Event-Driven Power Management

Tajana Šimunić, Luca Benini, *Member, IEEE*, Peter Glynn, and Giovanni De Micheli, *Fellow, IEEE*

***Abstract*—Energy consumption of electronic devices has become a serious concern in recent years. Power management (PM) algorithms aim at reducing energy consumption at the system-level by selectively placing components into low-power states. Formerly, two classes of heuristic algorithms have been proposed for PM: timeout and predictive. Later, a category of algorithms based on stochastic control was proposed for PM. These algorithms guarantee optimal results as long as the system that is power managed can be modeled well with exponential distributions. We show that there is a large mismatch between measurements and simulation results if the exponential distribution is used to model all user request arrivals. We develop two new approaches that better model system behavior for general user request distributions. Our approaches are event-driven and give optimal results verified by measurements. The first approach we present is based on renewal theory. This model assumes that the decision to transition to low-power state can be made in only one state. Another method we developed is based on the time-indexed semi-Markov decision process (TISMDP) model. This model has wider applicability because it assumes that a decision to transition into a lower-power state can be made upon each event occurrence from any number of states. This model allows for transitions into low-power states from any state, but it is also more complex than our other approach. It is important to note that the results obtained by renewal model are guaranteed to match results obtained by TISMDP model, as both approaches give globally optimal solutions. We implemented our PM algorithms on two different classes of devices: two different hard disks and client–server wireless local area network systems such as the SmartBadge or a laptop. The measurement results show power savings ranging from a factor of 1.7 up to 5.0 with insignificant variation in performance.**

***Index Terms*—Power consumption, stochastic processes, system analysis.**

I. INTRODUCTION

ENERGY consumption has become one of the primary concerns in electronic design due to the recent popularity of portable devices and environmental concerns related to desktops and servers. The battery capacity has improved very slowly (a factor of two to four over the last 30 years), while the computational demands have drastically increased over the same time frame. Better low-power circuit design techniques have helped to increase battery lifetime [1]–[3]. On the other hand, managing power dissipation at higher levels can considerably reduce energy consumption and, thus, increase battery lifetime [4].

Manuscript received September 18, 2000. This work was supported in part by the National Science Foundation under Grant CCR-9901190 and in part by the Marco Gigascale Research Center.

T. Šimunić and G. De Micheli are with the Computer Systems Laboratory, Stanford University, Stanford, CA 94305 USA.

L. Benini is with the Dipartimento di Elettronica, Informatica, e Sistemistica, University of Bologna, Bologna 10000, Italy.

P. Glynn is with the Management Science and Engineering Department, Stanford University, Stanford, CA 94305 USA.

Publisher Item Identifier S 0278-0070(01)04822-9.

System-level energy-conscious design is an effective way to reduce energy consumption. System-level *dynamic power management* [5] decreases the energy consumption by selectively placing idle components into lower power states. System resources can be modeled using state-based abstraction where each state trades off performance for power [6]. For example, a system may have an active state, an idle state, and a sleep state that has lower power consumption, but also takes some time to transition to the active state. The transitions between states are controlled by commands issued by a power manager (PM) that observes the workload of the system and decides when and how to force power state transitions. The PM makes state transition decisions according to the *power management policy.* The choice of the policy that minimizes power under performance constraints (or maximizes performance under power constraint) is a constrained optimization problem.

The most common power management policy at the system level is a *timeout policy* implemented in most operating systems (OSs). The drawback of this policy is that it wastes power while waiting for the timeout to expire [7], [8]. Predictive policies developed for interactive terminals [9], [10] force the transition to a low-power state as soon as a component becomes idle if the predictor estimates that the idle period will last long enough. An incorrect estimate can cause both performance and energy penalties. Both timeout and predictive policies are heuristic in nature and, thus, do not guarantee optimal results.

In contrast, approaches based on stochastic models can guarantee optimal results. Stochastic models use distributions to describe the times between arrivals of user requests (*interarrival times*), the length of time it takes for a device to service a user's request, and the time it takes for the device to transition between its power states. The system model for stochastic optimization can be described either with just memoryless distributions (exponential or geometric) [11]–[14] or with general distributions [15]–[18]. Power management policies can also be classified into two categories by the manner in which decisions are made: discrete time (or clock based) [11], [12] and event driven [13]–[18]. In addition, policies can be stationary (the same policy applies at any point in time) or nonstationary (the policy changes over time). All stochastic approaches except for the discrete adaptive approach presented in [12] are stationary. The optimality of stochastic approaches depends on the accuracy of the system model and the algorithm used to compute the solution. In both discrete and event-driven approaches optimality of the algorithm can be guaranteed since the underlying theoretical model is based on Markov chains. Approaches based on the discrete time setting require policy evaluation even when in low-power state [11], [12], thus wasting energy. On the other hand, event-driven models based on exponential distribution [13], [14] show little or no power savings when implemented in real systems since the exponential model does not describe well the request interarrival times of users [15]–[18].

In this paper, we introduce two new models for power management at the system level that enable modeling system transitions with general distributions, but are still event driven and guarantee optimal results. In order to verify our models, we implemented our power management algorithms on two different classes of devices: two different hard disks and client–server wireless local area network (WLAN) systems such as the SmartBadge [19] or a laptop. For each of these devices, we collected a set of traces that model typical user behavior well. We found the interarrival times between user requests are best modeled with a nonexponential distribution (a Pareto distribution shows the best fit, although our model applies to any distribution or direct data). These results are consistent with the observations on network traffic interarrival times presented in [20]. In addition, we measured the distributions of transition times between active, idle, and low-power states for each of the systems and found nonexponential transition times into or out of a low-power state. Traditional Markov chain models presented in previous work do not apply to these devices since user request arrivals and the transition times of a device are best modeled with nonexponential distributions. As a result, we formulated the policy optimization problem using two different stochastic approaches.

The first approach is based on renewal theory [21], [22]. It is more concise, but also is limited to systems that have only one decision state. The second approach is based on the time-indexed semi-Markov decision process (TISMDP) model. This model is more general, but also more complex. In both cases, the policy optimization problem can be solved *exactly* and in polynomial time by solving a linear program (LP). Clearly, since both approaches guarantee optimal solutions, they will give the same solution to a given optimization problem. Note that both approaches can handle general user request interarrival distributions, even though in the particular examples presented in this work we use the Pareto distribution since it showed a good fit to the data collected experimentally. The policy decisions are made only upon request arrival or upon finishing serving a request, instead of at every time increment as in discrete-time model. Since policy decisions are made in event-driven manner, more power is saved by not forcing policy re-evaluations as in discrete-time models.

We obtain globally optimal results for policy optimization using our models and in addition we present simulation and, more importantly, real measurement results. Our results show that the reduction in power can be as large as 2.4 times with a small performance penalty when power managing the laptop hard disk and 1.7 times for the desktop hard disk. This power reduction, which is compared against the Windows OS default timeout policy, is very significant and shows the overall benefits of our approach. Our algorithms perform better than any other power management algorithms tested in [23]. The measurements of optimal policy implemented on a laptop for the WLAN card show that the reduction in power can be as large as a factor of five with a small performance penalty. Finally, power management results on the SmartBadge show savings of as much as 70% in power consumption.

The remainder of the paper is organized as follows. Section III describes the stochastic models of the system components based on the experimental data collected. We develop the model for power management based on renewal theory in Section IV. Next, we present the TISMDP model for the dynamic power management policy optimization problem in Section V. We show simulation results for the SmartBadge, measured results for power managing WLAN card on a laptop, and both simulated and measured results for power managing a hard disk on a laptop and a desktop running Windows OS in Section VI. Finally, we summarize our findings and outline future directions of research in Section VII.

II. Related Work

The fundamental premise for the applicability of power management schemes is that systems or system components, experience nonuniform workloads during normal operation time. Nonuniform workloads are common in communication networks and in almost any interactive system. In the recent past, several researchers have realized the importance of power management for large classes of applications. Chip-level power management features have been implemented in mainstream commercial microprocessors [24]–[27]. Techniques for the automatic synthesis of chip-level power management logic are surveyed in [5].

Predictive policies for hard disks [28]–[32] and for interactive terminals [9], [10], [33] force the transition to a low-power state as soon as a component becomes idle if the predictor estimates that the idle period will last long enough. An incorrect estimate can cause both performance and energy penalties. The distribution of idle and busy periods for an interactive terminal is represented as a time series in [9] and approximated with a least-squares regression model. The regression model is used for predicting the duration of future idle periods. A simplified power management policy predicts the duration of an idle period based on the duration of the last activity period. The authors of [9] claim that the simple policy performs almost as well as the complex regression model and it is much easier to implement. In [10], an improvement over the prediction algorithm of [9] is presented, where idleness prediction is based on a weighted sum of the duration of past idle periods with geometrically decaying weights. The policy is augmented by a technique that reduces the likelihood of multiple mispredictions. All these policies are formulated heuristically, then tested with simulations or measurements to assess their effectiveness.

Another good example of heuristic power management policy is defined in the new IEEE 802.11 standard for wireless LAN at medium access control and physical layers [34]. The standard requires that a central access point (AP) send out a beacon every 100 ms followed by a traffic indication map (TIM). Each card that desires to communicate has to actively listen for the beacon in order to synchronize the clock with the AP and for the TIM to find out if any data is arriving for it. If it does not need to transmit or receive, the card can then go to the doze state until the next beacon. The IEEE standard does not address the need for power management at the system level. If the card is turned off when it is not being used, much larger power savings can be observed.

Stochastic models have also been introduced to obtain optimal power management algorithms. The optimality is guaranteed only under a set of assumptions that may or may not hold in real cases. Benini *et al.* [11] formulated a probabilistic system model based on discrete-time Markov decision processes (DTMDPs). They rigorously formulate the policy optimization problem and showed that it can be solved exactly and in polynomial time in the size of the system model. The DTMDP approach requires that all state transitions follow stationary geometric distributions, which is not true in many practical cases. Nonstationary user request rates can be treated using an adaptive policy interpolation procedure presented in [12]. A limitation of both stationary and adaptive DTMDP policies is that decision evaluation is repeated periodically, even when the system is idle, thus wasting power. For example, for a 10-W processor, the DTMDP policy with evaluation period of 1 s would waste as much as 1800 J of energy from the battery during a 30-min break. The advantage of the discrete time approach is that decisions are reevaluated periodically so the decision can be reconsidered, thus adapting better to arrivals that are not truly geometrically distributed.

An alternative to the DTMDP model is a continuous-time Markov decision process (CTMDP) model [13], [14]. In a CTMDP, the PM issues commands upon event occurrences instead of at discrete time settings. As a result, more energy can be saved since there is no need to continually reevaluate the policy in the low-power state. Results are guaranteed optimal assuming that the exponential distribution describes well the system behavior. Unfortunately, in many practical cases the transition times may be distributed according to a more general distribution. As a result, in real implementation the results are far from optimal [16]–[18]. Work presented in [14] uses series and parallel combinations of exponential distributions to approximate general distribution of transition times. Unfortunately, this approach is very complex and also gives a very poor approximation for the bursty behavior observed in real systems [16]–[18], [20]. In fact, the authors present only simulation results exclusively based on the exponential distribution.

In this paper, we present two new models for power management at the system level that accurately model system behavior, are event driven, and guarantee optimal results. In addition, we not only simulate, but also implement our power management policies on real systems, thus allowing us to get measurements of real power consumption. In the next section, we will develop our system model based on actual measurement results using realistic workloads.

III. System Model

In this paper, we focus on the systems that can be modeled with three components: the user, device, and the queue as shown in Fig. 1. While the methods presented in this paper are general, the optimization of energy consumption under performance constraints (or vice versa) is applied to and measured on two different classes of devices: two hard disks and client–server WLAN systems such as the SmartBadge [19] or a laptop [35]. The SmartBadge can be used as a corporate identity card, attached (or built in) to devices such as personal digital assistants (PDA) and mobile telephones, or incorporated in computing systems. In this paper, we use it as a PDA. The WLAN card is used as an internet access on the laptop computer. The hard disks are both part of Windows machines, one in the desktop and the other in the laptop. The queue models a memory buffer associated with each device. In all examples, the user is an application that accesses each device by sending requests via an OS.

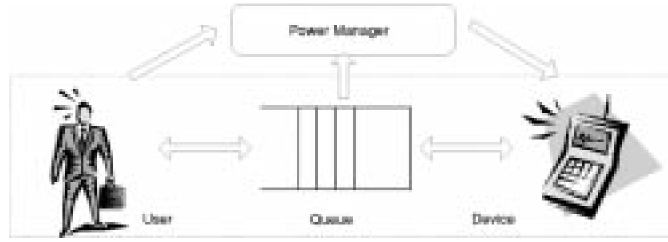


Fig. 1. System model.

Power management aims at reducing energy consumption in systems by selectively placing components into low-power states. Thus, at runtime, the PM observes user request arrivals, the state of the device's buffer, the power state, and the activity level of the device. When all user requests have been serviced, the PM can choose to place the device into a low-power state. This choice is made based on a policy. Once the device is in a low-power state, it returns to active state only upon arrival of a new request from a user. Note that a user request can come directly from a human user, from the OS, or even from another device.

Each system component is described probabilistically. The user behavior is modeled by a request interarrival distribution. Similarly, the service time distribution describes the behavior of the device in the active state. The transition distribution models the time taken by the device to transition between its power states. Finally, the combination of interarrival time distribution (incoming jobs to the queue) and service time distribution (jobs leaving the queue) appropriately characterizes well the behavior of the queue. These three categories of distributions completely characterize the stochastic optimization problem. The details of each system component are described in the next sections.

A. User Model

As the user's stochastic model is defined by the request interarrival time distribution, it is of critical importance to collect a good set of traces that do a good job of representing typical user behavior. We collected an 11-h user request trace for the PC hard disks running a Windows OS with standard software (e.g., Excel, Word, Visual C++). In the case of the SmartBadge, we monitored the accesses to the server during multiple long sessions. For the WLAN card, we used the *tcpdump* utility [36] to get the user request arrival times for two different applications (telnet and web browser).

The request interarrival times in the active state (the state where one or more requests are in the queue) for all three devices are exponential in nature. Fig. 2 shows the exponential cumulative distribution fitted to measured results of the hard disk. Similar results have been observed for the other two devices in the active state. Thus, we can model the user in active state with

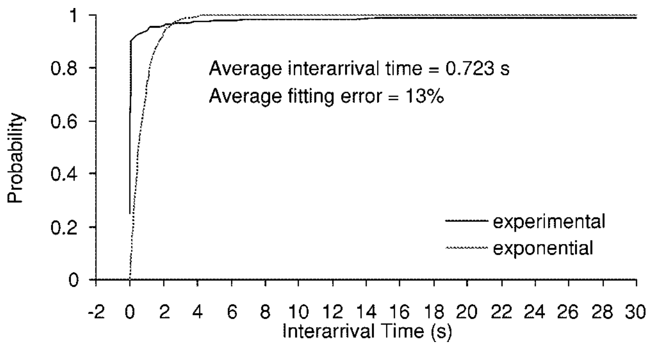


Fig. 2. User request arrivals in active state for hard disk.

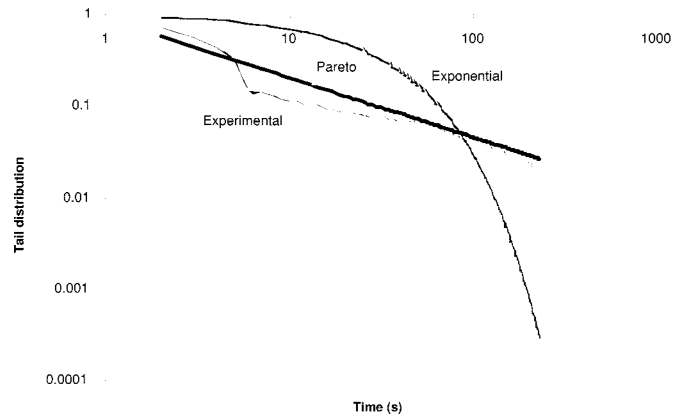


Fig. 3. Hard-disk idle state arrival tail distribution.

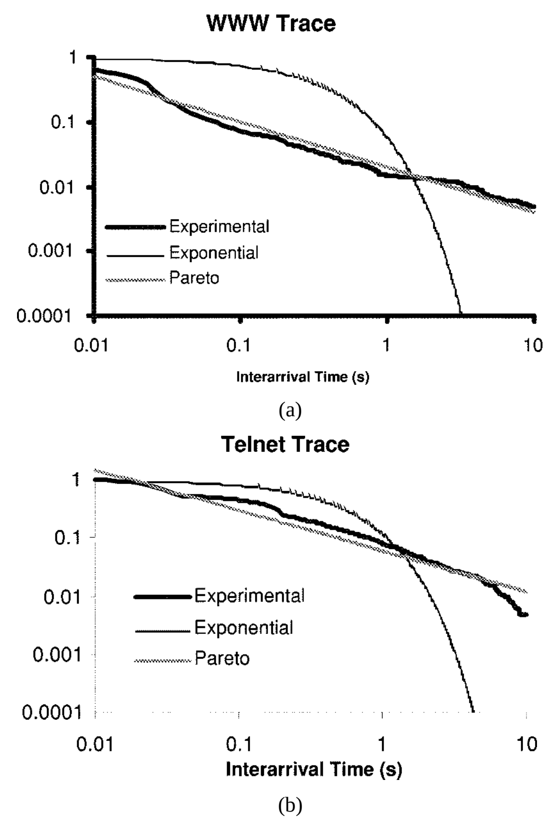


Fig. 4. WLAN idle state arrival tail distribution. (a) WWW trace. (b) Telnet trace.

rate λ_U and the mean request interarrival time $1/\lambda_U$, where the probability of the hard disk or the SmartBadge receiving a user request within time interval t follows the cumulative probability distribution shown below

$$F_U(t) = 1 - e^{-\lambda_U t}. \tag{1}$$

The exponential distribution does not model well arrivals in the idle state. The model we use needs to accurately describe the behavior of long idle times as the largest power savings are possible over the long low-power periods. We first filter out short user request interarrival times in the idle state in order to focus on the longer idle times. The filter interval is based on the particular device characteristics and not on the pattern of user access to the device. Filter interval is defined as a fraction of the *break-even* time of the device. Break-even time is the time the device has to stay in the low-power state in order to recuperate the cost of transitioning to and from the low-power state. Transitioning into a low-power state during idle times that are shorter than the break-even time is guaranteed to waste power. Thus, it is desirable to filter out very short idle times. We found that filter intervals from 0.5 s to about 2 s are most appropriate to use for the hard disk, while for the SmartBadge and the WLAN card filter intervals are considerably shorter (50–200 ms) since these devices respond much faster than the hard disk.

We use the tail distribution to highlight the probability of longer idle times that are of interest for power management. The tail distribution provides the probability that the idle time is greater than t. Fig. 3 shows the measured tail distribution of idle periods fitted with Pareto and exponential distributions for the hard disk and Fig. 4 shows the same measurements for the WLAN card. The Pareto distribution shows a much better fit for the long idle times as compared to the exponential distribution. The Pareto cumulative distribution is defined in (2). Pareto parameters are $a = 0.9$ and $b = 0.65$ for the hard disk, $a = 0.7$ and $b = 0.02$ for WLAN web requests, and $a = 0.7$ and $b = 0.06$ for WLAN telnet requests. SmartBadge arrivals behave the same way as the WLAN arrivals

$$F_U(t) = 1 - at^{-b}. \tag{2}$$

B. Portable Devices

Power-managed devices typically have multiple power states. Each device has one active state in which it services user requests and one or more low-power states. The PM can trade off power for performance by placing the device into low-power states. Each low-power state can be characterized by the power consumption and the performance penalty incurred during the transition to or from that state. Usually, higher performance penalty corresponds to lower power states.

1) SmartBadge: The SmartBadge shown in Fig. 5 is an embedded system consisting of a Sharp's display, WLAN RF link, StrongARM-1100 processor, Micron's SDRAM, FLASH, sensors, and modem/audio analog front-end on a printed circuit board powered by the batteries through a DC–DC converter. The initial goal in designing the SmartBadge was to allow a computer or a human user to provide location and environmental information to a location server through a heterogeneous network. The SmartBadge could be used as a corporate ID card,

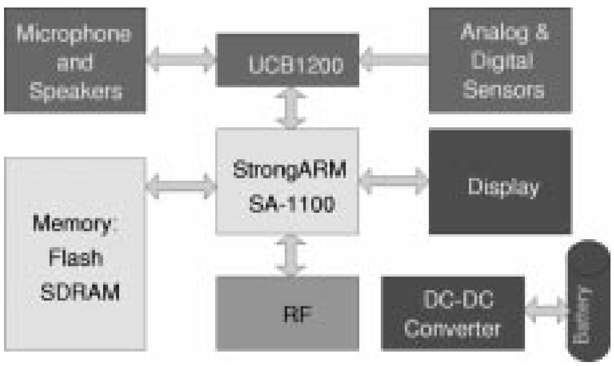


Fig. 5. SmartBadge.

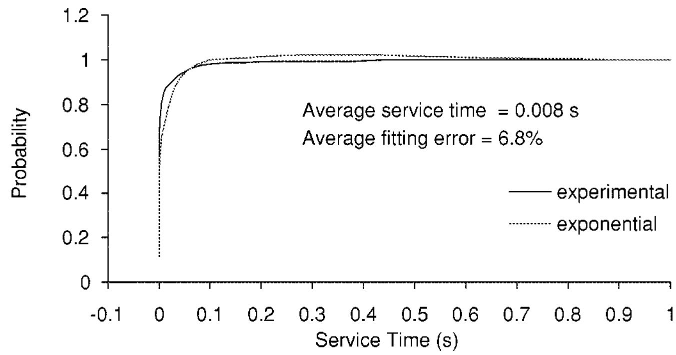


Fig. 6. Hard-disk service time distribution.

TABLE I
SMARTBADGE COMPONENTS

Component	Active Pwr (mW)	Idle Pwr (mW)	Standby Pwr (mW)	t_{sby} (ms)	t_{off} (ms)	t_{so} (ms)
Display	1000	1000	100	100	240	110
RF Link	1500	1000	100	40	80	20
SA-1100	400	170	0.1	10	35	10
FLASH	75	5	0.023	0.6	160	150
SRAM	115	17	0.13	5.0	100	90
DRAM	400	10	0.4	4.0	90	75
Total	3.5 W	2.2 W	200 mW	110 ms	705 ms	455 ms

attached (or built in) to devices such as PDAs and mobile telephones, or incorporated in computing systems. Both the SmartBadge and the WLAN card operate as a part of a client–server system. Thus, they initiate and end each communication session. The server just responds to their requests.

The SmartBadge supports three lower power states: idle, standby, and off. The idle state is entered immediately by each component in the system as soon as that particular component is not accessed. The standby and off state transitions can be controlled by the PM. The transition from standby or off state into the active state can be best described using the uniform probability distribution. Components in the SmartBadge, the power states, and the transition times of each component from standby ($t_{\rm sby}$) and off ($t_{\rm off}$) state into active state, and the transition times between standby and off states ($t_{\rm so}$) are shown in Table I. Note that the SmartBadge has two types of data memory—slower SRAM (1 MB, 80 ns) from Toshiba and faster DRAM (4 MB, 20 ns) from Micron that is used only during MPEG decode. Memory takes longer to transition from the off to the active state as contents of RAM have to be downloaded from FLASH and initialized. The power consumption of all components in the off state is 0 mW.

2) WLAN Card: The wireless card has multiple power states: two active states, *transmitting* and *receiving*, and two inactive states, *doze* and *off*. Transmission power is 1.65 W, receiving 1.4 W, and the power consumption in the doze state is 0.045 W [35] and in the off state it is 0 W. When the card is awake (not in the off state), every 100 ms it synchronizes its clock to the AP by listening to the AP beacon. After that, it listens to the TIM map to see if it can receive or transmit during that interval. Once both receiving and transmission are done, it goes into the doze state until the next beacon. This portion of the system is fully controlled from the hardware and, thus, is not accessible to the PM that has been implemented at the OS level.

The PM can control the transitions between the doze and the off states. Once in the off state, the card waits for the first user request arrival before returning back to the doze state. We measured the transitions between the doze and the off states using *cardmgr* utility. The transition from the doze state into the off state takes on average $t_{\rm ave} = 62$ ms with variance of $t_{var} = 31$ ms. The transition back takes $t_{\rm ave} = 34$ ms with $t_{var} = 21$ ms variance. The transition between doze and off states are best described using the uniform distribution.

3) Hard Disk: The Fujitsu MHF 2043AT hard disk in the Sony Vaio laptop we used in our experiments supports two states in which the disk is spinning—idle and active with average power consumption of 0.95 W. When the data is read or written, the power consumption is 2.5 W, but since the service rate is very high, the average power is 0.95 W. Service times on the hard disk in the active state most closely follow an exponential distribution as shown in Fig. 6. We found similar results for the SmartBadge and the WLAN card. The average service time is defined by $1/\lambda_D$, where λ_D is the average service rate. Equation (3) defines the cumulative probability of the device servicing a user request within time interval t

$$F_D(t) = 1 - e^{-\lambda_D t}. \tag{3}$$

The PM can control the transitions between the idle and the sleep state. The transition from the sleep to the active state requires spinup of the hard disk, which is very power intensive: 2.1 W. While in the sleep state, the disk consumes only 0.13 W. Once in the sleep state, the hard disk waits for the first service request arrival before returning to the active state. The transition between active and sleep states is best described using the uniform distribution, where t_0 and t_1 can be defined as $t_{\rm ave} - \Delta t$ and $t_{\rm ave} + \Delta t$, respectively. The cumulative probability function for the uniform distribution is shown below

$$F_D(t) = \begin{cases} 0 & t \le t_0 \\ \dfrac{t - t_0}{t_1 - t_0} & t_0 < t \le t_1 \\ 1 & t > t_1 \end{cases}. \tag{4}$$

Fig. 7 shows the large error that would be made if the transition to the sleep state were approximated using an exponential distribution. The transition from the active state into the sleep

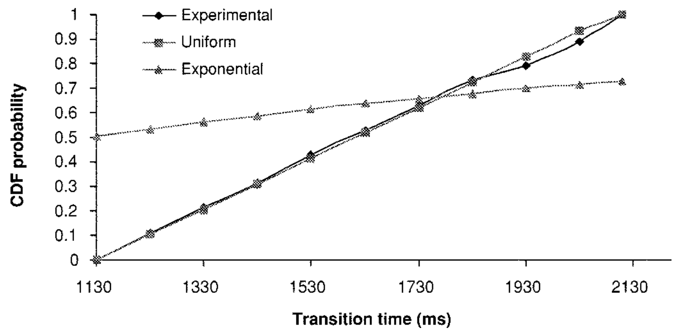


Fig. 7. Hard-disk transition from sleep to active state.

TABLE II
SYSTEM MODEL OVERVIEW

System Component	Component State	Distribution
User	Queue not empty	Exponential
	Queue empty	Pareto
Device	Active	Exponential
	Transition	Uniform

state takes on average 0.67 s with variance of 0.1 s. The transition back into the active state is much longer, requiring 1.6 s on average with 0.5 s variance.

C. Queue

Portable devices normally have a buffer for storing requests that have not been serviced yet. Since we did not have access to the detailed information about the real-time size of each queue, we measured the queue size of maximum ten jobs with an experiment on a hard disk using a typical user trace. Because the service rate in the SmartBadge and WLAN card is higher and the request arrival rate is comparable, we assume that the same maximum queue size can be used. As the requests arriving at the hard disk do not have priority associated with them and the SmartBadge requests by definition do not have priority, our queue model contains only the number of jobs waiting for service. Active and low-power states can be differentiated then by the number of jobs pending for service in the queue.

D. Model Overview

Table II shows the probability distributions used to describe each system component derived from the experimental results. User request interarrival times with at least one job in the queue are best modeled with the exponential distribution. On the other hand, we have shown that in all four applications, the Pareto distribution is best used to model the arrival of the user's requests when the queue is empty. Note that the queue is empty in either the idle state or a low-power state. The device is in the active state when at least one job is waiting to be serviced. We have also shown that the service times in active state are best modeled with the exponential distribution. The transitions to and from low-power states are better modeled with a uniform distribution. The combination of these distributions is used to derive the state of the queue. Thus, in the active state two exponential distributions define the number of jobs in the queue: the interarrival time and the service time distributions. During transitions, the queue state is defined by the transition distribution and the distribution describing user request arrivals. During transitions and in the low-power states, the first arrival follows the Pareto distribution, but the subsequent arrivals are modeled with the exponential distribution since for very short interarrival times the exponential distribution is very close to the Pareto distribution and the experimental results, as can be seen in Figs. 3 and 4.

Although in the experimental section of this paper we utilize the fact that nonexponential user and device distributions can be described with well-known functions (Pareto or uniform), the models we present are general in nature and, thus, can give optimal results with both experimental distributions obtained at runtime or commonly used theoretical distributions. We found that in the particular examples we present in this work Pareto and uniform distributions enabled us to obtain the optimal policy faster without sacrificing accuracy.

IV. POWER MANAGEMENT BASED ON RENEWAL THEORY

Renewal theory [21], [22] studies stochastic systems whose evolution over time contains a set of *renewals or regeneration times*, where the process begins statistically anew. Formally, a renewal process specifies that the random times between system renewals be independently distributed with a common distribution $F(x)$. Thus, the expected time between successive renewals can be defined as

$$E[\tau] = \int_0^\infty x\, dF(x). \tag{5}$$

Note that the Poisson process is a simple renewal process for which renewal times are distributed with the exponential distribution. In this case, the common distribution between renewals can be defined as $F(x) = 1 - e^{-\lambda x}$ and the mean time between renewals (or between exponential arrivals) is defined as $E[\tau] = 1/\lambda$. A process can be considered to be a renewal process only if there is a state of the process in which the whole system probabilistically restarts. This, of course, is the case in any system that is completely described by exponential or geometric distributions, since those distributions are not history dependent (they are memoryless).

In policy optimization for dynamic power management, the complete cycle of transition from the idle state, through the other states, and then back into the idle state can be viewed as one renewal of the system. When using renewal theory to model the system, the decision regarding transition to a lower power state (e.g., sleep state) is made by the PM in the idle state. If the decision is to transition to the lower power state, the system reenters the idle state after traversing through a set of states. Otherwise, the system transitions to the active state on the next job arrival and then returns to the idle state again once all jobs have been serviced.

The general system model shown in Fig. 1 defines the PM and three system components: user, device, and the queue. To provide concreteness in our examples, each component is

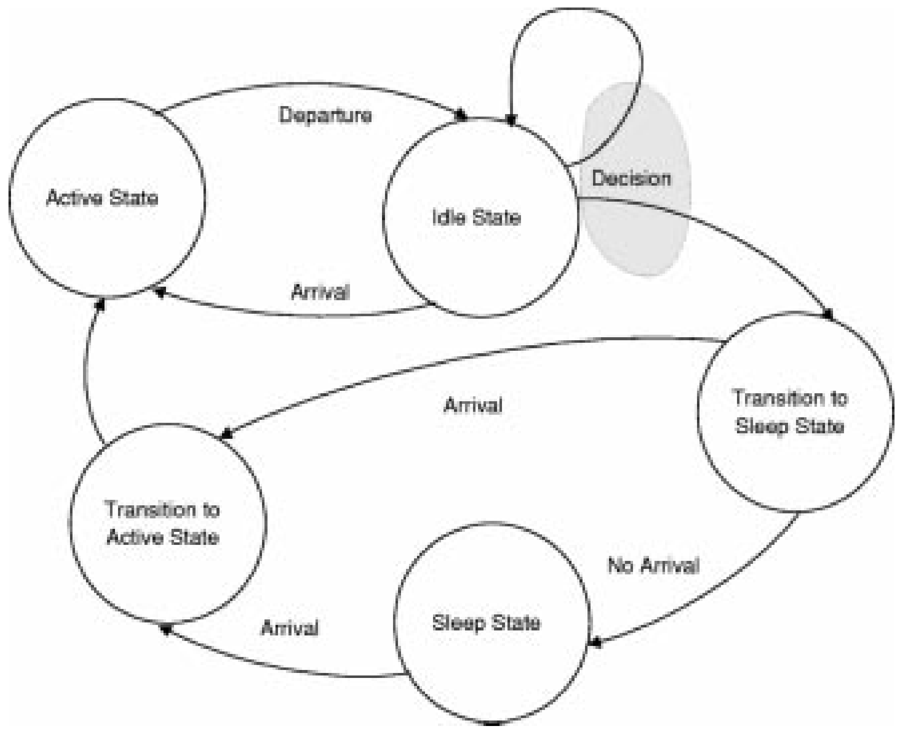


Fig. 8. System states for renewal theory model.

completely specified by the probability distributions defined in the previous section. With renewal theory, the search for the best policy for a system modeled using stationary non-exponential distributions can be cast into a stochastic control problem. System states used in the formulation of the renewal theory model are shown in Fig. 8. In the active state, the queue contains at least one job pending and the request arrivals and service times follow exponential distributions. Once the queue is emptied, the system transitions to the idle state, which is also a renewal and decision point in this system. Upon arrival of request, the system always transitions back into the active state. The PM makes a decision on when the transition to a low-power state from the idle state should occur. As soon as the command to place the system into the low-power state is given, the system starts a transition between the idle and the low-power states. The transition state highlights the fact that device takes a finite and random amount of time to transition into the low-power state (governed by a uniform distribution). If during the transition time a request arrives from the user (first request follows Pareto distribution, subsequent requests are exponential), the device starts the transition to active state as soon as the transition to off state is completed. If no request arrives during the transition state, the device stays in a low-power state until the next request arrives (Pareto distribution). Upon request arrival, the transition back into the active state starts. Once the transition into the active state is completed, the device services requests, and then again returns to the idle state where the system probabilistically renews again.

A. Renewal Theory Model

We formulate the power management policy optimization problem based on renewal theory in this section. We use upper-case bold letters (e.g., $\mathbf{M}$) to denote matrices, lower-case bold letters (e.g., $\mathbf{v}$) to denote vectors, calligraphic letters (e.g., $\mathcal{S}$) to denote sets, upper-case letters (e.g., S) to denote scalar constants, and lower-case letters (e.g., s) to denote scalar variables.

The problem of power management policy optimization is to determine the optimal distribution of the random variable Γ that specifies when the transition from the idle state to low-power state should occur based on the last entry into the idle state. We assume that Γ takes on values in $[0, h, 2h, \ldots, jh, \ldots, Nh)$, where j is an index, h is a fraction of the break-even time of the device, and N is the maximum time before the system goes to a low-power state (usually set to an order of magnitude greater than break-even time). The solution to the policy optimization problem can be viewed as a table of probabilities (Γ), where each element $p(j)$ specifies the probability of transition from idle to a low-power state indexed by time values jh.

We can formulate an optimization problem to minimize the average performance penalty under a power constraint ($P_{\rm constraint}$), using the results of the ratio limit theorem for renewal processes [22], as shown in (6). The average performance penalty is calculated by averaging $q(j)$, the time penalty user incurs due to transition to low-power state, over $t(j)$, the expected time until renewal. The power constraint is shown as an equality as the system will use the maximum available power in order to minimize the performance penalty. The expected energy ($\sum_j p(j)e(j)$) is calculated using $p(j)$, the probability of issuing command to go to low-power state at time jh, and $e(j)$, the expected energy consumption. This expected energy has to equal the expected power constraint ($\sum_j p(j)t(j)P_{\rm constraint}$) calculated using the expected time until renewal $t(j)$, the power constraint $P_{\rm constraint}$, and $p(j)$.

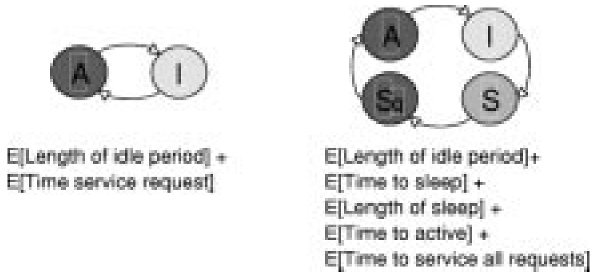


Fig. 9. Renewal cycles.

The unknown in the optimization problem is $p(j)$, the probability of issuing a command to go to low-power state at time jh. The full derivation of all the quantities follows:

$$\begin{aligned} &\min \frac{\sum_j p(j)q(j)}{\sum_j p(j)t(j)} \\ &\text{s.t. } \sum_j p(j)[e(j) - t(j)P_{\text{constraint}}] = 0 \\ &\qquad \sum_j p(j) = 1 \\ &\qquad p(j) \geq 0 \qquad \forall j. \end{aligned} \tag{6}$$

1) Computation of Renewal Time: Given the state space shown in Fig. 8, we can define the expected time until renewal $t(j)$ as follows. We define β as the time at which the first job arrives after the queue has been emptied. The first arrival is distributed using general probability distribution $P(jh)$. We also define the indicator function $I(jh)$, which is equal to one if we are in interval jh and is zero otherwise.

Further, as we showed in Section III, the subsequent user request arrivals follow a Poisson process with rate λ_U. Finally, the servicing times of the device also can be described using exponential distribution with parameter λ_D. We can now define the expected time until renewal for each time increment spent in the idle state as the sum of expected time until renewal if arrival comes before the system starts transitioning into low-power state jh [as shown by the first cycle in Fig. 9 and the first half of (7)] and if the arrival comes after the transitions has already started [the second parts of Fig. 9 and (7)]

$$\begin{aligned} t(j) = &E[t(j)I(\beta \leq jh)|\Gamma = jh] \\ &+ E[t(j)I(\beta > jh)|\Gamma = jh]. \end{aligned} \tag{7}$$

Each of the two terms in (7) is defined in (8) and (10). Note that Fig. 9 shows the components of each of the two terms. The expected time until renewal for arrival coming before transitioning to low-power state at time jh (the left portion of Fig. 9) is the expected time until arrival [the first term in (8)] and the time needed to work off the request that just arrived (the second term). Note that the second term can be computed based on the results from $M/M/1$ queueing theory due to the fact that the time to work off the request is governed by the exponential distribution with rate λ_D, while the arrivals in the active state are described by the exponential distribution with rate λ_U. The job departure rate has to be larger than the arrival rate ($\lambda_D \geq \lambda_U$), otherwise the queue would overflow. In all cases we studied, λ_D is at least order of magnitude larger than λ_U, leading to

$$\begin{aligned} E[\tau_j I(\beta \leq jh)|\Gamma = jh] = &\sum_{k=1}^{j} khP(\beta = kh) \\ &+ \frac{1}{\lambda_D - \lambda_U} P(\beta \leq jh). \end{aligned} \tag{8}$$

If the arrival comes after time jh when the system starts the transition to low-power state (the right portion of Fig. 9), then the expected time until renewal is the sum of the time until arrival (jh) with expected times for transition to low-power state and back to active states (EU_1, EU_2), expected length of the low-power period and the expected time to work off requests that arrived during the renewal period

$$\begin{aligned} &E[\tau_j I(\beta > jh)|\Gamma = jh] = P(\beta > jh) \\ &\quad \cdot \begin{bmatrix} jh + EU_1 + EU_2 \\ \quad + E[(\beta - jh + EU_1)I(\beta > jh + EU_1)] \\ \quad + E[(jh + EU_1 - \beta)I(jh < \beta \leq jh + EU_1)] \\ \quad \cdot \frac{\lambda_D}{\lambda_D - \lambda_U} + \frac{1}{\lambda_D - \lambda_U} + EU_2 \frac{\lambda_D}{\lambda_D - \lambda_U} \end{bmatrix}. \end{aligned} \tag{9}$$

2) Computation of Costs: We can define the performance penalty that the user experiences due to transition to low-power state [$q(j)$] and the expected energy consumption [$e(j)$] for each state using the same set of equations, just with different values for constants (c) as shown in Table III. Each state is labeled on the left side, while the expected time spent in that state multiplied by the constant c is on the right side.

The constants (c) equal the power consumption in a given state for energy consumption computation. For the performance penalty, the constants should be set to zero in low-power state and idle state and to one in all other states. For example, the constant c_i is set to power consumption of the device while in the idle state when calculating energy consumption (the first equation). Since there is no performance penalty to servicing users requests in the idle state, the constant c_i is set to zero for performance penalty calculation. On the other hand, the transition to the active state causes performance degradation, thus the constant c_{ta} is here set to one. The same constant is set to power required for the transition to the active state when calculating energy consumption.

The expected times spent in each state outlined in Table III are calculated as follows:

1) *Idle State:* The expected time spent in the idle state is the expected average of the idle time until the first request arrival [$\sum_{k=0}^{j} khP(\beta = kh)$] and the time spent in the idle state when the transition to low-power state occurs before the first arrival ($jhP(\beta \geq jh)$).
2) *Transition to Low-Power State:* The transition to low-power state occurs only if there has been no request arrival before the transition started [$P(\beta \geq jh)$]. The expected average time of the transition to low-power state is defined by the average of the uniform distribution that describes the transition (EU_1).
3) *Low-Power State:* Low-power state is entered only if no request arrival occurred while in the idle state ($P(\beta \geq$

TABLE III
CALCULATION OF COSTS

State	Performance penalty or Energy consumption
Idle	$c_i[\sum_{k=0}^{j} khP(\beta = kh) + jhP(\beta \geq jh)]$
To Low Power	$c_{ts}[EU_1 P(\beta \geq jh)]$
Low Power	$c_s[E[\beta - (jh + EU_1)]I(\beta > jh + EU_1)P(\beta \geq jh)]$
To Active	$c_{ta}[EU_2 P(\beta \geq jh)]$
Active	$c_a[\frac{1}{\lambda_D - \lambda_U} P(\beta \leq jh) +$ $EU_2 \frac{\lambda_D}{\lambda_D - \lambda_U} P(\beta \geq jh) +$ $\frac{1}{\lambda_D - \lambda_U} P(\beta \geq jh) +$ $E[(jh + EU_1 - \beta)I(jh \leq \beta \leq jh + EU_1)] \frac{\lambda_D}{\lambda_D - \lambda_U} P(\beta \geq jh)]$

jh)). The device stays in that state until the first request arrives ($E[\beta - (jh + EU_1)]I(\beta > jh + EU_1)$).

4) *Transition to Active State:* The transition to active state occurs only when there is a successful transition to low-power state ($P(\beta \geq jh)$). The transition length is the expected average of uniform distribution LP describes the transition to active state (EU_2).

5) *Active State:* The device works off the request that arrived in the idle state if no transition to low-power state occurred ($[1/\lambda_D - \lambda_U]P(\beta < jh)$). If the transition to low-power state did occur [terms containing $P(\beta \geq jh)$], then the system is in the active state for the time it takes to work off all the requests that arrived while transitioning between idle, low-power, and active states.

3) Problem Formulation: The optimization problem shown in (6) can be transformed into an LP using intermediate variables $y(j) = [p(j)/\sum_j p(j)t(j)]$ and $z(j) = 1/\sum_j p(j)t(j)$ [37]

$$\begin{aligned} \mathbf{LP}:\ \min\ & \sum_j q(j)y(j) \\ \text{s.t.}\ & \sum_j [e(j)y(j) - t(j)z(j)P_{\text{constraint}}] = 0 \\ & \sum_j t(j)y(j) = 1 \\ & z \geq 0. \end{aligned} \tag{10}$$

Once the values of intermediate variables $y(j)$ and $z(j)$ are obtained by solving the LP shown above, the probability of transition to low-power state from idle state at time jh, $p(j)$ can be computed as follows:

$$p(j) = \frac{y(j)}{z(j)}. \tag{11}$$

B. Policy Implementation

The optimal policy obtained by solving the LP given in (10) is a table of probabilities $p(j)$. The policy can be implemented in two different ways. If the probability distribution defined by $p(j)$ is used, then on each interval jh, the policy needs to be reevaluated until either a request arrives or the system transitions to a low-power state. This implementation has a high overhead as it requires multiple reevaluations. An alternative implementation gives the same results, but it requires only one evaluation upon entry to idle state. In this case a table of cumulative probabilities $P(j)$ is calculated based on the probability distribution described with $p(j)$. Once the system enters the idle state, a pseudorandom number RND is generated and normalized. The time interval for which the policy gives the cumulative probability $P(j)$ of going to the low-power state greater than RND is the time when the device will be transitioned into the low-power state. Thus, the policy works like a randomized timeout. The device stays in the idle state until either the transition to the low-power state as given by RND and the policy or until a request arrival forces the transition into the active state. Once the device is in the low-power state, it stays there until the

TABLE IV
SAMPLE POLICY

Idle Time	Transition Probability
j	P(j)
0	0
1	0.1
2	0.4
3	0.9
4	1.0

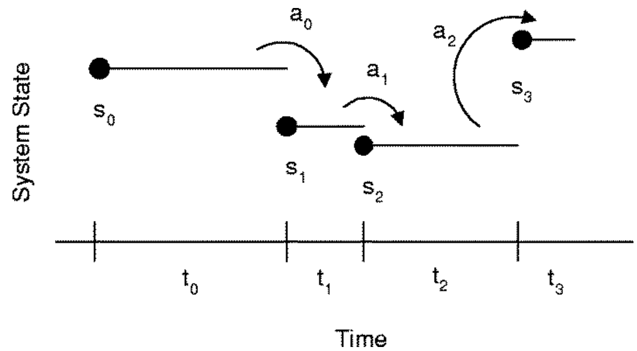


Fig. 10. SMDP progression.

first request arrives, at which point it transitions back into the active state.

Example IV.1: If a sample policy is given in Table IV and the pseudorandom number RND generated upon entry to idle state is 0.6, then the PM will give a command to transition the device to low-power state at time indexed by $j = 3$. Thus, if the time increment used is 0.1 s, then the device will transition into low-power state once it has been idle for 0.3 s. If a user request arrives before 0.3 s have expired, then the device transitions back to the active state.

V. POWER MANAGEMENT BASED ON TISMDP

In this section, we present the power management optimization problem formulation based on TISMDP. This model is more general than the model based on renewal theory as it enables multiple decision points (see Example V.1). Our goal is to minimize the performance penalty under an energy consumption constraint (or vice versa). We first present the average-cost semi-Markov decision process (SMDP) optimization problem [38] and then extend it to the TISMDP for modeling general interarrival times.

Example V.1: The SmartBadge has two states where decisions can be made: idle and standby. The idle state has higher power consumption, but also a lower performance penalty for returning to the active state, as compared to the standby state. From the idle state, it is possible to give a command to transition to the standby or the off states. From standby, only a command to transition to the off state is possible. The optimal policy determines when the transition between idle, standby, and off states should occur.

At each event occurrence, the PM issues a *command* (or *action*) that decides the next state to which the system should transition. In general, commands given are functions of the state history and the policy. Commands are modeled by decisions, which can be deterministic or randomized. In the former case, a decision implies issuing a command. In the later case, a decision gives the probability of issuing a command. The decisions taken by the PM form a discrete sequence $[\delta^{(1)}, \delta^{(2)}, \ldots)$. The sequence completely describes the PM *policy* π, which is the unknown of our optimization problem. Among all policies two classes are particularly relevant, as defined next.

Definition V.1: Stationary policies are policies where the same decision $\delta^{(i)} = \delta$ is taken at every decision point t_i, $i = 1, 2, \ldots$, i.e., $\pi = [\delta, \delta, \ldots)$.

For stationary policies, decisions are denoted by δ, which is a function of the system state s. Thus, stationarity means that the *functional dependency* of δ on s does not change over time. When s changes, however, δ can change. Furthermore, notice that even a constant decision *does not* mean that the *same command* is issued at every decision point. For randomized policies, a decision is a probability distribution that assigns a probability to each command. Thus, the actual command that is issued is obtained by randomly selecting from a set of available commands with the probabilities specified by δ.

Definition V.2: Markov stationary policies are policies where decisions δ do not depend on the entire history but only on the state of the system s at the current time.

Randomized Markov stationary policies can be represented as a $S \times A$ decision matrix $\mathbf{P}_\pi$. An element $p_{s,a}$ of $\mathbf{P}_\pi$ is the probability of issuing command a given that the state of the system is s. *Deterministic* Markov stationary policies can still be represented by matrices where only one element for each row has value one and all other elements are zero. The importance of these two classes of policies stems from two facts: first, they are *easy to store and implement*, second, we will show that for our system model, *optimal policies belong to these classes.* In the next sections, we will first present the average-cost SMDP, followed by the extension to TISMDP.

A. Semi-Markov Average-Cost Model

SMDP generalize Markov decision processes by allowing the decision maker to choose actions whenever the system state changes, to model the system evolution in continuous time, and to allow the time spent in a particular state to follow an arbitrary probability distribution. CTMDP [13], [15] can be viewed as a special case of SMDPs in which the intertransition times are always exponentially distributed. Fig. 10 shows a progression of the SMDP through event occurrences, called decision *epochs.* The PM makes decisions at each event occurrence. The *interevent time set* is defined as $\mathcal{T} = \{t_i,$ s.t. $i = 0, 1, 2, \ldots, i_{\max}\}$, where each t_i is the time between two successive event arrivals and $i_{\max}$ is the index of the maximum time horizon. We denote by $s_i \in \mathcal{S}_i$ the system state at decision epoch i. Commands are issued whenever the system state changes. We denote by $a_i \in \mathcal{A}$ an action that is issued at decision epoch i. When action a_i is chosen in system state s_i, the probability that the next event will occur by time t_i is defined by the cumulative probability distribution

$F(t_i|s_i, a_i)$. Also, the probability that the system transitions to state s_{i+1} at or before the next decision epoch t_i is given by $p(s_{i+1}|t_i, s_i, a_i)$.

The SMDP model also defines cost metrics. The average cost incurred between two successive decision epochs (events) is defined in (12) as a sum of the lump sum cost $k(s_i, a_i)$ incurred when action a_i is chosen in state s_i in addition to the cost in state s_{i+1} incurred at rate $c(s_{i+1}, s_i, a_i)$ after choosing action a_i in state s_i. We dfine $\mathcal{S}_{i+1}$ as the set of all possible states that may follow s_i

$$cost(s_i, a_i, u) = k(s_i, a_i) + \int_0^\infty \left[F(du|s_i, a_i) \sum_{s_{i+1} \in \mathcal{S}_{i+1}} \int_0^u \cdot c(s_{i+1}, s_i, a_i) p(s_{i+1}|t_i, s_i, a_i) \right] dt. \quad (12)$$

We can define the total expected cost for policy π until time t as a sum of all lump sum costs $k_v(s, a)$ up to time t and the costs incurred at the rate $c(s, a)$ while in each state s until time t

$$v_t^\pi(s) = E_s^\pi \left\{ \int_0^t c(s, a, u)\, du + \sum_{v=v_0^\pi}^{v_{i-1}^\pi} k_v(s, a) \right\} \quad (13)$$

and then we can define the average expected cost for all s

$$g^\pi(s) = \lim_{t \to \infty} \inf \frac{v_t^\pi(s)}{t}. \quad (14)$$

Theorem V.1: Finding the optimal power management policy minimizing (14) is equivalent to solving the following problem:

$$h(s) = \min_{a \in A} \left\{ cost(s, a) - g(s) y(s, a) + \sum_{j \in S} m(j|s, a) h(j) \right\} \quad (15)$$

where $h(s)$ is the so-called bias (the difference between long-term average cost and the average cost per period for a system in steady–state [38]), $g(s)$ is the average cost, $m(j|s, a)$ is the probability of arriving to state j given that the action a was taken in state s is defined by

$$m(j|s, a) = \int_0^\infty p(j|t, s, a) F(dt|s, a) \quad (16)$$

and expected time spent in each state is given by

$$y(s, a) = \int_0^\infty t \sum_{s \in S} p(j|t, s, a) F(dt|s, a). \quad (17)$$

Proof of Theorem V.1 is given in [38].

The following examples illustrate how the probability, the expected time and energy consumption can be derived.

Example V.2: In the active state with at least one element in the queue, we have two exponential random variables, one for the user with parameter λ_U and one for the device with parameter λ_D. The probability density function (pdf) of the jointly exponential user and device processes defines an $M/M/1$ queue and, thus, can be described by $F(dt|s, a) = \lambda e^{-\lambda t} dt$, where $\lambda = \lambda_U + \lambda_D$. In the same way, the probabilities of transition in $M/M/1$ queue $p(j|t, s, a)$ are defined as λ_U/λ for request arrival and λ_D/λ for request departure.

Using (16), we derive that the probability of transition to the state that has an additional element in the queue is λ_U/λ , while the probability of transition to the state with one less element is given by λ_D/λ. Note that in this special case, $p(j|t, s, a) = m(j|s, a)$. The expected time for transition derived using (17) is given by $1/\lambda$, which is again characteristic of $M/M/1$ queue. Energy consumption is given in (12). For this specific example, we define the power consumption in active state with P_a and we assume that there is no fixed energy cost for transition between active states. Then, the energy consumption can be computed as follows: $cost(s, a) = \int_0^\infty \lambda e^{-\lambda t}\, dt[\int_0^t P_a \lambda_D/\lambda du + \int_0^t P_a \lambda_U/\lambda\, du]$, which is equal to P_a/λ. Note that this solution is very intuitive, as we would expect the energy consumption to equal the product between the power consumption and the expected time spent in the active state.

The second example considers the transition from the sleep state into the active state with one or more elements in the queue.

Example V.3: The transition from sleep to active state is governed by two distributions. A uniform distribution describes device transitions: $F_U(dt|s, a) = dt/(t_{\max_a s} - t_{\min_a s})$, where $t_{\max_a s}$ and $t_{\min_a s}$ are maximum and minimum transition times. The request arrival distribution is exponential: $F_E(dt|s, a) = \lambda_U e^{-\lambda_U t} dt$. The probability of no arrival during the transition is given by $p(j|t, s, a) = e^{-\lambda_U t}$.

The probability of transition from the sleep state with a set number of queue elements into an active state with the same number of elements in the queue is given by $m(j|s, a) = \int_{t_{\min_a s}}^{t_{\max_a s}} [e^{-\lambda_U t}/(t_{\max_a s} - t_{\min_a s})]\, dt$. The expected transition time $y(s, a)$ is given by $(t_{\max_a s} + t_{\min_a s})/2$, which can be derived with (17). Finally, the energy consumed during the transition is defined by $cost(s, a) = \int_0^\infty [du/t_{\max_a s} - t_{\min_a s}] \int_0^u P_{sa}\, dt$, assuming that there is no fixed energy consumed during the transition and that the power consumption for the transition is given by P_{sa}. The energy consumption can further be simplified to be $[2P_{sa}/t_{\max_a s} + t_{\min_a s}]$. This is again equal to the product of power consumption with the expected transition time from the sleep state into the active state.

The problem defined in Theorem V.1 can be solved using policy iteration or by formulating and solving an LP. There are two main advantages of a linear programming formulation: additional constraints can be added easily and the problem can be solved in polynomial time (in $S \cdot A$). The primal LP derived from (15) defined in Theorem V.1 can be expressed as follows:

$$\begin{aligned} \mathbf{LPP:}\ & \min g(s) \\ & \text{s.t. } g(s)y(s, a) + h(s) - \sum_{j \in S} m(j|s, a) h(j) \\ & \qquad \geq \mathrm{cost}(s, a) \qquad \forall\, s, a \end{aligned} \quad (18)$$

where

s and a	state and command given in that state;
$g(s)$	average cost;
$h(s)$	bias;
$y(s, a)$	expected time;
$\mathrm{cost}(s, a)$	expected cost (e.g., energy);
$p(j\|s, a)$	transition probability between the two states.

Because the constraints of LPP are convex in $g(s)$ and the Lagrangian of the cost function is concave, the solution to the

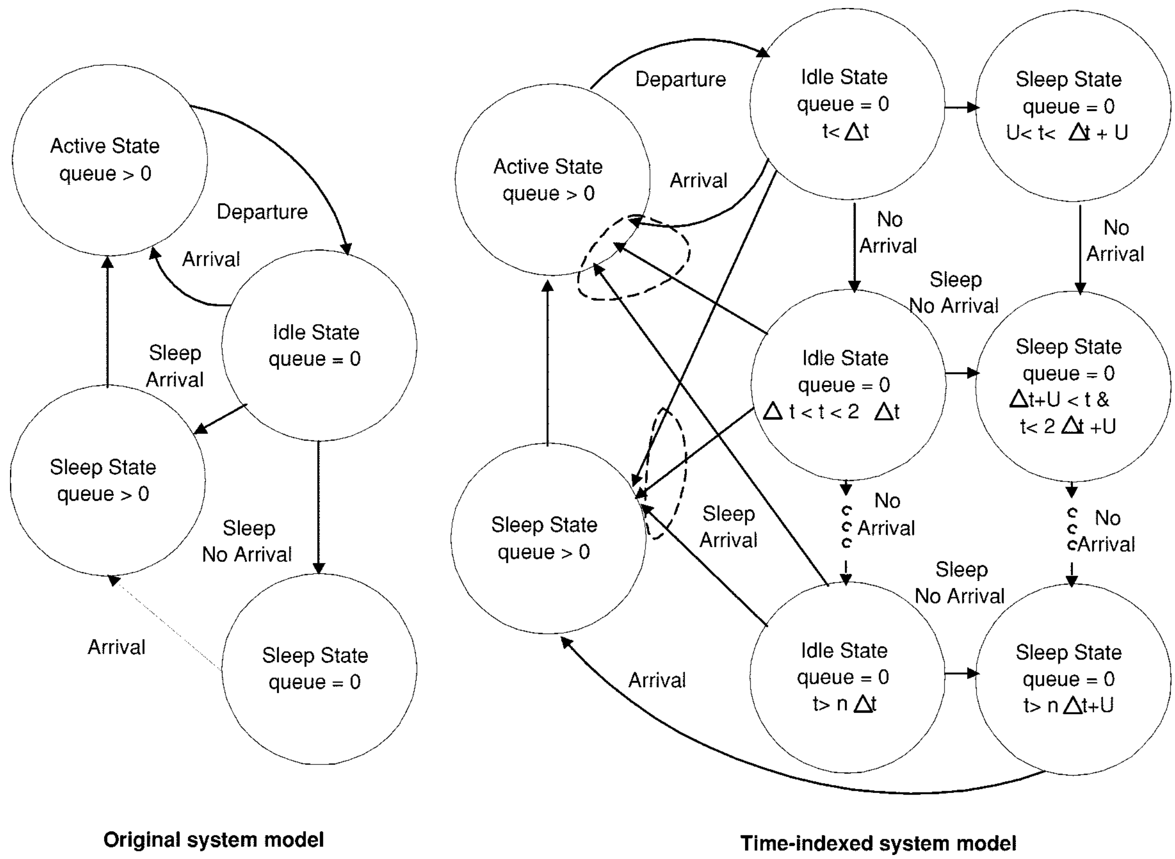


Fig. 11. Time-indexed SMDP states.

primal LP is convex. In fact, the constraints form a polyhedron with the objective giving the minimal point within the polyhedron. Thus, the *globally optimal* solution can be obtained that is both stationary and deterministic. The dual LP shown in (19) is another way to cast the same problem (in this case with the addition of a performance constraint). The dual LP shows the formulation for minimizing energy under performance constraint (opposite problem can be formulated in much the same way)

$$\begin{aligned}\mathbf{LPD}\colon \min &\sum_{s\in S}\sum_{a\in A} \mathrm{cost}_{\mathrm{energy}}(s, a) f(s, a)\\ \text{s.t. } &\sum_{a\in A} f(s, a) - \sum_{s'\in S}\sum_{a\in A} m(s'\,|s, a) f(s', a) = 0\\ &\sum_{s\in S}\sum_{a\in A} y(s, a) f(s, a) = 1\\ &\sum_{s\in S}\sum_{a\in A} \mathrm{cost}_{\mathrm{perf}}(s, a) f(s, a) < \mathrm{constraint}.\end{aligned} \tag{19}$$

The $A \cdot S$ unknowns in the LPD $f(s, a)$, called *state-action frequencies,* are the expected number of times that the system is in state s and command a is issued. It has been shown that the exact and the optimal solution to the SMDP policy optimization problem belongs to the set of Markovian randomized stationary policies [38]. A *Markovian randomized stationary policy* can be compactly represented by associating a value $x(s, a) \leq 1$ with each state and action pair in the SMDP. The probability of issuing command a when the system is in state s, $x(s, a)$ is defined as

$$x(s_i, a_i) = \frac{f(s_i, a_i)}{\displaystyle\sum_{a_i \in A} f(s_i, a_i)}. \tag{20}$$

B. *Time-Indexed Semi-Markov Average Cost Model*

The average-cost SMDP formulation presented above is based on the assumption that at most, one of the underlying processes in each state transition is not exponential in nature. On transitions where none of the processes are exponential, time-indexed Markov chain formulation needs to be used to keep the history information. Without indexing, the states in the Markov chain would have no information on how much time has passed. As for all distributions, but the exponential, the history is of critical importance, the state–space has to be expanded in order to include the information about time as discussed in [21]. Time-indexing is done by dividing the time line into a set of intervals of equal length Δt. The original state–space is expanded by replacing one idle and one low-power state with a series of time-indexed idle and low-power states as shown in Fig. 11. The expansion of idle and low-power states into time-indexed states is done only to aid in deriving in the optimal policy. A TISMDP can contain nonindexed states. Once the policy is obtained, the actual implementation is completely event-driven in contrast to the policies based on DTMDPs. Thus, all decisions are made upon event occurrences. So, the decision to go to a low-power state is made once upon entry to the idle state as discussed in Section IV-B. Other events are user request arrivals or service completions. Note that the technique we present is general, but in this paper, we will continue to refer to the examples shown in Section III.

If an arrival occurs while in the idle state, the system transitions automatically to the active state. When no arrival occurs during the time spent in a given idle state, the PM can choose to either stay awake in which case the system enters the next idle state or to transition into the low-power state. When the transition to the low-power state occurs from an idle state, the system can arrive to the low-power state with the queue empty or with jobs waiting in the queue. The low-power state with queue empty is indexed by the time from first entry into the idle state from the active state, much in the same way idle states are indexed, thus allowing accurate modeling of the first arrival. The LP formulation for average-cost SMDP still holds, but the cost, the probability, and the expected time functions have to be redefined for time-indexed states in SMDP. Namely, for the time-indexed states, (12) (that calculates cost assigned to the state s_i with action a_i) is replaced by

$$\text{cost}(s_i, a_i) = k(s_i, a_i) + \sum_{s_{i+1} \in S_{i+1}} c(s_{i+1}, s_i, a_i) y(s_i, a_i) \tag{21}$$

and (17) describing the time spent in the state s_i with action a_i is replaced by

$$y(s_i, a_i) = \int_{t_i}^{t_i+\Delta t} \frac{(1 - F(t))\, dt}{1 - F(t_i)}. \tag{22}$$

The probability of getting an arrival is defined using the time indices for the system state, where $t_i \le t \le t_i + \Delta t$

$$p(s_{i+1} | t_i, s_i, a_i) = \frac{F(t_i + \Delta t) - F(t_i)}{1 - F(t_i)}. \tag{23}$$

Equation (16) is replaced by the following set of equations. The probability of transition to the next idle state is defined to be $m(s_{i+1}|s_i, a_i) = 1 - p(s_{i+1}|t_i, s_i, a_i)$ and of transition back into the active state is $m(s_{i+1}|s_i, a_i) = p(s_{i+1}|t_i, s_i, a_i)$. The general cumulative distribution of event occurrences is given by $F(t_i)$.

An example below illustrates how the time indexing is done.

Example V.4: The cumulative distribution of user request arrival occurrences in the idle state follows a Pareto distribution $F(t_i) = 1 - at_i^{-b}$. The transition from the idle to the low-power state follows uniform distribution with average transition time $t_{\text{ave}} = (t_{\max_a s} + t_{\min_a s})/2$. The time increments are indexed by j. Thus, the probability of transition from the idle state at time increment $j\Delta t$ into the low-power state with no elements in the queue is given by $m(s_{i+1}|s_i, a_i) = [1 - F(j\Delta t + t_{\text{ave}})]/[1 - F(j\Delta t)]$. This equation computes conditional probability that there will be no arrivals up to time $(j + 1)\Delta t + t_{\text{ave}}$ given that there was no arrival up to time $j\Delta t + t_{\text{ave}}$. Note that in this way we are taking history into account. Similarly, we can define the probability of transition from the idle state into a low-power state with an element in the queue by $m(s_{i+1}|s_i, a_i) = [F(j\Delta t + t_{\text{ave}}) - F(j\Delta t)]/[1 - F(j\Delta t)]$.

The expected time spent in the idle state indexed with time increment $N\Delta t$ can be defined by $y(s, a) = \int_{N\Delta t}^{(N+1)\Delta t} [(1 - F(t))\, dt]/[1 - F(t_i)]$, which after integration, simplifies to $[((N+1)\Delta t)^{1-a} - (N\Delta t)^{1-a}]/[(1-a)(N\Delta t)^{-a}]$. With that, we can calculate energy consumed in the idle state, again assuming that there is no fixed energy cost and that the power consumption is defined by P_I: $[P_I((N + 1)\Delta t)^{1-a} - (N\Delta t)^{1-a}]/[(1 - a)(N\Delta t)^{-a}]$.

TISMDP policies are implemented in a similar way to the renewal theory model, but there are more possible decision points. Briefly, upon entry to each decision state, the pseudorandom number RND is generated. The device will transition into low-power state at the time interval for which the probability of going to that state as given by the policy is greater than RND. Thus, the policy can be viewed as randomized timeout. The device transitions into active state if the request arrives before entry into low-power state. Once the device is turned off, it stays off until the first request arrives, at which point it transitions into active state. The detailed discussion of how the policy is implemented if there is only one decision state has been presented in Section IV-B.

Example V.5: As mentioned in Example V.1, the SmartBadge has two states where decisions can be made: idle and standby. From the idle state, it is possible to give a command to transition to standby or to the off state. From standby, only a transition to the off state is possible. In this case, both the idle and the standby states are time-indexed. The optimal policy gives a table of probabilities determining when the transition between the idle, standby, and off states should occur. For example, a policy may specify that if the system has been idle for 50 ms, then the transition to the standby state should occur with probability of 0.4, the transition to the off state with probability of 0.2, and otherwise the device will stay idle. Once in the standby state for another 100 ms, the policy may specify that the transition into the off state should occur with probability of 0.9. When a user request arrives, the system transitions back into the active state.

In this section, we presented a power management algorithm based on TISMDPs. The TISMDP model is more complex than the SMDP model, but is more accurate and is also applicable to a wider set of problems, such as a problem that has more than one nonexponential transition occurring at the same time. The primary difference between the TISMDP model and the renewal theory model is that TISMDP supports multiple decision points in the system model, while renewal theory allows for only one state in which the PM can decide to transition the device to the low-power state. For example, in systems where there are multiple low-power states, the PM would not only have to make a decision to transition to low-power state, but also could transition the system from one low-power state into another. Renewal theory cannot be used for this case as there are multiple decision states. The main advantage of the renewal theory model is that it is more concise and thus computes faster. The renewal theory model has only five states, as compared to $O(N) + 2$ states in the TISMDP model (N is the maximum time index). In addition, each of the $O(N)$ states require evaluations of one double and two single integrals, compared with a very simple arithmetic formulation for the renewal theory model.

VI. Results

We perform the policy computation using the solver for LPs [39] based on the simplex method. The optimization runs in just under 1 min on a 300-MHz Pentium processor. We first verified the optimization results using simulation. Inputs to the simulator are the system description, the expected time horizon (the length of user trace), the number of simulations to be performed, and the policy. The system description is characterized by the power consumption in each state, the performance penalty, and the function that defines the transition time pdf and the probability of transition to other states given a command from the PM. Note that our simulation used both pdfs we derived from data and the original traces. When using pdfs, we just verified the correctness of our problem formulation and solution. With real traces, we were able to verify that indeed pdfs we derived do in fact match well the data from the real system and, thus, give optimal policies for the real systems. The results of the optimization are in close agreement with the simulation results.

In the next sections, we show large savings we *measured* on three different devices: laptop and desktop hard disks and the WLAN card and the simulation results showing savings in power consumption when our policy is implemented in a SmartBadge portable system. As the first three examples (two hard disks and WLAN) have just one state in which the decision to transition to low-power state can be made, the renewal theory model and the TISMDP model give the same results. The last example (SmartBadge) has two possible decision states—idle and standby state. In this case, the TISMDP model is necessary in order to obtain the optimal policy.

A. Hard Disk

We implemented the PM as part of a filter driver template discussed in [40]. A filter driver is attached to the vendor-specific device driver. Both drivers reside in the OS, on the kernel level, above the advanced configuration and power interface (ACPI) driver implementations. Application programs such as word processors or spreadsheets send requests to the OS. When any event occurs that concerns the hard disk, PM is notified. When the PM issues a command, the filter driver creates a power transition call and sends it to the device that implements the power transition using ACPI standard. The change in power state is also detected with the digital multimeter that measures current consumption of the hard disk.

We measured and simulated three different policies based on stochastic models and compared them with two bounds: *always-on* and *oracle* policies. Always-on policy leaves the hard disk in the active state and, thus, does not save any power. Oracle policy gives the lowest possible power consumption, as it transitions the disk into sleep state with the perfect knowledge of the future. It is computed off-line using a previously collected trace. Obviously, the oracle policy is an abstraction that cannot be used in runtime DPM.

All stochastic policies minimized power consumption under a 10% performance constraint (10% delay penalty). The results are shown in Figs. 12 and 13. These figures best illustrate the problem we observed when user request arrivals are modeled only with exponential distribution as in CTMDP model [13].

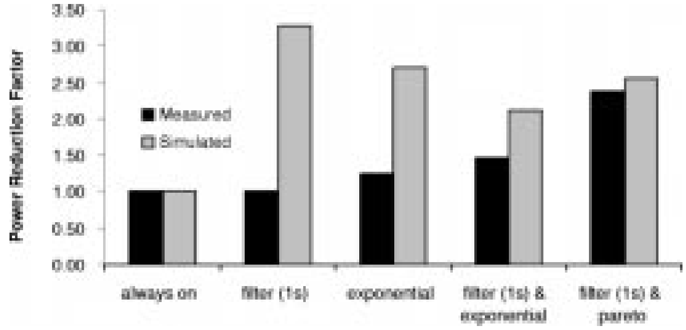


Fig. 12. Measured and simulated hard-disk power consumption.

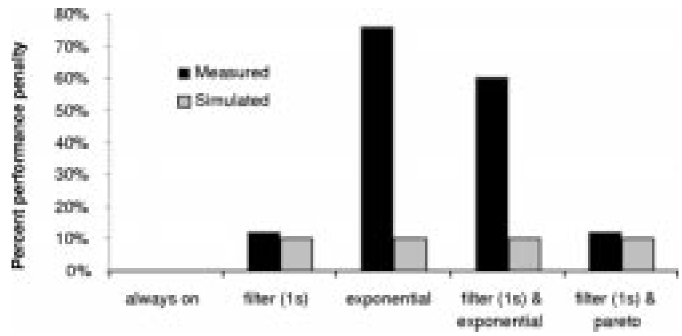


Fig. 13. Measured and simulated hard- disk performance.

The simulation results for the exponential model (CTMDP) show large power savings, but measurement results show no power savings and a very high-performance penalty. As the exponential model is memoryless, the resulting policy makes a decision as soon as the device becomes idle or after a very short filtering interval (filtered 1-s columns in Figs. 12 and 13). If the idle time is very short, the exponential model gets a large performance penalty due to the wakeup time of the device and a considerable cost in shutdown and wakeup energies. In addition, if the decision upon entry to idle state is to stay awake, large idle times, such as lunch breaks, will be missed. If the policy is forced to reevaluate until it shuts down (exponential), then it will not miss the long idle times. When we use a short timeout to filter out short arrival times and force the PM to reevaluate its decision (filtered exponential), the results improve. The best results are obtained with our policy. In fact, our policy uses 2.4 times less power than the always-on policy. These results show that it is critically important to not only simulate, but also measure the results of each policy and, thus, verify the assumptions made in modeling. In fact, we found that modeling based on simple Markov chains is not accurate and that we do require more complex model presented in this paper.

Comparison of all policies measured on the laptop is shown in Table V and for the desktop in Table VI. Karlin's algorithm analysis [7] is guaranteed to yield a policy that consumes at worst twice the minimum amount of power consumed by the policy computed with perfect knowledge of the user behavior. Karlin's policy consumes 10% more power and has worse performance than the policy based on our TISMDP model. In addition, our policy consumes 1.7 times less power than the default Windows timeout policy of 120 s and 1.4 times less power than the 30-s

TABLE V
LAPTOP HARD-DISK MEASUREMENT COMPARISON

Algorithm	Pwr (W)	N_{sd}	N_{wd}	$T_{ss}(s)$
Oracle	0.33	250		118
Ours	0.40	326	76	81
Adaptive	0.43	191	28	127
Karlin's	0.44	323	64	79
30s timeout	0.51	147	18	142
DTMDP	0.62	173	54	102
120s timeout	0.67	55	3	238
always on	0.95	0	0	0
CTMDP	0.97	391	359	4

TABLE VI
DESKTOP HARD-DISK MEASUREMENT COMPARISON

Algorithm	Pwr (W)	N_{sd}	N_{wd}	$T_{ss}(s)$
Oracle	1.64	164	0	166
Ours	1.92	156	25	147
Karlin's	1.94	160	15	142
Adaptive	1.97	168	26	134
30s timeout	2.05	147	18	142
120s timeout	2.52	55	3	238
DTMDP	2.60	105	39	130
always on	3.48	0	0	0
CTMDP	3.90	326	318	4

timeout policy on the laptop. Our policy performs better than the adaptive model [12] and significantly better than the policy based on DTMDP. The policy based on the simple CTMDP (without reevaluation and with initial 1-s filter) performs worse then the always-on policy, primarily due to the exponential interarrival request assumption. This policy both misses some long idle periods and tends to shut down too aggressively, as can be seen from its very short average sleep time. Better results can be obtained by using reevaluations with filtering. Similar results can be seen on the desktops.

Performance of the algorithms can be compared using three different measures. $N_{\rm sd}$ is defined as the number of times the policy issued the sleep command. $N_{\rm wd}$ gives the number of times sleep command was issued and the hard disk was asleep for shorter than the time needed to recover the cost of spinning down and spinning up the disk. Clearly, it is important to minimize $N_{\rm wd}$ while maximizing $N_{\rm sd}$. In addition, the average length of time spent in the sleep state ($T_{\rm ss}$) should be as large as possible while still keeping the power consumption down. From our experience with the user interaction with the hard disk, our algorithm performs well, thus giving us low-power consumption with still good performance.

As mentioned earlier, we filtered request arrivals using a fraction of hard disk break-even time. The effect of filtering arrivals into the idle state is best shown in Fig. 14 for the policy with the performance penalty of the laptop hard disk limited to 10%. For very short filter times, the power consumption is very high since

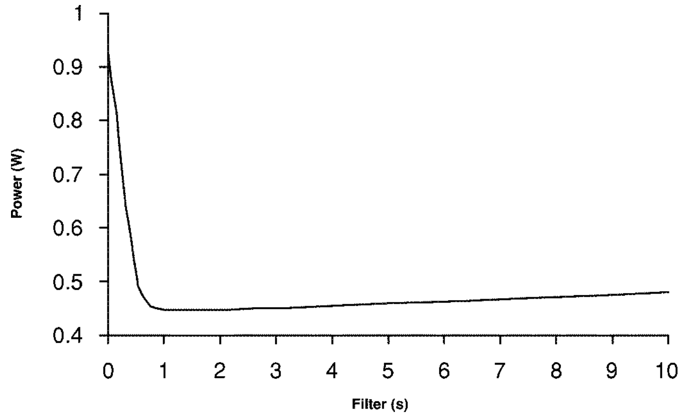


Fig. 14. Power consumption versus filter size.

the overhead of transition to and from low-power state has not been compensated. The power consumption grows for longer filter times since more time is spent in the idle state before transitioning to the low-power state, thus wasting some power. Note that the best filtering intervals are on the order of seconds since the hard disk break-even time is also on the order of seconds.

The event-driven nature of our algorithm as compared to algorithms based on discrete time intervals saves considerable amount of power while in sleep state as it does not require policy evaluation until an event occurs. Waking up a 10-W processor every 1 s for policy reevaluation that takes 100 ms to execute would use 1800 J of energy during a normal 30 min break. With an event-driven policy, the processor could be placed in a low-power mode during the break time, thus saving a large portion of battery capacity.

B. WLAN Card

For WLAN measurements, we used Lucent's WLAN 2-Mb/s card [35] running on the laptop. As a mobile environment is continually changing, it is not possible to reliably repeat the same experiment. As a result, we needed to use a trace-based methodology discussed in [41]. The methodology consists of three phases: collection, distillation, and modulation. We used *tcpdump* [36] utility to get the user's trace for two different applications: web browsing and telnet. During distillation, we prepared the trace for the modeling step. We had a LAN-attached host read the distilled trace and delay or drop packets according to the parameters we obtained from the measurements. In this way, we were able to recreate the experimental environment so that different algorithms can be reliably compared.

We implemented three different versions of our algorithm for each application, each with different power ($P_{\rm ave}$) and performance penalty ($T_{\rm penalty}$). The algorithms are labeled Ours a, b, c for web browser and Ours 1, 2, 3 for telnet. Since web and telnet arrivals behave differently (see Fig. 4), we observe through the OS what application is currently actively sending and use the appropriate power management policy. The performance penalty for WLAN is a measure of the total overhead time due to turning off the card. Note that for the hard disk, we measured instead the average time in the sleep state as the accurate real overhead was difficult to obtain. In addition to measuring the energy consumption (and then calculating average power), we also quantified the performance

TABLE VII
DPM FOR WLAN WEB BROWSER

Policy	N_{sd}	N_{wd}	$T_{penalty}$ (sec)	P_{ave} (W)
Oracle	395	0	0	0.467
Ours (a)	363	96	6.90	0.474
Ours(b)	267	14	1.43	0.477
Karlin's	623	296	23.8	0.479
Ours(c)	219	9	0.80	0.485
CTMDP	3424	2866	253.7	0.539
Default	0	0	0	1.410

TABLE VIII
DPM FOR WLAN TELNET APPLICATION

Policy	N_{sd}	N_{wd}	$T_{penalty}$ (sec)	P_{ave} (W)
Oracle	766	0	0	0.220
Ours(1)	798	21	2.75	0.269
Ours(2)	782	33	2.91	0.296
Karlin's	780	40	3.81	0.302
Ours(3)	778	38	3.80	0.310
CTMDP	943	233	20.53	0.361
Default	0	0	0	1.410

penalty using three different measures. Delay penalty T_p is the time the system had to wait to service a request since the card was in the sleep state when it should not have been. In addition, we measure the number of shutdowns N_{sd} and the number of wrong shutdowns N_{wd}. A shutdown is viewed as wrong when the sleep time is not long enough to make up for the energy lost during transition between the idle and off state. The number of shutdowns is a measure of how eager the policy is, while a number of wrong shutdowns tells us how accurate the policy is in predicting a good time to shut down the card.

The measurement results for a 2.5-h web browsing trace are shown in Table VII. Our algorithms (Ours a, b, c) show, on average, a factor of three in power savings with a low performance penalty. Karlin's algorithm [7] is guaranteed to be within a factor of two of the oracle policy. Although its power consumption is low, it has a performance penalty that is an order of magnitude larger than for our policy. A policy that assumes exponential arrivals only, CTMDP [13], has a very large performance penalty because it makes the decision as soon as the system enters idle state.

Table VIII shows the measurement results for a 2-h telnet trace. Again, our policy performs best with a factor of five in power savings and a small performance penalty. The telnet application allows larger power savings because on average it transmits and receives much less data then the web browser, thus giving us more chances to shut down the card.

C. SmartBadge

In contrast to the previous examples, where we implement and measure the decrease in power consumption when using our power management policies, in this case we perform a case study on the tradeoffs between power and performance for the SmartBadge. The SmartBadge is a good example of a system that has more than one decision point and, thus, requires the TISMDP model in order to obtain an optimal policy. We first study the tradeoffs between power and performance for policies with just one decision state (idle state) and then follow with an example contrasting policies with one state to policies that have two decision states (idle and standby).

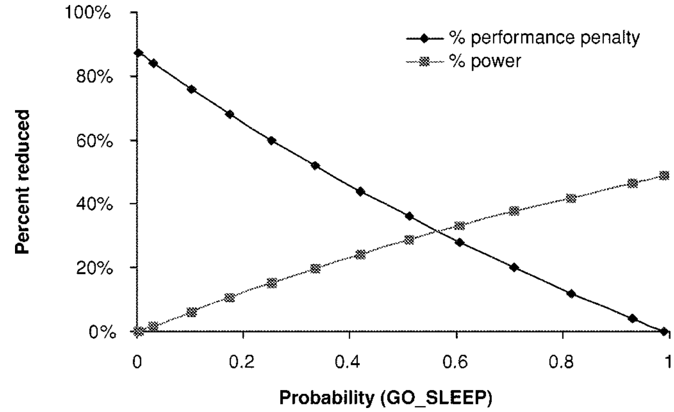


Fig. 15. SmartBadge DPM results.

TABLE IX
COMPARISON OF POLICIES BY DECISION STATE NUMBER

No. Decision States	Power (W)
One state	1.84
Two states	1.47

The results of simulation shown in Fig. 15 clearly illustrate the tradeoff between different policies for one decision state that can be implemented in the SmartBadge system. The performance penalty is defined as the percent of time the system spends in a low-power state with a nonempty queue. In general, the goal is to have as few requests as possible waiting for service. For systems with a hard real-time constraint, this penalty can be set to large values to force less aggressive power management, thus resulting in less requests queued up for service. In systems where it is not as critical to meet time deadlines, the system can stay in a low-power state longer, thus accumulating more requests that can be serviced upon return to the active state.

Because of the particular design characteristics of the SmartBadge, the tradeoff curves of performance penalty and power savings are very close to linear. When the probability of going to sleep is zero, no power can be saved, but the performance penalty can be reduced by 85% as compared to the case where the probability is one. On the other hand, about 50% of the power can be saved when the system goes to standby upon entry to idle state.

In addition to analyzing power and performance tradeoffs for policies that have only one decision state, we have also compared the one decision state (idle) policy to a policy with two decision states (idle and standby) with the same performance penalty. The results in Table IX clearly show that considerably larger power savings with the same performance penalty can be obtained when using a more complex policy optimization model that enables multiple decision points (TISMDP model) instead of just one decision point (renewal theory model).

VII. Conclusion

Dynamic power management policies reduce energy consumption by selectively placing components into low-power states. In contrast to heuristic policies, such as timeouts, policies based on stochastic models can guarantee optimal results. The quality of results of stochastic DPM policies depends strongly on the assumptions made. In this paper, we present and implement two different stochastic models for dynamic power management. The measurement results show large power savings.

The first approach requires that only one decision point be present in the system. This model is based on renewal theory. The second approach allows for multiple decision points and is based on SMDP model. The basic SMDP model can accurately model only one nonexponential transition occurring with the exponential ones. We presented TISMDP model as the extension to SMDP model in order to describe more than one nonexponential transition occurring at the same time. TISMDP model is very general, but also is more complex. Thus, it should be used for systems that have more than one decision point.

We presented large power savings using our approach on four different portable devices: the laptop and the desktop hard disks, the WLAN card, and the SmartBadge. The measurements for the hard disks show that our policy gives as much as 2.4 times lower power consumption as compared to the default Windows timeout policy. Thus, it is very beneficial to use our approach over the simple timeout. In addition, our policy obtains up to 5 times lower power consumption for the wireless card relative to the default policy. The power management results on the SmartBadge show savings of as much as 70% in power consumption. Finally, the comparison of policies obtained for the SmartBadge with the renewal model and TISMDP model clearly illustrate that whenever there is more than one decision point available, the TISMDP model should be used as it can utilize the extra degrees of freedom and, thus, obtain an optimal power management policy.

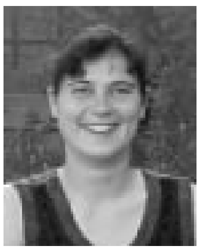
Acknowledgment

The authors would like to thank Hewlett-Packard Laboratories for their support.

References

[1] A. Chandrakasan and R. Brodersen, *Low Power Digital CMOS Design*. Norwell, MA: Kluwer, 1995.
[2] J. Rabaey and M. Pedram, Eds., *Low Power Design Methodologies*. Norwell, MA: Kluwer, 1996.
[3] W. Nabel, and J. Mermet, Eds., *Lower Power Design in Deep Submicron Electronics*. Norwell, MA: Kluwer, 1997.
[4] C. Ellis, "The case for higher-level power management," in *Proc. 7th IEEE Workshop on Hot Topics in Operating Systems*, Mar. 1999, pp. 162–167.
[5] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools*. Norwell, MA: Kluwer, 1997.
[6] Advanced Configuration and Power Interface Specification, Intel, Microsoft, and Toshiba. (1996). [Online]. Available: http://www.intel.com/ial/powermgm/specs.html
[7] A. Karlin, M. Manesse, L. McGeoch, and S. Owicki, "Competitive randomized algorithms for nonuniform problems," *Algorithmica*, vol. 11, no. 6, pp. 542–571, June 1994.
[8] D. Ramanathan and R. Gupta, "System level online power management algorithms," *Proc. Design, Automation, Test Eur.*, pp. 606–611, Mar. 2000.
[9] M. Srivastava, A. Chandrakasan, and R. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Trans. VLSI Syst.*, vol. 4, pp. 42–55, Mar. 1996.
[10] C.-H. Hwang and A. Wu, "A predictive system shutdown method for energy saving of event-driven computation," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1997, pp. 28–32.
[11] L. Benini, G. Paleologo, A. Bogliolo, and G. De Micheli, "Policy optimization for dynamic power management," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 813–833, June 1999.
[12] E. Chung, L. Benini, and G. De Micheli, "Dynamic power management for nonstationary service requests," *Proc. Design, Automation Test Eur.*, pp. 77–81, Mar. 1999.
[13] Q. Qiu and M. Pedram, "Dynamic power management based on continuous-time Markov decision processes," in *Proc. Design Automation Conf.*, June 1999, pp. 555–561.
[14] ——, "Dynamic power management of complex systems using generalized stochastic petri nets," in *Proc. Design Automation Conf.*, June 2000, pp. 352–356.
[15] T. Simunic, L. Benini, and G. De Micheli, "Event-driven power management," in *Proc. Int. Symp. System Synthesis*, Apr. 1999, pp. 18–23.
[16] ——, "Power management of laptop hard disk," *Proc. Design, Automation, Test Eur.*, p. 736, Mar. 2000.
[17] ——, "Energy efficient design of portable wireless devices," in *Proc. Int. Symp. Low Power Electronics Design*, Aug. 2000, pp. 49–54.
[18] ——, "Dynamic power management for portable systems," in *Proc. 6th Int. Conf. Mobile Computing Networking*, Oct. 2000, pp. 22–32.
[19] G. Q. Maguire, M. Smith, and H. W. P. Beadle, "SmartBadges: A wearable computer and communication system," in *Proc. 6th Int. Workshop Hardware/Software Codesign*, Apr. 1998, pp. 11–18.
[20] V. Paxson and S. Floyd, "Wide area traffic: The failure of Poisson modeling," *IEEE Trans. Networking*, vol. 3, pp. 226–244, June 1995.
[21] H. Taylor and S. Karlin, *An Introduction to Stochastic Modeling*. New York: Academic, 1998.
[22] S. Ross, *Stochastic Processes*. New York: Wiley, 1996.
[23] Y. Lu, E. Chung, T. Šimunić, L. Benini, and G. De Micheli, "Quantitative comparison of PM algorithms," *Proc. Design, Automation, Test Eur.*, pp. 20–26, Mar. 2000.
[24] S. Gary and P. Ippolito, "PowerPC 603, a microprocessor for portable computers," *IEEE Design Test Comput.*, vol. 11, pp. 14–23, Winter 1994.
[25] G. Debnath, K. Debnath, and R. Fernando, "The Pentium processor—90/100 microarchitecture and low power circuit design," in *Proc. IEEE Int. Conf. VLSI Design*, Jan. 1995, pp. 185–190.
[26] S. Furber, *ARM System Architecture*. Reading, MA: Addison-Wesley, 1997.
[27] L. Geppert and T. Perry, "Transmeta's magic show," *IEEE Spectrum*, vol. 37, pp. 26–33, May 2000.
[28] R. Golding, P. Bosch, and J. Wilkes, " Idleness is not sloth," Hewlett-Packard Laboratories, Palo Alto, CA, Tech. Rep. HPL-96-140, 1996.
[29] F. Douglis, P. Krishnan, and B. Bershad, "Adaptive disk spin-down policies for mobile computers," in *Proc. 2nd USENIX Symp. Mobile Location-Independent Computing*, Apr. 1995, pp. 121–137.
[30] D. Helmbold, D. Long, and E. Sherrod, "Dynamic disk spin-down technique for mobile computing," in *Proc. IEEE Conf. Mobile Computing*, 1996, pp. 130–142.
[31] Y. Lu and G. De Micheli, "Adaptive hard disk power management on personal computers," in *Proc. IEEE Great Lakes Symp. VLSI*, Mar. 1999, pp. 50–53.
[32] E. Chung, L. Benini, and G. De Micheli, "Dynamic power management using adaptive learning trees," in *Proc. Int. Conf. Computer-Aided Design*, Apr. 1999, pp. 221–227.
[33] L. Benini, R. Hodgson, and P. Siegel, "System-level power estimation and optimization," in *Proc. Int. Symp. Low Power Electronics Design*, Aug. 1998, pp. 173–178.
[34] *IEEE Draft Standard for Wireless LAN*, IEEE Standard P802.11D5.0, July 1996.

[35] *IEEE 802.11 WaveLAN PC Card—User's Guide*, Lucent, Murray Hill, NJ, 1998, p. A-1.
[36] *The "tcpdump" Manual Page*, Lawrence Berkeley Laboratory, Berkeley, CA, 1998.
[37] S. Boyd, "Convex optimization," Stanford Univ., Stanford, CA, Stanford Class Notes, 1999.
[38] M. Puterman, *Finite Markov Decision Processes*. New York: Wiley, 1994.
[39] Solver for linear programs, M. Berkelaar. [Online]. Available: www.cs.sunysb.edu/algorith/implement/lpsolve/implement.shtml
[40] Y. Lu, T. Šimunić, and G. De Micheli, "Software controlled power management," in *Proc. 7th Int. Workshop Hardware/Software Codesign*, Apr. 1999, pp. 157–161.
[41] B. Noble, M. Satyanarayanan, G. Nguyen, and R. Katz, "Trace-based mobile network emulation," in *Proc. Signal Communications Conf.*, Sept. 1997, pp. 51–61.



Tajana Šimunić received the M.S. degree in electrical engineering from the University of Arizona, Tucson. She is currently working toward the Ph.D. degree in electrical engineering at Stanford University, Stanford, CA.

She was a Senior Design Engineer with the Altera Corporation. She is currently with Hewlett-Packard Laboratories, Palo Alto, CA. Her research interests include low-power system design, embedded systems, and wireless system design.

Luca Benini (S'94–M'97) received the B.S. degree (summa cum laude) in electrical engineering from the University of Bologna, Italy, in 1991, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1994 and 1997, respectively.

Since 1998, he has been an Assistant Professor with the Department of Electronics and Computer Scienc, University of Bologna. He also holds Visiting Researcher positions at Stanford University and the Hewlett-Packard Laboratories, Palo Alto, CA. He is a member of the organizing commitee of the International Symposium on Low Power Design and a member of the technical program committee for several technical conferences, including the Design and Test in Europe Conference, International Symposium on Low Power Design, and the Symposiom on Hardware-Software Codesign. He has authored or coauthored more than 120 papers in international journals and conferences, a book, and several book chapters. His research interests include all aspects of computer-aided design of digital circuits, with special emphasis on low-power applications, and in the design of portable systems.

Peter Glynn is a Deputy Chair and a Professor of the Management Science and Engineering Department, Stanford University, Stanford, CA. He is the Editor of the Springer-Verlag Series in Operations Research and has coauthored or coedited seven other books and hundreds of technical articles. He is an Associate Editor of many journals and has been involved in organizing 20 conferences, pannels, workshops. His research interests include stochastic modeling and simulation of systems and networks of systems. Associate Editor of many other journals.

Dr. Glynn has been the Thomas W. Ford Professor of Engineering since 1996. He received the 1993 TIMS College on Simulation Best Paper Award.

Giovanni De Micheli (S'70–M'83–SM'89–F'94) received the Nuclear Engineer degree from the Politecnico di Milano, Milan, Italy, in 1979 and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley, in 1980 and 1983, respectively.

He is a Professor of Electrical Engineering and Computer Science at Stanford University, Stanford, CA. He is author of *Synthesis and Optimization of Digital Circuits* (New York: McGraw-Hill, 1994) and coauthor of four other books. His research interests include several aspects of design technologies for integrated circuits and sytems, with partcular emphasis on sythesis, system-level design, and hardware–software codesign, and low-power design.

Dr. De Micheli received a Presidential Young Investigator Award in 1988. He received the 1987 IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS Best Papaer Award and two Best Paper Awards at the Design Automation Conference in 1983 and 1993. He was Vice President (for publications) of the IEEE CAS Society from 1999 to 2000 and is currently the Editor-in-Chief of IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS. He was Program Chair and General Chair of the Design Automation Conference in 1996, 1997, and 2000, respectively, and was also the Program Chair and General Chair of the International Conference on Computer Design in 1988 and 1989, respectively.