Wireless Networks 11, 451–469, 2005
© 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

Self-Tuning Wireless Network Power Management

MANISH ANAND, EDMUND B. NIGHTINGALE and JASON FLINN
Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48105

Abstract. Current wireless network power management often substantially degrades performance and may even *increase* overall energy usage when used with latency-sensitive applications. We propose self-tuning power management (STPM) that adapts its behavior to the access patterns and intent of applications, the characteristics of the network interface, and the energy usage of the platform. We have implemented STPM as a Linux kernel module—our results show substantial benefits for distributed file systems, streaming audio, and thin-client applications. Compared to default 802.11b power management, STPM reduces the total energy usage of an iPAQ running the Coda distributed file system by 21% while also reducing interactive file system delay by 80%. Further, STPM adapts to diverse operating conditions: it yields good results on both laptops and handhelds, supports 802.11b network interfaces with substantially different characteristics, and performs well across a range of application network access patterns.

Keywords: power management, self-tuning, 802.11

1. Introduction

Wireless networks provide mobile computers with continuous Internet connectivity. Yet, power management is needed to ensure that the network interface does not overly tax the limited battery capacity of a mobile device. For example, our measurements show that using a 802.11b network card without power management can shorten the battery lifetime of a HP iPAQ 3870 handheld by almost 50%.

The popular IEEE 802.11 standard [10] provides a power-saving mode (PSM) that periodically disables the network interface during periods of no activity. However, PSM does not adapt to the power characteristics of the network interface and mobile computer, the intent and access patterns of applications, or the needs and expectations of users. While PSM provides excellent energy conservation in some circumstances, it can also substantially degrade interactive application performance and even *increase* the energy needed to perform certain activities. For instance, PSM causes an unacceptable 16-32x slowdown in the time to list directories stored in NFS.

We show that different power management strategies are needed in different circumstances. Rather than take a "one size fits all" approach, we propose self-tuning power management (STPM) that adapts to the characteristics of the network interface, mobile computer, and applications. We have implemented STPM as a Linux kernel module that runs on both handhelds and laptops.

STPM differs substantially from other adaptive strategies such as the PSPCAM mode of the Cisco Aironet 350 card [4] and the bounded slowdown protocol of Krashinsky and Balakrishnan [13]. STPM explicitly considers the time and energy costs of changing power modes. These transition costs can be quite large for current 802.11b cards—several hundred milliseconds in most cases. STPM also explicitly considers the base power usage of the mobile computer. Finally, STPM provides a simple interface that allows applications to disclose hints about their intent in using the network interface. For legacy applications that have not yet been modified to disclose such hints, STPM uses passive monitoring and heuristics to generate hints on their behalf. STPM then tunes its power management strategy to observed network access patterns.

Our results show that STPM provides significant energy conservation with minimal performance impact for applications such as distributed file systems, streaming audio, and thin-client remote X displays. For instance, STPM reduces the total energy usage of an iPAQ running the Coda distributed file system by 21% compared to PSM, while also reducing interactive file system delay by 80%. Further, STPM shows benefits across a diverse set of network interfaces and mobile devices.

We begin with a discussion of the limitations of current wireless power management. Section 3 outlines the principles we followed in the design of STPM. In Sections 4 and 5, we describe our implementation and compare its performance and energy conservation to that of other static and adaptive power management strategies. Finally, we conclude with a discussion of related and future work.

2. Motivation

Current 802.11b power management schemes can severely degrade the performance of latency-sensitive applications. For example, figure 1 shows how power management affects the time to list directories of varying sizes stored in the Network File System (NFS) [19]. These results were generated by executing `ls` on a HP iPAQ 3870 handheld with a Cisco Aironet 350 802.11b card.

The solid line at the bottom labeled "CAM" shows performance in continuously-aware mode (i.e. without power management). The dashed line at the top labeled "PSM-static" shows performance with the default 802.11b power saving mode (PSM). Thedifference between these two shows that

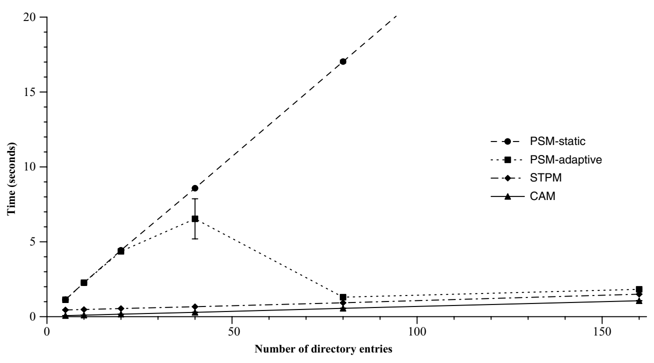


Figure 1. Effect of power management on NFS.

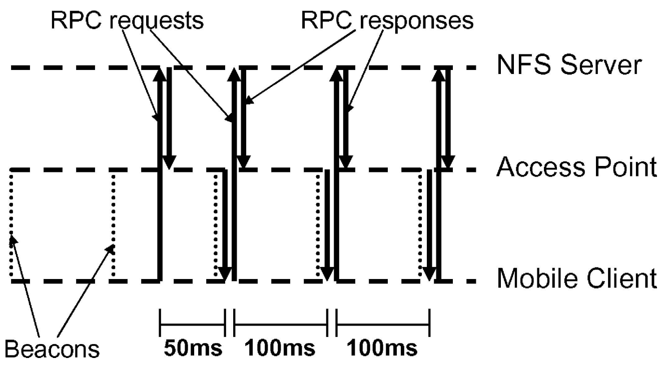


Figure 2. RPC delays due to power management.

PSM causes an unacceptable 16–32 times slowdown for NFS directory listings.

This poor performance is caused by the interaction of NFS remote procedure calls (RPCs) and 802.11b power management. When no packets are waiting for a mobile computer at the wireless access point, 802.11b power management disables the network interface to extend battery lifetime. The access point periodically sends a beacon informing the mobile computer if any packets have arrived—the client interface wakes up to listen to the beacon and goes back to sleep if no data is waiting.

As figure 2 shows, power management delays each RPC response at the access point until the next beacon. Since NFS does not usually issue concurrent RPCs, only one RPC is issued per beacon period. With a typical 100 ms beacon interval, the expected delay for the first RPC is approximately 50 ms. The delay for subsequent RPCs is almost 100 ms because each RPC request is sent soon after the response to the previous RPC is received. Since NFS makes two RPCs, a `lookup` and a `getattr`, for each file in the directory, the cumulative delay is quite large.

Similar observations have led to the development of adaptive power management strategies that switch between CAM and PSM depending upon traffic load [4,13]. Ideally, an adaptive strategy can yield good performance by switching to CAM when data is being transferred and by switching back to PSM when it is not. Many cards, including the Cisco Aironet 350 card, support such an adaptive mode. Our observations of the Cisco card reveal that it switches to CAM when more than one packet is waiting for the mobile computer at the access point, and that it switches back to PSM after approximately 800 ms without receiving a packet.

The dotted line labeled “PSM-adaptive” in figure 1 shows NFS directory listing performance when the Cisco adaptive mode is employed. For small and medium-sized directories, performance is 26 times slower. Because RPCs are issued sequentially and each RPC is typically small, NFS generates insufficient traffic to trigger a switch to CAM. For large directories, the initial read of directory data transmits enough data to trigger the switch—however, even for 160-entry directories, performance still lags CAM by 72% because several RPCs are made in PSM.

Even worse, power management may actually increase the energy used to perform interactive tasks. Compared to CAM, the iPAQ uses up to 17 times more energy to list NFS directories with PSM-static and up to 12 times more energy with PSM-adaptive. Although these modes decrease the average power used by the network interface, the network interface represents only a portion of total system power usage. Since energy usage is the integral of power over time, the substantial increase in execution time dominates the small decrease in power. Power management extends battery lifetime but the user accomplishes less work before the battery expires.

Of course, this is a worst-case example. The mobile computer will often be idle for a considerable amount of user think-time; during such idle periods, power management decreases energy usage without performance penalty. Yet, by replaying traces of interactive file system usage, we have found that the cost of using untuned power management during interactive episodes often dominates the benefits realized during idle periods. For many latency-sensitive applications, untuned power management introduces substantial performance and energy penalties. Examples of applications that have similar communication pattern include other distributed file systems such as AFS [9] and Coda [12], applications that use a remote X server [21] for display, and client-server systems based upon Java RMI [26] and SOAP [2].

To solve this problem, we have built a self-tuning module that adapts to application access patterns, network interface characteristics, and the system on which it is running. The line labeled “STPM” in figure 1 shows that our module decreases the time needed to list a NFS directory by up to 23 times compared to PSM-static and by up to 10 times compared to PSM-adaptive. STPM also reduces energy usage by up to 12 times compared to PSM-static and up to 5 times compared to PSM-adaptive.

3. Design principles

Self-tuning power management is based upon the following design principles:

- Know application intent

- Be proactive
- Respect the critical path
- Embrace the performance/energy tradeoff
- Adapt to the operating environment

3.1. Know application intent

A little information about application intent goes a long way. For example, consider why PSM-adaptive works poorly in the previous example. Most common applications issue file operations sequentially; thus, NFS often has only a single RPC in flight. Further, 802.11b power management effectively limits NFS to one RPC per beacon period. Although the data rate of NFS is low, its data rate would increase substantially without power management because several RPCs could complete during each beacon period. However, PSM-adaptive does not transition to CAM because it does not detect sufficient network traffic.

An alternative strategy would be to switch to CAM whenever an incoming packet is received. However, this aggressive strategy works poorly in other cases. For example, consider a stock ticker application that receives approximately 10 packets per second. When power management is enabled, NFS and the stock ticker receive roughly the same amount of data per second. However, the stock ticker performance will not improve when power management is disabled because it is already receiving at its maximum data rate.

Without knowing application intent, it is hard to distinguish these two applications. If an algorithm conservatively refuses to disable power management until a threshold data rate is achieved, it does not disable power management for NFS, leading to poor performance. Alternatively, if it liberally disables power management after the receipt of a few packets, it wastes energy by disabling power management for the stock ticker application.

Our approach is to allow each application to disclose hints about its intent in using the wireless network. This allows STPM to enable power management only when appropriate. Further, this hint-based approach helps STPM decide if the network interface can be disabled for periods longer than the beacon period. If each application discloses when it is transferring data and specifies the maximum delay on incoming packet arrivals it is willing to tolerate, then STPM can disable the network interface when it is not being used and ensure that application delay constraints are satisfied.

The main drawback of using hints is the requirement that applications be modified. To support legacy applications, STPM includes a hinting module that identifies non-hinting applications, observes their network traffic, and issues hints on their behalf. We have chosen to implement this module at the IP layer in order to support the widest possible range of legacy applications. Due to the lack of application support, the hinting module cannot precisely determine application intent. Instead, it relies on heuristics to estimate the intent of the applications.

3.2. Be proactive

If applications such as NFS were to disclose hints when each network transfer begins and ends, a possible strategy would be to enter CAM whenever at least one transfer is in progress and go back to PSM when no transfers are occurring. This purely *reactive* strategy requires that the transition cost of changing modes be low.

Unfortunately, we have found the transition costs for current 802.11b cards to be quite high. Although the device driver may complete the system call that initiates a mode transition in only a few tens of milliseconds, packet transmission and reception is delayed for a much longer period of time following each transition. We measured *transition time* for several cards by first initiating a transition to PSM or CAM and then immediately performing a single-packet ping of a nearby server. Transition times ranged from 200 ms to 600 ms—sample results are shown in figure 6.

A purely reactive strategy increases the time to perform a short RPC because transition time is greater than the latency reduction achieved by performing the RPC in CAM. However, a reactive strategy shows benefits for large requests. For instance, a 4 MB TCP transfer from an iPAQ client with an Orinoco Silver 802.11b card to a nearby server is 16% slower with power management enabled. These results, which confirm previously reported TCP throughput analysis [13], indicate that there is a break-even transfer size—for transfers larger than this size, the performance benefit of CAM outweighs the transition cost. The particular break-even size is dependent upon the data rate supported by the 802.11b card in each mode and the card's transition costs. STPM determines the break-even point for each card and switches to CAM when an application discloses that a forthcoming transfer will exceed the break-even size.

However, applications such as NFS are dominated by small transfers. For such applications, a proactive strategy is needed to amortize transition costs across multiple transfers. When a proactive strategy determines that a large number of transfers will soon occur, it switches to CAM, and then switches back to PSM after the last transfer. In the NFS example, the cumulative reduction in latency across all RPCs far exceeds the transition cost of changing power modes.

Clearly, the difficulty in implementing a proactive strategy is that it requires knowledge of the number of transfers that will occur in the near future. Applications like NFS and the X server do not have this information because they receive each application request sequentially. One possible approach would be for applications like `ls` and `make` to provide hints of future network activity. We rejected this approach because it requires modification of programs that are not normally network-aware. Further, since such applications do not usually care which type of file system they are using and do not know which blocks a file system may have cached locally, they do not know which requests will cause network activity.

Our approach is to have network-aware processes like NFS and the X server simply disclose the start and end of each transfer. STPM monitors the inter-arrival time of transfer hints, as well as the number of transfers that are closely correlated in time—we refer to such clusters of transfers as *runs*. Using an empirically-collected distribution of run lengths, STPM calculates the expected number of transfers in the current run given the number that it has already been seen. It then performs a cost-benefit analysis to determine if it should switch to CAM. For example, STPM might decide to switch to CAM after three short transfers occur close together, and also to switch back to PSM when 300 ms pass without a further transfer. The details of how this decision is made are explained in Section 4.3.

It is important to note that STPM supports both reactive and proactive strategies. For example, consider a hypothetical network interface that has negligible transition costs. Since the break-even transfer size is effectively zero, STPM would switch to CAM at the start of each transfer and return to PSM when the transfer completes. Although STPM's proactive mechanisms would be unused, STPM would benefit fully from the low transition costs.

3.3. Respect the critical path

Latency is often critical when data transfers are driven by an interactive application. The perception threshold beyond which delays become noticeable to human users is quite small—typically it is cited as being between 50 ms and 200 ms [6,15]. This means that only a few small transfers in PSM can cause a noticeable delay, and that the cumulative delay for operations such as NFS directory listings may certainly prove frustrating to the user. Thus, it is critical to disable power management when the network is being used by an interactive application.

However, there is also a substantial amount of network traffic for which latency is not critical. For example, the Coda distributed file system prefetches file data from servers to improve performance and guard against disconnection [12]. Coda also writes file modifications back to the server asynchronously [18]. For both prefetching and asynchronous writes, latency is not a critical constraint since a human user is not waiting for the transfer to complete. Similarly, streaming multimedia applications that buffer data on the client can tolerate delays commensurate with their buffer sizes.

To differentiate between these two types of network traffic, STPM enables applications to hint whether a transfer is a *foreground* transfer, in which latency is a constraint, or a *background* transfer that is not time-critical. In the former case, STPM tries to both reduce transfer time and conserve energy—in the latter case, STPM considers only energy conservation.

3.4. Embrace the performance/energy tradeoff

Disabling the 802.11 wireless interface reduces power consumption but increases the latency of packet delivery, creating an inherent tradeoff between performance and energy conservation. While this tradeoff seems unavoidable, it is important to evaluate it in the context of a mobile user's activity. If the mobile computer has a fully-charged battery and the user intends to operate on battery power for only a short time, then energy conservation is unnecessary, and the user should choose a power management strategy that maximizes performance. However, if the mobile computer's battery is nearly exhausted, then energy conservation is of primary importance.

A power management policy that statically balances these two competing goals cannot be correct in both contexts. Instead, a tunable strategy is needed. STPM provides a "knob" that can be adjusted to reflect different relative priorities for energy conservation and performance. Users can set the knob to maximum performance when they intend to operate on battery power for only a few minutes—STPM responds by keeping the wireless interface continuously active to minimize latency. We envision that STPM's knob might be set by higher-level energy-aware OS components such as Ecosystem [28] or Odyssey [7].

3.5. Adapt to the operating environment

To set the correct power management policy, STPM must understand not only the energy characteristics of the network interface, but also those of the computer using the interface. The goal of power management is to extend a mobile computer's battery lifetime—this means that the energy usage of the entire computer must be minimized, not simply that of the network interface.

To illustrate the difference, consider a hypothetical power management mode that reduces network power usage by 50% from 2 Watts to 1 Watt, but delays interactive activities by 10%. If this mode were employed by a handheld with a base power usage of 2 Watts, the total power of the mobile computer and interface would be reduced by 25% from 4 Watts to 3 Watts. Since interactive activities now take 10% longer to complete, the total energy used for each activity would be reduced by a still respectable 17.5%. However, if this mode were employed by a laptop with a base power usage of 15 Watts, total power would only be reduced by 5.9%. Further, the total energy used to perform interactive activities would actually increase by 3.5%.

This example illustrates two points. First, when used incorrectly, network power management can decrease the amount of useful work that a user can accomplish on battery power. Second, the correct power management strategy for one device may be inappropriate for the system as a whole. STPM avoids these pitfalls by explicitly considering the base power usage of the mobile computer.

4. Implementation

As shown in figure 3, we have implemented self-tuning wireless network power management as a Linux loadable kernel module. Applications link with a user-level library and disclose hints to the STPM module about their intent and activities. The library implements the STPM API described in Section 4.1 by first opening a Linux pseudo-device and then making `ioctls` on the device whenever an application calls a STPM function. This implementation enables the STPM

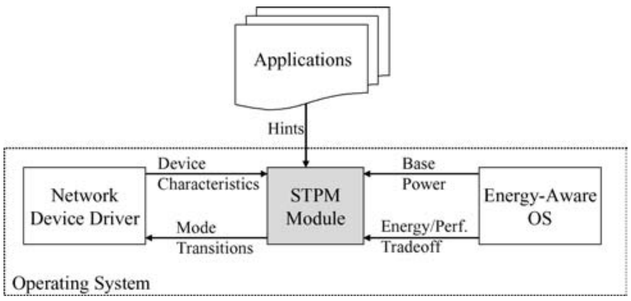


Figure 3. STPM software architecture.

module to detect when applications terminate and cancel any outstanding hints that the application made before exiting. Further, the kernel implementation improves performance by minimizing user-kernel boundary crossings.

The STPM module has two further inputs. First, the base power and current tradeoff between energy conservation and performance may be set by an energy-aware operating system [7,28] or by a user-level configuration tool. Second, for each wireless network card, the STPM module loads a device-specific characterization of power usage and transition costs. Section 4.2 describes how we derive these characterizations by executing a benchmark suite. The characterization is instantiated as a code component that is loaded by the STPM module in much the same way that the operating system loads the device driver for the wireless network card. Currently, the creation of each card-specific component is a manual task, but we plan to have our benchmark suite automatically create such components in the future.

Section 4.3 details the STPM algorithm that decides when to transition the power mode of the wireless network card. This algorithm is the simplest we could envision that meets the design principles of Section 3.

In Section 4.4 we describe our support for legacy applications that have not yet been modified to disclose hints. This support is implemented by an additional kernel module that snoops on network traffic at the IP layer and employs heuristics to generate hints on behalf of unmodified applications.

4.1. API

Figure 4 shows the STPM API. Applications use the first three functions to disclose hints about wireless network usage. Before each data transfer, an application calls `TransferHintBegin` and specifies whether the forthcoming transfer represents background or foreground activity. The application may optionally specify the expected amount of data to be sent and received. If the application leaves these values unspecified, the forthcoming transfer is assumed to be small. `TransferHintBegin` returns a unique identifier that the application later passes to `HintEnd` to specify that the transfer has completed. When an application terminates, the STPM module ends any outstanding hints for which the application has not called `HintEnd`.

```
TransferHintBegin  (IN fg_flag, IN xmit_size,
                    IN recv_size, OUT hint_id);
ListenHintBegin    (IN max_delay, OUT hint_id);
HintEnd            (IN hint_id);
SetKnob            (IN knob_value);
SetBasePower       (IN base_power);
```

Figure 4. Wireless power management API.

Processes that are listening for incoming packets may call `ListenHintBegin` to specify the maximum delay due to power management that they are willing to tolerate for incoming packets. This function also returns a unique hint identifier that may be passed to `HintEnd` to terminate the listen hint.

The `SetKnob` function adjusts the relative importance of performance and energy conservation. The value of the knob ranges from 0 to 100, where 0 represents maximum energy conservation and 100 represents maximum performance. The `SetBasePower` function specifies the base power usage of the mobile computer.

4.2. Characterizing network power costs

Wireless network interfaces differ substantially in the types of power saving modes that are supported and in the power that is used in each mode. Several 802.11b cards have custom adaptive algorithms implemented in firmware—examples include the Cisco Aironet 350 and Intel PRO/Wireless 2011B cards. In addition, we have found that the power usage of different cards can vary by a factor of two and that the transition cost of switching power modes differs by as much as 150 ms.

Our approach to handling this variability is to create a benchmark suite that can characterize the power usage of each new network interface card that we encounter. While running the benchmark suite, we use a digital multimeter to sample the current drawn by a mobile computer. We remove the batteries from the computer and sample current on the input power line. Since voltage is almost constant when power is drawn from the external power supply, current measurements alone are sufficient to determine power usage. Current samples are collected approximately 50 times per second. Many of our benchmarks require communication with a remote computer. For this purpose, we run a server program on a nearby computer that is LAN-connected with the wireless access point. The server program sends and receives bytes when requested.

The benchmark suite first measures the *base power* of the mobile computer—this is the power used when the machine is idle and the network card is not inserted. For each benchmark activity, we subtract the base power from the measured power usage to derive the additional power consumed by the activity.

Next, the benchmark suite measures the power used by the card in each mode. Every card that we have encountered supports at least CAM and PSM. Some support further adaptive modes that switch between CAM and PSM depending upon recent incoming traffic load. It may also be possible to disable the interface without turning the card completely off. For example, when the Cisco Aironet 350 interface is disabled, card

Card Mode	Cisco Aironet 350	Orinoco Silver
$P_{disabled}$	0.24 W	N/A
P_{PSM_idle}	0.39 W	0.19 W
P_{PSM_recv}	1.42 W	2.22 W
P_{PSM_send}	2.48 W	2.70 W
P_{CAM_idle}	1.41 W	1.21 W
P_{CAM_recv}	2.61 W	2.25 W
P_{CAM_send}	3.69 W	2.67 W

This figure shows the power usage of two 802.11b cards in several power modes. From top to bottom, the modes are: card disabled; card in PSM when idle, receiving, and sending data; and card in CAM when idle, receiving, and sending data. The standard deviation of all measurements is within 0.05 W.

Figure 5. Power usage of two 802.11b cards.

power usage is reduced by 150 mW. This saves less power than turning the card off completely but requires less time and energy for transitions.

The benchmark suite characterizes the cost of transitions between each supported power mode. It first initiates a mode transition and then exchanges a one byte ping with the server. The reported transition time is measured from the start of the mode change operation to the receipt of the ping response. The energy cost of the transition is the energy consumed by the network card during this interval (the cost of sending and receiving a byte is not significant within the granularity of our measurements).

The benchmark suite completes the characterization by measuring the power used to send and receive data in each power mode. It first sends 4 MB of data to the server using TCP and measures the average power usage. It then measures the average power used when the server retransmits the buffer to the client. Additionally, it measures the time to perform each transfer, which gives us an estimate of the maximum data rate that can be achieved in each power mode. Figures 5 and 6 show the results of running our benchmark suite for the Cisco Aironet 350 and Orinoco Silver PCMCIA cards.

Card characterization allows STPM to tune its behavior to the specific type of network card being employed. This modular approach means that we do not need to modify the STPM algorithm for each new type of wireless card. The benchmark suite takes approximately two hours to execute—however, characterization need only be done once for each model of wireless card. Ideally, a card manufacturer could run the benchmark suite and provide the characterization as part of the device driver.

4.3. Setting the power management policy

In this section, we describe the STPM algorithm. For the purpose of discussion, we first assume that the network card supports only two power modes: CAM and PSM. For such cards, STPM must decide when to transition to CAM and when to transition back to PSM. The next two sections describe how STPM makes these decisions. Section 4.3.3 then discusses how we generalize STPM to support cards with more than two power modes.

4.3.1. Transition to CAM

STPM transitions from PSM to CAM when:

- any application specifies a delay tolerance less than the maximum latency of PSM.
- any application discloses that the forthcoming transfer will be large enough such that the expected cost of performing the transfer in PSM is larger than the expected cost of switching to CAM and then performing the transfer.
- any application discloses a forthcoming transfer and, based on recent access patterns, STPM expects that there will be enough subsequent short transfers that the cumulative benefit of switching to CAM for the forthcoming transfers will be greater than the transition cost.

The first case is straightforward. The maximum delay of PSM is equal to the beacon interval, typically 100 ms. If a listen hint is specified that is less than the beacon interval, STPM switches from PSM to CAM.

Whenever a new transfer hint is disclosed, STPM checks for the second case. STPM performs a cost/benefit analysis

	Cisco Aironet 350		Orinoco Silver	
	Time	Energy	Time	Energy
PSM to CAM	0.40 s	0.51 J	0.23 s	0.24 J
CAM to PSM	0.41 s	0.53 J	0.26 s	0.31 J
Disable	0.00 s	0.00 J	N/A	N/A
Enable	0.39 s	0.51 J	N/A	N/A

This figure shows the time and energy to transition between CAM and PSM for each card. For the Cisco card, the time and energy to disable and re-enable the card are also shown. The standard deviation of all measurements is within 5%.

Figure 6. Cost of mode transitions.

by estimating the time and energy to perform the forthcoming transfer in both PSM and CAM. Through the transfer hint, the application has disclosed the expected number of bytes to send, B_s, and receive, B_r. The expected time, T, to perform the transfer is:

$$T = L + B_s/DR_s + B_r/DR_r \tag{1}$$

where the expected data rate for sending, DR_s, and receiving, DR_r, data in each mode is given by the card-specific characterization—this assumes that the wireless link is the bandwidth bottleneck in the transfer. The expected latency, L, is measured directly for CAM by observing the shortest time to complete a transfer over the recent past. For PSM, the expected latency is the CAM latency plus half the beacon interval.

The expected energy, E, to perform the transfer is calculated using the specified base power of the mobile computer, P_b, and the measured power usage of the card when idle, P_i, sending data, P_s, and receiving data, P_r. For each mode:

$$\begin{aligned} E = L \times (P_i + P_b) + B_s/DR_s \times (P_s + P_b) \\ + B_r/DR_r \times (P_r + P_b) \end{aligned} \tag{2}$$

Since equation (2) includes base power, it estimates the total energy used by the entire mobile computer, not just the network interface.

STPM calculates the total cost of switching to CAM by adding the estimated time and energy to perform the transfer in CAM to the transition costs given by the card-specific characterization. It compares the results to the estimated time and energy to perform the transfer in PSM. STPM transitions the card when it predicts that doing so will both save energy and improve performance. However, sometimes STPM estimates that a transition will improve either performance or energy conservation while hurting the other goal. In such cases, it uses the value of the knob that specifies the relative tradeoff between performance and energy conservation to decide whether to transition the card. Given n strategies, each of which has estimated time and energy costs, T_n and E_n, STPM first calculates the mean time, $\bar{T}$, and energy, $\bar{E}$, used by all strategies being compared. It then calculates a relative cost for each strategy, C_n as:

$$C_n = (T_n/\overline{T}) \times knob + (E_n/\overline{E}) \times (100 - knob) \tag{3}$$

For foreground hints, *knob* is the specified relative tradeoff between performance and energy conservation, ranging in value between 0 and 100. The intuition is that the time and energy values are first normalized by dividing them by the mean of the time or energy values being compared, then the knob is used to assign a relative weight for determining the final cost of the mode. For background hints, the knob is set to 0—since the transfer is latency-insensitive, only energy usage is considered.

In the third case, the time and energy of a single transfer is insufficient to justify switching to CAM. However, several subsequent transfers are expected and the time and energy saved over all transfers is expected to exceed the transition cost. To make this determination, STPM estimates the likelihood that subsequent transfers will occur in the near future.

STPM generates an empirical probability distribution of transfer hint frequency by observing the arrival of transfer hints. It maintains a histogram of the number of foreground transfer hints that occur closely correlated together in time. We refer to each such group of correlated hints as a *run*. A run begins when the first transfer hint is issued and ends when 150 ms pass with no foreground transfer being in progress. We chose 150 ms to differentiate the communication patterns of programs such as Web browsers that are driven by a human user from those of programs such as NFS that issue multiple sequential requests without human intervention. Our goal is to have each run correspond to a single interactive activity such as a NFS directory listing.

The *run length* is the number of foreground transfers issued during a run. STPM maintains a 1024 bucket histogram of observed run lengths—if a run exceeds 1024 transfers, STPM records it as having length 1024. The histogram data is periodically read from a `/proc` file system interface and saved to disk by a user-level daemon. When the module is first loaded, the daemon provides the initial histogram values from the saved data. In this fashion, STPM maintains data across reboots.

Currently, histogram entries are persistent, but we plan to investigate how histogram values can be aged to allow the module to adapt quicker to changes in network access patterns. We also plan to investigate the benefits of maintaining per-application histograms. The current approach of maintaining a single histogram for the client allows STPM to aggregate the access patterns of concurrently executing applications; yet, we feel that per-application histograms might allow STPM to adapt better to changing workloads.

We use the histogram to calculate the expected cost of switching to CAM prior to the nth transfer in each run. The expected time, T_n, to execute a run if a switch to CAM is done before the nth transfer is:

$$\begin{aligned} T_n = \sum_{i=1}^{n-1} L_{\text{PSM}} \times P(r \geq i) + \sum_{i=n}^{1024} L_{\text{CAM}} \times P(r \geq i) \\ + T_{TC} \times P(r \geq n) \end{aligned} \tag{4}$$

$P(r \geq x)$ is the probability that a run will equal or exceed length x—STPM derives this value from the run length histogram. T_{TC} is the time to switch to CAM. The intuition behind this equation is that STPM adds the total expected latency for transfers performed in PSM, the expected latency for transfers in CAM, and the expected transition time to derive the total expected time to execute the run of transfers. Similarly, STPM calculates the expected energy usage, E_n, as:

$$\begin{aligned} E_n = \sum_{i=1}^{n-1} L_{\text{PSM}} \times (P_{PSM\_idle} + P_b) \times P(r \geq i) \\ + \sum_{i=n}^{1024} L_{\text{CAM}} \times (P_{CAM\_idle} + P_b) \times P(r \geq i) \\ + (E_{TC} + (T_{TC} \times P_b)) \times P(r \geq n) \end{aligned} \tag{5}$$

Using equations (4) and (5), STPM calculates the expected time and energy if it switches to CAM after an application issues the nth foreground transfer hint in a run. Additionally, STPM calculates the expected time and energy if it never switches to CAM. It uses equation (3) to compare the different policies and chooses the one that has the minimum cost. Since this calculation is relatively time-consuming (2 ms) and the input data is slow to change, STPM performs this calculation every 10 minutes.

4.3.2. Transfer to PSM

STPM transitions from CAM to PSM when no transfers are in progress, no application has specified a delay tolerance less than the maximum latency of PSM, and it estimates that the network interface will be idle long enough to overcome the transition costs of switching modes. To aid in this decision, STPM maintains a 1024 bucket histogram of the length of the interval between each run. Each bucket corresponds to a 100 ms period; extremely long intervals are recorded as having the maximum value of 102.4 seconds. The interval histogram is maintained in an identical fashion to the run length histogram described in the previous section.

STPM transitions to PSM when it believes that the expected benefit of reduced power usage during the forthcoming idle period most exceeds the performance and energy cost of beginning the next run in PSM instead of CAM.

The expected time and energy to perform the next run starting in PSM, T_{init_PSM} and E_{init_PSM}, have already been calculated—these values are the expected time and energy of the policy chosen in Section 4.3.1. The expected time and energy to perform the next run starting in CAM, T_{init_CAM} and E_{init_CAM}, are:

$$T_{init\_CAM} = \sum_{i=1}^{1024} L_{\text{CAM}} \times P(r \geq i) \tag{6}$$

$$E_{init\_CAM} = \sum_{i=1}^{1024} L_{\text{CAM}} \times (P_{CAM\_idle} + P_b) \times P(r \geq i) \tag{7}$$

These equations assume small data transfers. Therefore, transfer time is dominated by latency, L_{CAM}, and card power usage is assumed to be close to its measured idle usage, P_{CAM_idle}. STPM next calculates the expected time and energy costs of switching to PSM n/10 seconds after the end of the previous run—these values correspond to the 100 ms histogram buckets. The expected time is:

$$\begin{aligned} T_n = &\sum_{i=1}^{1024} 0.1 \times P(l \geq i) + T_{init\_PSM} \times P(l \geq n) \\ &+ T_{init\_CAM} \times P(l < n) \end{aligned} \tag{8}$$

STPM calculates $P(l \geq n)$, the probability that the interval length will be greater than n/10 seconds, using the interval histogram. The first line of the previous equation is the expected length of the current interval, and the second line is the expected time to perform the succeeding run. The energy cost of switching to PSM n/10 seconds after the end of the previous run is calculated similarly:

$$\begin{aligned} E_n = &\sum_{i=1}^{n-1} 0.1 \times (P_{CAM\_idle} + P_b) \times P(l \geq i) \\ &+ \sum_{i=n}^{1024} 0.1 \times (P_{PSM\_idle} + P_b) \times P(l \geq i) \\ &+ E_{init\_PSM} \times P(l \geq n) + E_{init\_CAM} \times P(l < n) \end{aligned} \tag{9}$$

STPM calculates the expected time and energy costs of switching to PSM at each 100 ms interval, as well as the expected cost of remaining in CAM. It then chooses the policy that minimizes equation (3). This calculation is also performed once every ten minutes, at the same time that STPM decides when to switch to CAM.

4.3.3. Generalizing the model

The previous two sections have shown how STPM creates a transition policy for a card that supports only PSM and CAM. For cards that support more than two modes, there are many more possible policies from which STPM could choose. For instance, after a period of time with no transfers, STPM could transition the Cisco card from CAM to PSM, it could disable the card, or it could decide to switch to PSM and later disable the card if no further transfers occur for another period of time.

Since the number of possible strategies grows exponentially with the number of modes, we employ a heuristic to limit the search space. STPM first calculates the lowest cost policy that transitions to each mode. It then calculates the lowest cost hybrid policies that switch to one mode at the time calculated for the single-switch policy, then make a further transition at some later time. For example, STPM might decide that the lowest cost single-transition policy is to switch to PSM after 300 ms. It would then consider hybrid policies that switch to PSM after 300 ms and disable the card after some further period of time. Since most cards support only a few power modes, this strategy is computationally feasible.

4.4. Support for unmodified applications

We cannot reasonably expect all applications to be modified to use STPM. We must therefore support unmodified applications as best as possible. The strategy that we employ is to identify applications that are not disclosing hints, observe their network traffic, and issue hints on their behalf. Clearly, unmodified applications cannot hope to realize the full benefit of STPM. Two of the five design principles outlined in Section 3, knowing application intent and respecting the critical path, require explicit application hints. Since hints issued on behalf of unmodified applications are generated using heuristics, they often will be less accurate than hints explicitly issued by modified applications. So, we expect that STPM will provide less benefit for unmodified applications than it provides for modified ones. However, the remaining three design principles require no

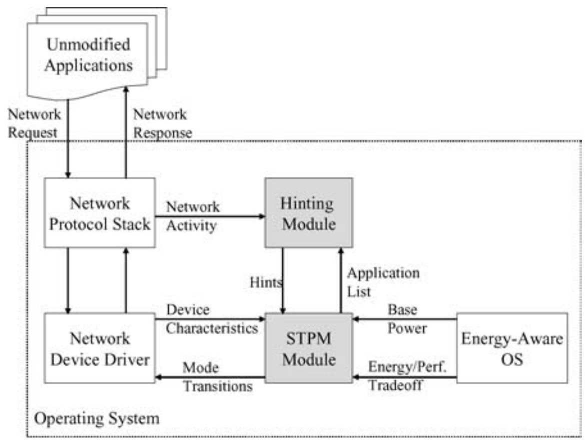


Figure 7. STPM for unmodified applications.

explicit application support. Thus, we expect that unmodified applications will still benefit from STPM.

As shown in the figure 7, we support unmodified applications by inserting a hinting module into the Linux kernel. The hinting module intercepts incoming and outgoing packets at the IP layer by adding a protocol handler to the network stack using the Linux `dev_add_pack` system call.

4.4.1. Detecting unmodified applications

Since the hinting module observes all network traffic on an interface, it is able to detect the presence of unmodified applications. It maintains a hash table that maps network ports to the processes that are reading from and writing to those ports. When the hinting module detects a packet that is being sent from or received by a port that is not in the hash table, it first determines the process that is associated with the new port. It then queries the STPM module to determine whether that process is disclosing hints. It adds the new port to the hash table, marking it as belonging to either a modified (hinting) or unmodified (non-hinting) application.

After the hinting module creates a mapping of port to process for the first packet, it can consult its hash table on the arrival of subsequent packets to determine whether the associated application is modified. The drawback of caching this mapping is that the hinting module will fail to detect an unmodified application that uses a port number that was previously used by a modified application. However, we expect this case to be extremely rare since Linux seldom reuses dynamically allocated port numbers. We further minimize the chances of this happening by evicting stale mapping from our hash table after two hours. In the unlikely event that an unmodified application is not detected, it will not benefit from using STPM.

4.4.2. Generating hints for unmodified applications

The STPM module uses hints to determine when transfers are taking place and whether those transfers represent background or foreground activity. The hinting module generates these hints for unmodified applications by observing the pattern of incoming and outgoing packets. The hinting module conservatively assumes that all transfers are foreground ones

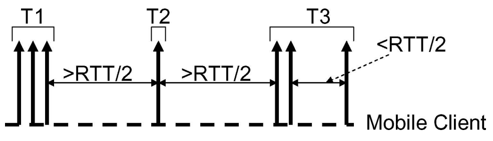


Figure 8. Example of a hinting strategy.

and latency-sensitive. It further assumes that all transfers are initiated by the mobile client. These two assumptions hold true for many common sources of network traffic, including communication with file, web, and mail servers.

Given these heuristics, the remaining problem faced by the hinting module is one of determining when each transfer begins and ends. Essentially, we need to define what constitutes a transfer by only observing the network activity. The hinting module assumes that packets with closely correlated arrival times are part of the same transfer. Since it also assumes that all transfers are client-initiated, outgoing network packets are considered to be part of the request and incoming packets are considered to be part of the response.

Specifically, the hinting module uses the expected round-trip time (RTT) to differentiate when one transfer ends and the next begins. If the inter-arrival time between two outgoing packets is less than half the expected round-trip time, then the two packets are considered to be part of the same request. We assume that each transfer is initiated only after the previous transfer has completed. If the gap between the two packets is much smaller than the expected round-trip time, it is unlikely that the second packet represents a new request issued after receiving a response to the first packet. Of course, this heuristic performs poorly if an application issues concurrent transfers. In that case, higher-layer knowledge such as explicit hints is needed to differentiate the transfers.

To illustrate this algorithm, figure 8 shows a sample time sequence of several outgoing packets. Using the heuristic described above, these packets are grouped into three distinct transfers because there are two idle periods that exceed half the expected round-trip time.

The hinting module considers all incoming packets on a network port to be part of the response to the last request that was issued. If no network activity occurs for a period that exceeds twice the round-trip time then the current transfer is assumed to have completed. Note that some timeout is necessary because if the last transfer is never assumed to complete, then the STPM module will not be able to transition the network interface to PSM.

Figure 9 shows a sample time sequence of several incoming and outgoing packets. As in figure 8, these packets are grouped into three distinct transfers. Since the gap between the third and fourth outgoing packets exceeds half the expected round-trip time, a new transfer, T2, is considered to begin with the fourth outgoing packet. All incoming packets prior to this are considered to be part of the first transfer, T1. Between transfers T2 and T3 there is a long idle period. After the network has been idle for twice the expected round-trip time, transfer T2 is assumed to have completed; the STPM module may transition

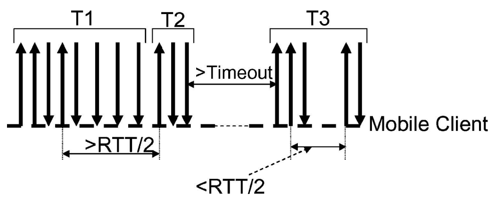


Figure 9. Example of hinting strategy with incoming and outgoing packets.

the network to PSM at this point. The next transfer, T3, begins when the next outgoing packet is observed.

The hinting module estimates round-trip time for each port using a weighted average of recent transfer times, in a manner similar to that employed by Noble [20] and Kim [11]. Unfortunately, we can only heuristically determine when transfers begin and end. We therefore base our estimation only upon round-trip time observations in which we have reasonable confidence. Specifically, we use only the round-trip time of transfers that begin after a long idle period because we are reasonably certain about the start time of such transfers.

Our task is complicated even further because round-trip time depends upon the current power-saving mode of the network interface. When the card is in PSM, we use the beacon period of the access point as the expected round-trip time. When the card is in CAM, we estimate round-trip time using the method described above.

Figure 10 shows the pseudo code for our hinting module. `ReceivePacket` is called when the hinting module observes an incoming packet, and `SendPacket` is called when the module observes an outgoing packet.

```
ReceivePacket (port) {
//check to see if this port belongs to a non-hinting application
    if (!hinted_port (port)) {
//check to see if a hint is already in progress for this port
        if (is_hint_in_progress (port)) {
            num_rcvd_packets[port]++;
            reset_timer (2*get_RTT());
        }
        if (first_send_flag (port) {
//Special handling for CAM mode to calculate the network delay
            if (mode == CAM) {
                this_RTT = now - first_send_time (port);
//By default, ALPHA is set to 0.5
                RTT = RTT * ALPHA + this_RTT (1 - ALPHA);
            }
            clear_first_send_flag (port);
        }
    }
}

on_timeout () {
    end_transfer_hint (port);
}

SendPacket (port) {
    if (!hinted_port (port)) {
        if (is_hint_in_progress (port)) {
//check to see if we need to reissue a hint if it is a new transfer
            if ((now - last_send_time) > get_RTT()/2 &&
                num_rcvd_packets > 0) {
                    end_transfer_hint (port);
                    begin_transfer_hint (port);
                }
            else {
                reset_timer (2*get_RTT());
            }
        }
        else {
            begin_transfer_hint (port);
//Special handling for CAM mode to calculate the network delay
            if (mode == CAM) {
                set_first_send_flag (port);
                first_send_time (port) = now;
            }
        }
        last_send_time = now;
    }
}
```

Figure 10. Pseudo code for the hinting module.

5. Evaluation

How much does self-tuning power management improve application performance and extend mobile computer battery lifetime?

To answer this question, we modified several network-intensive applications to disclose hints and measured application performance and energy usage when our self-tuning power management algorithm was used. We compared these results to those achieved using the static and adaptive power management algorithms natively supported by the 802.11b cards in figure 5.

5.1. Methodology

The primary client platform for our evaluation is a HP iPAQ 3870 running the Linux 2.4.18-rmk3 kernel. This handheld computer has a 206 MHz StrongArm processor, 64 MB of DRAM, and 32 MB of flash memory. The measured base power of the iPAQ (when idle with no network card inserted) is 1.44 Watts. Unless otherwise noted, the client uses the Cisco 802.11b card described in figure 5 to communicate with a Cisco Aironet 350 wireless access point with 100 ms beacon interval. The server in our experiments is a Dell Precision 350 with 3.06 GHz processor and 1 GB DRAM running the Linux 2.4.18–19.8.0 kernel—the server and access point are connected with a 100 Mb/s switch.

The Cisco PCMCIA card supports three power modes: CAM, where no power management is used; PSM-static, where the card's receiver is periodically disabled to save power; and PSM-adaptive, which switches between PSM-static and CAM depending upon the incoming traffic load. Since PSM-adaptive is implemented in the card firmware, it can change power modes faster than our STPM module—our measurements indicate that PSM-adaptive almost always transitions in less than 100 ms. In contrast, because STPM is implemented as a kernel module, its transition costs are approximately 400 ms.

We investigated the benefits of self-tuning power management for four network-intensive application scenarios: file access using the Coda distributed file system, file access using NFS, playing streaming audio using Xmms, and hosting thin-client remote X applications. We first ran each scenario using the power management methods natively supported by the 802.11b card. We then executed each scenario using STPM.

Unless otherwise noted, the tuning knob for STPM was set at 50 to equally weight performance and energy conservation. We also warmed the STPM prediction algorithm by specifying an initial probability distribution for network accesses. The alternative approach, starting with no access history, leads to a short period of initial volatility that makes experiments less repeatable. Further, the warming approach better reflects STPM steady-state performance.

To generate the initial access distribution, we replayed a trace of distributed file system accesses using Mummert's DFSTrace tool [18]. This tool re-executes previously recorded file system operations such as `open` and `mkdir`, preserving the inter-arrival time of each system call. We replayed a 30 minute segment of Mummert's `purcell` trace—this trace captures interactive software development activity such as file editing and compilation. We placed the files accessed by the trace in the Coda distributed file system [12]. During trace replay, the STPM kernel module observed Coda's network activity as it communicated with a file server to read and write data. We saved the access distribution observed by our module and used it to warm the STPM module before the execution of each scenario.

For each experiment, we measured application performance using the `gettimeofday` system call. To obtain energy measurements, we removed the iPAQ's batteries and powered the handheld through its external power supply. We sampled the current drawn by the iPAQ approximately 50 times per second using an Agilent 34401A digital multimeter. We calculated system power usage by multiplying each current sample by the mean voltage drawn by the iPAQ—separate voltage samples are not necessary since the variation in voltage drawn through the external power supply is very small. The total energy consumed by the system over a specific period of time is the sum of the power samples during that period multiplied by the measurement interval.

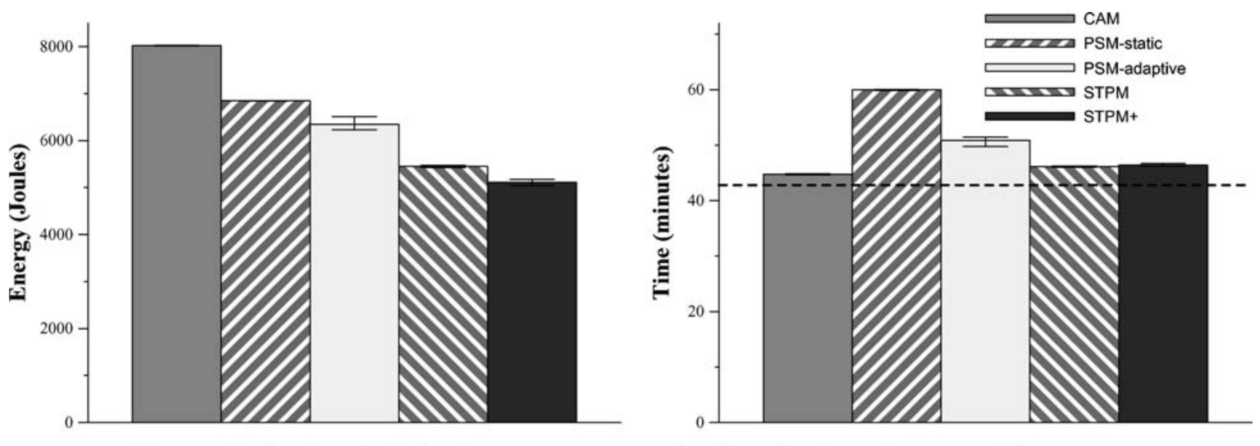
5.2. *Coda distributed file system*

We first examined the effectiveness of STPM for Coda file system [12] activity. Coda presents a single file system image across multiple computers by reading file data accessed by a client from remote file servers and writing modifications made on the client back to the servers. Coda caches file data on the client to improve performance and guard against disconnection—on the iPAQ, we used a 16 MB DRAM-based Coda cache. Since Coda uses RPCs for communication, we modified the Coda client to call `TransferHintBegin` before issuing each RPC and `HintEnd` after the RPC completes. The vast majority of Coda RPCs are synchronous—for these RPCs, we issue foreground hints. Although we issue background hints when Coda performs asynchronous writes and prefetches, these activities occur infrequently in our experiments. When Coda starts, it specifies a listen hint that is a fraction of the RPC timeout value—this ensures that server-initiated RPCs arrive at the client promptly. These modifications added 235 lines of source code.

To generate a realistic access pattern, we again used Mummert's `purcell` trace. From the trace, we selected a different trace segment than the one that we used to warm the STPM access history. The selected trace segment has 10,000 file operations and includes, in total, 42.8 minutes of delay between file operations. These delays typically represent user think time, but can also reflect application processing such as the time for gcc to compile a program. DFSTrace sleeps for the recorded inter-arrival time before issuing the next file request, so an infinitely fast system would complete trace replay in 42.8 minutes.

5.2.1. *Handheld results*

We first replayed the trace using the iPAQ handheld with the Cisco PCMCIA card—figure 11 shows the results. Since



These graphs show how the choice of power management policy affects the time and energy needed to execute a trace of Coda operations. The dashed line in the rightmost graph shows the amount of think time in the trace. Each bar shows the mean of three trials—the error bars show the value of the minimum and maximum trial.

Figure 11. Benefit of STPM for Coda.

CAM uses no power management, it yields the best possible performance. However, its energy cost is significantly higher than the other strategies. PSM-static (i.e. default 802.11 power management) reduces energy usage but has very poor performance, delaying trace execution by an additional 15 minutes. The dashed line in the rightmost graph shows the total amount of think time in the trace—thus, the cumulative delay added due to file system activity and power management is shown by the portion of each bar that exceeds the dashed line. This delay is especially bad for interactive applications because it represents time that the user must waste waiting for the application making the file accesses to respond. The Cisco card's PSM-adaptive strategy does substantially better by dynamically switching between PSM and CAM depending upon traffic load. Compared to CAM, PSM-adaptive adds only 6 minutes of cumulative delay and reduces energy usage 14%.

We next replayed the trace on the same hardware using STPM. To provide a fair comparison with card-based power management, we first limited STPM to use only CAM and PSM, i.e. only those modes available to the PSM-adaptive strategy. STPM initially chose to switch to CAM before the 3rd transfer hint in each run and to switch to PSM after 300 ms with no foreground activity. During trace replay, STPM usually became slightly more aggressive and switched to PSM after only 200 ms.

Compared to PSM-static, STPM reduces cumulative delay by 80% and energy usage by 21%. In figure 11, cumulative delay is compared by examining the height above the dashed line for each bar. Compared to PSM-adaptive, STPM reduces cumulative delay by 58% and energy by 14%. Compared to CAM, STPM adds slightly more than a minute of cumulative delay, but reduces energy consumption by 32%. This means that STPM allows the user to accomplish 48% more work before the battery expires.

How much more energy reduction is feasible? The minimum trace execution time, 44:45 minutes, is given by the time to execute the trace with the card in CAM. The minimum power usage, 1.83 Watts, occurs when the card is in PSM and no data is sent or received. The product of these two values, 4914 Joules, is a lower bound on the minimum energy usage achievable by an adaptive strategy that switches between CAM and PSM. This loose lower bound shows that an optimal strategy could at best be 10% more energy-efficient than STPM. Further, this lower bound is unachievable since it omits the cost of sending data, receiving data, and state transitions—thus, STPM is probably much closer to optimal.

When we allow STPM to disable the Cisco card, it reduces energy usage by an additional 8% as shown by the bars labelled "STPM+" in figure 11. The mean cumulative delay increases slightly but the difference between STPM and STPM+ is within experimental error. This demonstrates an important advantage of STPM: we did not design an entirely new algorithm to take advantage of the new power management mode. We simply made the benchmark data for disabling the card available to the STPM module and provided a function that performed the relevant state transitions. STPM automatically determined the instances in which the new power mode could be profitably employed.

One downside of STPM+ is that disabling the network interface prevents the client from accepting remote connections for ssh, telnet, or similar applications. In contrast, STPM only slightly delays such connections. To support remote connections, STPM+ could periodically poll for incoming connections at an interval specified by a global listen hint. Alternatively, the client could use an external signaling mechanism such as wake-on-wireless [23] to detect new connections and use STPM+ to manage the interface state after connection establishment.

5.2.2. *The importance of base power*

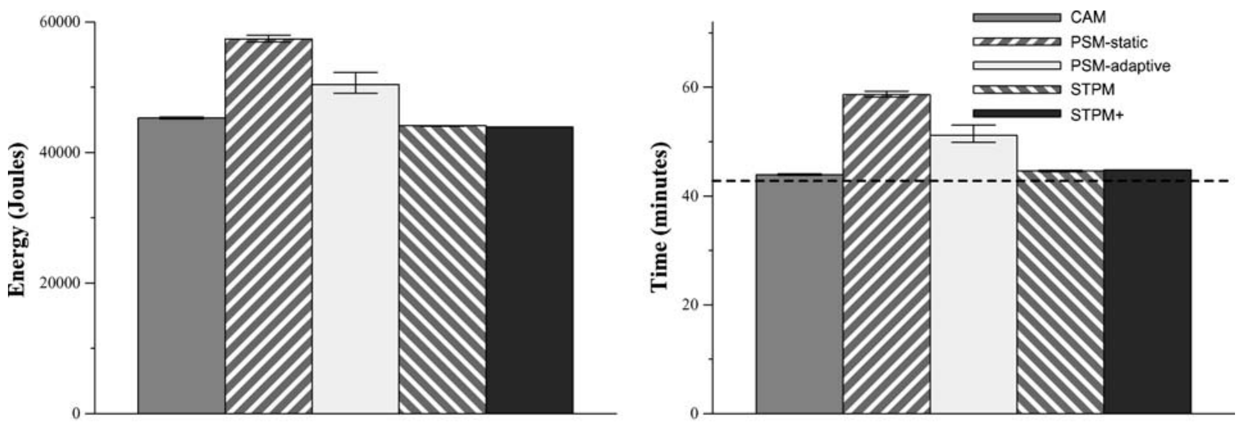
To demonstrate how well STPM adapts to different hardware platforms, we also replayed the Coda trace on an IBM T20 ThinkPad laptop computer. The base power of the laptop, 15.8 Watts, is an order of magnitude greater than that of the iPAQ.

Figure 12 shows a disturbing result: both PSM-static and PSM-adaptive increase the total energy used to replay the trace! This occurs because the wireless network represents only a small portion of the total power used by the laptop. While switching to PSM reduces the power used by the PCMCIA card by 72%, it only reduces total system power by 6%. Further, PSM-static and PSM-adaptive both increase the time needed to complete the trace because they delay file operations. Although this increase is only about 6 minutes for PSM-adaptive, laptop power usage exceeds 15 Watts during such delays. Thus, as base power increases, the benefit of PSM is less during periods without network activity, and the cost of PSM is higher during periods with frequent network activity. This implies that power management should be more conservative as base power increases.

The energy usage and performance of STPM on the laptop is shown by the fourth bar in each graph in figure 12. Because STPM accounts for base power, it is considerably more conservative when running on the laptop. On average, it only enters PSM after 15 seconds of inactivity. This strategy works: STPM decreases total system energy usage by 2.6% while increasing the total time to execute the trace by only 1.5%. When STPM is allowed to disable the card, results do not change much. The module rarely disables the wireless card because the benefit is so slight—disabling the card reduces system power usage by less than 1% compared to PSM.

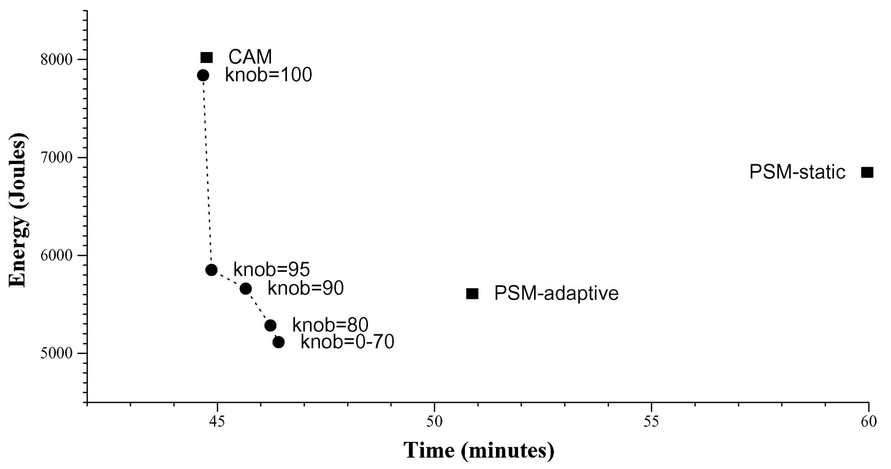
5.2.3. *Exploiting the performance/energy tradeoff*

Using the iPAQ and Cisco card, we explored the impact of changing the performance/energy knob for STPM. Figure 13 plots the tradeoff between energy usage and the execution time for the `purcell` trace. The square marks show the three native modes supported by the Cisco card, while the circles show STPM with five different knob settings. When the knob for STPM is set to 100, the performance is equivalent to CAM (PSM is employed once at the beginning of the trace, accounting for a slightly lower energy usage). The remaining four marks show that as the knob value is decreased, energy



These graphs show the time and energy costs of executing the Coda trace on an IBM T20 ThinkPad laptop. These results can be compared to those in figure 11 that were collected using an iPAQ client. Each bar shows the mean of three trials—the error bars show the value of the minimum and maximum trial.

Figure 12. Coda results on IBM T20 laptop.



This figure shows how performance and energy usage vary for the Coda scenario depending upon the value of the STPM knob parameter. Each circle represents results using STPM for a different knob value—knob values of 0 through 70 yield equivalent results. The boxes show the performance and energy usage achieved when the native modes of the Cisco card are used.

Figure 13. The performance/energy tradeoff.

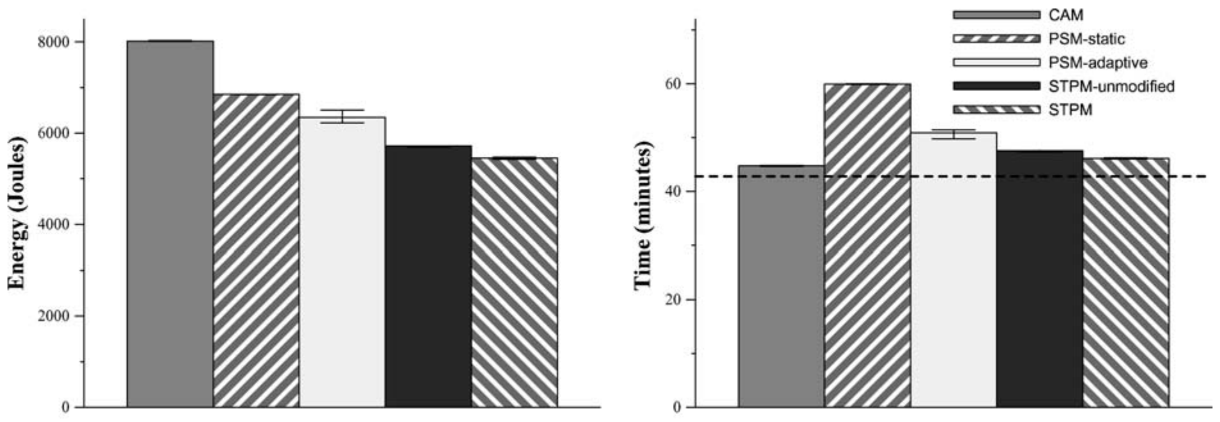
conservation improves but performance is decreased. Knob values below 70 yield an equivalent strategy to that realized with a knob value of 0, and hence have the same performance and energy usage.

These results have a powerful property: decreasing the knob value never yields increased energy usage, and increasing the knob value never yields reduced performance. It is clear that the effect of changing the knob value is non-linear. For instance, changing the knob from 100 to 95 substantially reduces energy usage, but changing the knob value from 70 to 0 has no effect. Partly, this occurs because STPM will not choose an inferior strategy if another is available that is expected to yield better performance and energy conservation. For instance, one might expect that STPM would behave identically to PSM-static with a knob value of 0. However, the strategy chosen by STPM yields better performance and greater energy conservation than PSM-static.

5.2.4. Benefit of application hints

We next examined the effectiveness of our hinting module by running an unmodified version of Coda that does not disclose any STPM hints. The first goal for this experiment was to determine whether STPM can improve application performance and energy conservation even without explicit application hints. The second goal was to quantify the additional benefit that can be realized by modifying applications. We ran this experiment on the iPAQ handheld using the Cisco PCMCIA card.

The results in figure 14 are the same as those in figure 11 except for the additional bars labeled "STPM-unmodified" that show results for STPM using the hinting module described in Section 4.4.2. These results show that STPM with the hinting module substantially outperforms both PSM-static and PSM-adaptive. Compared to PSM-static, STPM-unmodified reduces energy usage by 16% and improves latency by 72%.



These graphs show how the hinting module with STPM performs compared to other power management policies. Each bar shows the mean of three trials—the error bars show the value of the minimum and maximum trial.

Figure 14. Benefit of the hinting module for Coda.

However, STPM-unmodified performs poorer than STPM with explicit application hints. The unmodified version of Coda requires 5% more energy to execute the trace and takes 8% longer. This difference can primarily be attributed to the fact that hinting module must rely on heuristics to determine when each transfer begins and ends. These heuristics will sometimes be incorrect, especially when dealing with concurrent transfers. In contrast, the explicit hints provided by the applications are always accurate.

Overall, these results show that STPM with the hinting module is able to improve performance and save energy even without explicit hints. This provides some evidence that the heuristics we are employing are reasonable. During trace execution, we found that STPM with the hinting module was more aggressive in switching to CAM and a more conservative in switching back to PSM. For example, STPM with explicit hints switches to CAM after observing three transfers in a row, STPM without hints switches to CAM after observing one or two transfers.

We should note that Coda is a good target application for STPM-unmodified because its behavior closely matches the assumptions made by the hinting module. Most of Coda's transfers are client-initiated and the majority of transfers are foreground requests. Still, we believe that such behavior is not atypical for many applications important in mobile computing today.

5.3. Network file system

Next, we examined the benefit of STPM for the Network File System (NFS) version 2 [19]. As with Coda, NFS communication is RPC-based—we modified NFS to issue a transfer hint before each RPC begins and end the hint when the RPC completes. Unlike Coda, NFS revalidates cached files before using them—thus, each file access generates at least one RPC. NFS v2 does not delay writing modifications back to the server, nor does it prefetch whole file data. Since there are few sources of background traffic, all hints issued are foreground ones. These modifications added 92 lines of source code.

5.3.1. Cisco results

Figure 15 shows results from replaying the `purcell` trace on the iPAQ using NFS as the underlying distributed file system. For these experiments, we again used the Cisco PCMCIA card. The results for CAM and PSM-static are similar to those for Coda. However, PSM-adaptive does slightly better because NFS issues several concurrent RPCs for large reads and writes. This concurrency generates sufficient network activity for PSM-adaptive to switch immediately to CAM.

During trace execution, STPM becomes more aggressive in switching to CAM. With Coda, STPM switches to CAM before the third RPC; with NFS, STPM switches to CAM before the first or second RPC by the end of the trace. During replay, STPM learns that NFS is likely to issue many RPCs closely correlated in time due to its revalidation strategy and a general tendency to issue more RPCs per file operation. STPM gradually adjusts its power management strategy as it observes the NFS request distribution.

Using just CAM and PSM, STPM outperformed PSM-adaptive for NFS, eliminating over 3 minutes of interactive delay and reducing system energy usage by 6%. Using a calculation similar to that in Section 5.2.1, we can derive a loose lower bound of 4313 Joules for energy usage during NFS trace execution—this shows that STPM energy usage is within 11% of optimal.

When STPM was allowed to disable the Cisco card, it reduced energy usage by an additional 8% with no noticeable change in performance. Compared to CAM, STPM+ reduced iPAQ energy usage by 38%, which would allow 62% more work on battery power.

5.3.2. Adapting to interface characteristics

We also wished to verify that STPM adapts to 802.11b cards with significantly different attributes. For this purpose, we selected the Orinoco Silver 802.11b card detailed in figure 5. Whereas the maximum data rate for the Cisco card is 11 Mb/s, the Orinoco card is limited to 2 Mb/s. Further, the transition costs for the Orinoco card are lower than those for the Cisco

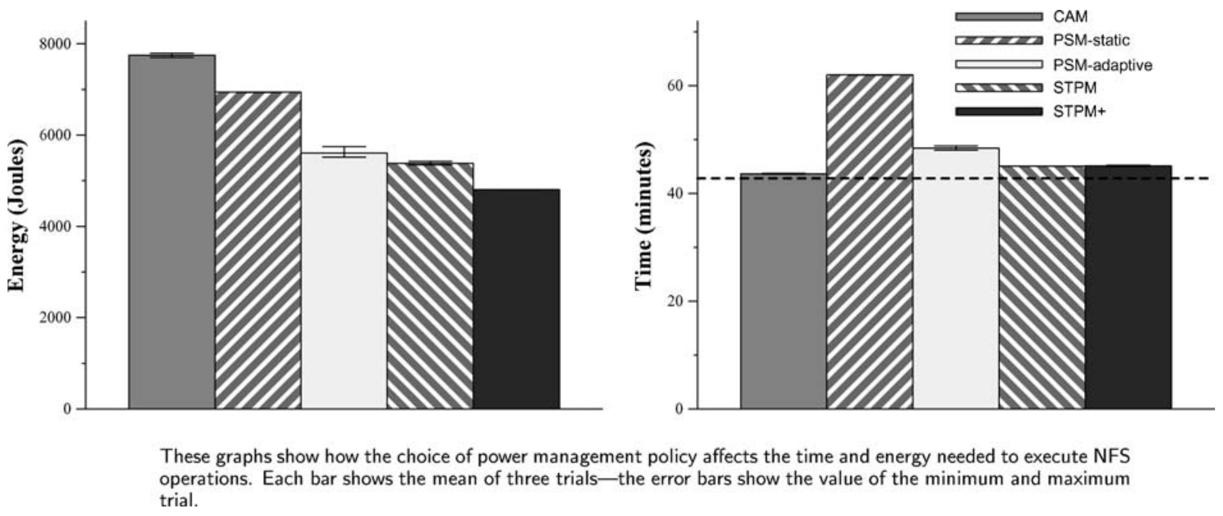


Figure 15. Benefit of STPM for NFS.

card. The Orinoco card cannot be easily disabled to save power, nor does it support an adaptive power management strategy.

Figure 16 shows the results from replaying the `purcell` trace for NFS running on an iPAQ with the Orinoco card. STPM is more aggressive for the Orinoco card—it switches to PSM after only 200 ms compared to a waiting period of 300 ms for the Cisco card. Since the transition costs for the Orinoco card are lower, STPM employs power saving modes more often. STPM significantly outperforms PSM-static, reducing energy usage by 15% and execution time by 23%.

5.4. Xmms streaming audio

We next modified XMMS-embedded to disclose hints when it streams live audio from an Internet server and plays it on an iPAQ. We chose this application because its communication pattern differs significantly from the file systems that we had explored previously. Since the data rate is low (128 Kb/s using the streaming MP3 format), the amount of client buffering is the primary factor that determines whether power management can be used. XMMS-embedded buffers several seconds of audio on the client; thus, it can tolerate the small delays in packet arrival caused by PSM. We modified XMMS-embedded to specify a maximum delay tolerance of 200 ms using `BeginListenHint`. Our modifications added 7 lines of source code.

As shown in figure 17, STPM reduces power usage by 25% compared to CAM. PSM-static is the optimal strategy for this application. However, STPM uses only 2% more energy than PSM-static—this difference reflects the overhead of our module. In contrast, PSM-adaptive always remains in CAM while audio is played. Since PSM-adaptive has no knowledge of application intent, it must assume that latency is critical for audio streaming traffic. In our experiments, no audio packets were dropped because application buffering was sufficient to overcome any jitter caused by power management.

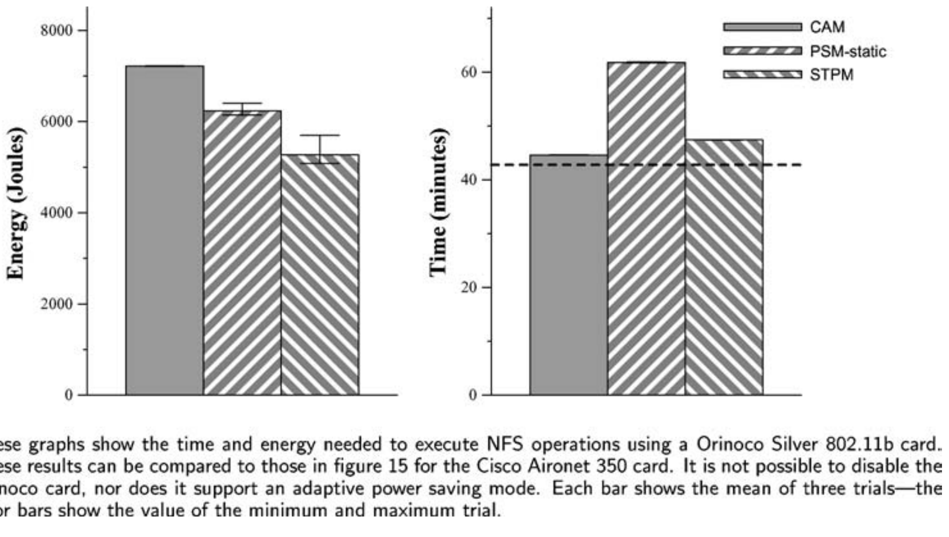
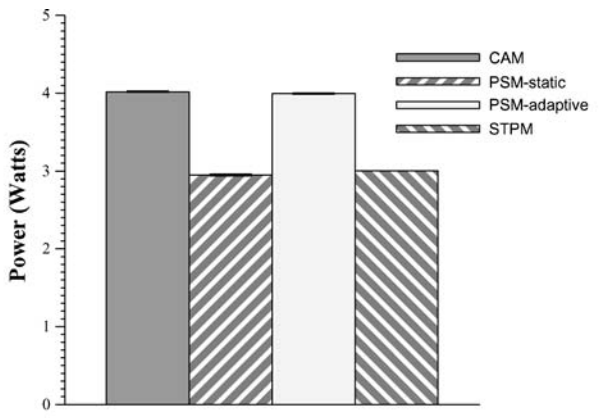


Figure 16. NFS results for Orinoco card.



These graphs show how the choice of power management policy affects the power used to play streaming audio on an iPAQ using XMMS. Each bar shows the mean of three trials—the error bars show the value of the minimum and maximum trial.

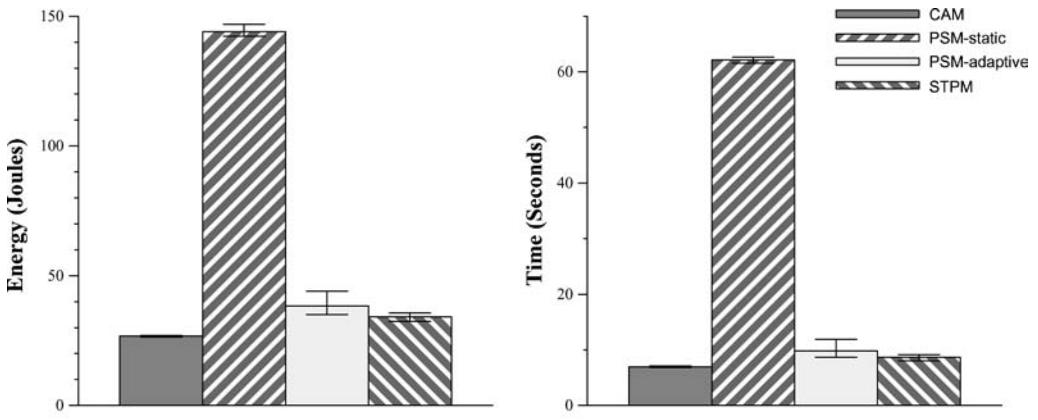
Figure 17. Benefit of STPM for Xmms.

5.5. *Thin-client using remote X*

Next, we examined thin-client display of remote X applications on the iPAQ. Remote display allows users to run applications that are too computationally expensive for a handheld, or which have not been ported to the handheld processor. We modified XFree86 4.3 to give application hints when reading incoming messages and transmitting responses to remote clients. To ensure acceptable interactive response time for long-running remote applications, the X server specifies a maximum delay on incoming traffic of 100 ms whenever a remote session exists. Our modifications added 51 lines of code.

In our experiments, we first started the Gnumeric spreadsheet on the server with its display hosted on a remote iPAQ. We used the iPAQ GUI to load a spreadsheet that contained three columns with roughly 4500 data points each, as well as two complex graphs. We then viewed the spreadsheet and closed the application. We measured the time to perform each action by observing network traffic on the server. We preceded each interactive action with a pause that enabled us to determine when the action began and ended.

Figure 18 shows the cumulative time and energy to perform all interactive activities—it does not reflect any user think time or pause between activities. STPM is over three times more energy efficient and six times faster than PSM-static. STPM uses 12% less energy than PSM-adaptive and is 13% faster. Although CAM is 25% faster and uses 28% less energy than STPM, it uses more power during user think time. Figure 19 shows how user think-time impacts energy usage. If think time exceeds 6.5 seconds, STPM uses less energy than CAM over the entire interactive episode.



These graphs show how the choice of power management policy affects the time and energy needed to run remote X applications. The rightmost graph shows the time to start the Gnumeric application running on the server with the display on the iPAQ client, load a spreadsheet, and close the application—the leftmost graph shows the total energy expended. Each bar shows the mean of three trials—the error bars show the value of the minimum and maximum trial.

Figure 18. Benefit of STPM for remote X.

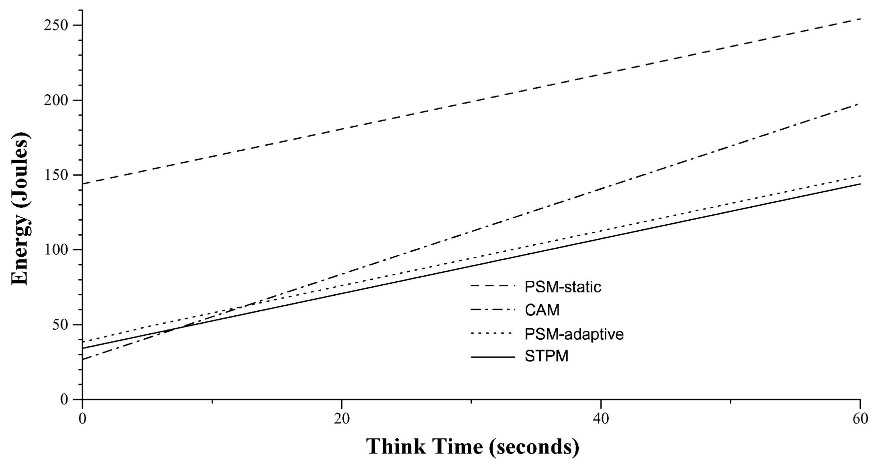


Figure 19. Effect of think time on X energy usage.

6. Related work

To the best of our knowledge, STPM is the first power management algorithm to simultaneously tune its behavior to the characteristics of the network interface, the base power usage of the mobile computer, and the intent and access patterns of applications. STPM differs from previous hint-based approaches to network power management because it requires applications to disclose only current activity, not future activity, and because it uses on-line modeling of application access patterns to set network power management parameters.

Lu et al. have also explored the use of application hints in power management [16]. They use a predictive policy to estimate when to wake up the network interface. Their hints are at a coarser granularity than those used by STPM and require applications to have knowledge of future activity: for instance, they allow an editor to specify that it requires the network interface over the next few minutes to auto-save a buffer. Their work focuses on scheduling network availability for applications with relatively flexible deadlines, while STPM focuses on supporting latency-sensitive applications. From a broader perspective, application hints have also been successfully used to improve disk power management [27] and I/O prefetching [17,22].

Kravets and Krishnan [14] also advocate an application level power management strategy. Their work provides a transport level protocol to improve energy usage. STPM complements their solution in that it predicts opportunities for performance improvement and energy conservation.

Wake-on-wireless [23] uses a low-power network to signal a mobile client when packets are waiting at the base station. Current wireless network transition costs are prohibitive for using this technique to disable and enable the network card between transfers for applications like NFS. However, wake-on-wireless seems extremely promising for supporting incoming connections for server processes like the X server and sshd. We can envision a hybrid strategy that uses wake-on-wireless to detect incoming connections during idle periods and uses STPM to manage existing connections.

Krashinsky's bounded slowdown (BSD) protocol [13] disables the network interface to save power while bounding the relative delay on transfer round-trip time. BSD differs from STPM in that it does not explicitly consider transition costs or the base power usage of the device. Unlike STPM, BSD is implemented without knowledge of application intent. The advantage of not using application hints is that BSD can show benefit for unmodified applications. The disadvantage is that BSD must behave conservatively, limiting the potential for energy savings. Further, BSD requires some small modifications to the 802.11b protocol, making deployment on current hardware more difficult.

Simunic's TISMDP [24] operates at a coarser granularity than STPM. TISMDP only decides when to transition from a higher-power state to a lower-power state. It immediately resumes upon the arrival of the next request. Thus, for intermittent activity like NFS RPCs, a transition cost must be paid for each RPC.

Chandra [3] explores how the regular nature of streaming multimedia can be exploited to improve power management. Stemm et al. [25] investigate methods that reduce energy consumption of network interfaces for electronic mail and web browsing applications on PDAs. Unlike STPM, both these techniques benefit only a limited set of specific applications.

7. Future work

Adaptive methods have previously applied to disk [5,8] and CPU [6] power management. Though these domains differ substantially from wireless network power management, we believe that the five design principles we have developed for STPM can also be applied. A logical next step in this work is therefore to apply STPM to the power management of other devices used in mobile computers.

In the specific case of disk power management, disk rotation is typically halted in order to save power. While spinning down the hard drive offers the opportunity for power savings during idle periods, there is also a substantial performance and energy transition cost that is incurred when the disk is spun up again to service the next request. Thus, mobile disk drives appear to offer a tradeoff between performance and energy conservation that is similar to the tradeoff offered by wireless network interfaces. This similarity encourages us to believe that STPM will also show significant benefits in this domain.

A similar tradeoff exists for processor power management when the CPU supports different clock speeds and voltages. Reducing clock frequency and voltage conserves energy but also adversely affects increasing interactive response time [6]. Thus, processor power management represents another arena in which STPM may prove useful.

Finally, concurrent power management of multiple devices remains an open research topic. Most current power management algorithms address only a single component. As results in this paper show, a myopic approach that optimizes the energy usage of only a single device can sometimes increase the energy usage of the system as a whole. STPM represents a first step toward concurrent power management of multiple devices because it expresses power management options in the common global currency of performance and energy usage rather than in device-specific terms.

8. Conclusion

Wireless network power management can severely degrade the performance of latency-sensitive applications and even increase the total energy needed to perform interactive activities on a mobile computer. In order to provide significant energy conservation without substantial performance degradation, a power management strategy should be tuned to reflect application intent and access patterns, as well as the power characteristics of the network interface and mobile computer. It is infeasible to expect a user to manually tune the power management algorithm for each combination of application, network interface, and mobile computer. Therefore, we have built a *self-tuning* power management module that adapts its behavior in response to a changing environment. Our results shows that self-tuning improves both performance and energy conservation compared to current power management strategies.

Our Linux implementation of self-tuning power management offers an important opportunity for future work. We plan to deploy our module to a small user community and gather detailed feedback about the benefits of self-tuning power management. In order to realize the full benefits of STPM, we will need to broaden the set of applications that disclose power management hints. Meanwhile, our hinting module is available to support applications that have not yet been modified.

Acknowledgments

We would like to thank Mark Corner, Brian Noble, and the anonymous reviewers for several helpful suggestions that improved the quality of the previous version of this paper that appeared in Mobicom 2003 [1]. This research was partially supported by the National Science Foundation under grant CCR-0306251 and by an equipment grant from Intel Corporation. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF, Intel, the University of Michigan, or the U.S. government.

References

[1] M. Anand, E.B. Nightingale and J. Flinn, Self-tuning wireless network power management, in: *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking (MOBICOM '03)*, San Diego, CA (Sept. 2003) pp. 176–189.

[2] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H.F. Nielsen, S. Thatte and D. Winer, Simple object access protocol (SOAP) 1.1. Technical report, W3C Note (May 2000).

[3] S. Chandra, Wireless network interface energy consumption implications of popular streaming formats, in: *Proceedings of Multimedia Computing and Networking*, San Jose, CA (2002) pp. 85–99.

[4] Cisco Systems, Inc., *Cisco Aironet wireless LAN Client Adapters Installation and Configuration Guide for Linux*.

[5] F. Douglis, P. Krishnan and B. Bershad, Adaptive disk spin-down policies for mobile computers, in: *Proceedings of the 2nd USENIX Symposium on Mobile and Location-Independent Computing*, Ann Arbor, MI (April 1995) pp. 121–137.

[6] K. Flautner, S.K. Reinhardt and T.N. Mudge, Automatic performance setting for dynamic voltage scaling, in: *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MOBICOM '01)*, Rome, Italy (July 2001) pp. 260–271.

[7] J. Flinn and M. Satyanarayanan, Energy-aware adaptation for mobile applications, in: *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP)*, Kiawah Island, SC (Dec. 1999) pp. 48–63.

[8] P. Greenawalt, Modeling power management for hard disks, in: *Proceedings of the Symposium on Modeling and Simulation of Computer Telecommunication Systems*, Durham, NC (Jan. 1994) pp. 62–66.

[9] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham and M.J. West. Scale and performance in a distributed file system, ACM Transactions on Computer Systems, 6(1) (Feb. 1998).

[10] IEEE Local and Metropolitan Area Network Standards Committee, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std 802.11-1997, New York, New York (1997).

[11] M. Kim, L. P. Cox and B.D. Noble, Safety, visibility, and performance in a wide-area file system, in: *Proceedings of the 1st USENIX Conference on File and Storage Technologies* (Jan. 2002).

[12] J.J. Kistler and M. Satyanarayanan, Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems* 10(1) (Feb. 1992).

[13] R. Krashinsky and H. Balakrishnan, Minimizing energy for wireless web access with bounded slowdown, in: *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking (MOBICOM '02)*, Atlanta, GA (July 2002).

[14] R. Kravets and P. Krishnan, Application-driven power management for mobile communication, ACM Wireless Networks 6(4) (2000) 263–277.

[15] J.R. Lorch and A.J. Smith, Improving dynamic voltage scaling algorithms with PACE, in: *Proceedings of ACM SIGMETRICS*, Cambridge, MA (June 2001).

[16] Y.-H. Lu, L. Benini and G.D. Micheli, Power-aware operating systems for interactive systems, IEEE Transactions on VLSI, 10(2) (2002) 119–134.

[17] T.C. Mowry, A.K. Demke and O. Krieger, Automatic compiler-inserted I/O prefetching for out-of-core applications, in: *Proceedings of the 2nd*

USENIX Symposium on Operating Systems Design and Implementation, Seattle, WA (Oct. 1996) pp. 3–17.

[18] L. Mummert, M. Ebling and M. Satyanarayanan, Exploiting weak connectivity in mobile file access, in: *Proceedings of the 15th ACM Symp. on Op. Syst. Principles*, Copper Mountain, CO (Dec. 1995).

[19] Network Working Group, NFS: Network file system protocol specification, RFC 1094 (March 1989).

[20] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn and K.R. Walker, Agile application-aware adaptation for mobility, in: *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP)*, Saint-Malo, France (Oct. 1997) pp. 276–287.

[21] A. Nye, editor, *X Protocol Reference Manual* (O'Reilly and Associates, Inc., 1990).

[22] R. Patterson, G. Gibson, E. Ginting, D. Stodolsky and J. Zelenka, in:formed prefetching and caching, in: *Proceedings of the 15th ACM Symp. on Op. Syst. Principles*, Copper Mountain, CO (Dec. 1995).

[23] E. Shih, P. Bahl and M.J. Sinclair, Wake on wireless: An event-driven energy saving strategy for battery operated devices, in: *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking (MOBICOM '02)*, Atlanta, GA (July 2002).

[24] T. Simunic, L. Benini, P. Glynn and G.D. Micheli, Dynamic power management for portable systems, in: *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MOBICOM '00)*, Boston, MA (Aug. 2000) pp. 11–19.

[25] M. Stemm and R.H. Katz, Measuring and reducing energy consumption of network interfaces in hand-held devices. *IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Science* 80(8) (1997) 1125–1131.

[26] Sun Microsystems, Inc., *Java Remote Method Invocation Specification* (Dec. 1999).

[27] A. Weissel, B. Beutel and F. Bellosa, Cooperative I/O: A novel I/O semanatics for energy-aware applications, in: *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, Boston, MA (Dec. 2002) pp. 117–129.

[28] H. Zeng, C.S. Ellis, A.R. Lebeck and A. Vahdat, ECOSystem: Managing energy as a first class operating system resource, in: *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, San Jose, CA (Oct. 2002).

Manish Anand obtained his B.E. in computer science and engineering from Birla Institute of Technology, India, in 1998. He obtained his Masters in Computer Science from University of Illinois, Urbana Champaign, in 2000. He is currently working on his Ph.D in the department of electrical engineering and computer science at University of Michigan, Ann Arbor. He is working on the pervasive computing research team at University of Michigan and his research interest include mobile systems, pervasive computing, operating systems, dynamic power management and distributed systems.

Edmund Nightingale is currently in his third year pursuing a Ph.D. in Computer Science at the University of Michigan. He received his B.A. from DePauw University in 2002 and his M.S. from the University of Michigan in 2004. His research interests include distributed file systems, mobile file system and operating system design, and dynamic power management.

Jason Flinn is an assistant professor in the Electrical Engineering and Computer Science department at the University of Michigan. He received his PhD from Carnegie Mellon University in 2001. His research interests include operating systems, mobile computing, and dynamic power management.