Energy-aware adaptation for mobile applications

Jason Flinn and M. Satyanarayanan
School of Computer Science
Carnegie Mellon University

Abstract

In this paper, we demonstrate that a collaborative relationship between the operating system and applications can be used to meet user-specified goals for battery duration. We first show how applications can dynamically modify their behavior to conserve energy. We then show how the Linux operating system can guide such adaptation to yield a battery-life of desired duration. By monitoring energy supply and demand, it is able to select the correct tradeoff between energy conservation and application quality. Our evaluation shows that this approach can meet goals that extend battery life by as much as 30%.

1 Introduction

Energy is a vital resource for mobile computing. There is growing consensus that advances in battery technology and low-power circuit design cannot, by themselves, meet the energy needs of future mobile computers — the higher levels of the system must also be involved [1, 7].

In this paper, we explore how applications can dynamically modify their behavior to conserve energy. To guide such adaptation, the operating system monitors energy supply and demand. When energy is plentiful, application behavior is biased toward a good user experience; when it is scarce, the behavior is biased toward energy conservation.

To validate the energy benefits of adaptation, we present results from a detailed study of applications running on the Odyssey platform for mobile computing. Our results show energy reductions in the range of 7% to 72%, with a mean of 36%. Combined with hardware power management, we achieve overall reductions between 31% and 76%, with a mean of 50% — in effect, doubling battery life.

This research was supported by the National Science Foundation (NSF) under grant number CCR-9901696, and the Air Force Materiel Command (AFMC) under DARPA contract number F19628-96-C-0061. Addition support was provided by IBM. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of NSF, AFMC, DARPA, IBM, CMU, or the U.S. Government.

SOSP-17 12/1999 Kiawah Island, SC

Our measurements also suggest a novel approach to reducing the energy drain of the display, an important but difficult challenge. Using this approach, we project a further energy reduction ranging from 7% to 29%.

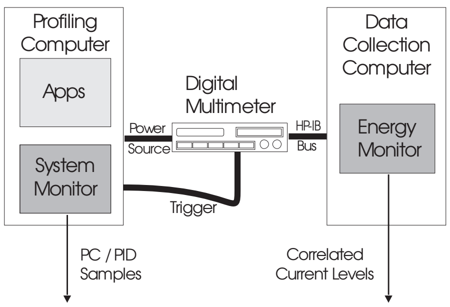
Finally, we show how the operating system can control adaptation by concurrent applications to give a battery life of user-specified duration. To perform this control, we have extended Odyssey to predict future energy demand from measurements of past usage. When there is substantial mismatch between predicted demand and available energy, Odyssey notifies applications to adapt. Using this approach, we demonstrate that Odyssey can extend battery-life to meet user-specified goals that vary by as much as 30%.

We begin with brief overviews of PowerScope, a tool we built to profile energy usage, and Odyssey. Three major sections follow: Section 3, on energy savings through adaptation; Section 4, on reducing display energy usage; and Section 5, on achieving a desired battery life. We close with a summary of related work and future plans.

2 Background

2.1 The PowerScope energy profiler

PowerScope is a tool for mapping energy consumption to specific software components. Its functionality and design are inspired by CPU profilers such as `prof` and `gprof` that help expose code components wasteful of processor cycles. Using PowerScope, one can determine what fraction of the total energy consumed during a certain time period is due to specific processes. Further, one can determine the energy



This hardware setup is used during PowerScope data collection. A data collection computer distinct from the profiling computer controls the multimeter and stores samples from it. Later, program counter and process id samples are correlated offline with current levels to yield energy profiles.

Figure 1. Data collection in PowerScope

Process	CPU Time(s)	Total Energy(J)	Average Power(W)
/usr/odyssey/bin/xanim	66.57	643.17	9.66
/usr/X11R6/bin/X	35.72	331.58	9.28
Kernel	50.89	328.71	6.46
Interrupts-WaveLAN	18.62	165.88	8.91
/usr/odyssey/bin/odyssey	12.19	123.40	10.12
Total	183.99	1592.75	8.66

Energy Usage Detail for process /usr/odyssey/bin/odyssey

Procedure	CPU Time(s)	Total Energy(J)	Average Power(W)
_Dispatcher	0.25	2.53	10.11
_IOMGR_CheckDescriptors	0.17	1.74	10.23
_sftp_DataArrived	0.16	1.68	10.48
_rpc2_RecvPacket	0.16	1.67	10.41
_ExaminePacket	0.16	1.66	10.35

This figure shows a sample energy profile. The first table summarizes the energy usage by process, while the table below shows a portion of the detailed profile for a single process. Only part of the full profile is shown.

Figure 2. Example of an energy profile

consumption of individual procedures within a process. By providing fine-grained feedback, PowerScope helps expose system components most responsible for energy consumption. Since PowerScope was recently described in detail [8], we only provide a brief overview here.

PowerScope uses statistical sampling to profile the energy usage of a computer system. To reduce overhead, profiles are generated in two stages. During the data collection stage, shown in Figure 1, the tool samples power consumption as well as the program counter (PC) and process identifier (PID) of the code executing on the profiling computer. A digital multimeter, currently a Hewlett Packard 3458a, samples the current drawn by the profiling computer through its external power input. Since the input voltage on computers is well-controlled (to within 0.25% in our measurements), current samples alone are adequate to infer energy consumption. The output of this stage consists of a sequence of current level samples and a correlated sequence of PC/PID samples. In a later off-line stage, PowerScope combines these sequences with symbol table information from binaries and shared libraries on the profiling computer. The result is an energy profile such as that shown in Figure 2.

2.2 The Odyssey platform for adaptation

The design rationale and architecture of Odyssey were presented in an earlier paper [17]. Adaptation in Odyssey involves the trading of data quality for resource consumption. For example, a client playing full-color video data from a server could switch to black and white video when bandwidth drops, rather than suffering lost frames. Similarly, a map application might fetch maps with less detail rather than suffering long transfer delays for full-quality maps.

Odyssey captures this notion of data degradation through an attribute called *fidelity*, that defines the degree to which data presented at a client matches the reference copy at a

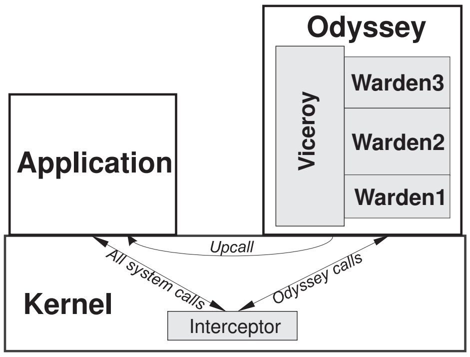


Figure 3. Odyssey architecture

server. Fidelity is a type-specific notion since different kinds of data can be degraded differently. Since the minimal level of fidelity acceptable to the user can be both time and application dependent, Odyssey allows each application to specify the fidelity levels it currently supports.

Odyssey supports concurrent adaptation by diverse applications. The client architecture providing this support is shown in Figure 3. Odyssey is conceptually part of the operating system, even though it is implemented in user space for simplicity. The *viceroy* is the Odyssey component responsible for monitoring the availability of resources and managing their use. Code components called *wardens* encapsulate type-specific functionality. There is one warden for each data type in the system. We have built four adaptive applications on top of Odyssey: a video player, a speech recognizer, a map viewer, and a Web browser. Such multimedia and speech-enabled applications are of growing importance in mobile computing, although they are not yet as common as spreadsheets and word processors. Relevant details of these applications are presented later.

Odyssey is integrated into Linux as a new VFS file system, along with a set of API extensions for expressing resource expections. If resource levels stray beyond an application's expectation, Odyssey notifies it through an upcall. The application then adjusts its fidelity to match the new resource level, and communicates a new set of expectations to Odyssey. Some applications, such as our Web browser and map viewer, use a proxy to avoid modifications to application source code. Other applications, such as our video player and speech recognizer, are modified to interact directly with Odyssey.

The initial Odyssey prototype only supported network bandwidth adaptation. The work reported here extends Odyssey to support energy adaptation.

3 Energy impact of fidelity

Does lowering data fidelity yield significant energy savings? This was the crucial question facing us when we began this work. Incorporating support for energy-aware adaptation into Odyssey is futile if the potential savings are meager.

We were also keen to confirm that the energy savings from lowering fidelity could enhance those achievable through well-known hardware power management techniques such as turning off the disk or slowing the CPU [6, 13, 16]. Although these distinct approaches to energy savings seem composable, we wanted to verify this experimentally.

3.1 Methodology

To answer these questions, we measured the energy used by the Odyssey video player, speech recognizer, map viewer, and Web browser. We first observed the applications as they operated in isolation, and then as they operated concurrently. To maintain fidelity constant throughout an experiment, we disabled Odyssey's dynamic adaptation capability.

We explored sensitivity of energy consumption to data content by using four data objects for each application: that is, four video clips, four speech utterances, four maps, and four Web images. We first measured the baseline energy usage for each object at highest fidelity with hardware power management disabled. Next, we measured energy usage with hardware-only power management. Then, for each lower fidelity level we measured energy usage with hardware power management enabled.

This sequence of measurements is directly reflected in the format of the graphs presenting our results: Figures 6, 8, 10 and 13. Since a considerable amount of data is condensed into these graphs, we explain their format here even though their individual contents will not be meaningful until the detailed discussions in Sections 3.3 through 3.6.

For example, consider Figure 6. There are six bars in each of the four data sets on the X axis; each data set corresponds to a different video clip. The height of each bar shows total energy usage, and the shadings within each bar show energy usage by software component. The component labelled "Idle" aggregates samples that occurred while executing the kernel idle procedure — effectively a Pentium `hlt` instruction. The component labelled "WaveLAN" aggregates samples that occurred during network interrupts.

For each data set, the first and second bars, labelled "Baseline" and "Hardware-Only Power Mgmt.", show energy usage at full fidelity with and without hardware power management. Each of the remaining bars show the energy usage at a different fidelity level with hardware power management enabled. The difference between one of these bars and the first bar ("Baseline") gives the combined benefit of hardware power management and fidelity reduction. The difference between one of these bars and the second one ("Hardware-Only Power Mgmt.") gives the savings directly attributable to reduction in fidelity.

The measurements for the bars labelled "Hardware-Only Power Mgmt." were obtained by powering down as many hardware components as possible for each application. For example, we placed the disk in standby mode after 10 seconds of inactivity. Further, we modified the network communication package used by Odyssey to place the wireless network interface in standby mode except during remote procedure calls or bulk transfers. Finally, we turned off the display during the speech application. To ensure good experimental control, we disabled BIOS-level hardware power management. While these hardware power management techniques are simple, they combine to yeild up to a 34% reduction in energy usage.

Component	State	Power (W)
Display	Bright	4.54
	Dim	1.95
WaveLAN	Idle	1.46
	Standby	0.18
Disk	Idle	0.88
	Standby	0.24
Other	Idle	3.20

Background (display dim, WaveLAN & disk standby) = 5.6 W.

This figure shows the measured power consumption of components of the IBM 560X laptop. Power usage is slightly but consistently superlinear; for example, the laptop uses 10.28 W when the screen is brightest and the disk and network are idle — 0.21 W more than the sum of the individual power usage of each component. The last row shows the power used when the disk, screen, and network are all powered off. Each value is the mean of five trials — in all cases, the sample standard deviation is less than 0.01 W.

Figure 4. Power consumption of IBM ThinkPad 560X

3.2 Experimental setup

All measurements reported in this paper were obtained on a 233 MHz Pentium IBM ThinkPad 560X laptop with 64 MB of memory, running the Linux 2.2 operating system. This machine was configured as an Odyssey client and communicated with servers over a 2 Mb/s wireless WaveLAN network operating at 900 MHz. The servers were 200 MHz Pentium Pro desktop computers with 64 MB of memory.

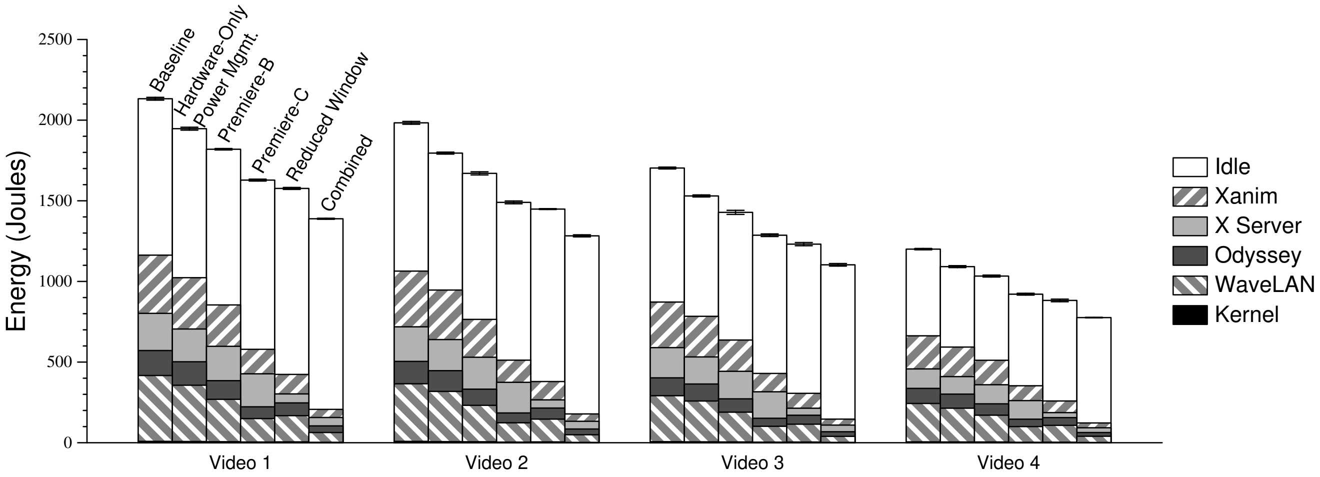
We profiled energy usage on the client with PowerScope, sampling approximately 600 times per second. To avoid confounding effects due to non-ideal battery behavior, the client used an external power supply. Further, to eliminate the effects of charging, the client's battery was removed.

Figure 4 shows the power usage of several hardware components of this laptop. To obtain these measurements, we used PowerScope to measure the change in power usage as we ran benchmarks which varied the power states of individual hardware components.

3.3 Video player

3.3.1 Description

We first measured the impact of fidelity on the video application shown in Figure 5. Xanim fetches videos from a server through Odyssey and displays them on the client. It supports two dimensions of fidelity: varying the amount of



This figure shows the energy used to display four QuickTime/Cinepak videos from 127 to 226 seconds in length, ordered from right to left above. For each video, the first bar shows energy usage without hardware power management or fidelity reduction. The second bar shows the impact of hardware power management alone. The next two show the impact of lossy compression. The fifth shows the impact of reducing the size of the display window. The final bar shows the combined effect of lossy compression and window size reduction. The shadings within each bar detail energy usage by software component. Each value is the mean of five trials — the error bars show 90% confidence intervals.

Figure 6. Energy impact of fidelity for video playing

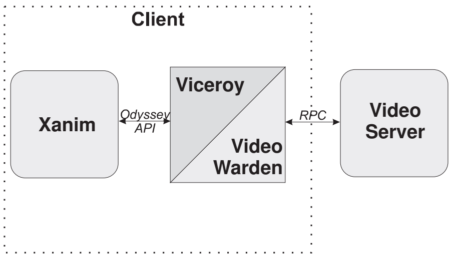


Figure 5. Odyssey video player

lossy compression used to encode a video clip, and varying the size of the window in which it is displayed. There are multiple tracks of each video clip on the server, each generated off-line from the full-fidelity video clip using Adobe Premiere. They are identical to the original except for size and the level of lossy compression used in frame encoding.

3.3.2 *Results*

Figure 6 shows the energy used to display four videos at different fidelities. At baseline fidelity, much energy is consumed while the processor is idle because of the limited bandwidth of the wireless network — not enough video data is transmitted to saturate the processor. Most of the remaining energy is consumed by asynchronous network interrupts, the Xanim video player, and the X server.

For the four video clips, hardware-only power management reduces energy consumption by a mere 9–10%. There is little opportunity to place the network in standby mode since it is nearly saturated. Most of the reduction is due to disk power management — the disk remains in standby mode for the entire duration of an experiment.

The bars labelled Premiere-B and Premiere-C in Figure 6 show the impact of lossy compression. Premiere-C, at the highest compression, consumes 16–17% less energy than hardware-only power management. Note that these gains are understated due to the bandwidth limitation imposed by our wireless network. With a higher-bandwidth network, we could raise baseline fidelity and thus transmit better video quality when energy is plentiful. The relative energy savings of Premiere-C would then be higher.

By examining the shadings of each bar in Figure 6, we see that compression significantly reduces the energy used by Xanim, Odyssey and the WaveLAN device driver. However, the energy used by the X server is almost completely unaffected by compression. We conjecture that this is because video frames are decoded before they are given to the X server, and the size of this decoded data is independent of the level of lossy compression.

To validate this conjecture, we measured the effect of halving both the height and width of the display window, effectively introducing a new dimension of fidelity. As Figure 6 shows, shrinking the window size reduces energy consumption 19–20% beyond hardware-only power management. The shadings on the bars confirm that reducing window size significantly decreases X server energy usage. In fact, within the bounds of experimental error, X server energy consumption is proportional to window area.

Finally, we examined the effect of combining Premiere-C encoding with a display window of half the baseline height and width. This results in a 28–30% reduction in energy usage relative to hardware-only power management. Relative to baseline, using all the techniques (hardware, Premiere-C, and reduced window) together yields about a 35% reduction.

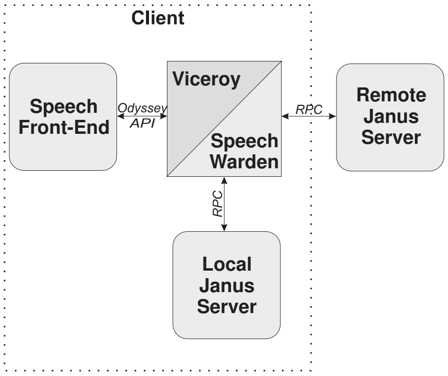


Figure 7. Odyssey speech recognizer

From the viewpoint of further energy reduction, the rightmost bar of each data set in Figure 6 offers a pessimistic message: there is little to be gained by further efforts to reduce fidelity. Virtually all the energy used at this fidelity is in the idle state, with the display consuming a large share. Turning off the display is, of course, not an option when watching video. We will return to this point in Section 4.

3.4 Speech recognizer

3.4.1 Description

Our second application is an adaptive speech recognizer. As shown in Figure 7, it consists of a front-end that generates a speech waveform from an utterance and submits it via Odyssey to a local or remote instance of the Janus speech recognition system [23].

Local recognition avoids network transmission and is unavoidable if the client is disconnected. In contrast, remote recognition incurs the delay and energy cost of network communication but can exploit the CPU, memory and energy resources of a remote server that is likely to be operating from a power outlet rather than a battery. The system also supports a hybrid mode of operation in which the first phase of recognition is performed locally, resulting in a compact intermediate representation that is shipped to the remote server for completion of the recognition. In effect, the hybrid mode uses the first phase of recognition as a type-specific compression technique that yields a factor of five reduction in data volume with little computational overhead.

Fidelity is lowered in this application by using a reduced vocabulary and a less complex acoustic model. This substantially reduces the memory footprint and processing required for recognition, but degrades recognition quality. The system alerts the user of fidelity transitions using a synthesized voice. The use of low fidelity is most compelling in the case of local recognition on a resource-poor disconnected client, although it can also be used in hybrid and remote cases.

Although reducing fidelity limits the number of words available, the word-error rate may not increase. Intuitively, this is because the recognizer makes fewer mistakes when choosing from a smaller set of words in the reduced vocabulary. This helps counterbalance the effects of reducing the sophistication of the acoustic model.

3.4.2 Results

Figure 8 presents our measurements of client energy usage when recognizing four pre-recorded utterances using local, remote and hybrid strategies at high and low fidelity. The baseline measurements correspond to local recognition at high fidelity without hardware power management. Since speech recognition is compute-intensive, almost all the energy in this case is consumed by Janus.

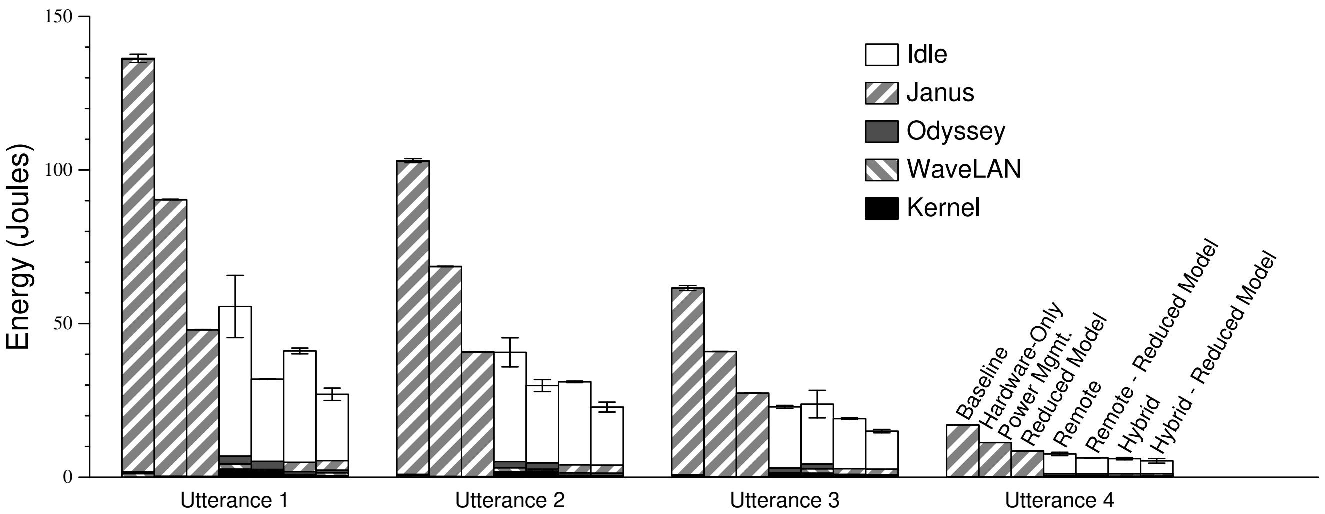
Hardware-only power management reduces client energy usage by 33–34%. Such a substantial reduction is possible because the display can be turned off and both the network and disk can be placed in standby mode for the entire duration of an experiment. This assumes that user interactions occur solely through speech, and that disk accesses can be avoided because the vocabulary, language model and acoustic model fit entirely in physical memory. More complex recognition tasks may trigger disk activity and hence show less benefit from hardware power management.

Lowering fidelity by using a reduced speech model results in a 25–46% reduction in energy consumption relative to hardware-only power management. This corresponds to a 50–65% reduction relative to the baseline.

Remote recognition at full fidelity reduces energy usage by 33–44% below that obtained by using hardware-only power management. If fidelity is also reduced, the corresponding savings is 42–65%. These figures are comparable to the energy savings for remote execution reported in the literature for other compute-intensive tasks [18]. As the shadings in the fourth and fifth bars of each data set in Figure 8 indicate, most of the energy consumed by the client in remote recognition occurs with the processor idle — much of this is time spent waiting for a reply from the server. Lowering fidelity speeds recognition at the server, thus shortening this interval and yielding additional energy savings.

Hybrid recognition offers slightly greater energy savings than remote recognition: 47–55% at full fidelity, and 53–70% at low fidelity, both relative to hardware-only power management. Hybrid recognition increases the fraction of energy used by the local Janus code; but this is more than offset by the reduction in energy for network transmission and idle time.

Overall, the net effect of combining hardware power management with hybrid, low-fidelity recognition is a 69–80% reduction in energy usage relative to the baseline. In practice, the optimal strategy will depend on resource availability and the user's tolerance for low-fidelity recognition.



This figure shows the energy used to recognize four spoken utterances from one to seven seconds in length, ordered from right to left above. For each utterance, the first bar shows energy consumption without hardware power management or fidelity reduction. The second bar shows the impact of hardware power management alone. The remaining bars show the additional savings realized by adaptive strategies. The shadings within each bar detail energy usage by activity. Each measurement is the mean of five trials — the error bars show 90% confidence intervals.

Figure 8. Energy impact of fidelity for speech recognition

3.5 Map viewer

3.5.1 Description

The third application that we measured was an adaptive map viewer named Anvil. As shown in Figure 9, Anvil fetches maps from a remote server via Odyssey and displays them on the client. Fidelity can be lowered in two ways: filtering and cropping. Filtering eliminates fine detail and less important features (such as secondary roads) from a map. Cropping preserves detail, but restricts data to a geographic subset of the original map. The client annotates the map request with the desired amount of filtering and cropping. The server performs any requested operations before transmitting the map data to the client.

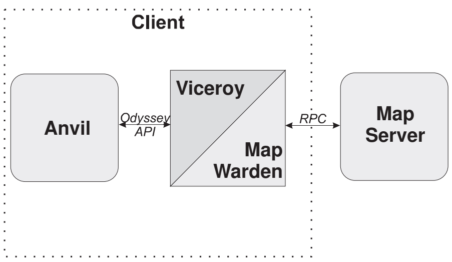


Figure 9. Odyssey map viewer

3.5.2 Results

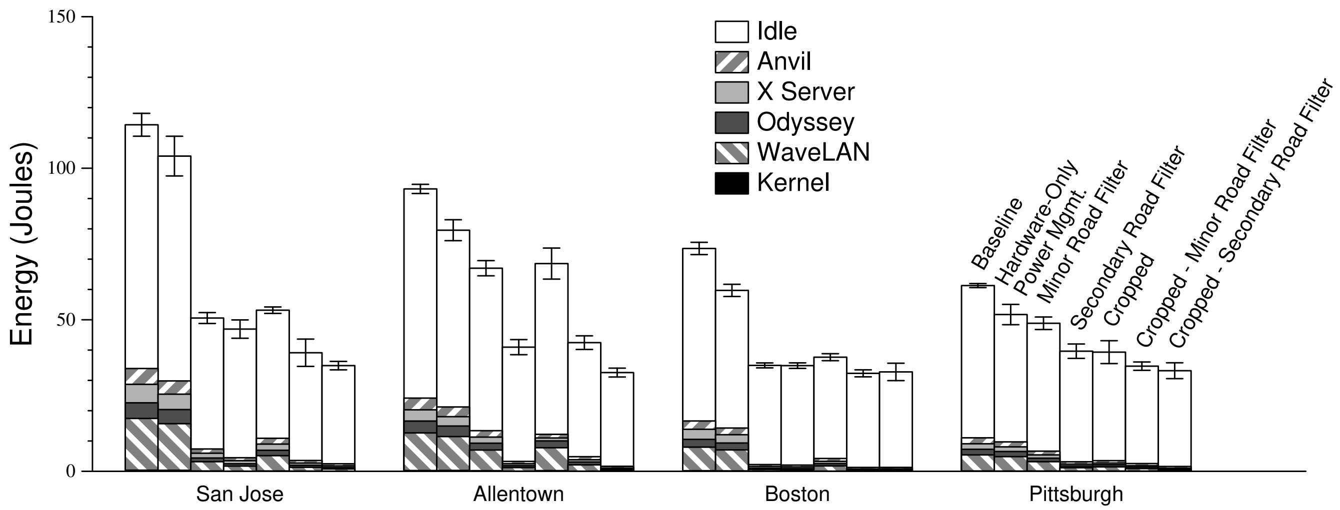
We measured the energy used by the client to fetch and display maps of four different cities. Viewing a map differs from the two previous applications in that a user typically needs a non-trivial amount of time to absorb the contents of a map after it has been displayed. This period, which we refer to as *think time*, should logically be viewed as part of the application's execution since energy is consumed in keeping the map visible. In contrast, the user needs negligible time after the display of the last frame or the recognition of an utterance to complete use of the video or speech application.

Think time is likely to depend on both the user and the map being displayed. Our approach to handling this variability was to use an initial value of 5 seconds and then performing sensitivity analysis for think times of 0, 10 and 20 seconds. For brevity, Figure 10 only presents detailed results for the 5 second case; for other think times, we present only the summary information in Figure 11.

The baseline bars in Figure 10 show that most of the energy is consumed while the CPU is idle; a significant portion of this goes to keeping the display backlit during the five second think time. The shadings on the bars indicate that network communication is a second significant drain on energy. The comparatively larger confidence intervals for this application result from variation in the time required to fetch a map over the wireless network.

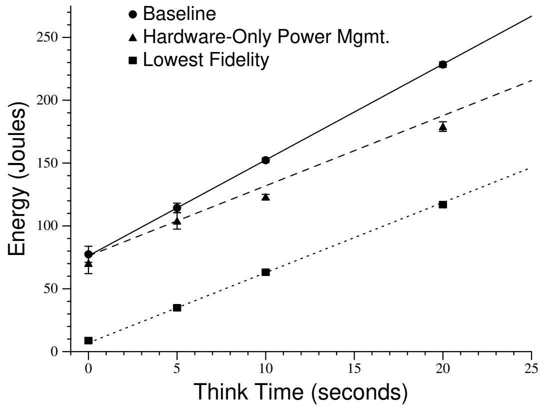
Hardware-only power management reduces energy consumption by about 9–19% relative to the baseline. Although there is little opportunity for network power management while the map is being fetched, the network can remain in standby mode during think time. Since the disk is never used, it can always remain in standby mode.

The third and fourth bars of each data set show the effect of fidelity reduction through two levels of filtering. One filter omits minor roads, while the more aggressive filter omits both minor and secondary roads. The savings from the minor road filter range from 6–51% relative to hardware-only power management. The corresponding figure for the secondary road filter is 23–55%.



This figure shows the energy used to view four U.S.G.S. maps. For each map, the first bar shows energy usage without hardware power management or fidelity reduction, with a 5 second think time. The second bar shows the impact of hardware power management alone. The remaining bars show the additional savings realized by degrading map fidelity. The shadings within each bar detail energy usage by activity. Each measurement is the mean of ten trials—the error bars are 90% confidence intervals.

Figure 10. Energy impact of fidelity for map viewing



This figure shows how the energy used to view the San Jose map from Figure 10 varies with think time. The data points show measured energy usage. The solid, dashed and dotted lines represent linear models for energy usage for the baseline, hardware-only power management and lowest fidelity cases. The latter combines filtering and cropping, as in the rightmost bars of Figure 10. Each measurement is the mean of ten trials — the error bars are 90% confidence intervals.

Figure 11. Effect of user think time for map viewing

The fifth bar of each data set shows the effect of lowering fidelity by cropping a map to half its original height and width. Energy usage at this fidelity is 14–49% less than with hardware-only power management. In other words, cropping is less effective than filtering for these samples. Combining cropping with filtering results in an energy savings of 36–66% relative to hardware-only power management, as shown by the rightmost bars of each data set. Relative to the baseline, this is a reduction of 46–70%. There is little savings left to be extracted through software optimization — almost all the energy is consumed in the idle state.

After examining energy usage at 5 seconds of think time, we repeated the above experiments at think times of 0, 10 and 20 seconds. At any given fidelity, energy usage, E_t increases with think time, t. We expected a linear relationship: $E_t = E_0 + t \cdot P_B$, where E_0 is the energy usage for this fidelity at zero think time and P_B is the background power consumption on the client (5.6 W from Figure 4).

Figure 11 confirms that a linear model is indeed a good fit. This figure plots the energy usage for four different values of think time for three cases: baseline, hardware-only power management, and lowest fidelity. The divergent lines for the first two cases show that the energy reduction from hardware-only power management scales linearly with think time. The parallel lines for the second and third cases show that fidelity reduction achieves a constant benefit, independent of think time. The complementary nature of these two approaches is thus well illustrated by these measurements.

3.6 Web browser

3.6.1 Description

Our fourth application is an adaptive Web browser based on Netscape Navigator, as shown in Figure 12. In this applica-

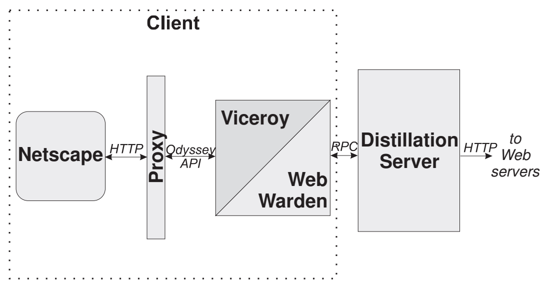
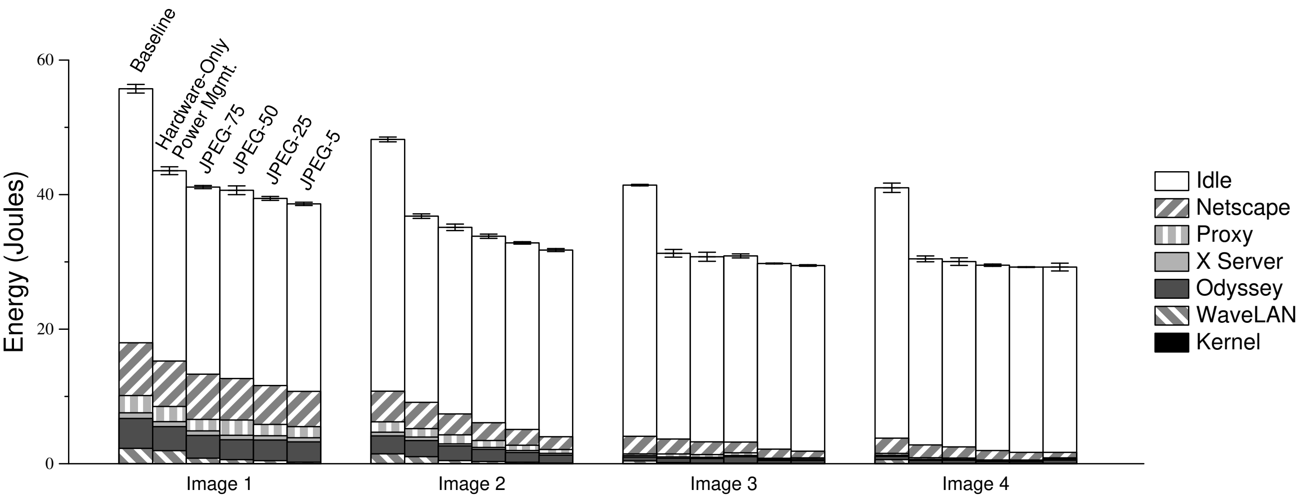


Figure 12. Odyssey Web browser



This figure shows the energy used to display four GIF images from 110 B to 175 KB in size, ordered from right to left above. For each image, the first data bar shows energy consumption at highest fidelity without hardware power management, assuming a think time of five seconds. The second bar shows the impact of hardware power management alone. The remaining bars show energy usage as fidelity is lowered through increasingly aggressive lossy JPEG compression. The shadings within each bar detail energy usage by activity. Each measurement is the mean of ten trials — the error bars show 90% confidence intervals.

Figure 13. Energy impact of fidelity for Web browsing

tion, Odyssey and a distillation server located on either side of a variable-quality network mediate access to Web servers. Requests from an unmodified Netscape browser are routed to a proxy on the client that interacts with Odyssey. After annotating the request with the desired level of fidelity, Odyssey forwards it to the distillation server which transcodes images to lower fidelity using lossy JPEG compression. This is similar to the strategy described by Fox et al [9], except that control of fidelity is at the client.

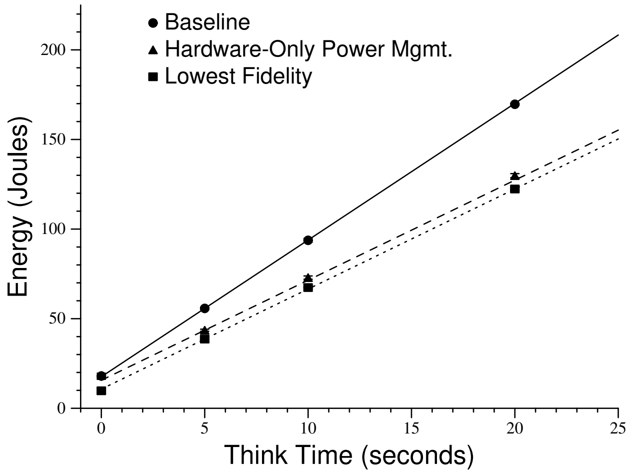
3.6.2 Results

As with the map application, a user needs some time after an image is displayed to absorb its contents. We therefore include energy consumed during user think time as part of the application. We use a baseline value of five seconds and perform sensitivity analysis for 0, 10 and 20 seconds.

Figure 13 presents our measurements of the energy used to fetch and display four GIF images of varying sizes. Hardware-only power management achieves reductions of 22–26%. The shadings on the first and second bars of each data set indicate that most of this savings occurs in the idle state, probably during think time.

The energy benefits of fidelity reduction are disappointing. As Figure 13 shows, the energy used at the lowest fidelity is merely 4–14% lower than with hardware-only power management; relative to baseline, this is a reduction of 29–34%. Of course, these results apply only to the specific form of fidelity reduction used in our experiments, namely transcoding using lossy JPEG compression.

The effect of varying think time is shown in Figure 14. The linear model introduced in Section 3.5.2 fits observations well for all three cases: baseline, hardware-only power



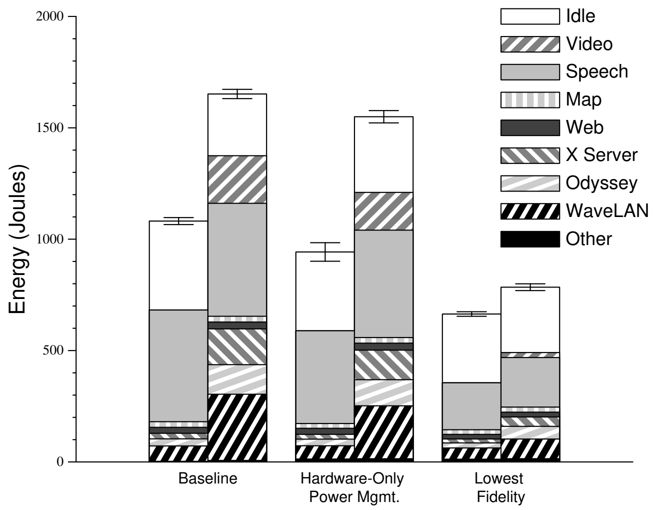
This figure shows how the energy used to display Image 1 from Figure 13 varies with user think time. The data points on the graph show measured energy usage for user think times of 0, 5, 10, and 20 seconds. The solid, dashed and dotted lines represent linear models for energy consumption for the baseline, hardware-only power management and lowest fidelity cases. Each measurement represents the mean of ten trials — the error bars are 90% confidence intervals.

Figure 14. Effect of user think time for Web browsing

management and lowest fidelity. The close spacing of the lines for the two latter cases reflects the small energy savings available through fidelity reduction. The divergence of the lines for the first two cases shows the importance of hardware power management during think time.

3.7 Effect of concurrency

How does concurrent execution affect energy usage? One can imagine situations in which total energy usage goes



Each data set in this figure compares energy usage for the composite application described in Section 3.7 in isolation (left bar), with total energy usage when a video application runs concurrently (right bar). Each measurement is the mean of five trials — the error bars are 90% confidence intervals.

Figure 15. Effect of concurrent applications

down when two applications execute concurrently rather than sequentially. For example, once the screen has to be turned on for one application, no additional energy is required to keep it on for the second. One can also envision situations in which concurrent applications interfere with each other in ways that increase energy usage. For example, if physical memory size is inadequate to accommodate the working sets of two applications, their concurrent execution will trigger higher paging activity, possibly leading to increased energy usage. Clearly, the impact of concurrency can vary depending on the applications, their interleaving, and the machine on which they run.

What is the effect of lowering fidelity? The measurements reported in Sections 3.3 to 3.6 indicate that lowering fidelity tends to increase the fraction of energy consumption attributable to the idle state. Concurrency allows background energy consumption to be amortized across applications. It is therefore possible in some cases for concurrency to enhance the benefit of lowering fidelity.

To confirm our intuition, we compared the energy usage of a composite application when executing in isolation and when executing concurrently with the video application described in Section 3.3. The composite application consists of six iterations of a loop that involves the speech, Web and map applications described in Sections 3.4 to 3.6. The loop consists of local recognition of two speech utterances, access of a Web page, access of a map, and five seconds of think time. The composite application models a user searching for Web and map information using speech commands, while the video application models a background newsfeed. This experiment takes between 80 and 160 seconds.

Figure 15 presents the results of our experiments for three cases: baseline, hardware-only power management, and minimal fidelity. In the first two cases, all applications ran at full fidelity; in the third case, all ran at lowest fidelity. For each data set, the left bar shows energy usage for the composite application in isolation, while the right bar shows energy usage during concurrent execution.

For the baseline case, the addition of the video application consumes 53% more energy. But for hardware-only power management case, it consumes 64% more energy. This difference is due to the fact that concurrency reduces opportunities for powering down the network and disk. For the minimum fidelity case, the second application only adds 18% more energy. This is because the significant background power usage of the client, which limits the effectiveness of lowering fidelity, is amortized by the second application. In other words, for this workload, concurrency does indeed enhance the energy impact of lowering fidelity.

3.8 Summary

We began Section 3 by asking whether lowering data fidelity yields significant energy savings. The results of Sections 3.3 to 3.6 confirm that such savings are indeed available over a broad range of applications relevant to mobile computing. Further, those results show that lowering fidelity can be effectively combined with hardware power management. Section 3.7 extends these results by showing that concurrency can magnify the benefits of lowering fidelity.

At the next level of detail, Figure 16 summarizes the results of Sections 3.3 to 3.6. For clarity, the data in each row is normalized to the baseline values.

The key messages of Figure 16 are:

- *There is significant variation in the effectiveness of fidelity reduction across data objects.*

  The reduction can span a range as broad as 29% (0.38–0.67 for Map, at think time 5). Video is the only application that shows little variation across data objects.

- *There is considerable variation in the effectiveness of fidelity reduction across applications.*

  Holding think time constant at 5 seconds and averaging across data objects, the energy usage for the four applications at lowest fidelity is 0.84, 0.28, 0.51 and 0.93 relative to their baseline values. The mean is 0.64, corresponding to an average savings of 36%.

- *Combining hardware power management with lowered fidelity can sometimes reduce energy usage below the product of their individual reductions.*

  This is seen most easily in the case of the video application, where the last column is 0.65 rather than the expected value of 0.76, obtained by multiplying 0.9 and 0.84. Intuitively, this is because reducing fidelity decreases hardware utilization, thereby increasing the opportunity for hardware power management.

Application	Think Time (s.)	Baseline	Hardware Power Mgmt.	Fidelity Reduction	Combined
Video	N/A	1.00	0.90–0.91	0.84–0.84	0.65–0.65
Speech	N/A	1.00	0.66–0.67	0.22–0.36	0.20–0.31
Map	0	1.00	0.80–1.01	0.06–0.13	0.07–0.18
	5	1.00	0.81–0.91	0.38–0.67	0.31–0.54
	10	1.00	0.74–0.84	0.53–0.77	0.42–0.58
	20	1.00	0.76–0.78	0.69–0.89	0.51–0.67
Web	0	1.00	0.85–1.06	0.40–0.75	0.32–0.54
	5	1.00	0.74–0.78	0.88–0.97	0.66–0.71
	10	1.00	0.75–0.78	0.93–0.98	0.70–0.74
	20	1.00	0.74–0.77	0.96–0.99	0.72–0.73

This table summarizes the impact of data fidelity on application energy consumption. Each entry shows the minimum and maximum measured energy consumption for four data objects. The entries are normalized to baseline measurements of full fidelity objects with no power management. This data was extracted from Figures 6, 8, 10 and 13.

Figure 16. Summary of energy impact of fidelity

4 Zoned backlighting

The display is the Achilles heel of power management. As shown in Figure 4, it is responsible for nearly 35% of the background energy usage of the IBM 560X laptop used in our experiments. This is consistent with similar studies of other mobile computers [13]. As shown by Figures 6, 8, 10 and 13, the idle state dominates energy usage after all our improvements have been applied. Reducing display energy drain is thus crucial to further progress.

4.1 Description

An intriguing solution is suggested by the results of Sections 3.3 and 3.5. In those cases, energy usage was reduced by shrinking the physical size of video and map images. Yet, the entire display had to remain backlit. Suppose it were possible for the X server to selectively control illumination in different parts of the screen. Then, lowering fidelity by image size reduction could yield even greater energy savings since the background power usage of the laptop would also be decreased.

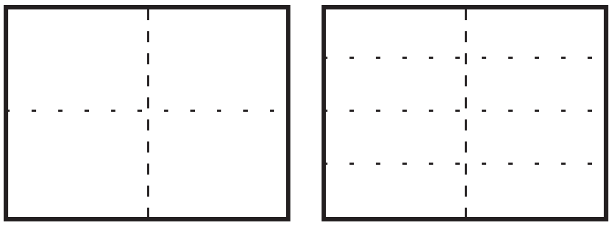
We define *zoned backlighting* as a display feature that allows independent control of illumination level for different regions of the screen under software control. If we view such a screen as a grid, the energy usage of each grid element can be independently controlled. We refer to each grid element as a *zone*. Typically, zone control would be exercised by the X server in Unix systems, and by the window management subsystem in Windows operating systems. This is analogous to device drivers for the disk and network controlling the energy states of those devices.

When energy is plentiful, the large high-fidelity image generated by an application would span many zones. As the battery drains, applications would generate smaller low-fidelity images that span fewer zones. One can envision a "snap-to" feature provided by window managers that would move windows slightly so as to straddle the fewest possible zones. Further, window managers could support user control over illumination of peripheral zones — in a typical configuration, only the window in focus might be brightly illuminated, while the rest of the screen is dim or dark. In effect, zoned backlighting introduces a new degree of freedom into the energy budgets of laptops. It allows the use of large screens for improved usability, without compromising the ability to minimize energy usage at low battery levels.

4.2 Results

To the best of our knowledge, no existing display supports zoned backlighting. Indeed, it is possible that design or manufacturing limitations may preclude mass-production of such displays. However, we were curious to see what the energy performance of such a display would be for the applications in Section 3.

For this estimate, we extrapolated from the design characteristics of the IBM ThinkPad 560X used in our experiments. We considered 4-zone and 8-zone versions of this display, as shown in Figures 17(a) and 17(b). We assumed that each zone was illuminated by one lamp, and that the power used by each zone was proportional to its area — 0.25 and 0.125 of the value from Figure 4 for the 4-zone and 8-zone cases.



(a) 4-Zone Display (b) 8-Zone Display

Figure 17. Zoned backlighting

			Hardware-Only Power Management			Combined		
App.	Think Time (s.)	Baseline	No Zones	4 Zones	8 Zones	No Zones	4 Zones	8 Zones
Video	N/A	1.00	0.90–0.91	0.74–0.76	0.74–0.76	0.65–0.65	0.49–0.50	0.46–0.47
Map	0	1.00	0.80–1.01	0.80–1.01	0.75–0.95	0.07–0.18	0.07–0.16	0.06–0.15
	5	1.00	0.81–0.91	0.81–0.91	0.75–0.85	0.31–0.54	0.26–0.45	0.24–0.43
	10	1.00	0.74–0.84	0.74–0.84	0.68–0.78	0.42–0.58	0.39–0.48	0.33–0.46
	20	1.00	0.76–0.78	0.76–0.78	0.70–0.72	0.51–0.67	0.42–0.55	0.40–0.52

This table summarizes the projected impact of zoned backlighting on application energy consumption. Each entry shows the minimum and maximum energy consumption for four data objects. The entries are normalized to baseline measurements of full fidelity objects with no power management. The column labelled "Combined" corresponds to measurements at lowest fidelity, with hardware power management enabled.

Figure 18. Energy impact of zoned backlighting

Figure 18 shows projected energy usage of the video and map applications using a zoned display. These results are derived from those presented in Sections 3.3.2 and 3.5.2 respectively. As in Figure 16, all results are normalized to baseline energy usage. We omit the other two applications since we expect them to benefit little from zoned displays: the entire display can be turned off for speech, and Netscape was almost full-screen at all fidelities in our experiments.

There are two key results in Figure 18. First, zoned backlighting offers noticeable benefits for both the video and map applications, even at full fidelity. Second, lowering fidelity enhances the energy savings due to zoned backlighting.

The video at full fidelity fits within one zone for the 4-zone case, and within two zones for the 8-zone case. In both cases, zoned backlighting reduces energy usage by 17–18%. At lowest fidelity, the video fits entirely within one of the 8 zones. This reduces energy consumption by 24% for the 4-zone case and by 28–29% for the 8-zone case.

The map at full fidelity occupies all zones in the 4-zone case and hence shows no benefits with zoned backlighting. But it occupies only six zones in the 8-zone case, and yields an energy reduction of 7–8% at a think time of 5 seconds. At lowest fidelity, the map output only occupies two zones in the 4-zone case, yielding an energy reduction of 17% at a think time of 5 seconds. In the 8-zone case, the map output only occupies three zones and this yields an energy reduction of 21%. The energy reduction increases with think time, approaching 8% for the 4-zone case with hardware power management, 17% for the 4-zone case at lowest fidelity, and 22% for the 8-zone case at lowest fidelity.

Based on these results, we plan to explore collaborations with display manufacturers to provide support for zoned backlighting. While savings in the range of 7–29% may not seem dramatic, it does represent an amount large enough to be worth expending effort to realize. It is important to remember that these savings are available *after* applying all the other energy-savings techniques described in this paper. We therefore anticipate an important future role for zoned backlighting in mobile computing.

5 Goal-directed energy adaptation

A mobile user often has a reasonable estimate of how long a battery needs to last — for example, the expected duration of a flight, commute, or meeting. Given such an estimate, one can ask whether energy-aware adaptation can be exploited to yield the desired battery life. To explore this question, we have extended Odyssey to monitor energy supply and demand, and to use this information to direct application adaptation to meet a user-specified time duration.

5.1 Prototype design and implementation

Our primary design goal was to ensure that Odyssey meets the specified time duration whenever feasible. An infeasible duration is one so large that the available energy is inadequate even if all applications run at lowest fidelity; the user should be alerted to this as early as possible. If the initial estimate is incorrect, the user can respecify the time goal. When this happens, the system should either adapt to meet the new goal or notify the user that it is infeasible.

An important secondary goal is to provide the best user experience possible. This translates into two requirements: first, applications should offer as high a fidelity as possible at all times; second, the user should not be jarred by frequent adaptations. In our prototype, we balance these opposing concerns by striving to provide high average fidelity while using hysteresis to reduce the frequency of fidelity changes.

Odyssey must perform three tasks periodically. First, it must determine the residual energy available. Second, it must predict future energy demand. Third, based on these two pieces of information, it must decide if applications should change fidelity, and notify them accordingly. Currently, Odyssey performs these actions twice a second, using samples collected every 100 milliseconds.

5.1.1 Determining residual energy

In our current implementation, Odyssey measures power with a on-line version of PowerScope. The data collection computer transmits current samples to Odyssey on the mobile computer. At each sample, Odyssey calculates residual energy, assuming a known initial value and constant power consumption between samples.

Although adequate for prototype validation, this solution is not portable, since PowerScope currently uses external hardware. We see three possible alternatives for the future. First, the BIOS on a mobile computer could provide power usage and residual battery level information — the SmartBattery API [19] that is currently being standardized as part of the Advanced Configuration and Power Interface Specification [10] is a promising candidate for this purpose. Second, if a compact digital multimeter in PCMCIA form factor were available, PowerScope could be modified to use it. Third, laptops such as the Apple Macintosh Duo already incorporate support for monitoring energy usage [15], and Odyssey could use this built-in functionality.

5.1.2 Predicting future energy demand

To predict future energy demand, Odyssey relies on smoothed observations of present and past power usage. This is in contrast to requiring applications to explicitly declare their future energy usage — an approach that we felt placed unnecessary burden on applications and was unlikely to be accurate. We use an exponential smoothing function of the form $new = (1-\alpha)(this\ sample) + (\alpha)(old)$, where α is a parameter that determines the relative weights of current and past power usage. Once future power usage has been estimated, it is multiplied by the time remaining until the goal to obtain future energy demand.

Odyssey varies α as energy drains, thus changing the tradeoff between agility and stability. When the goal is distant, Odyssey uses a large α. This biases adaptation toward stability by reducing the number of fidelity changes — there is ample time to make adjustments later, if necessary. As the goal nears, Odyssey decreases α so that adaptation is biased toward agility. Applications now respond more rapidly, since the margin for error is small.

Currently, we set α so that the half-life of the decay function is 10% of the time remaining until the goal. For example, if 30 minutes remain, the present estimate will be weighted equally with more recent samples after 3 minutes have passed. The choice of 10% is based on a sensitivity analysis, discussed in Section 5.3.

5.1.3 Triggering adaptation

When predicted demand exceeds residual energy, Odyssey issues upcalls so that applications can adapt to reduce energy usage. Conversely, when residual energy significantly exceeds predicted demand, applications are notified to increase data fidelity.

The amount by which supply must exceed demand to trigger fidelity improvement is indicative of the level of hysteresis in Odyssey's adaptation strategy. This value is the sum of two components: a variable component, currently 5% of residual energy, and a constant component, currently 1% of the initial energy available. The variable component reflects our bias toward stability when energy is plentiful and toward agility when it is scarce; the constant component biases against fidelity improvements when residual energy is low. As a guard against excessive adaptation due to energy transients, Odyssey caps fidelity improvements at a maximum rate of once every 15 seconds.

When multiple applications are executing concurrently, Odyssey must decide which to notify. A simple scheme based on user-specified priorities is used for this. Odyssey always tries to degrade a lower-priority application before degrading a higher-priority one — upgrades occur in the reverse order. Priorities are currently static, but we are implementing an interface to allow users to change priority dynamically, thus offering better control over adaptation.

5.1.4 Overhead

The power overhead imposed by Odyssey is the sum of the overhead of measuring power usage and the overhead of using these measurements to predict energy demand. Since we presently use external hardware to measure power usage, it is difficult to precisely quantify the measurement overhead. However, we believe the measurement overhead in a deployed version of our system will be quite low. Several SmartBattery solutions can provide power measurements at the frequency we require using less than 10 mW [3, 22]. The measured prediction overhead of our prototype is only 4 mW. Therefore, we expect that the total power overhead imposed by our solution will be less than 14 mW — only 0.25% of the background power consumption of our laptop.

5.2 Validation

5.2.1 Experiment design

To validate our prototype, we used the two applications described in Section 3.7: a composite application involving speech recognition, map viewing and Web access, run concurrently with a background video application. To obtain a continuous workload, we ran the composite application every 25 seconds rather than for six iterations. To avoid confounding effects due to non-ideal battery behavior, we used an external power supply and removed the battery.

At the beginning of each experiment we provided Odyssey with an initial energy value and a time goal. We then ran the applications, allowing them to adapt under Odyssey's direction until the time goal was reached or the residual energy dropped to zero. The former outcome represents a successful completion of the experiment, while the latter represents a failure. We noted the total number of adaptations of each application during the experiment. We also noted the residual energy at the end of the experiment — a large value suggests that Odyssey may have been too conservative in its adaptation decisions, and that average fidelity could have been higher.

All our experiments used an initial energy value of 12,000 J. This lasts 19:27 minutes when applications operate at highest fidelity, and 27:06 minutes at lowest fidelity. This represents a 39.3% extension in battery life. Our choice of initial energy value was deliberately small, so that experiments could be conducted in a reasonable amount of time. The value of 12,000 J is only about 14% of the nominal energy in the IBM 560X battery. Extrapolating to full nominal energy, our workload would have run for 2:18 hours at highest fidelity, and 3:13 hours at lowest fidelity.

5.2.2 *Results*

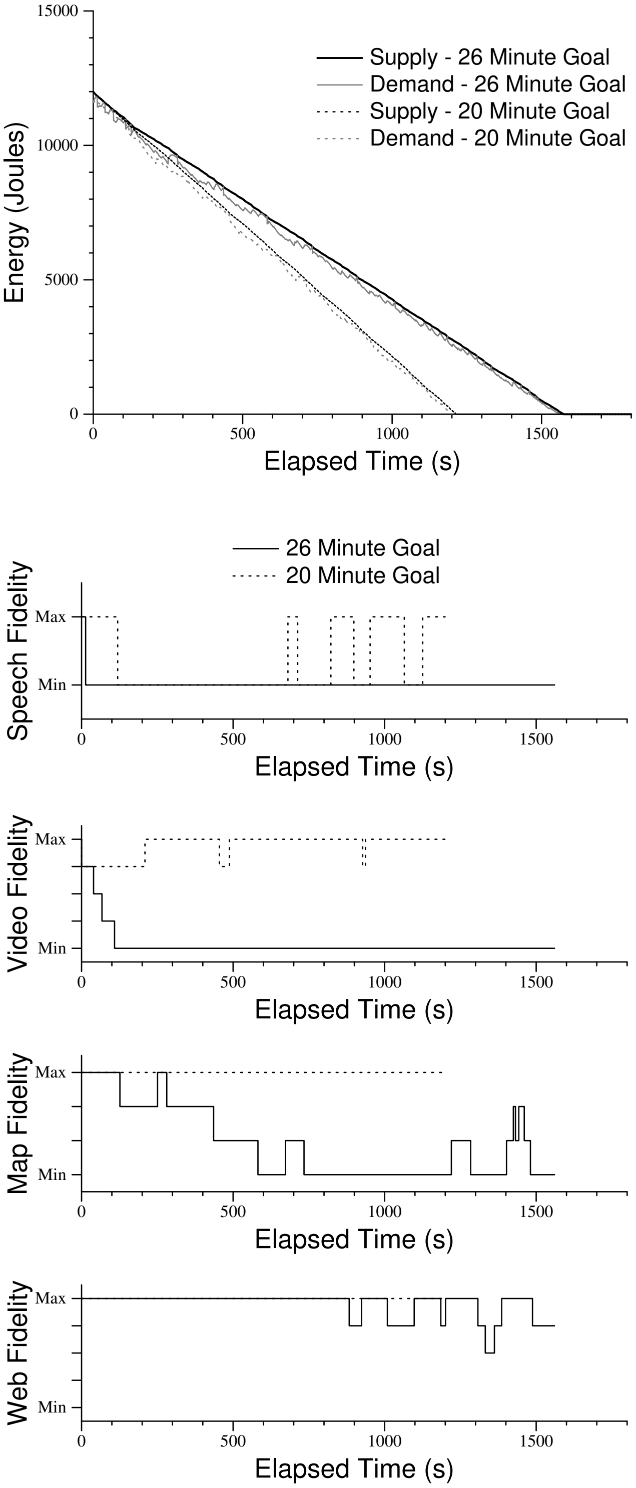
Figure 19 shows detailed results from two typical experiments: one with a 20 minute goal, and the other with a 26 minute goal. The top graph shows how the supply of energy and Odyssey's estimate of future demand change over time. The graph confirms that estimated demand tracks supply closely for both experiments.

The four bottom graphs of Figure 19 show how the fidelity of each application varies during the two experiments. For the 20 minute goal, the high priority Web and map applications remain at full fidelity throughout the experiment; the video degrades slightly; and speech runs mostly at low fidelity. For the 26 minute goal, the highest priority Web application runs mostly at the highest fidelity, while the other three applications run mostly at their lowest fidelities. For both goals, applications are more stable at the beginning of the experiment, and exhibit greater agility as energy drains.

Figure 20 summarizes the results of our experiments for goals of 20, 22, 24, and 26 minutes. These results confirm that Odyssey is doing a good job of energy adaptation. The desired goal was met in every trial of each experiment. In all cases, residual energy was very low, with the largest residue being only 1.2% of the initial energy value. The average number of adaptations by applications is generally low, but there are some cases where it is high. This is an artifact of the small initial energy value, since the system is designed to exhibit greater agility when energy is scarce.

5.3 Sensitivity to half-life

Since the choice of a smoothing function was an important factor in the prototype design, we examined the performance of our prototype using several different values for the half-life used in calculating α. In each experiment, we displayed a half-hour video clip, and specified that the energy supply should last for the entire video.



This figure shows how Odyssey meets user-specified goals for battery durations of 20 and 26 minutes when running the composite and video applications described in Section 3.7. The top graph shows how the supply of energy and the estimated demand change over time, while the other graphs show the corresponding changes in application fidelity. The applications are prioritized with Speech having the lowest priority, and Video, Map, and Web having successively higher priority.

Figure 19. Example of goal-directed adaptation

Specified Duration (s)	Goal Met	Residue				Number of Adaptations							
		Energy (J)		Time (s)		Speech		Video		Map		Web	
1200	100%	145.2	(25.3)	15.3	(1.9)	10.8	(1.6)	11.0	(4.0)	0.4	(0.9)	0.0	(0.0)
1320	100%	107.5	(61.5)	12.9	(7.2)	2.8	(0.4)	28.2	(5.2)	1.6	(2.6)	0.0	(0.0)
1440	100%	101.2	(22.3)	13.0	(4.5)	5.0	(7.9)	22.6	(9.8)	9.6	(3.8)	1.2	(1.8)
1560	100%	60.2	(28.7)	8.7	(5.9)	1.0	(0.0)	6.0	(2.8)	15.4	(4.6)	7.6	(5.9)

This figure shows system behavior when the composite application executes concurrently with the video player. In each experiment, we assume a 12,000 J energy supply. Each row shows the result of specifying a different battery-duration goal. The second column shows the percentage of trials in which the energy supply lasted for at least the specified duration. The next two columns show the residual energy at the end of the experiment. The remaining columns show the number of adaptations performed by each application. Each entry represents the mean of five trials with standard deviations given in parentheses.

Figure 20. Summary of goal-directed adaptation

Figure 21 summarizes the results of these experiments. A half-life of 1% is clearly too unstable — the system produces the largest residue with this value, and the video player adapts excessively. As the half-life increases, the system becomes more stable. However, with a 15% half-life, the system was insufficiently agile, failing to meet the goal in one trial. We have seen similar results in less detailed analysis of other adaptive applications. This led to our current decision to use a 10% half-life to calculate the value of α.

Half-Life	Goal Met	Residue (J)		Adaptations	
0.01	100%	204.6	(17.7)	93.6	(3.7)
0.05	100%	124.1	(38.0)	33.2	(4.0)
0.10	100%	129.2	(21.6)	14.6	(5.4)
0.15	80%	97.6	(22.2)	6.8	(2.9)

This figure shows sensitivity to the half-life value used for smoothing. In each experiment, we assume a 13,000 J energy supply. The second column shows the percentage of trials in which the energy supply lasted for at least the specified duration. The next column shows the residual energy at the end of the experiment. The final column shows the number of adaptations performed. Each entry is the mean of five trials with standard deviation in parentheses.

Figure 21. Sensitivity to half-life

5.4 Longer-duration experiments

The short duration (20–30 minutes) of the experiments in Sections 5.2 and 5.3 allowed us to explore the behavior of Odyssey for many parameter combinations. Having established the feasibility of goal-directed adaptation, we ran a small number of longer-duration experiments to confirm its benefits in more realistic scenarios.

We began each experiment with an energy supply of 90,000 J, roughly matching a fully-charged ThinkPad 560X battery. We specified an initial time duration of 2 hours and 45 minutes, but extended this goal by 30 minutes at the end of the first hour. This change reflects the possibility of a user modifying the estimate of how long the battery needs to last. Finally, we used a simple stochastic model to construct an irregular workload. During any given minute, each of the four applications may independently be active or idle. An active application executes a fixed workload for one minute; for example, the video application shows a one minute video, and the map application fetches five maps, with five seconds of think time after each. After each minute, there is a 10% chance of switching states; that is, an active application stays active, and an idle one stays idle, with probability 0.9.

Figure 22 presents the results of five trials of this experiment, each generated with a different random number seed. In every case, Odyssey succeeded in meeting its time goal. Four of the five trials ended with a residual energy that was less than 1% of the initial supply. Only in one trial (Trial 3) was the residue noticeably higher (2.8%), implying that Odyssey was too conservative in its adaptation.

In spite of the bursty workload, Figure 22 shows fewer adaptations than Figure 20, which had a steady workload. This is a consequence of two interactions between our hysteresis strategy and the longer-duration goal of Figure 22. First, the zone of hysteresis is much larger, since it is proportional to total energy supply. Second, smoothing is more aggressive when the goal is distant. Combined, these two factors cause Odyssey to ignore minor fluctuations in power usage except toward the end of each trial.

Trial	Goal Met	Residual Energy (J)	Number of Adaptations			
			Speech	Video	Map	Web
1	Yes	345	1	5	5	1
2	Yes	381	1	10	7	11
3	Yes	2486	8	13	5	0
4	Yes	554	2	10	6	8
5	Yes	464	5	6	14	0

This figure shows system behavior for bursty workloads. In each trial, we assume a 90,000 J energy supply. After one hour, the initial goal of 2:45 hours is extended by a half hour. Each row shows the result of a trial using a different randomly-generated workload. The second column shows whether the energy supply lasted for at least the specified duration. The next column shows the residual energy at the end of the experiment. The remaining columns show the number of adaptations performed by each application.

Figure 22. Longer duration goal-directed adaptation

6 Related work

We believe this is the first study to demonstrate the energy benefits of application adaptation. Further, it is also the first to demonstrate how a collaborative relationship between the operating system and applications can be used to meet user-specified goals for battery duration.

The most closely related work is Ellis' Milly Watt project [7], which is exploring the development of a power-based API that allows a partnership between applications and the system in setting energy use policy. The need for such an API is motivated with an adaptive map application that runs on a Palm Pilot. While this work shares many of our goals, it is at an earlier stage of maturity.

Less closely related are two pieces of work that bear some similarity to ideas proposed here. First, the use of remote compute services for speech recognition by Odyssey is reminiscent of the work on remote process execution for energy savings reported by Rudenko et al [18]. Second, the use of variable bit-width representation of data by Tong et al. [21] is an example of lowering fidelity for reduced power consumption in speech recognition.

From a broader perspective, there has been a substantial body of work on hardware power management for mobile computers. This includes efforts that span multiple components [10, 16], as well as those focusing on a single component such as the network [11, 20], disk [4, 5, 6, 12, 25], and CPU [14, 24]. As shown in this paper, such efforts aimed at hardware components are complementary to reducing energy usage through application-driven fidelity reduction.

7 Conclusion

Relentless pressure to make mobile computers lighter and more compact places severe restrictions on battery capacity. At the same time, mobile software continues to grow in complexity, hence increasing energy demand. Reconciling these opposing concerns by exploiting remote infrastructure is possible [2], but uses energy for wireless communication.

Energy-aware adaptation introduces flexibility into this overconstrained solution space. Rather than making static tradeoffs in hardware and software design, we defer these tradeoffs. At runtime, more accurate knowledge of energy supply and demand allows better decisions to be made in resolving the tension between energy conservation and usability. Our results confirm that this approach yields substantial energy savings, and can be effectively combined with hardware-centric approaches proposed by other researchers.

We see many ways in which this work can be extended. First, our experiments were conducted on a single hardware and operating system platform. We would like to explore the robustness of our results across diverse platforms. Second, we would like to broaden the range of mobile applications studied. Third, all our results have been obtained from controlled experiments. We would like to verify that these results apply in real life. This will involve making Odyssey robust enough for serious use, nurturing a user community, and studying its use of the system. Only serious use can validate many of our design decisions.

There is no silver bullet for the energy problem in mobile computing. Only through a combined effort, involving improvements at every level of the system from the battery, through hardware and the operating system, to applications can the high-end mobile systems of the future hope to operate untethered for many hours, days or possibly weeks. We are confident that energy-aware adaptation by mobile applications will be an essential part of such future systems.

Acknowledgements

This paper was made possible by the work and ideas of the Odyssey team: Dushyanth Narayanan, Brian Noble, Eric Tilton, and Kevin Walker. Bob Baron and Jan Harkes helped us solve several implementation problems. We are grateful to Matthew Reynolds for creating the Anvil software, donating it to Carnegie Mellon, and assisting in its integration into Odyssey. Jennifer Anderson, Keith Farkas, Brian Noble, David Petrou, and the anonymous reviewers provided many helpful comments that improved the paper.

References

[1] Board on Army Science and Technology, National Research Council, Washington, DC. *Energy-Efficient Technologies for the Dismounted Soldier*, 1997.

[2] Brodersen, R. W. InfoPad — past, present and future. *Mobile Computing and Communications Review*, 3(1):1–7, January 1999.

[3] Dallas Semiconductor Corp., 4401 South Beltwood Parkway, Dallas, TX. *DS2437 Smart Battery Monitor*, 1999.

[4] Douglis, F., Cáceres, R., Kaashoek, F., Li, K., Marsh, B., and Tauber, J. Storage alternatives for mobile computers. In *Proceedings of the First USENIX Symposium on Operating System Design and Implementation (OSDI)*, pages 25–37, Monterey, CA, November 1994.

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

[6] Douglis, F., Krishnan, P., and Marsh, B. Thwarting the power-hungry disk. In *Proceedings of 1994 Winter USENIX Conference*, pages 293–307, San Francisco, CA, January 1994.

[7] Ellis, C. S. The case for higher-level power management. In *The 7th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VII)*, pages 162–167, Rio Rico, AZ, March 1999.

[8] Flinn, J. and Satyanarayanan, M. PowerScope: a tool for profiling the energy usage of mobile applications. In *Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications*, pages 2–10, New Orleans, LA, February 1999.

[9] Fox, A., Gribble, S. D., Brewer, E. A., and Amir, E. Adapting to network and client variability via on-demand dynamic distillation. In *Proceedings of the Seventh International ACM Conference on Architectural Support for Programming Languages and Operating Systems*, pages 160–170, Cambridge, MA, October 1996.

[10] Intel, Microsoft, and Toshiba. *Advanced Configuration and Power Interface Specification*, February 1998. http://www.teleport.com/~acpi/ (as of 9/99).

[11] Kravets, R. and Krishnan, P. Power management techniques for mobile communication. In *Proceedings of The Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'98)*, pages 157–168, Dallas,TX, October 1998.

[12] Li, K., Kumpf, R., Horton, P., and Anderson, T. A quantitative analysis of disk drive power management in portable computers. In *Proceedings of the 1994 Winter USENIX Conference*, pages 279–291, San Francisco, CA, January 1994.

[13] Lorch, J. R. A complete picture of the energy consumption of a portable computer. Master's thesis, Department of Computer Science, University of California at Berkeley, 1995.

[14] Lorch, J. R. and Smith, A. J. Scheduling techniques for reducing processor energy use in MacOS. *Wireless Networks*, 3(5):311–324, October 1997.

[15] Lorch, J. R. and Smith, A. J. Apple Macintosh's energy consumption. *IEEE Micro*, 18(6):54–63, November/December 1998.

[16] Lorch, J. R. and Smith, A. J. Software strategies for portable computer energy management. *IEEE Personal Communications*, 5(3):60–73, June 1998.

[17] Noble, B. D., Satyanarayanan, M., Narayanan, D., Tilton, J. E., Flinn, J., and Walker, K. R. Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium on Operating Systems and Principles*, pages 276–287, Saint-Malo, France, October 1997.

[18] Rudenko, A., Reiher, P., Popek, G. J., and Kuenning, G. H. Saving portable computer battery power through remote process execution. *Mobile Computing and Communications Review*, 2(1):19–26, January 1998.

[19] SBS Implementers Forum, http://www.sbs-forum.org/ (as of 9/99). *Smart Battery Data Specification, Revision 1.1*, December 1998.

[20] Stemm, M. and Katz, R. H. Measuring and reducing energy consumption of network interfaces in handheld devices. *IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Science, Special Issue on Mobile Computing*, 80(8):1125–1131, August 1997.

[21] Tong, Y. F., Rutenbar, R. A., and Nagle, D. F. Minimizing floating-point power dissipation via bit-width reduction. In *Power-Driven Microarchitecture Workshop*, Barcelona, Spain, June 1998.

[22] USAR Systems, Inc., 568 Broadway, Suite 405, New York, NY. *USAR ACPITroller II - Zero-Power ACPI KBC with Built-in Smart Battery System Manager*, 1999.

[23] Waibel, A. Interactive translation of conversational speech. *IEEE Computer*, 29(7):41–48, July 1996.

[24] Weiser, M., Welch, B., Demers, A., and Shenker, S. Scheduling for reduced CPU energy. In *Proceedings of the First USENIX Symposium on Operating System Design and Implementation (OSDI)*, pages 13–23, Monterey, CA, November 1994.

[25] Wilkes, J. Predictive power conservation. Technical Report HPL-CSP-92-5, Hewlett-Packard Laboratories, February 1992.