# PowerScope:
## A Tool for Profiling the Energy Usage of Mobile Applications

Jason Flinn and M. Satyanarayanan
School of Computer Science
Carnegie Mellon University
{jflinn,satya}@cs.cmu.edu

## Abstract

*In this paper, we describe the design and implementation of PowerScope, a tool for profiling energy usage by applications. PowerScope maps energy consumption to program structure, in much the same way that CPU profilers map processor cycles to specific processes and procedures. Our approach combines hardware instrumentation to measure current level with kernel software support to perform statistical sampling of system activity. Postprocessing software maps the sample data to program structure and produces a profile of energy usage by process and procedure. Using PowerScope, we have been able to reduce the energy consumption of an adaptive video playing application by 46%.*

## 1. Introduction

Energy is a critical resource for mobile computers [5, 8]. In spite of many improvements in low-power hardware design and battery life, there is now growing awareness that a strategically viable approach to energy management must include higher levels of the system [2]. For example, a network application that offers acceptable service while temporarily disconnected can save a considerable amount of energy by suppressing non-essential wireless communication. The resulting savings add to those offered by energy-efficient hardware. In contrast, efforts aimed solely at the hardware cannot benefit from application-specific knowledge.

Progress in energy-efficient software design requires the ability to attribute energy consumption to specific software components, in much the same way that CPU profilers such as `prof` and `gprof` help expose code components that are wasteful of processor cycles. In this paper, we describe a tool called *PowerScope* that we have built to profile energy usage.

PowerScope maps energy consumption to program structure. Using PowerScope, one can determine what fraction of the total energy consumed during a certain time period is due to specific processes in the system. Further, one can drill down and determine the energy consumption of different procedures within a process. By providing such fine-grained feedback, PowerScope allows attention to be focused on those system components responsible for the bulk of energy consumption. As improvements are made to these components, PowerScope quantifies their benefits and helps expose the next target for optimization. Through successive refinement, a system can be improved to the point where its energy consumption meets design goals.

Our initial experience with this tool has been rewarding. By using PowerScope, we have obtained a 46% reduction in total energy consumption when an adaptive video application is run on the Odyssey platform for mobile computing [6]. Some of the steps in the path to achieving this reduction were counterintuitive — certain obvious changes did not produce anticipated savings. Thus, although this research is still in its early stages, we are convinced that an energy profiling tool such as PowerScope is indispensable in building mobile computing systems. In the rest of this paper, we present the design and implementation of PowerScope and describe its use in the adaptive video case study.

## 2. Design considerations

The most important design consideration was for PowerScope to gather sufficient information to produce a detailed picture of system activity. The usefulness of a profiling tool is directly related to how definitively it assigns costs to specific application events. Attributing costs in detail enables attention to be focused quickly on problem areas in

This research was sponsored by the Defense Advanced Research Projects Agency (DARPA), Air Force Materiel Command, USAF under agreement number F19628-96-C-0061, the Intel Corporation, and AT&T Corporation. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Intel, AT&T, DARPA, or the U.S. Government.

the code. With this in mind, we felt it insufficient to map energy costs only to specific processes — we also desired to map costs to the procedure level.

A second consideration was that our tool monitor the activity of *all* processes executing on a computer system. We have found that profiling the activity of only a single process omits critical information about total energy usage. For instance, a task which blocks frequently may expend large amounts of energy on the screen, disk, and network when the processor is idle. Furthermore, asynchronous activity, such as network interrupts, can account for a significant portion of energy consumption. An energy profiler which monitors energy usage only when a specific process is executing will not account for the energy expended by these activities.

Finally, we have taken care to minimize the overhead generated by our tool. A profiler necessarily induces some overhead on the system that it monitors. For an energy profiler, this is reflected both in additional CPU usage and in additional energy expended during execution. We have striven to reduce this overhead.

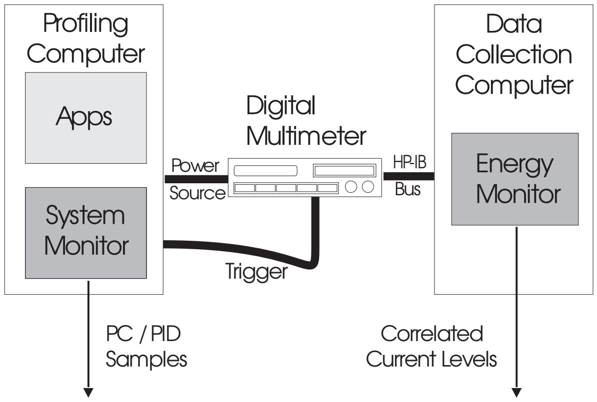
## 3. Implementation

### 3.1. Overview

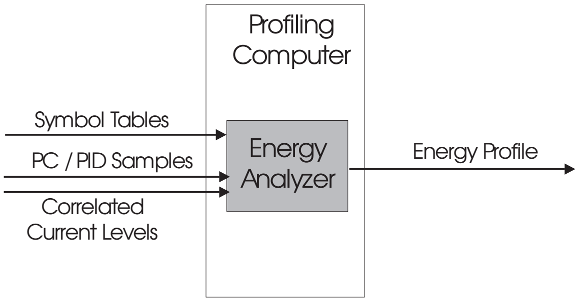
The prototype version of PowerScope, shown in Figure 1, uses statistical sampling to profile the energy usage of a computer system. To reduce overhead, profiles are generated by a two-stage process. During the data collection stage, the tool samples both the power consumption and the system activity of the profiling computer. PowerScope then generates an energy profile from this data during a later analysis stage. Because the analysis is performed off-line, it creates no profiling overhead.

During data collection, we use a digital multimeter to sample the current drawn by the profiling computer through its external power input. We require that this multimeter have an external trigger input and output, as well as the ability to sample DC current at high frequency. Our present implementation uses a Hewlett Packard 3458a digital multimeter, which satisfies both these requirements. A separate data collection computer controls the multimeter and stores current samples.

We considered an alternate implementation in which measurement and data collection were performed on the profiling computer using an on-board digital multimeter with a PCI or PCMCIA interface. However, this implementation makes it very difficult to differentiate the energy consumed by the profiled applications from the energy used by data collection and by the operation of the on-board multimeter. Further, our current implementation allows easy switching of the measurement equipment among different profiling targets.




(a) Data Collection




(b) Off-Line Analysis

This figure shows how PowerScope generates an energy profile. As applications execute on the profiling computer, the System Monitor samples system activity and the Energy Monitor samples power consumption. Later, the Energy Analyzer uses this information to generate an energy profile.

**Figure 1. PowerScope Architecture**

Because our tool requires a small set of kernel modifications, we require that a source-available operating system run on the profiling computer. At present, we are using the NetBSD operating system. There is no specific operating system requirement for the data collection computer; we currently use Windows 95 to take advantage of manufacturer-provided device drivers for our multimeter.

The functionality of PowerScope is divided among three software components. Two components, the System Monitor and Energy Monitor, share responsibility for data collection. The System Monitor samples system activity on the profiling computer by periodically recording information which includes the program counter (PC) and process identifier (PID) of the currently executing process. The Energy Monitor runs on the data-collection computer, and is responsible for collecting and storing current samples. Because data collection is distributed across two monitor processes, it is essential that some synchronization method en-

```
pscope_init (u_int size);
pscope_read (void* sample,
             u_int size,
             u_int* ret_size);
pscope_start (void);
pscope_stop (void);
```

**Figure 2. PowerScope API**

sure that they collect samples closely correlated in time. We have chosen to synchronize the components by having the digital multimeter signal the profiling computer after taking each sample.

The final software component, the Energy Analyzer, uses the raw sample data collected by the monitors to generate the energy profile. The analyzer runs on the profiling computer since it uses the symbol tables of the executables on disk to map samples to specific procedures. There is an implicit assumption in this method that the executables being profiled are not modified between the start of profile collection and the running of the off-line analysis tool.

### 3.2. The System Monitor

The System Monitor consists of a user-level daemon process and a small set of modifications to the NetBSD kernel. Its design is similar to the sampling components of continuous profilers such as Morph [9] and DCPI [1]. Our current implementation samples system activity when triggered by the digital multimeter. Each twelve byte sample records the value of the program counter (PC) and the process identifier (PID) of the currently executing process, as well as additional information such as whether the system is currently handling an interrupt. This assumes that the profiling computer is a uniprocessor — a reasonable assumption for a mobile computer.

Samples are written to a circular buffer residing in kernel memory. This buffer is emptied by the user-level daemon, which writes the samples to disk. The daemon is triggered when the buffer grows more than 7/8 full, or at the end of data collection.

The System Monitor records a small amount of additional information to assist in the generation of energy profiles. The kernel `fork()`, `exec()`, and `exit()` routines are instrumented to record the pathname associated with each currently executing process. The NetBSD run-time system loader is also instrumented to record the loading of shared libraries. This information is written to the sample buffer during data collection, and is used during off-line analysis to associate each sample with a specific executable image on disk.

We created a small number of system calls, shown in Figure 2, to allow applications to control profiling. The user-level daemon calls `pscope_init()` to set the size of the kernel sample buffer. Since there is a tension between excessive memory usage and frequent reading of the buffer by the user-level daemon, the buffer size has been left flexible to allow efficient profiling of different workloads. The `pscope_read()` system call is used by the user-level daemon to read samples out of the buffer. The `pscope_start()` and `pscope_stop()` system calls allow application programs to precisely indicate the period of sample collection. Multiple sets of samples may be collected one after the other; each sample set is delineated by start and end markers written into the sample buffer.

### 3.3. The Energy Monitor

The Energy Monitor runs on the data collection computer and communicates with the digital multimeter. It configures the multimeter to sample the current being drawn by the laptop from its external power source. In our experience, the voltage variation is extremely small (measured at less than 0.25%). Therefore, current samples alone are sufficient to determine the energy usage of the system. The battery is removed from the laptop while measurements are taken to avoid extraneous power drain caused by charging. Current samples are transmitted asynchronously to the Energy Monitor which stores them on disk for later analysis.

Sample collection is driven by the multimeter clock. Synchronization with the System Monitor is provided by connecting the multimeter's external trigger input and output to pins on the parallel port of the profiling computer. Immediately after the multimeter takes a current sample, it toggles the value of a parallel port pin. This causes a system interrupt on the profiling computer, during which the System Monitor samples system activity. Upon completion, the System Monitor triggers the next sample by toggling another parallel port pin (unless profiling has been halted by the `pscope_stop` system call). The multimeter buffers this trigger until the time to take the next sample arrives.

Our original design used the clock of the profiling computer to drive sample collection. Although simpler to implement, that design had the disadvantage of biasing the profile values of activities correlated with the system clock. Using the multimeter clock also allows us to generate interrupts at a finer granularity then that allowed by the kernel `stat_clock` routine. The user may specify the sample period as a parameter when the Energy Monitor is started. For all measurements reported in this paper, we used a sample period of approximately 1.6 ms.

| Process | Elapsed 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 |
| /netbsd (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 |

(a) Summary of Energy Usage by Process

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

User-level procedures:

| Procedure | Elapsed 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 |

(b) Partial Detail of Process Energy Usage

Energy Usage Detail for process Interrupts-WaveLAN

Kernel-level procedures:

| Procedure | Elapsed Time (s) | Total Energy (J) | Average Power (W) |
|---|---|---|---|
| _xferDMAbuffer | 16.66 | 147.38 | 8.85 |
| _pwlread | 0.30 | 2.90 | 9.65 |
| _pwlget | 0.30 | 2.68 | 8.93 |
| _pwlintr | 0.24 | 2.31 | 9.62 |

(c) Partial Detail of WaveLAN Interrupt Energy Usage

This figure shows a sample energy profile for an adaptive video playing application. Part (a) summarizes the energy usage by process. Part (b) shows a portion of the detailed profile for a single process, while part (c) shows a portion of the detailed profile for WaveLAN interrupts.

**Figure 3. Sample Energy Profile**

### 3.4. The Energy Analyzer

The Energy Analyzer generates an energy profile of system activity. Total energy usage can be calculated by integrating the product of the instantaneous current and voltage over time. We can approximate this value by simultaneously sampling both current, $I_t$, and voltage, $V_t$, at regular intervals of time $\Delta t$. Further, in our current implementation, $V_t$ is constant within the limits of accuracy for which we are striving. We therefore calculate total energy over $n$ samples using a single measured voltage value, $V_{meas}$, as follows:

$$E \approx V_{meas} \sum_{t=0}^{n} I_t \Delta t \qquad (1)$$

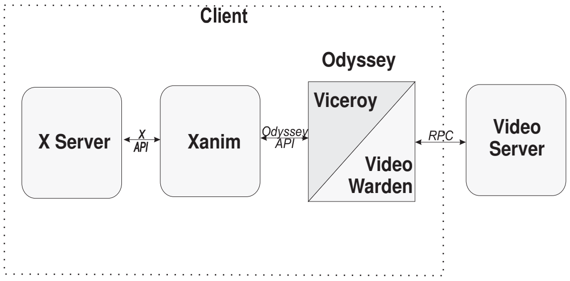
The Energy Analyzer reads the raw data generated by the monitors and associates each current sample collected by the Energy Monitor with the corresponding sample collected by the System Monitor. The analyzer assigns each sample to a process bucket using the recorded PID value. Samples that occurred during the handling of an asynchronous interrupt, such as the receipt of a network packet, are not attributed to the currently executing process but are instead attributed to a bucket specific to the interrupt handler. If no process was executing when the sample was taken, the sample is attributed to a kernel bucket. The energy usage of each process is calculated as in Equation 1 by summing the current samples in each bucket and multiplying by the measured voltage ($V_{meas}$) and the sample interval ($\Delta t$).

The analyzer then generates a summary of energy usage by process, such as the one shown in Figure 3(a). Each entry displays the total time spent executing the process, the total energy usage of the process, and the average power usage (simply calculated by dividing the energy value by the time value). We envision extending this summary to include histograms of energy usage over time.

The analyzer repeats the above steps for each process to determine the energy usage by procedure. The process and shared library information stored by the System Monitor is used to reconstruct the memory address of each procedure from the symbol tables stored on disk. Then, the PC value of each sample is used to place the sample in a procedure bucket. When the profile is generated, procedures that reside in shared libraries and kernel procedures can be displayed separately. Figure 3(b) shows a partial profile of one typical process, and Figure 3(c) shows a partial profile of WaveLAN interrupts.

## 4. Case study: adaptive video

We decided to test the effectiveness of our tool by using it to reduce the energy consumption of an adaptive video-player. The application, first described in an earlier paper on Odyssey [6], is based on *xanim*, a public-domain software package that can generate video animation from data stored in various formats in a local file. As shown in Figure 4, we split its monolithic implementation into a client and server. Video frames are prefetched from the remote server by Odyssey, decoded by the xanim client, and displayed by the X server.




This figure shows the architecture of an adaptive video-playing application. Odyssey prefetches frames from the remote server. They are subsequently decoded by the xanim process and displayed by the X server.

**Figure 4. Video Application**

The application currently adapts to changing network conditions by varying the video quality. When bandwidth drops, the client plays tracks encoded with a greater amount of lossy compression. We were curious to see if the adaptivity could be extended to conserve client energy usage. In this design, when battery life is low, the player would use less energy by displaying a degraded version of the current video. Therefore, one of the goals of our case study was to identify a method for creating significant energy savings by slightly degrading the quality of the video. Of course, we also sought to reduce the total energy consumption of the video player across all track qualities.

For the following study, the client machine is a 75 MHz 486 IBM 701C laptop with 24 MB of memory and the server is a 200 MHz Pentium Pro with 64 MB of memory. The client and server communicate using a wireless 900 MHz Lucent WaveLAN network. Figure 5 shows the measured power consumption for key components of the client machine.

### 4.1. Effect of lossy compression

We first used PowerScope to measure the energy used to play three video tracks which varied only in the amount of lossy compression used for encoding. Each track was generated from the same 184 second 320x240 pixel video clip using Adobe Premiere. For reference, we have labeled these compression levels Premiere-A, Premiere-B, and Premiere-C. Premiere-A is the highest quality track, and Premiere-C

| Component | State | Power (W) |
|---|---|---|
| Display | Bright | 4.87 |
| | Dim | 3.99 |
| WaveLAN | Idle | 1.57 |
| | Standby | 0.15 |
| Disk | Idle | 1.06 |
| | Standby | 0.17 |
| Other | Idle | 1.11 |

This figure shows the measured power consumption of components of the IBM 701C laptop in their various power states. Power consumption is slightly but consistently superlinear; for example, the machine consumes 8.69 W when the screen is at its brightest and the disk and network are idle, which is 0.08 W more than the sum of the individual power consumption of each component when measured separately. The last row displays the power consumed by the laptop when the disk, screen, and network are all powered off. Each reported value is the mean of five trials — in all cases, the sample standard deviation is less than 0.01 W.

**Figure 5. Power Measurements for IBM 701C**

| Encoding | Display | Size | Data Rate |
|---|---|---|---|
| Premiere-A | 320x240 | 12.1 MB | 515 Kb/s |
| Premiere-B | 320x240 | 7.0 MB | 296 Kb/s |
| Premiere-C | 320x240 | 2.8 MB | 120 Kb/s |
| Premiere-A | 160x120 | 4.9 MB | 206 Kb/s |
| Premiere-C | 160x120 | 1.0 MB | 44 Kb/s |

This figure shows the characteristics of the five video tracks used in this case study. All tracks were generated from the same video clip and are 184 seconds in length. From left to right, the columns describe the amount of lossy compression used to encode the video, the size of the video display (in pixels), the size of the video (in megabytes) and the data rate (in kilobits per second). All tracks were encoded in QuickTime CinePak format using Adobe Premiere. Premiere-A is the highest quality track. Premiere-C is the lowest.

**Figure 6. Video Track Characteristics**

is the lowest. The basic characteristics of the video tracks are shown in the first three rows of Figure 6.

The results of this experiment are shown in Figure 7(a). To our surprise, playing tracks with greater compression results in only a small (13.5%) reduction in energy usage. Analysis of the energy usage by process revealed that compression significantly reduces the energy used by network-related activities (the Odyssey process and asynchronous WaveLAN interrupts) and the xanim video player. However, the energy consumed by the X server remains relatively unaffected by the amount of compression, most probably because it operates only on data that has already been decoded.

## 4.2. Effect of display size

We next decided to try to reduce the X server's energy usage by reducing the size of the video display from 320x240 pixels to 160x120. We therefore generated two 160x120 tracks, one encoded at Premiere-A and one encoded at Premiere-C, from the same video clip used in the previous experiment. Figure 7(b) shows the effects of reducing the display size for both encodings.

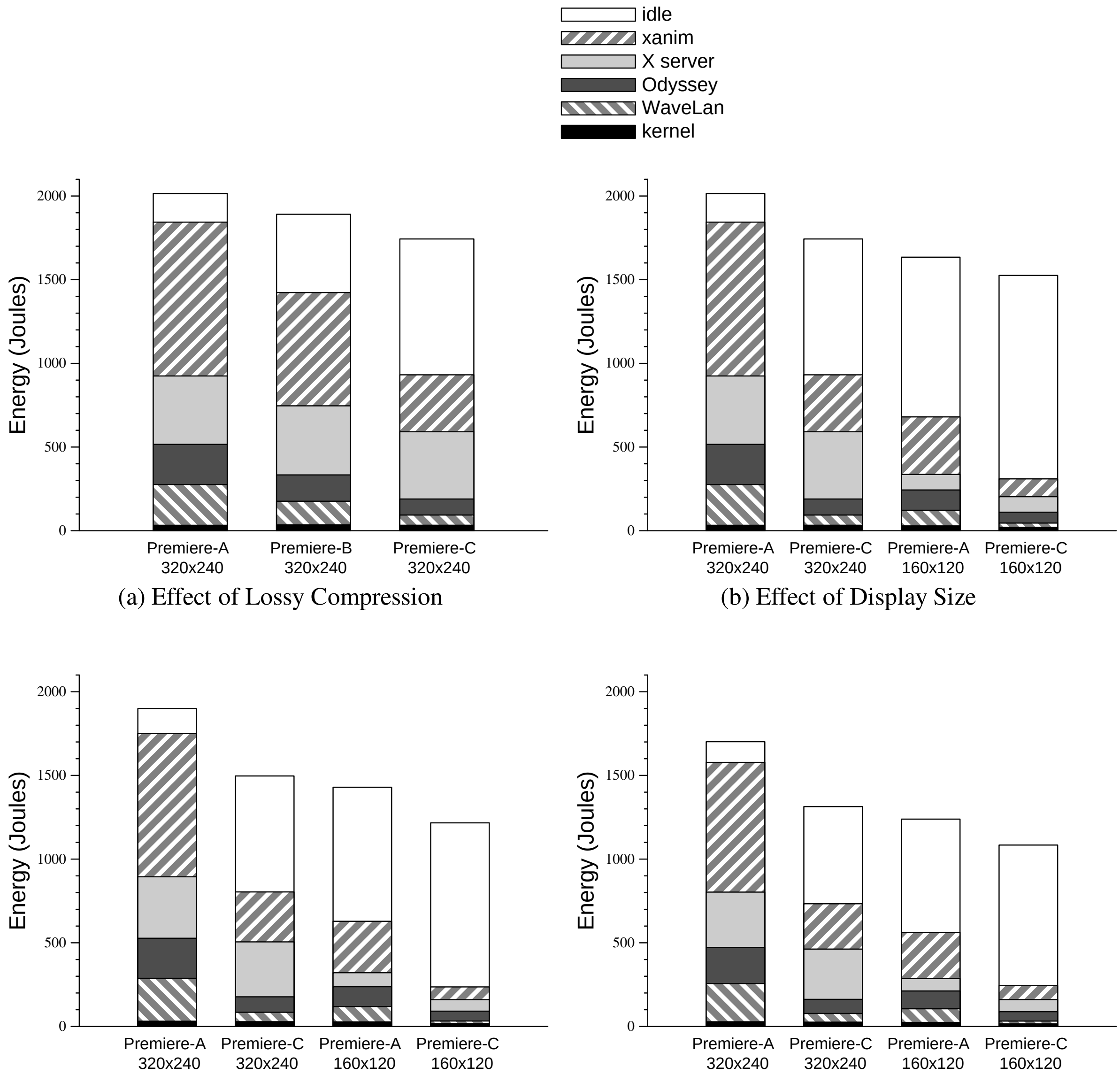
Despite having a greater data rate, the Premiere-A 160x120 video consumes less energy than the Premiere-C 320x240 video. The greater energy usage for network-related activities is more than compensated for by an approximately 75% reduction in energy usage by the X server. When the Premiere-C 160x120 video track is played, the largest energy reduction (24.3%) is achieved. The majority of the energy consumption for this track occurs when the CPU is idle.

We conjectured that this behavior indicated that most of the energy was now being consumed by keeping hardware devices such as the network and disk in their idle states. To test this, we measured the power expended by the profiling computer with the screen on, and the CPU, network, and disk idle. This configuration uses 7.80 Watts, which projected over the length of the video accounts for 1434 Joules, or 94% of the total energy consumed by the Premiere-C 160x120 track. Clearly, this value represents the upper bound on the energy savings that can be achieved by degrading the video without changing the power state of client hardware components.

## 4.3. Effect of network power management

These results led us to explore the feasibility of powering down hardware components when they are not in use. Although the display accounts for the majority of the energy consumed by the client, it is difficult to imagine how it could be disabled while the video is playing. Our efforts therefore concentrated on the network interface and the hard drive.

As shown in Figure 5, completely powering down the WaveLAN interface achieves a power savings of 1.57 Watts. However, restarting the interface from this state requires 137 milliseconds, which is less than ideal for a network-intensive application like our video player. Fortunately, our WaveLAN hardware supports the ability to disable only those components that can be quickly restarted, including the RF-modem. We modified our NetBSD WaveLAN driver by adding `ioctl` operations which support transitions to and from this standby mode. Our measurements indicate that the interface consumes only 0.15 Watts while in standby mode and requires only 0.81 milliseconds to resume transmission.




(a) Effect of Lossy Compression

(b) Effect of Display Size

(c) Effect of Network Power Management

(d) Effect of Disk Power Management

This figure shows the amount of energy used by the video player when various power saving strategies are applied. Part (a) shows the effect of varying the amount of lossy compression. Part (b) shows the effect of reducing the size of the video display. Part (c) shows the effect of switching the network interface to standby mode when not in use. Part (d) shows the effect of powering down the disk. Note that the effects are applied cumulatively, so that part (d) shows the aggregate effect of all four power-saving strategies. Each reported result is the mean of five trials. The number of dropped or late frames was less than 4% in every trial. Refer to Figure 6 for the size and data rate of each video track.

**Figure 7. Reducing the Energy Consumption of a Video Player**

We next modified Odyssey to manage the power state of the WaveLAN interface. Odyssey fetches frames from the server using a RPC protocol. After each RPC completes, Odyssey puts the interface into standby mode until it is ready to initiate the next RPC. Here, we assume that the video player is the only application using the network.

As shown in Figure 7(c), network power management reduces the energy consumption of all tracks, achieving greater reductions for tracks which require less network activity. The video player consumes 33.1% less energy playing the Premiere-C 160x120 track (the rightmost bar in Figure 7(c)) compared to playing the Premiere-A 320x240 track (the leftmost bar in Figure 7(c)). When compared to playing the Premiere-A 320x240 track with no network power management (the leftmost bar in Figure 7(a)), the energy reduction is 36.9%.

### 4.4. Effect of disk power management

Since the video player buffers frames entirely in memory, the disk is not required when playing a video. We therefore modified Odyssey to power down the disk when the video first starts to play. As shown in Figure 7(d), this reduces the amount of energy needed to play each track by approximately 195 Joules. With both network and disk power management, degrading the quality of the video provides an energy savings of 36.3%. This value is calculated by comparing the Premiere-C 160x120 track (the rightmost bar in Figure 7(d)) with the 320x240 Premiere-A track (the leftmost bar in Figure 7(d)). Cumulatively applying all optimizations in this case study achieves a 46.2% energy savings compared to playing the Premiere-A 320x240 track without power management (the leftmost bar in Figure 7(a)).

## 5. Related work

To the best of our knowledge, PowerScope is the first tool that maps energy consumption to program structure. At the same time, our development of the tool was motivated by previous work in CPU profiling. In particular, the implementation of the System Monitor is closely related to similar components of continuous profilers such as Morph [9] and DCPI [1].

Several researchers have previously investigated power management in the context of wireless networks and disks. Stemm and Katz [7] measured the power consumption of several network interfaces. They also presented simulation results showing how powering down the network interface could reduce energy consumption for e-mail and web-browsing applications. Kravets and Krishnan [4] proposed a transport-level protocol which powers down the network interface for short periods and measured the reduction in energy consumption for communication patterns typical of common applications. Douglis et al. [3] investigated the energy implications of various strategies for powering down the disk.

## 6. Conclusion

We believe PowerScope is a valuable tool that can facilitate further research in developing energy-efficient mobile applications. We are greatly encouraged by our initial study which reduced the energy consumption of an adaptive movie player by 46%. Our current plans for the tool include several enhancements.

In the short-term, we plan to repeat the movie player case study for at least one additional laptop computer. This will have the dual purpose of verifying the robustness of our tool across multiple hardware platforms and allowing us to obtain measurements for a more modern computer. We also plan additional experiments to carefully calibrate the performance of the tool.

In the longer-term, we plan to incorporate a more detailed model of the relationship between energy usage and battery life. In addition to total energy usage, other factors such as peak power levels can significantly effect battery life. Our enhancement would be mostly to the analyzer component since we currently collect sufficiently detailed sample data to support more complex analysis.

Finally, we plan to use our tool to investigate the behavior of multiple adaptive applications which concurrently access hardware components such as the network and disk. In this and other areas of investigation, PowerScope promises to be of considerable assistance.

## Acknowledgements

Tom Martin helped us considerably with his many useful suggestions and his assistance with our experimental setup. Bob Baron provided us with his NetBSD kernel expertise. Dushyanth Narayanan, David Petrou, Eric Tilton, and Kip Walker provided much useful feedback during the course of this project.

## References

[1] J. M. Anderson, L. M. Berc, J. Dean, S. Ghemawat, M. R. Henzinger, S.-T. A. Leung, R. L. Sites, M. T. Vandevoorde, C. A. Waldspurger, and W. E. Weihl. Continuous Profiling: Where Have All the Cycles Gone? In *Proceedings of the 16th ACM Symposium on Operating Systems and Principles*, Saint-Malo, France, October 1997.

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

[3] F. Douglis, P. Krishnan, and B. Marsh. Thwarting the Power-Hungry Disk. In *Proceedings of 1994 Winter USENIX Conference*, January 1994.

[4] R. Kravets and P. Krishnan. Power Management Techniques for Mobile Communication. In *Proceedings of The Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'98)*, October 1998.

[5] J. R. Lorch and A. J. Smith. Software Strategies for Portable Computer Energy Management. *IEEE Personal Communications*, 5(3), June 1998.

[6] 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 and Principles*, Saint-Malo, France, October 1997.

[7] M. Stemm and R. H. Katz. Measuring and Reducing Energy Consumption of Network Interfaces in Hand-Held Devices. *IEICE Transactions on Communications, Special Issue on Mobile Computing*, 80(8), August 1997.

[8] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. In *Proceedings of the First USENIX Symposium on Operating System Design and Implementation (OSDI)*, Monterey, CA, November 1994.

[9] X. Zhang, Z. Wang, N. Gloy, J. B. Chen, and M. D. Smith. System Support for Automated Profiling and Optimization. In *Proceedings of the 16th ACM Symposium on Operating Systems and Principles*, Saint-Malo, France, October 1997.