ENERGY MANAGEMENT IN MOBILE DEVICES WITH THE CINDER OPERATING SYSTEM

Arjun Roy, Stephen M. Rumble, Ryan Stutsman, Philip Levis, David Mazières, Nickolai Zeldovich
PROBLEM

• memory and CPU time treated as first-class resources

• mobile devices are „the dominant end-user computing platform of the decade“

• **energy is the new speed**

• energy is not controllable at all

• (at least not like memory and CPU time)
MECHANISM

Reserve

• right to use a given quantity of a resource
• when the resource is used, the reserve is consumed

Tap

• conduit between a source and a sink reserve
• transfers specific rate of resource allowance
QUOTAS

ergy isolation, subdivision and delegation

Figure 1. A 15 kJ battery, or root reserve, connected to a reserve via a tap. The battery is protected from being misused by the web browser. The web browser draws energy from an isolated reserve which is fed by a 750 mW tap.

3.4 Resource Consumption Graph

Reserves and taps form a directed graph of resource consumption rights. The root of the graph is a reserve representing the system battery; all other reserves are a subdivision of this root reserve. Figure 1 shows a simple example of a web browser whose consumption is rate limited using a tap. The tap guarantees that even if the browser is aggressively using energy the battery will last at least 5 hours (15,000 J at 0.750 J/s is about 5.6 hours).

3.5 Access Control & Security

Any thread can create and share reserves or taps to subdivisions and delegate its resources. This ability introduces a problem of fine-grained access control. To solve this, reserves and taps are protected by a security label, like all other kernel objects. The label describes the privileges needed to observe, modify, and use the reserve or tap.

Using resources from a reserve requires both observe and modify privileges: observe because failed consumption indicates the reserve level (zero) and modify for when consumption succeeds. Since a tap actively moves resources between a source and sink reserve, it needs privileges to observe and modify both reserve levels; to aid with this, taps can have privileges embedded in them.

4. Cinder on the HTC Dream

Controlling energy requires measuring or estimating its consumption. This section describes Cinder's implementation and its energy model. The Cinder kernel runs on AMD64, i386, and ARM architectures. All source code is freely available under open-source licenses. Our principal experimental platform is the HTC Dream (Google G1), a modern smartphone based on the Qualcomm MSM7201A chipset.

4.1 Energy accounting

Energy accounting on the HTC Dream is difficult due to the closed nature of its hardware. It has a two-processor design, as shown in Figure 2. The operating system and applications run on an ARM11 processor. A secure, closed ARM9 coprocessor manages the most energy hungry, dynamic, and informative components (e.g. GPS, radio, and battery sensors). The ARM9, for example, exposes the battery level as an integer from 0 to 100.

Recent work on processors has shown that fine-grained performance counters can enable accurate energy estimates within a few percent [Economou 2006; Snowdon 2009]. Without access to such state in the HTC Dream, however, Cinder relies on the simpler well-tested technique of building a model from offline measurements of device power states in a controlled setting [Flinn 1999b; Fonseca 2008; Zeng 2002]. Phones today use this approach, and so Cinder has equivalent accuracy to commodity systems.

4.2 Power Model

Our energy model uses device states and their duration to estimate energy consumption. We measured the Dream's energy consumption during various states and operations. All measurements were taken using an Agilent Technologies E3644A, a DC power supply with a current sense resistor that can be sampled remotely via an RS-232 interface. We sampled both voltage and current approximately every 200 ms, and aggregated our results from this data.

While idling in Cinder, the Dream uses about 699 mW and another 555 mW when the backlight is on. Spinning the CPU increases consumption by 137 mW. Memory-intensive instruction streams increase CPU power draw by 13% over a simple arithmetic loop. However, the HTC Dream does not have hardware support to estimate what percentage of instructions are memory accesses. The ARM processor also lacks a floating point unit, leaving us with only integer, control flow, and memory instructions. For these reasons, our CPU model currently does not take instruction mix into account and assumes the worst case power draw (all memory intensive operations).

4.3 Peripheral Power

The baseline cost of activating the radio is exceptionally high: small isolated transfers are about 1000 times more expensive, per byte, than large transfers. Figure 3 demonstrates the cost of activating the radio and sending UDP packets to an echo server that returns the same contents. Results demonstrate that the overhead involved dominates the total
Figure 4. Total energy consumption. The average cost is 14.3 J (minimum: 9.5 J, maximum 14.3 J) to allow users and applications to provision and manage their limited budgets. Accurate energy accounting is a critical component of energy management. Cinder's aim is to leverage advances in energy accounting techniques to improve energy efficiency. In particular, Cinder focuses on energy wrap, a technique that allows applications to be bundled into a single executable that can be run on any device. Cinder's implementation is designed to be easy to use, with a simple interface for creating and managing energy reservations.

Cinder's energy wrap allows applications to be provisioned and managed in a way that is transparent to the user. This is achieved by wrapping the application with an energy sandbox, which manages the application's energy consumption and ensures that it does not exceed the available budget. The sandbox also provides a way to monitor the application's energy usage, allowing users to track their energy consumption and make informed decisions about their energy use.

Cinder's energy wrap is particularly useful for applications that are resource-intensive, such as those that require high computational power or large amounts of data. By wrapping these applications, Cinder can ensure that they do not impact other applications running on the device, while still allowing users to access the benefits of these applications.

Cinder can be used to run applications on devices that do not have dedicated energy management capabilities, or on devices that have limited energy resources. By using Cinder's energy wrap, users can be confident that their applications will run efficiently and effectively, without impacting other applications or the device's overall energy consumption.

The diagram shows the energy consumption of applications running on a device over a period of 10 seconds, with different packet sizes and rates. The x-axis represents the number of packets per second, while the y-axis represents the amount of energy consumed in joules. The different lines represent different packet sizes, with larger packet sizes resulting in higher energy consumption.

The data shows that for short flows lasting less than 10 seconds, the energy consumption is reasonable, with the average plateau consuming an additional 9.5 J of energy. For longer flows, the energy consumption increases, with the maximum 11.9 J for flows lasting up to 35 seconds.

In conclusion, Cinder's energy wrap is a powerful tool for managing energy consumption on devices, allowing applications to be run efficiently and effectively without impacting other applications or the device's overall energy consumption. By using Cinder, users can make informed decisions about their energy use and ensure that their applications run efficiently on a wide range of devices.
LIMIT ENERGY HOARDING

![Diagram showing energy flow between Root, Browser, and Plugin]

- **Root** to **Browser**: 0.1x, 700 mW
- **Browser** to **Root** and **Plugin**: 0.1x, 70 mW

The diagram illustrates energy flow between different components, emphasizing the principles of energy management and hoarding prevention in the context of resource allocation.
TIDBITS

• applications can inspect their reserve and adapt
• different reserves for foreground and background operation
• executing a service depletes the caller’s reserve
• multiple caller’s can pool reserves to pay device startup cost
• netd short-sales reserve when receiving
EVALUATION

![Stacked graph of Cinder's CPU energy accounting estimates during isolated process execution. Process A's energy consumption is isolated from other processes' energy use despite B's periodic spawning of child processes (B1 and B2). The sum of B's energy, not accounting for A, is only slightly above A's and is about half of B's power. Figure 9 shows that both A and B's estimates of their power of B's tap, such that after spawning both they are using half of B's power. This leaves the problem of how to charge for incoming packets since energy has already been spent to receive them.

To facilitate this, threads can debit their own reserves up to their-fact. This allows user space accounting; for example, in practice, some genuine threads may be charged for energy while others are not. This situation is analogous to the radio case: a thread may be charged for energy while another is not. However, the radio case is more complex because energy is consumed in a continuous manner.

In this section, we evaluate whether Cinder can control energy, provide visibility into the energy of a running system, and provide subdivision, delegation, as well as isolation? Furthermore, we evaluate whether Cinder meets the requirements described in Sections 1 and 6. Evaluation cooperation between applications to increase the responsiveness of systems.

In this section, we evaluate whether Cinder can control energy, provide visibility into the energy of a running system, and provide subdivision, delegation, as well as isolation? Furthermore, we evaluate whether Cinder meets the requirements described in Sections 1 and 6. Evaluation cooperation between applications to increase the responsiveness of systems.

In this section, we evaluate whether Cinder can control energy, provide visibility into the energy of a running system, and provide subdivision, delegation, as well as isolation? Furthermore, we evaluate whether Cinder meets the requirements described in Sections 1 and 6. Evaluation cooperation between applications to increase the responsiveness of systems.

In this section, we evaluate whether Cinder can control energy, provide visibility into the energy of a running system, and provide subdivision, delegation, as well as isolation? Furthermore, we evaluate whether Cinder meets the requirements described in Sections 1 and 6. Evaluation cooperation between applications to increase the responsiveness of systems.
ADAPTIVE APPLICATION

When image requests use energy-aware scaling as in Figure 10, the transfer rate is higher compared to the non-adaptive viewer due to adaptation to reduced available energy. The line in Figure 10 represents the amount of data downloaded per image without dynamic scaling of image quality. The bars in Figure 11 highlight the need for Cinder to prevent large-scale hoarding. The configuration is the same except the foreground tap gives 300 mW of power. Because 300 mW is greater than the CPU cost of 137 mW, applications in the foreground can accumulate excess energy.

The system-wide half-life both caps the total energy in the downloader's reserve while the bars represent the transfer rate (KiB) over time. Each batch contains the same number of images, and the average bytes transferred per image is similar size. Over the course of the test, the level of energy present in the reserve decreases since the user pauses after each batch was downloaded. When image download sizes are not scaled back as in Figure 11, the transfer rate decreases with the smaller image data. Pausing between batches until enough energy is available for the reserve at the start of the batch decreases since the user continues to run according to its background power share of 10% utilization. At about 10 seconds, the task manager sets Process A to the background by setting its foreground tap to 0 mW.

The system-wide half-life both caps the total energy in the downloader's reserve while the bars represent the transfer rate (KiB) over time. Each batch contains the same number of images, and the average bytes transferred per image is similar size. Over the course of the test, the level of energy present in the reserve decreases since the user pauses after each batch was downloaded. When image download sizes are not scaled back as in Figure 11, the transfer rate decreases with the smaller image data. Pausing between batches until enough energy is available for the reserve at the start of the batch decreases since the user continues to run according to its background power share of 10% utilization. At about 10 seconds, the task manager sets Process A to the background by setting its foreground tap to 0 mW.

The images were of similar size (2.7 MiB) and each batch contained 50 images. We tracked the energy in the downloader's reserve while the bars represent the transfer rate (KiB) over time. Each batch contained 50 images and the average bytes transferred per image over time. Each batch contained 50 images and the average bytes transferred per image over time. Each successive pause being 5 seconds shorter than the previous one.

Figure 12a and 12b illustrate the behavior of two applications, A and B, under different power conditions. Initially, both applications are in the background. The background tap provides 50% of the power, while the foreground tap provides 50% of the power. After 14 mW, B moves to the background and the background tap provides 80 mW of power. Shortly thereafter, B moves to the background and the background tap provides 80 mW of power. After B exhausts its energy, it returns to its original position in the background and continues to run according to its background power share of 10% utilization. At about 10 seconds, the task manager sets Process A to the foreground by setting its foreground tap to 300 mW of power. Because 300 mW is greater than the CPU cost of 137 mW, applications in the foreground can accumulate excess energy.

The system-wide half-life both caps the total energy in the downloader's reserve while the bars represent the transfer rate (KiB) over time. Each batch contains the same number of images, and the average bytes transferred per image is similar size. Over the course of the test, the level of energy present in the reserve decreases since the user pauses after each batch was downloaded. When image download sizes are not scaled back as in Figure 11, the transfer rate decreases with the smaller image data. Pausing between batches until enough energy is available for the reserve at the start of the batch decreases since the user continues to run according to its background power share of 10% utilization. At about 10 seconds, the task manager sets Process A to the background by setting its foreground tap to 0 mW.

The system-wide half-life both caps the total energy in the downloader's reserve while the bars represent the transfer rate (KiB) over time. Each batch contains the same number of images, and the average bytes transferred per image is similar size. Over the course of the test, the level of energy present in the reserve decreases since the user pauses after each batch was downloaded. When image download sizes are not scaled back as in Figure 11, the transfer rate decreases with the smaller image data. Pausing between batches until enough energy is available for the reserve at the start of the batch decreases since the user continues to run according to its background power share of 10% utilization. At about 10 seconds, the task manager sets Process A to the background by setting its foreground tap to 0 mW.

The system-wide half-life both caps the total energy in the downloader's reserve while the bars represent the transfer rate (KiB) over time. Each batch contains the same number of images, and the average bytes transferred per image is similar size. Over the course of the test, the level of energy present in the reserve decreases since the user pauses after each batch was downloaded. When image download sizes are not scaled back as in Figure 11, the transfer rate decreases with the smaller image data. Pausing between batches until enough energy is available for the reserve at the start of the batch decreases since the user continues to run according to its background power share of 10% utilization. At about 10 seconds, the task manager sets Process A to the background by setting its foreground tap to 0 mW.
Figure 12. Stacked graph of Cinder’s CPU energy accounting estimates as processes A and B spin on the CPU. Together, they are allowed 14 mW while in the background. The task manager runs A in the foreground in the 10 - 20 second interval and B in the foreground during the 30 - 40 second interval. (a) shows the results for the foreground tap providing the process with 137 mW (the precise cost of using the CPU at 100%). (b) shows the foreground tap providing the process with 300 mW. The dotted line shows actual power measurements compensated for baseline power draw with an idle CPU and averaged over 1 second intervals.

Figure 13. Two background applications, a pop3 mail and an RSS fetcher, each poll every sixty seconds. a) Since they are not coordinated, their use of the radio is staggered, resulting in increased power consumption. Each application uses the radio for at most a few seconds, but neither takes advantage of the other having brought the radio out of the low power idle state. b) The same mail and RSS background applications using reserves and limits to coordinate their access to the radio data path. Enough energy is allocated to each application to turn the radio on every two minutes. By pooling their resources, they are able to turn the radio on at most every sixty seconds.

Figure 14. The level of the reserve into which the two background applications transfer their allotted joules. When the reserve reaches a level sufficient to pay for the cost of transitioning the radio to the active state, it is debited, the radio is turned on, and the processes proceed to use the network. Although Figure 4 showed an average 9.5 J cost to power up the radio, netd requires 125% of this level before turning the radio on, essentially mandating that applications have extra energy to transmit and receive subsequent packets. Therefore, the reserve does not empty to 0.

Table 1. Improvements in energy consumption and active radio time using cooperative resource sharing in Cinder. Energy use due to the radio is reduced, resulting in a 12.5% total system power reduction over the 20 minute experiment.
SUMMARY

• token-bucket shaping for energy use
• throttling threads when energy reserve is depleted
• enables energy isolation and controlled delegation
• applications can adapt and pool
Figure 9: Video playback power breakdown. Aggregate power excluding backlight is 453.5 mW.

Figure 10: Power breakdown for sending an SMS. Aggregate power consumed is 302.2 mW, excluding backlight.

The contacts application and selecting a contact, typing and sending a 55-character message, then returning to the home screen; lasting a total of 62 seconds. To ensure the full cost of the GSM transaction is included, we measured power for an additional 20 seconds. The average result of 10 iterations of this benchmark are shown in Figure 10. Again, the power for four backlight brightness levels is shown.

Power consumed is again dominated by the display components. The GSM radio shows an average power of 66.3 ± 20.9 mW, only 7.9 mW greater than idle over the full length of the benchmark, and accounting for 22% of the aggregate power (excluding backlight). All other components showed an RSD of below 3%.

3.3.4 Phone call

Figure 11 shows the power consumption when making a GSM phone call. The benchmark is trace-based, and includes loading the dialer application, dialing a number, and making a 57-second call. The dialed device was configured to automatically accept the call after 10 seconds. Thus, the time spent in the call was approximately 40 seconds, assuming a 7-second connection time. The total benchmark runs for 77 seconds.

GSM power clearly dominates in this benchmark at 832.4 ± 99.0 mW. Backlight is also significant, however note that its average power is lower than in other benchmarks, since Android disables the backlight during the call. The backlight is active for approximately 45% of the total benchmark.

3.3.5 Emailing

For this benchmark, we used Android's email application to measure the cost of sending and receiving emails. The workload consisted of opening the email application, downloading and reading 5 emails (one of which included a 60 KiB image) and replying to 2 of them. The results of the benchmark are shown in Figure 12, averaged over 10 iterations.

The power breakdown between the GPRS and WiFi

Figure 12: Power consumption for the email macro-benchmark. Aggregate power consumption (excluding backlight) is 610.0 mW over GPRS, and 432.4 mW for WiFi.
Figure 9: Video playback power breakdown. Aggregate power excluding backlight is 453.5 mW.

Figure 10: Power breakdown for sending an SMS. Aggregate power consumed is 302.2 mW, excluding backlight.

The contacts application and selecting a contact, typing and sending a 55-character message, then returning to the home screen; lasting a total of 62 seconds. To ensure the full cost of the GSM transaction is included, we measured power for an additional 20 seconds. The average result of 10 iterations of this benchmark are shown in Figure 10. Again, the power for four backlight brightness levels is shown.

Power consumed is again dominated by the display components. The GSM radio shows an average power of 66.3 ± 20.9 mW, only 7.9 mW greater than idle over the full length of the benchmark, and accounting for 22% of the aggregate power (excluding backlight). All other components showed an RSD of below 3%.

3.3.4 Phone call

Figure 11 shows the power consumption when making a GSM phone call. The benchmark is trace-based, and includes loading the dialer application, dialing a number, and making a 57-second call. The dialled device was configured to automatically accept the call after 10 seconds. Thus, the time spent in the call was approximately 40 seconds, assuming a 7-second connection time. The total benchmark runs for 77 seconds.

GSM power clearly dominates in this benchmark at 832.4 ± 99.0 mW. Backlight is also significant, however note that its average power is lower than in other benchmarks, since Android disables the backlight during the call. The backlight is active for approximately 45% of the total benchmark.

3.3.5 Emailing

For this benchmark, we used Android's email application to measure the cost of sending and receiving emails. The workload consisted of opening the email application, downloading and reading 5 emails (one of which included a 60 KiB image) and replying to 2 of them. The results of the benchmark are shown in Figure 12, averaged over 10 iterations.

The power breakdown between the GPRS and WiFi

The power consumed is dominated by the display components. The GSM radio shows an average power of 66.3 ± 20.9 mW, only 7.9 mW greater than idle over the full length of the benchmark, and accounting for 22% of the aggregate power (excluding backlight). All other components showed an RSD of below 3%.
DISCUSSION

• interesting mechanism, but does it allow useful management?

• thinking inside the box: energy should be like CPU time
  • hard: deadlines are inherent, energy cap is not

• thinking outside the box: system does only useful work
  • efficiency: order requests to use resource better
  • adaptivity: quality-resource tradeoff