Operating System Energy Accounting

Andreas Mosthaf
Friedrich-Alexander University Erlangen-Nuremberg
andreas.mosthaf@e-technik.stud.uni-erlangen.de

ABSTRACT

Besides performance, energy is an important issue for operating systems—especially when they are running on a mobile platform. Targeting an optimal lifetime of the system battery, operating systems have to provide mechanisms for energy accounting and energy controlling. This document describes the prerequisites for an effective energy management. It also introduces ECOSystem and the Cinder operating system, two implementations of an energy-aware operating system, along with applications developed for these systems that demonstrate the benefits of energy management on operating systems.

1. INTRODUCTION

Since mobile phones—the dominating mobile systems on the market—are capable of running general-purpose operating systems like Linux, there is a great demand for making operating systems more energy-aware. Improvements have been made to make energy consumption more visible by means of smart device interfaces. In addition, modern devices provide a lot of energy-saving modes to avoid wasting energy. These improvements allow operating systems a coarse control of the platforms' energy behavior.

Energy management requires energy accounting on the one side and energy controlling on the other side. This document describes the prerequisites for an effective energy management on operating systems. An overview over methods and techniques, providing relevant parameters concerning energy resources, is given in Section 2. Section 3 then describes the required power model and control mechanisms. Section 4 and Section 5 contain a brief introduction to ECOSystem [5] and the Cinder operating system [3]—two implementations of energy aware operating systems. Thereafter, Section 6 describes three applications, developed on the Cinder operating system, demonstrating the benefits of an energy aware operating system in typical use cases.

2. ENERGY VISIBILITY

Energy management on operating system level requires at first precise information about energy consumption of the managed devices on the one hand and characteristics of the battery on the other hand. Although modern hardware components may provide interfaces to retrieve the relevant parameters, in most cases measurements are necessary to get accurate results.

Due to the closed nature of mobile systems like smartphones, measuring the power consumption of individual functional units is difficult. In such cases measuring the total system power and estimating the consumption of single components will lead to inaccurate results. However, there are mechanisms that allow operating systems to work around this kind of error.

Discharge Rate	Usable Battery Capacity (normalized to 1C discharge rate)
C/5	107 %
C/2	104 %
C	100 %
2C	94 %
4C	86 %

Table 1: Characteristics of a lithium-ion battery, adapted from [1].

2.1 Battery Characteristics

Battery lifetime is an important factor for mobile systems and therefore it is a primary target of energy management to maximize it. The discharge rate of a battery has a nonlinear impact on the usable battery capacity. High discharge rates can lower the capacity to 70 %–80 %. Table 1 shows the characteristics of a typical lithium-ion battery.

The Smart Battery interface in the ACPI specification [2] allows operating systems to query all relevant values like current voltage, remaining capacity, and discharge rate. Unfortunately, queries to this interface are slow and return only averaged values of power consumption.

2.2 Power Consumption Measurement

To measure the power consumption of managed devices one has to consider the various states a device could reach. The consumption of a processor, for example, depends on the usage of the built-in functional units and therefore on the current workload. Additionally modern devices offer power saving modes we also have to consider. The transition between the different modes of such devices is also a source of energy consumption that has to be taken into account.

Table 2 shows the measured energy consumption of an IBM Travelstar 12GN harddisk. Spinup, spindown and accessing a block on the disk drains a fixed amount of energy, whereas energy consumption in one of the three idle states depends on the time the harddisk spends in the respective state.

State	Cost	Time Out (Sec)
Access	1.65 mJ/Block	
Idle 1	1600 mW	0.5
Idle 2	650 mW	2
Idle 3	400 mW	27.5
Standby (disk down)	0 mW	
Spinup	6000 mJ	
Spindown	6000 mJ	

Table 2: Measured power state values and time-out values of an IBM Travelstar 12GN hard disk, adapted from [3]

2.3 Energy Profiling

To provide precise energy accounting we have to assign the measured energy consumption to the application that causes it. A simple approach is to sample power consumption and to assign the sample values to the thread currently using the processor. The results will be inaccurate due to the fact that simultaneously active hardware components could have been triggered by two different applications.

A more precise method to account energy online makes use of performance counters embedded in modern processors. Performance counters are implemented as hardware registers. By registering events that imply the consumption of a certain amount of energy at these counters, the operating system is able to change its behavior and therefore to adapt the expected power drain. For example, a high number of main memory references, although only a little number of instructions were executed, could indicate that performance is dominated by the main memory latency. Therefore, throttling the processor speed will improve the energy efficiency without a significant negative impact on system performance.

3. ENERGY CONTROL

Targeting a given battery lifetime means a limitation of the discharge rate of the battery, shown in Table 1. Based on the information provided by hardware measurements, energy-aware operating systems must be able to adapt their scheduling so that the systems' overall power consumption never exceeds the demanded discharge rate. To provide energy-aware scheduling, mechanisms for accounting and distribution of available energy resources among all competing tasks are required.

3.1 Energy Model

A prerequisite for energy accounting is an abstraction for the resource energy. Such an abstraction is provided by the *Currentcy Model* [5]. The model uses a common unit to account and allocate energy. The word currentcy is made up of the words currency and current since this unit is used to pay for a certain amount of current. As the operating system targets a certain consumption rate, one unit represents the right to consume a certain amount of energy in a fixed time interval.

3.2 Control Mechanisms

Energy management on operating systems requires mechanisms for accurate accounting and effective distribution of the resource energy. The following subsections describe three basic mechanisms: isolation, delegation, and subdivision.

3.2.1 Isolation

Isolation is an important security mechanism of process and memory management in operating systems. Since applications should not be able to drain energy from other applications, isolation is therefore an important factor for energy-management, too.

3.2.2 Delegation

Delegation allows a task to loan its available energy to other tasks. Therefore this is an important mechanism of inter-application cooperation. Donor and recipient of a delegation are both able to consume the delegated resource. By delegating resources from more than one donor to a single recipient, applications are able to contribute to expensive operations.

3.2.3 Subdivision

Subdivision allows a task the partitioning of its available energy. Combined with delegation, subdivision enables tasks to loan parts of their energy to other tasks. This is especially important for applications with child processes. In this case the main task might want to be assured that child processes are not able to starve other components of the application.

4. ECOSYSTEM

ECOSystem is an implementation of the currentcy model, based on the Linux operating system. For handling energy as a first class resource, the kernel was modified to provide mechanisms for allocating and accounting energy. This includes a resource container object for the storage of energy units. The prototype of ECOSystem was evaluated on an IBM Thinkpad T20 laptop with a set of microbenchmarks targeting the CPU, the disk, and the network interface.

4.1 Currentcy Allocation

ECOSystem provides an interface to allow the user to set a target battery lifetime. The maximum discharge rate for the battery depends on the targeted lifetime, as shown in Section 1.1. The difference between the current and the maximum discharge rate determines the amount of energy that can be allocated in a fixed time interval. This value corresponds to an amount of energy units that can be distributed between all competing tasks. The distribution of energy is performed by a periodic kernel thread that modifies the values of all tasks' resource containers periodically. If a task does not use all of its energy units, it can accumulate these units up to a certain limit.

4.2 Currentcy Accounting

Performing system operations, like the execution of an instruction or requesting data from a disk, requires a certain amount of energy. If a task wants to perform such an operation, the operating system checks the corresponding energy value. If the value is sufficiently high, the operation will be executed. Otherwise, the task is blocked until it accumulated enough energy units for the requested operation.

Currentcy Model

App	CPU (mJ)	HD (mJ)	Net (mJ)	Total (mJ)
DiskW	430	339,319	0	339,749
NetRecv	256,571	0	553,838	810,409
Compute	8,236,729	0	0	8,236,729

Program Counter Sampling

App	CPU (mJ)	HD (mJ)	Net (mJ)	Total (mJ)
DiskW	430	16	24	470
NetRecv	256,571	9,235	20,206	286,012
Compute	8,236,729	326,404	531,789	9,094,922

Table 3: Energy accounting: currentcy model vs. program counter sampling, adapted from [5].

Due to the numerous, different energy characteristics of managed devices, ECOSystem uses energy charging policies with respect to these differences:

- CPU: Execution of instructions on the processor is charged in dependence of a fixed power value and the processing time. A task running out of energy units will be preempted until it accumulates more units.
- Hard Disk: In addition to a fixed amount of energy for every block accessed, all tasks that accessed the disk within the same session share the costs for spinning up and down the disk.
- Network Interface: Sending or receiving of data packets is charged in dependence of the transmit power, the size of the packet, and the bitrate.

Assigning the costs of CPU operations to the correct task is quite simple, whereas tracking the energy consumption of the disk or the network interface is more complex. Files in ECOSystem are accessed through file related system calls. Hereby the container ID of the calling task is stored within the buffer cache entry of the disk. When the cache entry is actually written, the appropriate resource container will be charged for the operation. Source tasks of network operations are identified by their associated source socket.

4.3 Energy Accounting

The result of an evaluation of the effectiveness of energy accounting implemented in ECOSystem is shown in Table 3. In this experiment three benchmarks were run simultaneously on the system, each addressing a separate functional unit. DiskW writes 4 KB of data every four seconds to the disk, NetRecv receives continuously data from the network interface at the highest bitrate, and Compute is a batch job running continuously on the processor. For comparison Table 3 shows the results of a similar run using a program counter sampling technique.

ECOSystem assigns energy consumption to the appropriate application, whereas the results of the program counter sampling show significant accounting errors. With program counter sampling, energy consumption is measured periodically and the values are assigned to the task currently running on the CPU. The error in Table 3 arises from device operations currently being executed but triggered by a task that it is not running.

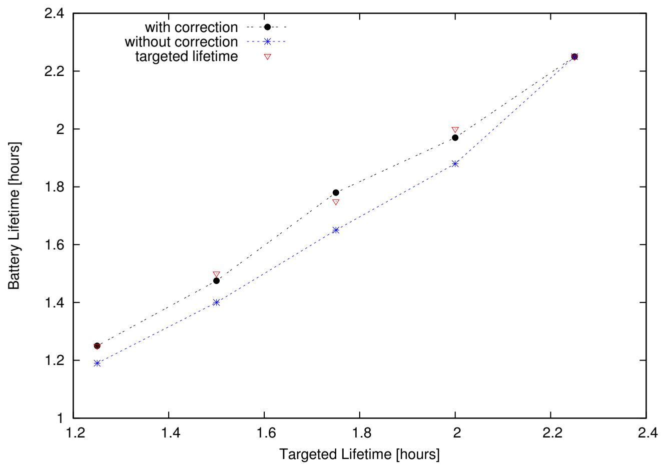

Figure 1: Targeted battery lifetime compared to the measured lifetime, adapted from [5].

4.4 Battery Lifetime

The primary design objective for the development of ECOSystem was to achieve a targeted battery lifetime. As mentioned in Section 2, offline measurements of closed mobile systems could be inaccurate. This experiment shows a possible impact of such an error on the battery lifetime.

In this experiment the system runs a CPU-intensive microbenchmark. Additionally an accounting error is modeled by decreasing the power consumption value for the usage of the CPU. The usage of the CPU is now cheaper than it should be and will lead to a shorter lifetime of the battery. Figure 1 shows the achieved lifetime of the system battery in comparison to the targeted lifetime.

To handle such errors the operating system can make use of the Smart Battery interface. By periodically checking the remaining capacity of the battery, the discharge rate of the battery and hence the amount of allocated energy can be dynamically adapted.

4.5 Energy Sharing

This experiment demonstrates that ECOSystem is able to manage energy sharing between simultaneously running applications. One of the applications is ijpeg, a CPU-intensive graphical application. The other application is netscape, which is mainly using the network interface. A performance indicator for ijpeg is the computation speed and therefore its CPU utilization. The performance of network applications like netscape depends on response times and therefore on page load latencies. Table 4 shows the measurement results for various energy ratios between these two applications, whereby the amount of allocated energy for both applications is set to 5 W.

The measurements show that both tasks match their targeted allocation. Although netscape makes use of all three functional units, ECOSystem tracks the power consumption across all devices accurately. The CPU utilization of ijpeg, and therefore its performance, increases nearly equivalently to the allocated power, whereas netscape shows a non-linear behavior due to non-linear energy characteristics of the network interface.

a)

Energy Share	Power Alloc (W)	Ave Power Used (W)	CPU Util (%)
70%:30%	3.5	3.507	22.55%
60%:40%	3.0	3.008	19.34%
50%:50%	2.5	2.500	16.08%
40%:60%	2.0	2.008	12.91%
30%:70%	1.5	1.503	9.67%
20%:80%	1.0	1.005	6.46%

b)

Energy Share	Power Alloc (W)	Ave Power Used (W)	Page Delay (s)
70%:30%	1.5	1.49	29.205
60%:40%	2.0	2.006	17.441
50%:50%	2.5	2.457	9.928
40%:60%	3.0	2.961	6.322
30%:70%	3.5	3.443	3.934
20%:80%	4.0	3.663	3.032

Table 4: Results of energy sharing between ijpeg (a) and netscape (b), adapted from [5].

5. CINDER OPERATING SYSTEM

Cinder is an operating system designed for modern mobile phones. It extends HiStar [4], a secure operating system that provides *containers* and *gates* in addition to conventional kernel objects.

Every object in HiStar has to be referenced by a container including containers ifself. The hierarchical structure of containers is used by the Cinder operating system to deallocate resources and therefore to implement a mechanism for delegating and subdividing resources. Gates provide secure inter-process communication by using security labels for all objects. This security concept of HiStar ensures isolation of containers.

The Cinder operating system extends HiStar with two additional kernel objects: *reserves* and *taps*. These two extensions to the kernel are explained in the following paragraphs.

5.1 Reserves

Reserves are kernel objects that permit the usage of energy. To determine the amount of energy a task is allowed to use, a power model similar to the currentcy model is used. The value of a reserve is therefore a permission to use a certain amount of energy in a fixed time interval. If the value of a reserve is not high enough for the operation the corresponding task wants to perform, the kernel will prevent execution.

Although it is possible to partition a reserve's value and assign it to other reserves, this is in most cases not practical, because threads rarely need to delegate fixed amounts of energy. Usually threads provide other threads with energy by using a fixed rate.

5.2 Taps

In order to guarantee a certain battery lifetime the discharge rate of the battery has to be limited. Transferring energy between reserves with a fixed rate is therefore a better approach than delegating fixed amounts of energy.

Taps are kernel objects that transfer a certain quantity of energy between two reserves with a fixed rate. Therefore a tap contains a rate, a source reserve, and a sink reserve. Instead of transferring a fixed amount of energy, proportional taps transfer a certain fraction of the source reserves' energy to the sink reserve. In practice, taps are implemented as a thread whose job is to transfer energy values between corresponding reserves periodically.

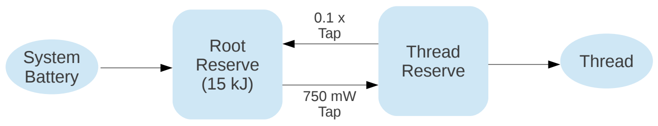

Figure 2: Consumption graph, adapted from [3].

5.3 Energy Consumption Graph

Due to the hierarchical structure of reserves and taps, it is suitable to model it with a directed graph. The root of such a directed graph represents the reserve connected to the system battery; all other reserves are a subdivision of this root reserve. A simple example of such a directed graph is shown in Figure 2. In this example the root reserve is connected to the 15 kJ system battery. A single application draws energy from a separate reserve connected to the root reserve over a 750 mW tap.

5.4 Energy Hoarding

Tasks can accumulate unused amounts of energy to use it for future operations. However, to improve the performance of the whole system, it is good practice to reclaim unused energy and assign it to other tasks. This problem can be solved by using backward taps. A backward tap transfers a fraction of the accumulated energy back to the source reserve. The fraction of the backward tap is also a limitation for the amount of energy the sink reserve is able to hoard. In Figure 2, a backward tap transfers 10 % of the reserve's accumulated energy back to the root reserve. When the sink reserve level in the example reaches 700 mW, both taps have the same rate and therefore the level cannot raise anymore.

6. APPLICATIONS

Improving the battery lifetime of a mobile system is just one goal of energy management. Another target is the performance the user expects. A common use case of modern smartphones is to play music while browsing the internet. In this case user experience depends on the delay of network requests and a disruption-free playback of the music file. Although a user may accept varying delays, a disruption of the playback will have an unacceptable impact on his experience.

The following subsections describe applications that were developed on the Cinder operating system using reserves and taps. They represent typical use cases operating systems have to handle and will show the ability of this operating system to control energy on mobile systems.

The platform on which Cinder is implemented and the following applications are evaluated is an HTC Dream, which is also known as Google G1. Because of its closed, integrated structure measuring becomes a challenging task. Unfortunately the ARM11 core processor provides no performance

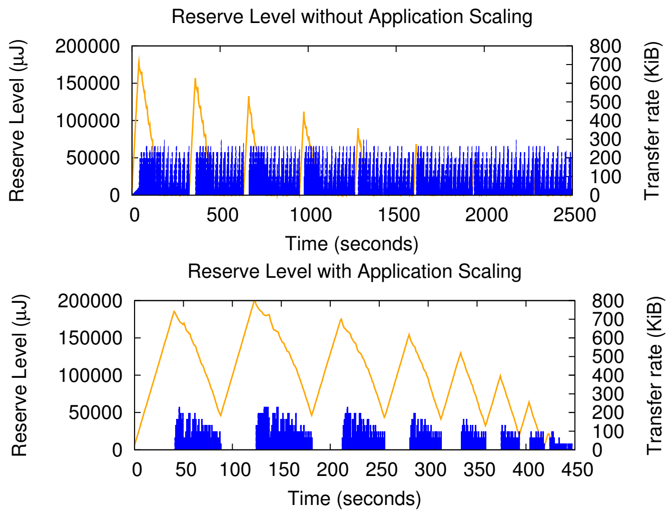

Figure 3: The results of the image viewer application, adapted from [3].

counters. Therefore energy accounting is based on offline measurements of device power states.

6.1 Energy Aware Applications

Reserves and taps allow developers fine-grained control of resources. Adapting the image quality of a graphical application to a given energy level is a good example for an energy aware application.

The application described here is a network picture gallery. A thread separate from the main thread with a separate reserve handles the downloading of pictures. The rate of the tap between the reserves of the main application and the download thread depends on the frequency of image requests and the size of the pictures. By periodically checking the resources' energy level the application is able to adapt its behavior if the downloader is consuming energy too quickly. In this case the application requests only partial data from the interlaced PNG images.

Figure 3 shows the results of test runs with and without energy aware scaling where a batch of image requests is made periodically. The line represents the energy level of the downloader thread's reserve, and the bars show the amount of data downloaded per image. A pause between the batch requests allows the reserve of the downloader to accumulate energy when using scaling. Without scaling the transfer rate stays constant until the reserve runs out. This leads to a disruption of the downloader until the reserve has accumulated enough energy again. Therefore this example makes a trade-off between constant transfer rates and picture quality.

6.2 Background Applications

Background applications are another challenge for energy management. Despite being invisible for the user they may interfere with foreground applications and could therefore have a negative impact on user experience.

The example in this section models an RSS reader and a mail client. Each application has a reserve connected to a

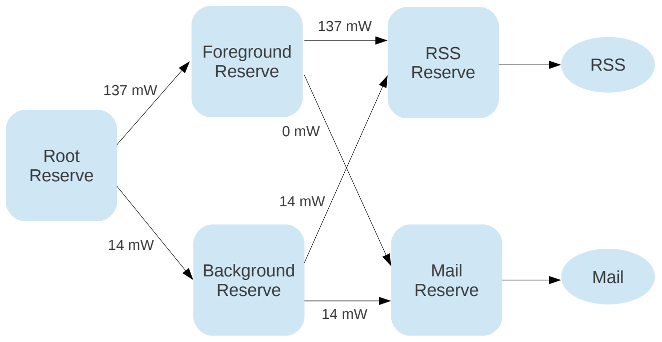

Figure 4: The energy consumption graph for the background application, adapted from [3].

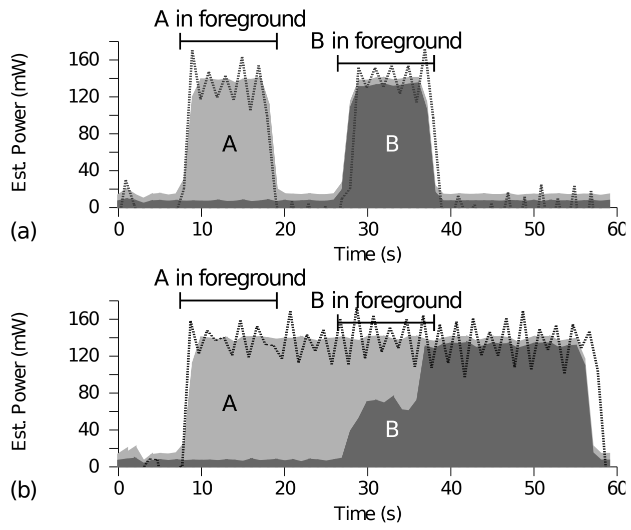

Figure 5: The results of test runs with two tasks spinning from foreground to background, adapted from [3].

foreground and a background reserve, as shown in Figure 4. The taps to the background reserve have a constant small rate of 14 mW for both applications. The rates of the taps to the foreground reserve depend on the current state of a task. The task manager sets the foreground task tap to a rate of 137 mW, which is the power the processor requires to run. The rate of the background task is set to 0 mW and therefore, the background task is not able to run the processor until it accumulates enough energy from the background reserve.

Figure 5a shows the results of a test run. The dark areas in Figure 5 show the power estimation for the two running tasks and the dotted line is the result of an actual power measurement. Task A runs in the foreground in the 10 s till 20 s interval, task B between 30 s and 40 s. Figure 5b shows a test run with a higher foreground rate of 300 mW. With the rate exceeding the required processor power, task A is able to accumulate the unused energy. Therefore task A has still enough energy to use the processor although set to the background and interferes with task B until energy runs out after 40 s.

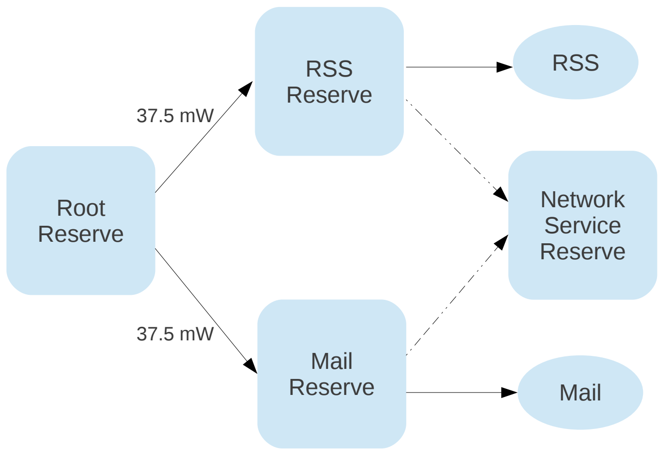

Figure 6: The energy consumption graph for the network application, adapted from [3].

6.3 Network Applications

Network interfaces are very expensive regarding their energy consumption. This is aggravated by the fact that they have a non-linear energy characteristic. Initial activation costs of a radio interface are very high but can be compensated through bulk transfers. A common use case are background applications that make periodic use of the network interface.

This example describes the network access of two applications over a network daemon, shown in Figure 6. Both applications should make a poll every 60 seconds. Giving each task enough energy to activate the network daemon every 60 seconds leads to the result shown in Figure 7a. Polling uses the network interface just for a few seconds. The time between the polls of the two different applications is therefore sufficient for the network interface to switch back to its idle mode and both tasks have to pay the full activation costs for their polls.

A better approach is to implement the network daemon with its own reserve to which the polling tasks can delegate their energy to. To activate the network interface, the sum of the reserves' energy of the calling task and the network daemon must reach the activation costs. If the sum is not high enough, the calling task delegates its energy to the daemon's reserve and blocks until enough energy is accumulated. Both of the polling tasks now only get enough energy to activate the network interface if they work in unison. The result of this implementation is shown in Figure 7b. Both tasks are still polling every 60 seconds but now the network interface is activated only once in this period which leads to a significantly decreasing overall energy consumption.

7. CONCLUSION

The performance of a mobile platform depends highly on the lifetime of the system battery. Improving energy management is therefore an important topic of research, especially considering that energy is a more and more limited and expensive resource.

Abstracting energy to a first class resource and providing mechanisms to isolate, delegate, and subdivide it enables operating systems to manage energy on a fine-grained level. The evaluation of the presented applications developed on

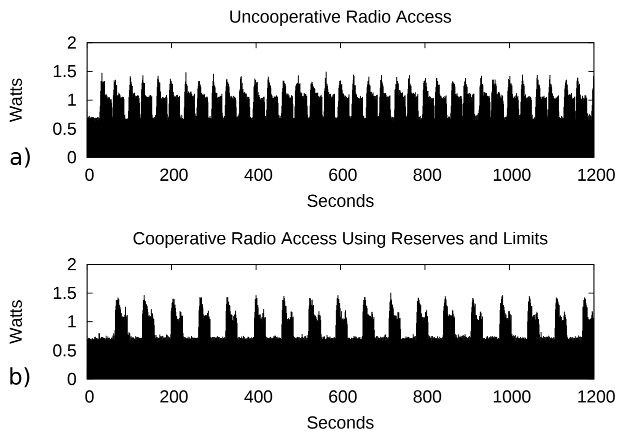

Figure 7: The results of test runs with (a) and without (b) a coordinated network stack, adapted from [3].

the Cinder operating system and ECOSystem shows that these systems are able to reach a given battery lifetime. Furthermore, these systems allow developers to implement energy aware applications that will improve the user experience by adapting the applications' behavior to the current energy level.

However, it is uncertain if this approach improves the overall energy efficiency, since there were no measurements made to compare energy consumption, performance and battery lifetime of the evaluated systems with standard systems. Providing the ability to account and control energy on such a fine-grained and accurate level requires a lot of additional computation time that causes higher energy consumption and a lower system performance. Consequently, it is doubtful if managing energy as a first class resource is a promising approach to improve energy efficiency on mobile platforms.

8. REFERENCES

[1] Bellosa, F. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the 9th ACM SIGOPS European Workshop* (2000), pp. 37–42.

[2] Intel Corporation and Microsoft Corporation and Toshiba Corporation. *Advanced Configuration and Power Interface Specification.* http://www.teleport.com/acpi, 1996.

[3] Roy, A., Rumble, S. M., Stutsman, R., Levis, P., Mazieres, D., and Zeldovich, N. Energy management in mobile devices with the Cinder operating system. In *Proceedings of the European Conference of Computer Systems 2011* (2011), pp. 139–152.

[4] Zeldovich, N., Boyd-Wickizer, S., Kohler, E., and Mazieres, D. Making information flow explicit in HiStar. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation* (2000), pp. 263–278.

[5] Zeng, H., Ellis, C. S., Lebeck, A. R., and Vahdat, A. ECOSystem: Managing energy as a first class operating system resource. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems* (2003), pp. 123–132.