Resource Isolation in Terms of Power and Energy Consumption

Maximilian Ott
Friedrich-Alexander University Erlangen-Nürnberg (FAU)
maximilian.o.ott@fau.de

ABSTRACT

The benefits of power-aware applications are generally recognized. By adapting to the current power consumption, apps are able to optimize their runtime behavior. Today, operating systems provide interfaces for apps to query such information, however, those approaches suffer mostly from two drawbacks: Firstly, the accounted power consumption for one process or process group might be distorted due to power entanglement. Secondly, those approaches do not offer any isolation between different processes in terms of power as a resource. To address those problems, this work proposes the combination of power sandboxing [5], a counter mechanism for power entanglement, and tabs/reserves [11], an OS-level abstraction of power consumption.

1 INTRODUCTION

The increasing demand of power-aware applications, especially on mobile devices, requires operating system developers to add or adapt interfaces in order to provide accurate measurements in a secure manner. In the course of this, two fundamentally different approaches of power metering have emerged: Firstly, operating systems can use power models to approximate their power consumption. Hereby, a power model can be viewed as a function of the estimated power consumption for a given period of time [3]. The second approach uses hardware sensors directly and reports the actual measured data [4].

While both power-metering mechanism will produce valid data, which indeed can be already used by the power-aware applications, the actual reasoning of those may often be rather difficult. This is mainly due to the fact that the data is computed or measured for a whole subsystem or even for the whole system and then accounted to the running threads according to chosen heuristics [2]. Therefore, it is not possible to directly provide fine-grained observation for a process or process group. Furthermore, those approaches might lead to a serious security vulnerabilities known as power side-channel attacks. A malicious application is able to constantly poll the system power meter and the reported data will give a hint about the power consumption of a simultaneously running application. It is demonstrated that a malicious background application can infer the current location information on mobile devices by exploiting the collected power meter data [8]. To address those problems, power observations can be sandboxed as proposed in [5].

The vertical environment, as shown in Figure 1, provides an abstraction of the measured power consumption by virtualizing the underlying subsystem. Based on memory balloons of virtual machines, resource balloons are introduced as a mechanism to provide exclusive access to a subsystem and therefore to offer power metering without any entanglement.

However, power sandboxing does not prevent malicious apps from draining excessive amounts of the battery capacity. To increase the battery runtime, especially for mobile devices, another regulation mechanism must be added. One possible approach towards power consumption regulation would be the usage of reserves and taps which were first introduced in [11]. A reserve can be compared to a water reservoir but instead of storing water it will keep track of the available energy. Whereas, a tap will provide an abstraction of the actual provided power. Based on these two mechanism, isolation, delegation and subdivision can be achieved.

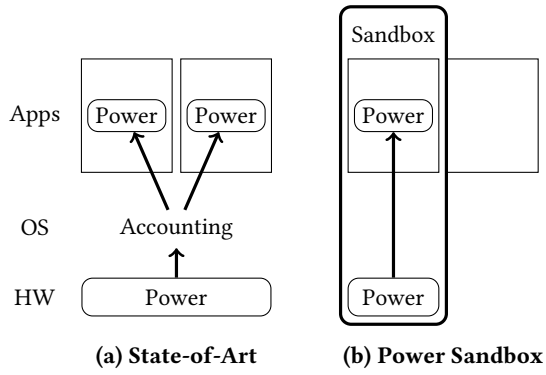


Figure 1: Vertical environment. (a) The power consumption is measured for a whole subsystem and accounted to different apps. (b) The sandbox allows measurements of a specific application by providing an exclusive vertical environment.

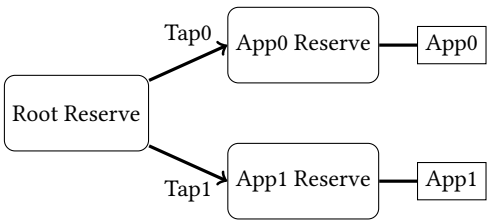


Figure 2: Taps and Reserves. Each application was granted a specific amount of usable energy in form of a reserve. These reserves can be refilled using taps.

As shown in Figure 2, each process or process group will utilize their own reserve and drain power from it. Therefore, their power consumption only depends on their own reserve and isolation is achieved. To fill a reserve, the available energy of another reserve can be subdivided and delegated.

This paper contributes the combination of two already existing operating system abstractions, namely power sandboxing [5] and taps/reserves [11], to provide fine-grained isolation in terms of power metering and power consumption for independent processes and process groups.

2 A CASE FOR OS ABSTRACTIONS

This section examines the typical setup of power-aware applications and specifies requirements for the proposed operating system abstractions.

2.1 Power-Aware Applications

Especially on mobile devices, power-aware applications benefit from fine-grained power metering as it provides key insights of possible optimizations [7]. The typical setup of such optimized apps can be illustrated as shown in Figure 3.

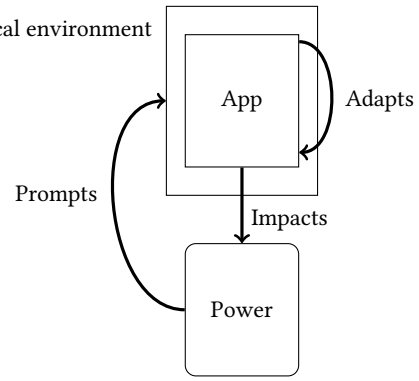


Figure 3: Closed Control Loop

While the applications is running, power is consumed. Depending on the load of the task at a given moment, the relative power consumption might increase or decrease. To adapt to the situation, the application will query the current power consumption from the vertical environment, which in turn gets those information from the system power meter. As a result, the applications might change content fidelity [9], algorithms accuracy [6], application preferences [1] or network scheduling strategy [10]. Due to the changed behavior of the application, the power consumption will change and the control loop is closing.

2.2 Power Entanglement

As motivated in Section 1, the reported power consumption might be distorted due to power entanglement. The main source of entanglement is the concurrent usage of a metered subsystem by several independent threads. Hereby, power entanglement can be classified in two groups:

The first kind of power entanglement concerns spatial concurrency in the hardware. The most familiarly example of this category are multicore CPUs with a shared power supply as multiple independent threads can run concurrently but the overall power can only be metered as a whole [15]. This might not only make reasoning more difficult but also introduces potential threats of side-channel attacks.

The second source of power entanglement corresponds to blurry request boundaries. Commonly, IO devices offer queues in order to receive requests from the CPU. Those requests are handled by the device asynchronously later on. Therefore, it is not possible to pinpoint the exact start and end of a request. If several independent threads submit requests concurrently, the accounting of the spent execution time and the associated fine-grained power consumption is rather imprecise.

2.3 Global Power Management

With this background knowledge, this section discusses to actual setup of power management in a global scope. The goal of the operating system in this work is about providing a secure and fair environment for each process or process group in terms of power consumption.

Following this requirement, an isolated view of the current available energy is needed. The operating system will assign at least one reserve to each process or process group independently and guarantees that it is not possible to drain energy from a different reserve without sufficient permissions. Due to its fine-grained nature, the design of reserves differs from previous approaches [14]. This leads to the second requirement as energy, or rather power, should be allowed to be delegated between two reserve while maintaining all necessary security policies. Taps will be the base for this power transfer if all permission checks are passed. The setup has one last implicit requirement: Currently, energy is stored in one's own isolated reserve and power can be delegated from a reserve to another but a process must be able to subdivide their own resource before delegating to another process or process group.

3 DESIGN

To provide an accurate and secure environment for power-aware applications, the following key characteristics will form the basis for the design in this work:

(1) Isolated Power Metering: The vertical environment should provide accurate and reasonable measurements without the exposure of any power-related security flaws.
(2) Dynamic Power Sandboxing: The application itself should be allow to choose its environment. This means in particular to enter or leave power sandbox freely.
(3) Tracking of available Energy and Costs: The operating system should keep track of available energy resources for the managed processes or process groups.
(4) Ensuring Boundaries: The operating systems should take action if applications exceed their delegated resource amount.

3.1 Power Sandboxing

The design of the power sandbox is based on resource balloons. Hereby, the operating system will allocate a portion of resource which will be granted one process or process group exclusively. Therefore, the concurrent usage of a subsystem by independent threads can be confined. As the final result, power entanglement is restricted and accurate/reasonable measurements are made possible. Based on Section 2.2, the power sandbox must distinguish between two fundamental types of resource balloons:

(1) Spatial balloons: Those will handle concurrency in subsystem which can be used by several threads simultaneously and are schedulable by the operating system. For instance, spatial balloons are able to confine usage of several CPU cores at the same time.

(2) Temporal balloons: This kind of resource balloon will confine asynchronous request of IO device by several independent entities. The device may only process one request at the moment but there are almost no constraints on the order of submitted requests by different threads. Therefore, the operating system must ensure that there are no in-flight requests by another process while the resource granted exclusively to an entity.

Both kind of resource balloons confine concurrency within a subsystem, which was identified as the main source of power entanglement. As a result, isolated power metering (per process or process group) was made possible while reducing the attack surface. The first key characteristic of the overall design (see Section 3), is met.

Concerning the second property, the power sandbox is designed as an on-demand service. All apps are allow to enter or leave their power sandbox freely:

```
1 // Create a power sandbox
2 box = psbox_create(HW_CPU); // optional argument
3 box_enter(box);
4 // Collection of power samples
5 psbox_sample(box, &buf, NUM_SAMPLES);
6 psbox_leave(box);
```

Listing 1: Usage of the power sandbox. This example code was taken from [5].

If the application wants to observe its own power consumption, it will enter the power sandbox and have exclusive access to the metered devices (line 3 of Listing 1). Hereby, the application can select the demanded subsystem optionally. After reading the collected data (line 5), the app will leave the power sandbox for full execution speed (line 6). To keep fairness among the applications, the lost sharing opportunities due to the exclusive access will be fined to the sandboxed applications in future scheduling decisions. Therefore, the overhead of the power sandbox should not be noticeable for the remaining applications.

3.2 Reserves and Taps

Reserves form the basis for isolation in terms of controlled power consumption. By storing the current amount of usable resources (in this case energy), it serves as one important parameter for scheduling decisions. In particular, an operating system, which supports the reserves/taps mechanism, will only schedule a thread of a process or process group if the reserve yields enough resource for this operation. This implies that the scheduler won't select a thread with an empty reserve as the next running thread. While rescheduling, the used resources will be accounted and the associated value of reserve will be decreased.

To refill the balance of a process or process group, the reserve (most often the battery of the mobile device) can be subdivided and the newly created reserve can be delegated to the corresponding process or process group. For example, if the mobile device has a 15kJ battery, it may split a 100mJ reserve from it and delegate it to the process. However, this fixed-quantity approach is rather unhandy and unnatural. Instead, a rate of energy (i.e. power) makes more sense. For instance, if the same mobile device should be usable for at least 5 hours, the reserve should be recharged with an rate of 0.75J/s. For this occasion, a tap can be used. A tap can be described as triple consisting of a source, destination and a given rate. Combining reserves and taps, energy can be stored, used and restored. Noteworthy, reserves and taps exist in the kernel space and are treated like every other control structure. Therefore, changes of those by the user space require sufficient permissions.

Reserves and tabs provide basic isolation between processes in terms of power/energy consumption as every process or process group has only access to their own reserves and taps. However, those two alone will not prevent applications from hoarding energy. An application might idle most of the time and its reserve will be almost always completely filled. As a result, the saved energy is no longer available to the rest of the system and therefore wasted. One possible counter measurement includes the usage of proportional backward tabs. Instead of transferring energy described by a fixed rate, a fraction from the source reserve can be reallocated.

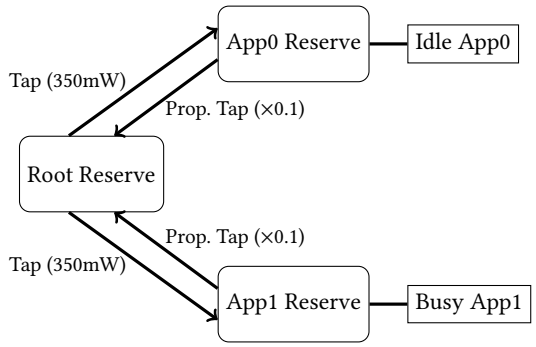


Figure 4: Taps and Backward Taps. The taps will refill the reserve of App 0 and App 1 with a fixed rate of 350mW while the proportional backward tap will tax 10% of each reserve.

As shown in Figure 4, a tenth of the current reserve resource count will be taxed and reclaimed for the overall system. Therefore, an energy-hoarding application will be penalized with a larger tax in quantitative terms. Alternatively, a more aggressive approach can be used. Hereby, the reserve resource amount can be halved and reallocated after a predefined period of idle time. This approach is similar to the "idle memory tax" of virtual machines [12].

Finally, as each process or process group utilize their own reserves and energy hoarding is confined, isolation in terms of power consumption can be achieved. Also, the operating system ensures that the amount of delegated resources is not exceeded. Therefore, the third and fourth requirement of Section 3 is fulfilled.

3.3 Combined approach

The overall design goal is about providing an accurate and reasonable environment for power-aware applications while ensuring process isolation in terms of energy as a system resource. In addition, user space applications must be protected from power-related security threats. We build our design starting with the described reserves/taps mechanism as the foundation. Most of the previously done work remains unchanged. The used reserve(s) and their associated amount of delegated energy are still the final step of the

scheduling decision and prevent threads with an insufficient energy balance from running. Hereby, two adaptions must be made:

(1) Adapted Power Accounting: Depending on the context, an application may be sandboxed or not. Therefore, the accounting mechanism of the reserve must be modified in order to handle both input methods correctly.
(2) Penalties due to Lost Sharing Opportunities: As the original scheduling mechanism of reserves are not aware of the sandboxes environment and their overhead due to exclusive device access, the penalties for the sake of fairness must be accounted as well.

While adding the power sandboxing mechanism upon the reserves/taps abstraction, the kernel interface must be slightly extended. Now, not only the user space will utilize the metered data but also the kernel will use them. Otherwise, the basic idea of power sandbox and the user-space interface remains the same. Depending on the usage state of the sandbox by the executed thread, the reserve accounting mechanism must be able to deal with either the sandboxed data or the plain measurements with potential power entanglement. Hereby, this work advocates that the plain measurements are also a sufficient resource for the reserve accounting mechanism in the kernel. This statement is based on the following two observations: Firstly, the power sandbox mechanism still guarantees the protection of user space application as those have only access to the metered data if they are currently running with the sandboxed environment. Due to the exclusive nature of the underlying resource balloons, power entanglement is still confined and therefore no power side-channel attacks are possible. Secondly, if a malicious application tries to manipulate the cost accounting of a simultaneously running application (which is not sandboxed) by consuming excessive amounts of energy, the malicious application will receive penalties due to the extreme power consumption. The observed impact on the other applications should be negligible for accounting. Therefore, we believe that this is a reasonable design choice as otherwise all kind of concurrency in metered subsystems must be confined all the time. That being said, the first adaption of the reserve/tap mechanism should be done.

Concerning the second aspect, the overhead of a power sandboxed application must be accounted for future scheduling decisions. Therefore, the combined approach will also have to take care of the penalty and modify the scheduling credit of the thread accordingly.

4 IMPLEMENTATION

This section covers a possible implementation for the previously described design. Hereby, the focus lies on the different isolation mechanisms in terms of power as a system resource.

4.1 Power Sandboxing

Resource balloons form the foundation for power sandboxing by confining concurrency in a metered subsystem. As a common example, spatial balloons handle concurrency within multicore CPUs. The following summary will serve as a high-level description of the multicore scheduler which confines power entanglement of different CPUs. However, the basic idea can be applied analogously to any other device covered by spatial/temporal balloon.

(1) Schedule-In: The scheduler on core n ($Sched_n$) will pick the process group with the best scheduling credit from its queue. From this group, one thread is selected and prepared for execution.
(2) Task Shootdown: Afterwards, the scheduler $Sched_n$ will notify all other schedulers instances on the remaining cores to coschedule threads of the same process group. This notification can be realized as an inter-processor interrupt (IPI). After receiving that notification, the remaining schedulers $Sched_{\overline{n}}$ will pick another thread from the group or schedule an idle thread if the group is empty. Each scheduler will compute the difference in scheduling credit between the selected thread and their most favorable thread in the local scheduling queue which would have ran otherwise. This difference is denoted as Δ_j for the scheduling instance of core j.
(3) Running and loan update: The scheduled threads are executing and the running costs are accounted and added to scheduling credit change Δ_j.
(4) Schedule out: If one scheduling instance is about to reschedule another thread (of a different process group), all other schedulers are notified to perform a task shootdown as well.
(5) Loan redistribution & repayment: The accounted cost are split evenly among the coscheduled threads to ensure fairness among them.

4.2 Reserves and Taps

Reserves and taps are used to provide isolation between different processes by storing the amount of available resource (in this case energy). The adapted scheduling mechanism can be simplified and described as following:

(1) Reserve Check: The scheduler will check the current amount of usable resource within the available reserves for the selected thread. If the associated reserves do not yield enough resources, the scheduler will block the thread from running and postpone it in favor of the next entry within the scheduling queue.
(2) Running and Accounting: Following, the scheduler can restart the selected thread while accounting the consumed power. The caused costs will be subtracted from the associated reserves.
(3) Evaluation of Taps: To refill the reserves, all corresponding taps are evaluated. Therefore, the elapsed time is tracked and will be multiplied by the tap rate. The resulting amount of energy will be added to the reserves.

5 EVALUATION

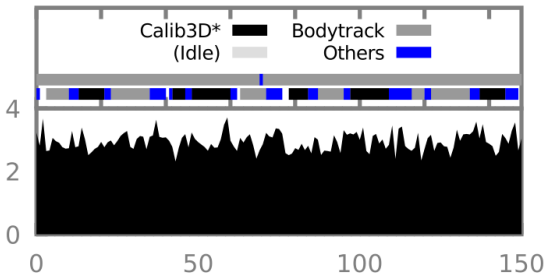
We evaluate the effectiveness of the combination of power sandboxing and reserves/taps by looking at following questions:

(1) Does power sandboxing allow accurate and reasonable measurements without power entanglement?
(2) Is process isolation in terms of energy as a system resource achievable?

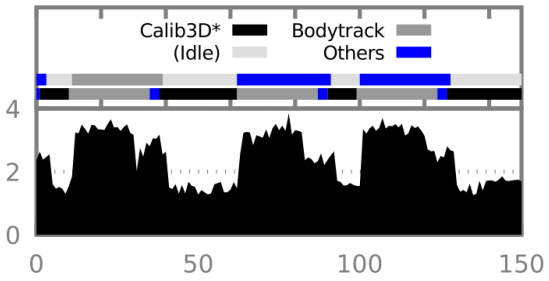
Below, the setups of the experiments and their measurement results are taken from [5] and [11].

5.1 Reasonable & Accurate power-consumption

Power entanglement was identified as the main source of measurement distortions due to the concurrent usage of one metered subsystem. To prove the functionality of the power sandboxing mechanism or rather the exclusive nature of the underlying resource balloons abstraction, the following experiment is performed: Several different applications are started concurrently on a dual-core CPU while their activity and power consumption is tracked. Hereby, the operating system will start multiplexing the applications and schedule them on the two CPU cores. The used applications include the calib3d and bodytrack benchmark. Both benchmarks will create a noticeable CPU load. Now, the experiment will be performed twice: During the first run, no applications will be sandboxed and a normal multiplexing behavior should be observable. For the second run, the calib3d applications will be power sandboxed.



(a) Multiplexing without Power Sandbox



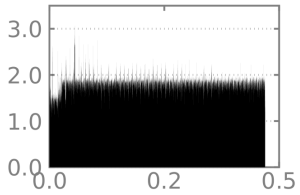
(b) Multiplexing with Power Sandbox

Figure 5: Resource Multiplexing while tracking Application Activity and Power Consumption, x-axis: Time/ms, y-axis: Power/W

Figure 5 shows the measured and tracked data of both runs. The first plot illustrates the multiplexing behavior of the non-sandboxed run. Hereby, almost no idle time can be observed and most of the time two applications are scheduled and make use of both CPU cores. Also, the power-consumption graph indicates a more or less stable value without any outstanding break-in. However, the second subfigure points out the sandboxing effects clearly. Around 50ms, 100ms and 140ms on the plot time line, the sandboxed calib3d application is scheduled in. Due to the exclusive nature of the power sandbox, all concurrent threads will be shot down and an idle thread will be scheduled instead. Noteworthy, the measured power consumption also drops heavily during the sandboxed execution as the idle thread will hardly consume power. This proofs the effectiveness of the underlying resource balloon mechanism.

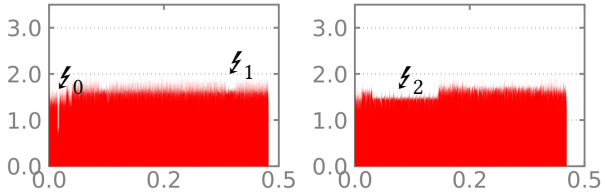
Furthermore, it is necessary to test whether the reported values of the power sandbox match the actual power consumption behavior of the application without any power entanglement. Therefore, a second experiment must be started. Hereby, the calib3d benchmark will be repeated three times. During the first run, the calib3d application is measured isolated without any other application. This measurement serves as the baseline for comparison. Concerning the second run, the calib3d application will be started simultaneously with the bodytrack and dedup benchmark once. For measuring, a fine-grained kernel-level accounting mechanism is chosen [13]. For the third run, the calib3d application will be sandboxed and started with the bodytrack and dedup benchmark once but this time, the measurements are produced by the virtualized power meter of the sandbox.

(a) calib3d Alone



alone: 936mJ

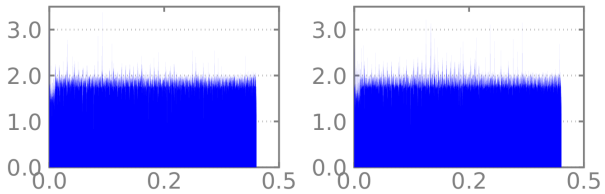
(b) calib3d Co-Running



w/ body: 853mJ (-8.9%)

w/ dedup: 804mJ (-14.1%)

(c) Sandboxed calib3d Co-Running



w/ body: 922mJ (-1.5%)

w/ dedup: 925mJ (-1.2%)

Figure 6: Sandboxed calib3d benchmark, x-axis: Time/s, y-axis: Power/W

Figure 6 demonstrates the difference between the kernel-level accounting mechanism and the metering of the vertical environment. As seen in the second subfigure, the measurement of the power consumption of calib3d while co-running with the body benchmark includes some unmatching spikes in comparison to the baseline measurement (marked by $\lightning_0$ and $\lightning_1$). An even worse effect can

be noticed while co-running with the dedup benchmark as a non-matching valley (marked by $\lightning_2$). In contrast, the sandboxed data equals pretty much the base measurement. Also, the accumulated energy consumption of the sandboxed measurement (922mJ and 925mJ) shows less deviation in comparison to kernel-level approach (853mJ and 804mJ).

5.2 Resource Isolation

The second goal of the overall design is about providing resource isolation in terms of energy for processes or process groups. Hereby, the operating system ensures that delegated amount of energy is available and not exceeded. To confirm, the following experiment is performed: Two processes A and B are started simultaneously. After about 5 seconds B spawns a third process B1 and then, after 10 seconds in total, B spawns the fourth process B2.

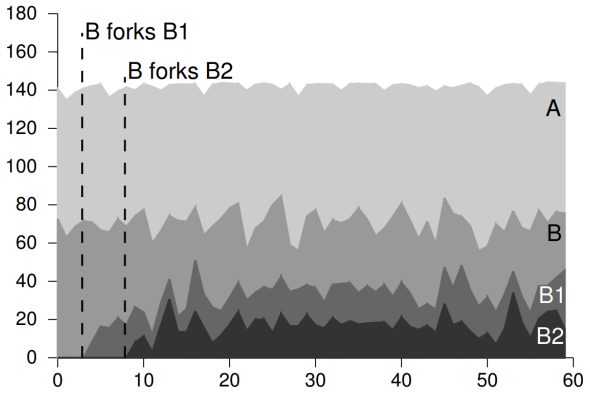


Figure 7: Stacked Graph of isolated Processes, x-axis: Time/s, y-axis: Power/mW

Figure 7 shows the power consumption of all four processes as a stacked graph. Normally, the spawned processes B1 and B2 would receive the same CPU share as the processes A and B. Therefore, a drop should be noticeable for process A in the power-consumption graph. However, due to the fact that the reserves (and the associated CPU share) of process A is isolated from the remaining processes, A is still allowed to run at the same level. Furthermore, the process B is also able to isolate itself from its children B1 and B2 by creating two separate reserves respectively. This should proof the ability of a process to isolate itself from other processes in terms of energy as a system resource.

6 CONCLUSION

Power-aware applications benefit from their ability to optimize their runtime behavior according to the current power consumption. By combining two already existing approaches, namely power sandboxing and reserves/taps, the quality of power metering and energy resource management can be improved. As a result, not only the accuracy is increasing but potential power-related security threats can be confined. Furthermore, abstractions regarding energy as another important system resource have found their way into operating system design and will impact future scheduling decisions.

REFERENCES

[1] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. 2009. Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement (IMC '09)*. Association for Computing Machinery, New York, NY, USA, 280–293. https://doi.org/10.1145/1644893.1644927

[2] Mian Dong, Tian Lan, and Lin Zhong. 2014. Rethink Energy Accounting with Cooperative Game Theory. In *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking (MobiCom '14)*. Association for Computing Machinery, New York, NY, USA, 531–542. https://doi.org/10.1145/2639108.2639128

[3] Mian Dong and Lin Zhong. 2011. Self-Constructive High-Rate System Energy Modeling for Battery-Powered Mobile Systems. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys '11)*. Association for Computing Machinery, New York, NY, USA, 335–348. https://doi.org/10.1145/1999995.2000027

[4] Jason Flinn and M. Satyanarayanan. 1999. Energy-Aware Adaptation for Mobile Applications. In *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles (SOSP '99)*. Association for Computing Machinery, New York, NY, USA, 48–63. https://doi.org/10.1145/319151.319155

[5] Liwei Guo, Tiantu Xu, Mengwei Xu, Xuanzhe Liu, and Felix Xiaozhu Lin. 2018. Power Sandbox: Power Awareness Redefined. In *Proceedings of the Thirteenth EuroSys Conference (EuroSys '18)*. Association for Computing Machinery, New York, NY, USA, Article 37, 15 pages. https://doi.org/10.1145/3190508.3190533

[6] Henry Hoffmann. 2015. JouleGuard: Energy Guarantees for Approximate Applications. In *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP '15)*. Association for Computing Machinery, New York, NY, USA, 198–214. https://doi.org/10.1145/2815400.2815403

[7] Mohammad Hosseini, Alexandra Fedorova, Joseph Peters, and Shervin Shirmohammadi. 2012. Energy-aware adaptations in mobile 3D graphics. *MM 2012 - Proceedings of the 20th ACM International Conference on Multimedia*, 1017–1020. https://doi.org/10.1145/2393347.2396371

[8] Yan Michalevsky, Aaron Schulman, Gunaa Arumugam Veerapandian, Dan Boneh, and Gabi Nakibly. 2015. PowerSpy: Location Tracking Using Mobile Device Power Analysis. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, Washington, D.C., 785–800. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/michalevsky

[9] Shivajit Mohapatra, Nalini Venkatasubramanian, Nikil Dutt, Cristiano Pereira, and Rajesh Gupta. 2004. *Energy-Aware Adaptations for End-to-End Videostreaming to Mobile Handheld Devices.* Springer US, Boston, MA, 255–273. https://doi.org/10.1007/1-4020-8076-X_14

[10] Sergiu Nedevschi, Lucian Popa, Gianluca Iannaccone, Sylvia Ratnasamy, and David Wetherall. 2008. Reducing Network Energy Consumption via Sleeping and Rate-Adaptation. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI'08)*. USENIX Association, USA, 323–336.

[11] Arjun Roy, Stephen M. Rumble, Ryan Stutsman, Philip Levis, David Mazières, and Nickolai Zeldovich. 2011. Energy Management in Mobile Devices with the Cinder Operating System. In *Proceedings of the Sixth Conference on Computer Systems (EuroSys '11)*. Association for Computing Machinery, New York, NY, USA, 139–152. https://doi.org/10.1145/1966445.1966459

[12] Carl Waldspurger. 2003. Memory resource management in VMWare ESX server. *ACM SIGOPS Operating Systems Review* 36 (01 2003). https://doi.org/10.1145/1060289.1060307

[13] Chanmin Yoon, Dongwon Kim, Wonwoo Jung, Chulkoo Kang, and Hojung Cha. 2012. AppScope: Application Energy Metering Framework for Android Smartphone Using Kernel Activity Monitoring. In *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. USENIX Association, Boston, MA, 387–400. https://www.usenix.org/conference/atc12/technical-sessions/presentation/yoon

[14] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. 2002. ECOSystem: Managing Energy as a First Class Operating System Resource. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X)*. Association for Computing Machinery, New York, NY, USA, 123–132. https://doi.org/10.1145/605397.605411

[15] Meng Zhu and Kai Shen. 2016. Energy Discounted Computing on Multicore Smartphones. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. USENIX Association, Denver, CO, 129–141. https://www.usenix.org/conference/atc16/technical-sessions/presentation/zhu