

Software-Defined Batteries

By Anirudh Badam, Ranveer Chandra, Jon Dutra, Anthony Ferrese, Steve Hodges, Pan Hu, Julia Meinershagen, Thomas Moscibroda, Bodhi Priyantha, and Evangelia Skiani

Abstract

Different battery chemistries perform better on different axes, such as energy density, cost, peak power, recharge time, longevity, and efficiency. Mobile system designers are constrained by existing technology, and are forced to select a single chemistry that best meets their diverse needs, thereby compromising other desirable features. In this paper, we present a new hardware–software system, called Software Defined Battery (SDB), which allows system designers to integrate batteries of different chemistries. SDB exposes application programming interfaces (APIs) to the operating system, which controls the amount of charge flowing in and out of each battery, enabling it to dynamically trade one battery property for another depending on application and/or user needs. Using micro-benchmarks from our prototype SDB implementation, and through detailed simulations, we demonstrate that it is possible to combine batteries which individually excel along different axes to deliver an enhanced collective performance when compared to traditional battery packs.

1. INTRODUCTION

The utility of a mobile device is often constrained by the capabilities of its battery. While integrated circuit performance has doubled every 18 months according to Moore's law, the same is far from true for battery technology. Battery performance can be evaluated in many different ways (see Table 1), but no matter which metric we look at, it has taken more than a decade to double performance.

Furthermore, the various properties of batteries are often at odds with each other. For example, batteries with higher

power densities tend to have lower volumetric and gravimetric energy densities, and vice versa. Similarly, making a conformable battery that fits a particular industrial design compromises its performance characteristics.

Such tradeoffs are present even within a given physical battery. For example, energy delivered by a battery in a single charge–discharge cycle (energy capacity) is inversely related to the rate at which the battery is drained (discharge rate). This is because the resistance losses inside a battery are proportional to the square of the current. Similarly, a battery's longevity—its ability to perform consistently following many charge–discharge cycles—is inversely related to the discharge and recharge rates. This is because higher currents speed up the creation of fissures in the electrodes that reduce the amount of energy a battery can store.

In summary, no single battery type can deliver the ever-growing list of requirements of modern devices: fast charging, high capacity, low cost, less volume, light weight, less heating, better longevity, and high peak discharge rates.

A growing range of battery chemistries are under development, each of which delivers a different set of benefits. We believe that combining multiple of these heterogeneous batteries instead of using a single battery chemistry can allow a mobile system to dynamically trade between their capabilities and thereby offer attractive tradeoffs.

However, traditional methods of integrating multiple batteries are not suitable for heterogeneous batteries. Connecting them in series or parallel does not provide enough control over usage: batteries connected in series can only supply the same amount of current; batteries connected in parallel must operate at the same voltage and can only supply currents that are inversely proportional to their internal resistances.

We propose a new system, called Software Defined Battery (SDB), that allows heterogeneous batteries with different chemistries to be integrated in a mobile system. SDB consists of hardware and software components. The hardware enables fine-grained control of the amount of power going in and out of each battery using smart switching circuitry. The software, which resides in the operating system (OS), computes how much power to draw from each battery, and how to recharge each battery.

Deciding how much power to draw from and how to charge each battery is nontrivial. It depends on the efficiency of each battery under different workloads, the age of each battery, and also the usage profile. For example, if a high power workload is anticipated in the future, then it could be worth-while

Table 1: A number of battery characteristics.

Battery characteristics	Units
Energy capacity	J
Volume	mm ³
Mass	kg
Discharge rate	W
Recharge rate	W
Gravimetric energy density	J/kg
Volumetric energy density	J/L
Cost	\$/J
Discharge power density	W/kg
Recharge power density	W/kg
Cycle count	Number of discharge/recharge cycles
Longevity	% of original capacity after N cycles
Internal resistance	Ohm
Efficiency	% of energy turned into heat
Bend radius	mm

These are often in tension with each other, for example, increasing recharge rate compromises longevity.

The original version of this paper was published in *Proceedings of ACM SOSP'15*.

conserving charge on the battery that is more capable of handling such a workload in an efficient manner.

The SDB software component that resides in the OS implements a set of policies, and uses simple application programming interfaces (APIs) to communicate with the SDB hardware. The algorithms implemented by this software use various metrics to decide the ratios in which to discharge and charge each battery, such that the charge–discharge duration of the device is increased, and degradation of the batteries is reduced. We present the details of the APIs and policies in Section 3.3.

The SDB design is cross-layer and involves new chemistries, additional hardware, and new OS components. This approach opens up new battery parameters, previously unavailable to OS designers, for resource optimization. In existing mobile devices, the battery is usually treated as a black box, and is simply assumed as a reservoir of charge. As we show in Section 5, OS techniques yield substantial gains in battery usage. This design also allows a system designer to select any combination of batteries for an optimal design, including new chemistries as they are developed using just software updates.

Even with existing batteries, SDB enables several new scenarios, such as: (i) fast-charging devices that can gain a significant percentage of their charge in just a few minutes without causing unexpected battery degradation, (ii) long-lived wearables created by combining flexible bendable batteries with traditional batteries, and (iii) efficient 2-in-1 laptop-tablet convertible devices with battery usage tailored to the user’s behavior.

2. BATTERY BACKGROUND

A Li-ion battery contains a negative electrode (the anode), which is usually made of graphite and a positive electrode (the cathode), which is typically a metal oxide. A separator ensures physical separation between the anode and the cathode to prevent shorting, and the battery is filled with an electrolyte composed of a lithium-based salt whose ions can easily pass through the separator. Current is discharged when the electrodes are connected externally over a resistive load while positive lithium ions flow from the anode to the cathode through the electrode and the separator. During charging, Li-ion batteries store energy by trapping positive lithium ions in the anode when an external potential is applied.

Li-ion battery capabilities, such as longevity, energy density, and internal resistance, are largely determined by the materials used for the electrodes and the separator. The battery’s gravimetric and volumetric energy densities are affected by the strength of the separator. The resistance of the battery, and hence its inefficiencies, depend on the resistance of the separator, which typically increases with the age of the battery. The power density of the battery is also affected by aging. The structural integrity of the electrodes determines how much energy they can store—some lithium ions get permanently trapped in the anode. The anodes can develop cracks as they age, which can ultimately reduce both energy and power densities.

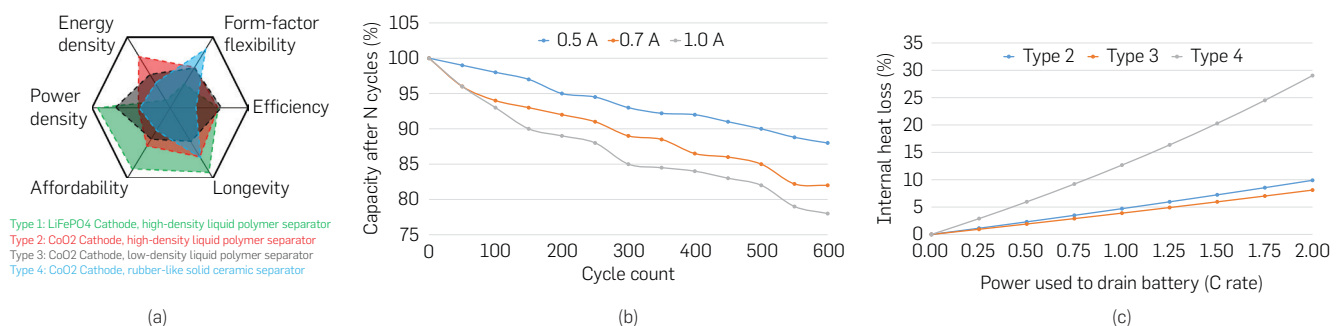
Figure 1a demonstrates the capabilities of four different Li-ion batteries, which differ in the chemistry of materials used for the cathode and the separator. Batteries of Type 1 are typically used in powered tools that need to charge quickly and provide high power for a short duration of time. Such batteries are a poor choice for mobile devices because of their poor energy density—a Type 1 battery is usually double the volume of a Type 2 battery with the same energy capacity. Type 2 batteries are commonly used in most mobile devices today. We measure the loss in capacity with respect to number of charge–discharge cycles for a sample Type 2 battery, and observe that the battery degrades much faster when discharged at higher current (Figure 1b).

Type 3 batteries are an emerging variation over Type 2 that have a slightly higher power density at the expense of some energy density. This is achieved by making the separator less dense allowing more lithium ions to pass through per unit time. This usually leads to decreased energy density as separators cannot store energy—only the electrodes can. Finally, Type 4 is another emerging battery that is flexible and bendable because of the physical properties of the rubber-like (ceramic-based) separator used—while the electrodes are implemented by coating material along the cell’s walls. Unfortunately, such separators increase the resistance to passage of ions and thereby result in higher inefficiency, as shown in Figure 1c.

2.1. Typical power management

Figure 2 shows a block diagram of the typical power management hardware. It consists of a (i) battery, (ii) fuel gauge, (iii) battery charger, and (iv) voltage regulator.

Figure 1. Li-ion battery properties. (a) Li-ion batteries compared. (b) Charging rate affects longevity. (c) Discharging rate versus lost energy.



A *Battery pack* has one or more battery cells. Multiple cells are used to achieve higher voltage or higher capacity. While such multi-cell configurations exist today, for example in the Surface Pro, Galaxy Tab, and iPad, these cells have the same chemistry and are either connected in series, parallel, or a combination thereof. They are treated as a single monolithic battery by the OS. Our aim is to use a heterogeneous set of cells and achieve wide dynamic characteristics by exposing the cells directly to the OS.

The *Fuel gauge* keeps track of the state of charge (SoC) of the battery by measuring the voltage across the battery terminals, and the coulombs flowing in and out of it. This information is exposed to the OS.

The *Battery charger* charges the battery with an appropriate charging current profile based on the battery's SoC, the terminal voltage (the potential difference between the anode and the cathode), and the capability of the power source.

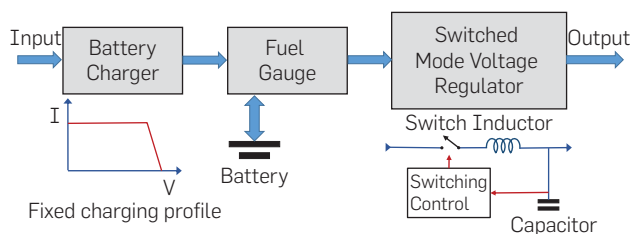
Due to the battery's internal resistance R , the battery terminal voltage changes with the load current I due to the IR voltage drop. The internal resistance and the battery voltage themselves change with the SoC. The job of the *Voltage Regulator* is to hide these terminal voltage variations due to changing potentials at the electrodes and the changing internal resistance and present a constant voltage to the load. Mobile devices use switched mode voltage regulators due to their high efficiency. As the name implies, a switch mode power supply contains a switch that opens and closes to transfer packets of energy (Figure 2). A control loop maintains a constant voltage under varying load currents by changing the energy per packet or the packet switching frequency.

Typically, all these modules are contained in a single power management integrated circuit (PMIC), which communicates with the OS over a serial bus. In current designs, the interactions between the OS and PMIC are limited to *query* operations, such as inquiring about remaining SoC. However, none of these APIs allow the OS to set the battery parameters, and in particular to change the amount of charge to be drawn from or provided to each cell within a battery pack. Through the SDB system, we propose enabling fine grain control by exposing a richer software API to the OS to dynamically change the amount of charge to be drawn from or provided to each battery.

3. SDB DESIGN

SDB allows a device to use diverse batteries through fine-grain control of the amount of charge flowing in and out

Figure 2. Traditional power management hardware.



of each battery. SDB provides APIs to the OS to change the aforementioned power values based on user workload. We describe the SDB system in detail in this section.

3.1. System overview

The SDB system spans components across three layers: the batteries and their chemistry, the battery management circuit, and the OS. We outline these components and their interactions in Figure 3.

SDB allows a system designer to combine diverse batteries. The particular batteries chosen depend on the scenario, such as a fast charging battery and a high energy battery for a tablet, or a bendable battery and high energy battery for a smart-watch.

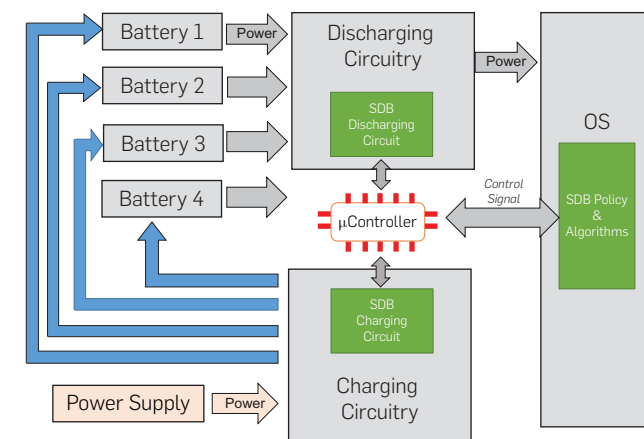
However, combining different battery types is not trivial. These batteries might have different capacities and different terminal voltages. Therefore, we design a new power distribution circuit for fine-grain control of how multiple batteries are discharged to support system load. A microcontroller interfaces between this power distribution circuitry and the mobile device OS to control the charging and discharging of batteries accordingly.

To enable flexibility in design, and to allow quick changes in policy, we only implement the mechanisms in hardware, and all policies are managed and set by the OS. A runtime component in the OS monitors the charging and discharging behavior and accordingly sets policies that meet user expectations in terms of daily battery life and longevity of the battery-pack.

3.2. SDB hardware

The SDB hardware needs to support discharging and charging across multiple, heterogeneous batteries. For discharging, it has to provide a flexible mechanism for fine-grain control of how the load current is supplied from each battery. This should support two things: coarse grain *switching* of the load across multiple batteries where the total load is supplied by a particular battery for an extended period of time and the fine grain *sharing* of the load where a certain fraction of the load is drawn from each battery.

Figure 3. SDB system overview.



For charging, the SDB hardware has to support control over how batteries are charged. In contrast to existing solutions where batteries are charged according to a fixed charging profile, SDB requires setting of charging currents and charging profiles dynamically based on OS policies. Under certain circumstances, it should even be possible to charge a battery from another one.

Designing these flexible charging and discharging circuits are challenging for two reasons. First, due to the high currents that flow in these circuits, any electronic component in series with the current flow will cause energy losses. Hence, these circuit designs should introduce as few of these components as possible. Second, each extra component we introduce can increase the weight, volume, and bill of material (BoM) cost of the device, which will make the proposed solution unattractive in the competitive hardware market.

SDB discharging circuit design. A simple discharging circuit can be implemented using a combination of an electronic switch and a capacitor as shown in Figure 4a. The microcontroller achieves load switching by connecting the appropriate battery to the load. To achieve load sharing, the load is switched between the batteries at a high frequency in round-robin fashion. The ratio of the current draw is determined by the fraction of time the switch is connected to a particular battery. The capacitor acts as an energy store to smooth out the discontinuities due to switching. Parasitic battery capacitance and external capacitors smooth out the high frequency battery current.

However, this naive implementation has two main drawbacks. First the switch, typically implemented using a Field Effect Transistor (FET), has a finite on resistance that causes significant power loss at high load currents. Second, a switch with high power handling capability and the necessary capacitors increase the BoM cost and space required.

To overcome these shortcomings, we designed a new switched mode regulator architecture that integrates fine-grain battery switching into the regulator itself. As shown in Figure 4c, we restructure the built-in switch to achieve voltage regulation and support switching between multiple batteries—by drawing packets of energy from the batteries in a weighted round-robin fashion. We reuse the storage

capacitor to smooth out the load current variations due to switching. We have evaluated the correctness of the proposed solution under different battery voltages and load conditions by running LTSPICE¹¹ simulations.

SDB charging circuit design. The SDB charging circuit should have the ability to charge batteries at a configurable rate and also charge them from each other. Given that such charging should be possible irrespective of the battery voltage, the batteries should be connected through a buck-boost regulator, such that the energy source is at the input and the energy sink is at the output of the regulator. A buck-boost regulator is a particular form of switching regulator where the regulator output voltage can be either less than or greater than its input voltage.

Apart from different charging configurations, SDB requires dynamic fine-grain control over the charging profile. This is achieved by instrumenting each switched mode regulator with multiple charging profiles where the SDB microcontroller dynamically selects the appropriate charging profile based on OS policy decisions.

Figure 4b shows how these modules can be combined to implement a flexible charging circuit. However, a major drawback of this configuration is the large number of switching regulators ($O(N^2)$ for N batteries) required, which negatively impacts the device BoM cost and space requirements.

Instead, we design an optimized charging circuit as shown in Figure 4c, which requires only $O(N)$ switched mode regulators to charge N batteries. When an external supply is present, the microcontroller configures both R_1 and R_2 in *buck* mode to charge the batteries. When external power is removed, R_1 and R_2 are disabled. When B_2 is to be charged from B_1 , R_1 operates in *reverse buck* mode while R_2 operates in *buck* mode and vice versa.

3.3. SDB policies and APIs

Our current SDB software architecture is illustrated in Figure 5. An *SDB Runtime* encapsulates the SDB microcontroller from the rest of the OS. The SDB Runtime is responsible for all scheduling decisions affecting the charging and discharging of batteries. It takes clues from the rest of the OS, and communicates the charging and discharging scheduling decisions to the SDB controller.

Figure 4. (a) A simple switch and capacitor-based battery switching solution. (b) A naive implementation of a flexible charging circuit consisting of two buck regulators and two buck-boost regulators with dynamic charging parameters. (c) SDB hardware architecture for an example two-battery system. The switched mode regulator implements discharge across multiple batteries and the reverse buck regulators implement charge.

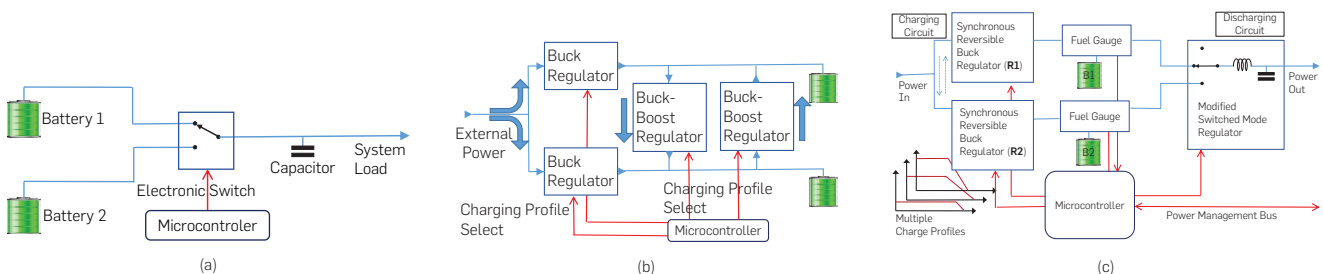
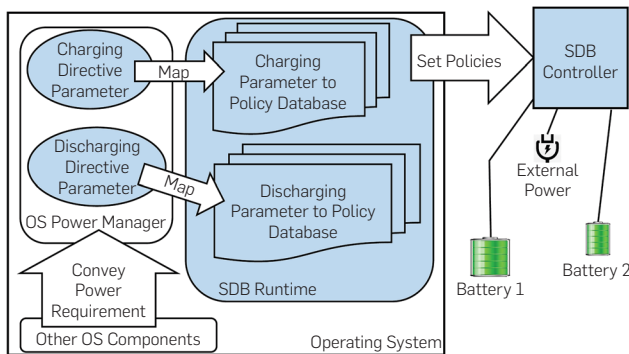


Figure 5. SDB software architecture.



APIs: For a system with N batteries, the SDB Runtime maintains two N -tuples (c_1, \dots, c_N) and (d_1, \dots, d_N) of non-negative values, one for charging and one for discharging. In both cases, the N values add up to one and represent power ratios, that is, the numbers represent the fraction of power that must go in and out of each of the N batteries. The runtime communicates with the SDB microcontroller using the following four APIs:

- Charge (c_1, c_2, \dots, c_N) : Charge N batteries in proportion to c_1, c_2, \dots, c_N , when being charged from an external source.
- Discharge (d_1, d_2, \dots, d_N) : Discharge N batteries in proportion to d_1, d_2, \dots, d_N , when being discharged.
- ChargeOneFromAnother (X, Y, W, T) : Charge battery Y from battery X with a power of W for time T .
- QueryBatteryStatus $()$: Returns an array with SoC, terminal voltages and cycle counts for each battery.

The SDB Runtime affects changes in the charging and discharging behavior by adapting the $2N$ numbers and sending them to the microcontroller using the above APIs, which enforces the ratios. Such changes can be triggered for example by a change of the user’s needs, the battery state, workload patterns, or external factors such as a change in device temperature, etc. Determining optimal battery charging/discharging policies, is nontrivial, and the underlying algorithmic problems are deep and interesting. Often, various battery properties are in tension with one another. For example, fast-charging a battery all the time can greatly accelerate its aging. In this paper, we only scratch the surface of these algorithmic problems and instead describe a set of natural policy heuristics that exhibit good albeit non-optimal performance.

Metrics: Two key metrics any charging/discharging policy seeks to optimize are *Cycle Count Balance* (CCB) and *Remaining Battery Lifetime* (RBL). The RBL metric simply captures the remaining battery lifetime of the device, assuming that no further charging occurs in the future. In other words, RBL is the amount of useful charge in the batteries. The CCB metric reflects that—ideally—the charging and discharging policies should maximize longevity of the device, by balancing the charging cycles of each battery. In

a heterogeneous battery system, each battery is a unique precious resource that excels on a few metrics of interest described in Table 1. Therefore, having a metric-like the CCB ensures that these batteries are aging such that the properties of a battery that the user is most interested in are preserved over time. For example, a battery that has the ability to charge fast must be treated as a precious resource for a user who relies on fast charging during low-battery situations.

Concretely, let χ_i be the number of charging cycles tolerable by battery i before its capacity drops below some acceptable threshold, and let cc_i be the number of charging cycles of battery i . The *wear-ratio* $\lambda_i = cc_i/\chi_i$ describes what fraction of the tolerable recharge cycles have already been consumed by battery i . We define CCB as the ratio $CCB = \max_i \lambda_i / \min_j \lambda_j$, that is, the ratio between the most and least worn-out battery, normalized to each battery’s total tolerable cycle count. A device’s longevity is maximized by balancing CCB.

Charge/discharge algorithms. The heuristics currently driving our SDB Runtime are simple and driven by the following observation: It is possible to derive charging and discharging algorithms that, in isolation, optimize the CCB and the instantaneous RBL metric. We use these four “optimal” algorithms (CCB-Charge, RBL-Charge, CCB-Discharge, and RBL-Discharge) and weigh them by means of two parameters—Charging and Discharging Directive Parameters—handed to the SDB Runtime by the rest of the OS. Essentially, these parameters guide the SDB Runtime to weigh one of the algorithms more heavily at any moment in time. For example, a low value of the Charging Directive Parameter indicates that the user is in no hurry (e.g., charging at night), and that the Runtime should prioritize the use of the CCB-Charge algorithm. On the other hand, a high value of this parameter would lead the Runtime to prioritize the RBL-Charge algorithm in order to increase the useful charge (and thus the remaining battery lifetime) in the batteries as quickly as possible—say just before boarding an airplane. The discharge scenario is similar.

The CCB-Charge and CCB-Discharge algorithms are simple. These policies essentially enforce the controller to schedule the batteries (either for charging or discharging) in such a way that the resulting CCB is minimized, that is, is as close to 1 as possible. Our RBL-algorithms are more complex. Consider the discharge case, and let y_1, \dots, y_N be the amount of current drawn from each of the batteries. The key underlying insight is that we can maximize the instantaneous RBL of the battery system by minimizing the total resistive losses across all the batteries. This can be achieved *if the resistances of the batteries are proportional to the square-root of their DCIR-to-SoC ratios*. Thus, the RBL-Discharge algorithm seeks to allocate the currents y_1, \dots, y_N in such a way that the effective resistances of batteries are as much as possible proportional to the square-root of their DCIR-to-SoC rates minimizing the total energy wasted through resistive losses. Mathematically speaking, let δ_i be the instantaneous derivative of battery i ’s DCIR curve, and let R_i be the current resistance. Then, the RBL-Discharge algorithm balances $R_i' = \sqrt{\delta_i/\lambda}$, where $R_i' = R_i + \delta_i y_i$ and λ is a

Lagrangian multiplier constant. Again, the case for charging (RBL-Charge) is similar. The SDB runtime calculates these power values at coarse granular time steps and updates the ratios based on the DCIR-SoC curves given by the manufacturer of the batteries.

A word of caution is necessary. The above RBL-algorithms are “optimal” only in an instantaneous sense. They minimize the instantaneous decrease of RBL (when discharging), or maximize the instantaneous increase of RBL (when charging). However, they are not globally optimal. Across the length of an entire workload, these algorithms might not actually maximize battery lifetime as we show in Section 5, that is, if we had knowledge of the future workload, we could improve upon the above instantaneously optimal algorithms by making temporarily suboptimal choices from which the system can profit later, for example, keeping a battery fully charged, if we know that this battery will be particularly helpful in the way of CCB or RBL for a future workload. For example, the overall cycle life or daily battery life may be improved when compared to using instantaneous mechanisms all the time.

Exploring these and other algorithmic nuances is interesting, but beyond the scope of this paper. We just note that the SDB resource optimization problem differs from traditional resource scheduling mechanisms, such as for Big.Little processors, hybrid storage, and SSD wear leveling, because of the resource in question—batteries. The main focus of traditional resource management algorithms is to multiplex a resource efficiently across a number of entities, such as users, processes, virtual machines, or erase blocks in case of SSDs over some fixed periods of time. The challenge of battery resource scheduling is threefold: daily battery life cannot be simply extended by minimizing instantaneous power losses; their long-term cycle life cannot be simply extended by balancing cycle life across batteries. Knowledge of impending workload can be used to improve the latter two metrics by picking strategies that may not be an instantaneous optimums as we demonstrate in Section 5. We hope that exposing the appropriate APIs will help system and algorithm designers to customize the scheduling algorithms for their battery configuration, and user workloads based on predicted as well as expected user behavior.

4. PROTOTYPE AND MICROBENCHMARKS

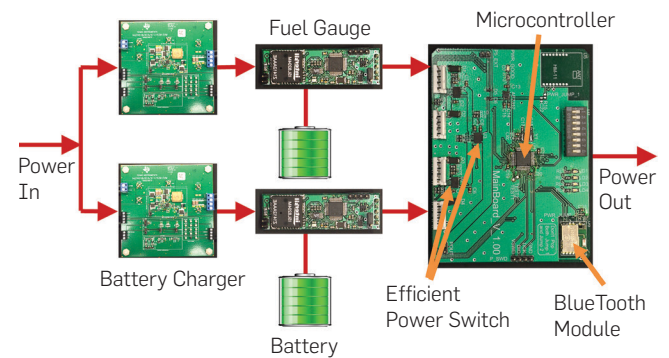
In this section, we describe the implementation of SDB and present microbenchmarks to evaluate it.

We built a hardware prototype of the SDB hardware architecture in Figure 4c. Figure 6 shows the components of our prototype.

We built a custom controller board with a ARM Cortex M3 microcontroller and a low-loss switching circuit. We also built a custom fuel gauge module that consists of a coulomb counter and a controller. We modified an off-the-shelf battery-charger evaluation board to enable dynamic charge current setting by the microcontroller on the control board. These hardware modules were interconnected as shown in Figure 6.

We used an ideal diode to switch between the batteries. The switching between batteries is extremely fast, and

Figure 6. SDB prototype implementation.



hence the battery sees a constant, smooth current draw. We note that the small power-loss due to this switch underestimates the efficiency achievable by the proposed solution. As mentioned in Section 3, the extra power-loss and the high component cost can be eliminated by augmenting existing switching regulators to switch across multiple batteries.

The boards were designed with Altium Designer,¹ a circuit board development package. The firmware was written in C using the IAR for ARM V7.40 tool chain. The board firmware contains ≈ 3500 lines of code. We also developed the prototype SDB Runtime shown in Figure 5 with 1200 lines of code.

We conducted simulations and microbenchmark experiments to evaluate the efficiency and accuracy of our hardware design, as well as to evaluate the correctness of the firmware and the runtime. Circuit simulations were done in LTSPICE,¹¹ a simulation program with integrated circuit emphasis (SPICE). We generated the circuits as shown in Figure 4, in LTSPICE, and conducted extensive simulations at various power loads to validate system correctness, stability, and responsiveness.

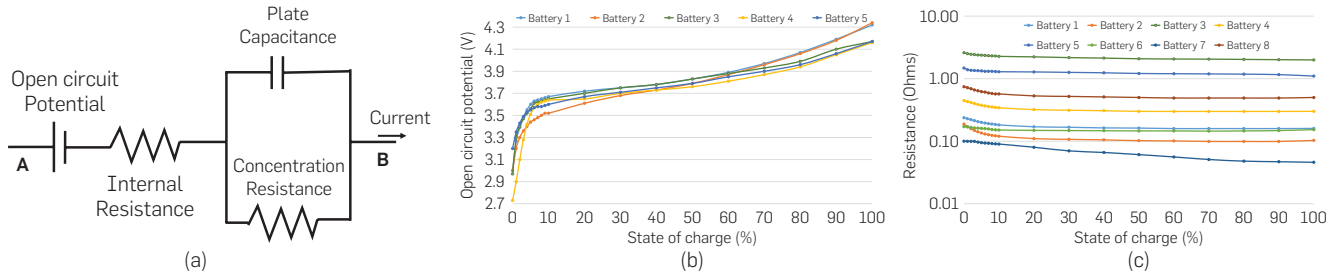
4.1. SDB emulator

We build a model for batteries based on Thevenin’s model as built by other battery researchers^{3-6,9} to simulate batteries used in production devices. The simplified Thevenin model is reproduced in Figure 7a. The model has four parameters: open circuit potential, internal resistance, concentration resistance, and plate capacitance.

The open circuit potential of a battery is the voltage across the terminals of the battery when no load is applied. It increases with the amount of energy left in a given battery. The internal resistance of a battery is the resistance across the terminals of the battery when a load is applied. It decreases with the amount of energy left in a given battery. In Figure 7b and c, we plot the open circuit potential and resistance, respectively, of a few batteries as the energy left in them increases.

The concentration resistance and the plate capacitance of a battery are fixed values for a given battery. We measure the open circuit potential, internal resistance, concentration resistance, and the plate capacitance for several kinds of batteries. We use the industry standard Arbin BT-2000²

Figure 7. Battery simulator: (a) Battery modeled with four variables that are learned using experimentation: open circuit potential, internal resistance, concentration resistance, and plate capacitance. This model allows us to conduct experiments in a scalable manner. (b) The open circuit potential of a battery increases with the state of charge (amount of energy left) of the battery. (c) The internal resistance of a battery decreases with the state of charge.



and Maccor 4200¹² battery cycling and testing hardware for measuring the battery properties.

The model takes the initial SoC, OCP versus SoC, resistance versus SoC, concentration resistance, and plate capacitance to emulate a battery. At each time step, based on the SoC, it estimates OCP, and resistance. Using the updated values, it calculates the values for the SoC after the time step.

We build the battery model using a few batteries and validate the models against other batteries of the same type. The validation results for one of the batteries are shown in Figure 8. The results show that our model is accurate to 97.5%. We modeled 15 batteries in total: two of Type 4, two of Type 3, eight of Type 2, and three more of other types (refer to Figure 1a).

We implement a simple software layer that takes the input power requirement and splits it across a given number of batteries according to the power policies set by an OS. The model and the SDB emulator are integrated into the OS using 4800 lines of code across modules written in C#.

We focus on two hardware platforms: a tablet and a watch. The tablet is a “2-in-1” development device with Intel Core i5 CPU, 4GB DRAM, 128GB SSD, and 12 inch display. The watch is a Qualcomm Snapdragon 200 development board with hardware similar to several smart-watches. Devices are instrumented to obtain fine grained (100 Hz) power-draw measurements. The power-draw is then fed into the emulator to calculate the energy drawn from the batteries.

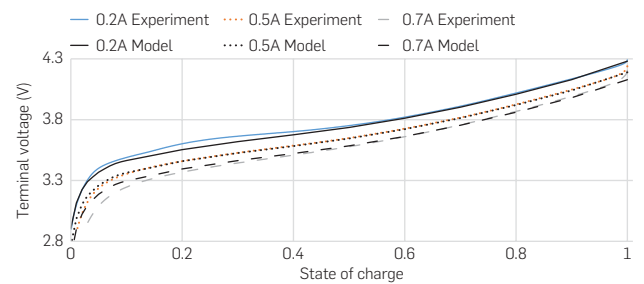
5. SDB APPLICATIONS

In this section, we describe two scenarios that benefit from using SDB with heterogeneous batteries. We also show how SDB policies can be customized for the different scenarios, and demonstrate the benefits of integrating future workload knowledge in the SDB system.

5.1. Adopting flexible batteries

Flexibility and bendability are important structural properties for wearable devices, for example, a watch-strap that is flexible and bendable tends to be easier to wear. Coincidentally, there are a few emerging battery chemistries that enable bendability. The bendability, unfortunately, comes at the cost of other battery properties. Such batteries use a solid (rubber-like) electrolyte in place of a traditional liquid (polymer) electrolyte. Unfortunately, the solid

Figure 8. Validating the model against battery testing hardware reveals that our model is 97.5% accurate.



(elastic) state of the electrolyte increases the resistance for the Li-ions and therefore, such batteries have higher internal losses. Several prototype bendable batteries we tested are excellent at handling low power workloads but often are very inefficient for high power workloads.

SDB can enable a scenario where a small traditional Li-ion battery in smart-watches is augmented with bendable batteries. This helps design better wearables that utilize the strap space to increase capacity but are still able to execute high power workloads like GPS tracking while running and cycling. The reduction in the size of the rigid Li-ion battery also allows for the design of a less bulky watch body.

The bendability of the battery in the strap is a boon, but its low efficiency is a bane that has to be intelligently managed to maximize effective battery life of the device. It is important to preserve energy in the efficient battery for times when the user is expected to perform power-intensive tasks. For example, the user may exercise, run or bicycle during certain times of the day, which all require high power. Therefore, the SDB policies should preserve the efficient battery for such times.

Since smart-watch usage will vary across users, we compare two extreme parameter values to demonstrate the benefits of SDB: One that minimizes instantaneous losses by drawing appropriate amounts of power from both the batteries and one that draws higher amounts of power from the inefficient battery to conserve the efficient battery.

Figure 9 demonstrates the setting and the results. We use a 200mAh Li-ion battery in combination with a

200mAh bendable battery for the setting. For a typical user who spends the entire day checking messages on his smart-watch and goes for a run in the evening, we plot the workload and the instantaneous losses in the batteries. We find that the latter method minimizes the total losses and therefore increases overall battery life by over an hour. These results provide evidence that mobile OSes that are aware of a user’s day-to-day schedule may be able to provide better battery life by setting the right parameter. On the other hand, it is interesting to note that if the user had not gone for a run then the first policy would have given better battery life suggesting that the knowledge of an impending workload can help save energy in heterogeneous battery settings.

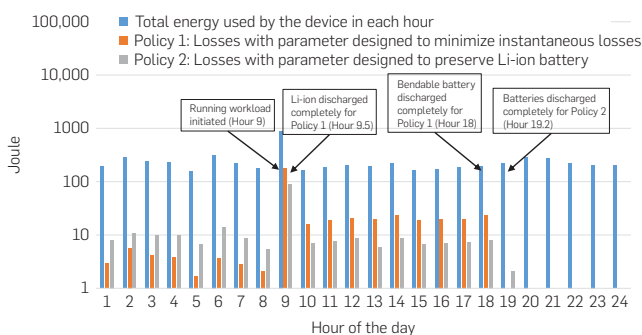
5.2. Battery management for 2-in-1s

2-in-1 devices are tablets that have a detachable keyboard. Some such devices have another battery under the keyboard. In such a setting, there are two batteries exposed to the OS, often with different capacities but the same internal chemistry—traditional Li-ion. However, efficiency of the battery in the base is less as it is used solely to charge the battery in the tablet. Significant amount of energy is lost in charging the internal battery with the external one, yet the reason why device manufacturers have chosen this route is to simplify design.

SDB via the OS can improve the battery life of a combined internal and external battery by understanding user behavior and expectations. The power drawn from an external battery can either be used toward running the system, for charging the main battery or both. For a user who rarely unplugs an external battery, the better solution would be to draw power simultaneously from both batteries as the internal losses are proportional to the square of the current (resistive losses = I^2R). Splitting the power draw across the two batteries, therefore, reduces the internal losses and increases the energy delivered to the system.

However, this strategy may not be ideal for a user who mostly operates in tablet-only mode. For such users, it makes more sense to draw as much power for as long as possible from the external battery to handle system load and also for charging the internal battery.

Figure 9. Fixed priority levels are bad. Priority levels have to be changed according to expected user schedules and workloads.



The OS sets a low parameter value for times when the external battery is expected to be plugged in for longer duration while high parameter values are for times when the external battery is plugged during battery crises.

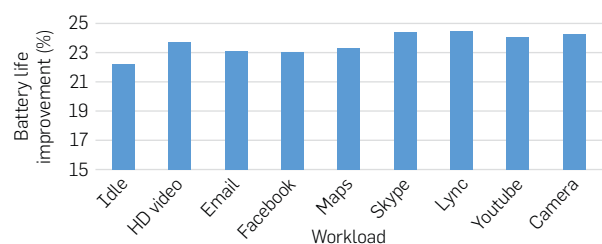
Figure 10 shows the comparison of two extreme parameters for various application workloads on a development 2-in-1 device with two equal sized traditional Li-ion batteries. Results show that the parameter causing simultaneous power draw from both batteries provides 22% more battery life than the parameter that causes one battery to charge another. However, this gain is not realizable for a user who only keeps the base with the secondary battery plugged in for short periods of time. The OS must, therefore, learn, predict and adapt to user behavior to set appropriate parameters.

6. CONCLUSION AND FUTURE WORK

Device requirements are typically hard to meet with a single battery since these requirements are often in conflict with each other. We present the SDB system that allows a device to use multiple heterogeneous batteries, and get the best of all of them. The SDB hardware is designed to be low cost, and provides rich functionality to the OS. The SDB APIs allow an OS to dynamically route charge to, and from the batteries based on application workload such that the overall goals (battery life, cycle count, fast charge, etc.) are met. We show several new scenarios that can be enabled with SDB, and demonstrate its feasibility using a prototype, and detailed emulations.


Moving forward, we are taking the SDB work in two main directions. First, we are tying personal assistants like Siri, Cortana, and Google Now understand user behavior and the user’s schedule and by using this information, an OS can perform better parameter selection. For example, if the user’s profile suggests that the user plays video games in the evening, then it SDB could preserve a higher power-density battery for that workload. Second, we are working on additional devices that would benefit from this technology, such as drones, smart glasses, and electric vehicles (EVs). Each would require a different combination of battery chemistries, and the SDB logic might be different too. For example, we are building on the techniques proposed for Hybrid Energy Storage in the Grid^{7,10} or hybrid power sources in Data Centers,⁸ to improve the lifetime of EVs. An EV’s NAV system could provide the vehicle’s route as a hint to the

Figure 10. Drawing power simultaneously from internal and external batteries is more energy efficient than depleting the external battery for conserving and charging the internal one.



SDB Runtime, which could then decide the appropriate batteries based on traffic, hills, temperature, and other factors. Our preliminary analysis shows that SDB might help these systems achieve tradeoffs that until now were considered to be at odds with each other.

Acknowledgments

Insightful comments from Jon Crowcroft, Srinivasan Keshav, Matthew Lentz, and Vasuki Narasimha Swamy greatly improved the final version of this paper. 

References

1. Altium Designer. <http://www.altium.com/altium-designer/overview>.
2. Arbin BT-2000 Battery Testing Equipment. <http://www.arbin.com/products/battery>.
3. Chen, M., Rincon-Mora, G.A. Accurate electrical battery model capable of predicting runtime and IV performance. *IEEE Trans. Energy Convers.* 21, 2 (2006), 504–511.
4. Chiasserini, C.-F., Rao, R.R. Energy efficient battery management. *IEEE J. Sel. Areas Commun.* 19, 7 (2001), 1235–1245.
5. Erdinc, O., Vural, B., Uzunoglu, M. A dynamic lithium-ion battery model considering the effects of temperature and capacity fading. In *Proceedings of the IEEE International Conference on Clean Electrical Power* (Capri, Italy, June 2009).
6. Gao, L., Liu, S., Dougal, R.A. Dynamic lithium-ion battery model for system simulation. *IEEE Trans. Compon. Pack. Technol.* 25, 3 (2002), 495–505.
7. Ghiassi-Farrokhfal, Y., Rosenberg, C., Keshav, S., Adjaho, M.-B. Joint optimal design and operation of hybrid energy storage systems. *IEEE J. Select. Areas Commun.* 34, 3 (Nov. 2016), 639–650.
8. Govindan, S., Sivasubramaniam, A., Urgaonkar, B. Benefits and limitations of tapping into stored energy for datacenters. In *International Symposium on Computer Architecture* (2011).
9. He, H., Xiong, R., Zhang, X., Sun, F., Fan, J. State-of-charge estimation of the lithium-ion battery using and adaptive extended Kalman filter based on an improved Thevenin model. *IEEE Trans. Veh. Technol.* 60, 4 (2011), 1461–1469.
10. Kim, Y., Chang, N. *Hybrid Electrical Energy Storage Systems Design*. Springer International Publishing, Cham, Switzerland, 2014, 19–25.

11. LTSpice: Linear Technologies Simulator Program with Integrated Circuit Emphasis.

12. Maccor 4200 Battery Testing Equipment. <http://www.maccor.com/Products/Model4200.aspx>.

Anirudh Badam, Ranveer Chandra, Jon Dutra, Julia Meinershagen, and Bodhi Priyantha ([anbadam, ranveer, jodutra, juliam, bodhip]@microsoft.com), Microsoft, Redmond, WA.

Pan Hu (lghupan@gmail.com), University Massachusetts Amherst, MA.

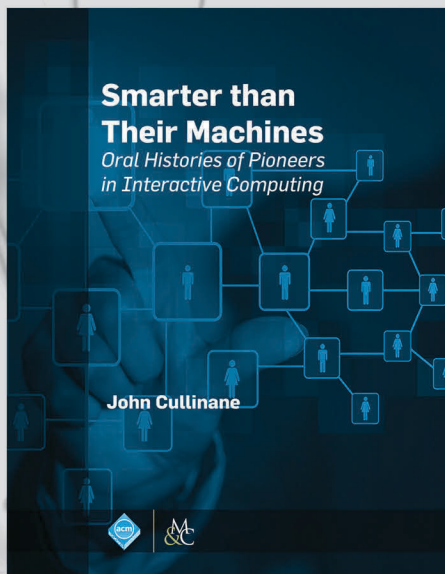
Thomas Moscibroda (moscitho@microsoft.com), Microsoft, Beijing, China.

Anthony Ferrese (anthony.ferrese@gmail.com), Tesla Motors, Palo Alto, CA.

Evangelia Skiani (valia@ee.columbia.edu), Department of Electrical Engineering, Columbia University, New York, NY.

Steve Hodges (shodges@microsoft.com), Microsoft, Cambridge, U.K.

© 2016 ACM 0001-0782/16/12 \$15.00



A personal walk down the computer industry road. BY AN EYEWITNESS.

Smarter Than Their Machines: Oral Histories of the Pioneers of Interactive Computing

is based on oral histories archived at the Charles Babbage Institute, University of Minnesota. These oral histories contain important messages for our leaders of today, at all levels, including that government, industry, and academia can accomplish great things when working together in an effective way.



ISBN: 978-1-62705-550-5 DOI: 110.1145/2663015

<http://books.acm.org>

<http://www.morganclaypoolpublishers.com/acm>