TrustDER: Trusted, Private and Scalable Coordination of Distributed Energy Resources

Agreement DE-OE0000919

Battery Abstraction Layer Design Document

Principal Investigators:
Professor Ram Rajagopal, Stanford University
ramr@stanford.edu

Professor Philip Levis, Stanford University
pal@cs.stanford.edu

Submitted to:
U. S. Department of Energy, National Energy Technology Laboratory

DOE Project Officer:
Ms. Carol Painter, carol.painter@NETL.DOE.gov

Battery Abstraction Layer

Philip Levis, Sonia Martin, Nicholas Mosier, Yancheng Ou, Ram Rajagopal, and Oskar Triebe
June 29, 2021

Abstract

The Battery Abstraction Layer (BAL) is a software abstraction for battery energy storage. Its intended use case is battery systems being used as distributed energy resources (DERs) in the electric grid. The core idea in BAL is virtualizing batteries through software, such that they can be flexibly managed as a resource. Using the BAL, a system administrator or user can aggregate multiple batteries into a single battery, or partition a single battery into multiple batteries. The BAL defines the semantics and behavior of these operations as well as resource management policies for battery partitioning.

This document describes the Battery Abstraction Layer, its abstractions, and their semantics. It describes a software API for controlling and monitoring BAL devices. It describes how a BAL device can be represented in the IEC 61850 standard for communication networks and systems for power utility automation.

1. Introduction

Energy storage is an increasingly important component of the electrical grid. Energy storage allows grid operators as well as consumers to time-shift generation and loads away from peak periods. Large-scale energy storage (e.g., pumped-storage hydropower) has shown that centralized approaches can reduce peak loads. As consumers increasingly deploy solar and other renewable energy sources, the benefits of time shifting the generated energy has led to increased interest and deployment of local, small-scale storage (e.g., home battery units). Furthermore, as battery manufacturing continues to scale up, and as more electric vehicles connect to the grid, other battery storage configurations and usage models will appear.

Managing these distributed battery systems will be a tremendous challenge for the grid. Tapping their resources and benefits could improve reliability, reduce costs, and increase the value and deployment of renewable energy sources. But to be able to tap and use these resources, we must first be able to manage them.

This document describes a proposal for managing, organizing, and monitoring distributed battery resources. The key idea in this proposal is that if we can abstract away the behavior and details of specific batteries through a simple *virtual battery* interface, then we treat these batteries like fungible resources that can be partitioned, combined, and composed to meet the needs of many use cases, business models, and user requirements. It proposes the concept of a *logical battery* which behaves like a physical battery but may be an *aggregate* of many batteries, a *partition* of a battery, or complex compositions of aggregates and partitions. The leaves of a battery topology are *physical batteries,* actual battery storage devices; the logical batteries can form a complex topology of resource management, providing greater reliability and flexibility than purely physical resources.

The principal design and research question in logical batteries is defining how aggregating and partitioning batteries behave and what values they report. For partitioning batteries, the principal complication is what happens when the underlying battery's values unexpectedly fluctuate. This question is especially important because batteries are controlled and accessed over a network. As home battery systems typically use consumer Internet connections and WiFi routers, their connectivity is unreliable and batteries can become unavailable at unpredictable times. The BAL proposes three policies for partitioned

batteries: *proportional*, in which each partition receives a fixed portion of the underlying battery, *tranched*, in which surpluses beyond the expected values go to the highest tranches first and deficits are taken from the lowest tranches first, and *reserved*, in which both surpluses and deficits go to the lowest tranches first.

In aggregating batteries, the principal question is what values an aggregating battery should report. Simply reporting the sum of the underlying batteries' values is incorrect, as it can cause an aggregating battery to report performance it cannot achieve. Instead, an aggregating battery has the sum of the underlying capacities (in terms of energy), but its maximum current is determined by the minimum C-Rate of its constituent batteries.

The rest of this document describes these results in greater detail. Section 2 describes the Battery Abstraction Layer. Section 3 describes how aggregate batteries behave. Section 4 describes how partitioned batteries behave. Section 5 describes a Python-based API for the Battery Abstraction Layer. Section 6 describes how the Battery Abstraction Layer can be represented in IEC 61850 as logical nodes. Section 7 discusses open questions and areas of future work.

2. Battery Abstraction Layer

Enabling innovative solutions for networked batteries requires having functional abstractions of how those batteries can be used, managed and controlled. The traditional approach to achieving this is to design abstractions for each domain or use case: there would be an interface to DER for industrial users wishing to reduce peaks, an interface for arbitrage, and an interface for microgrid operators, and every other use case.

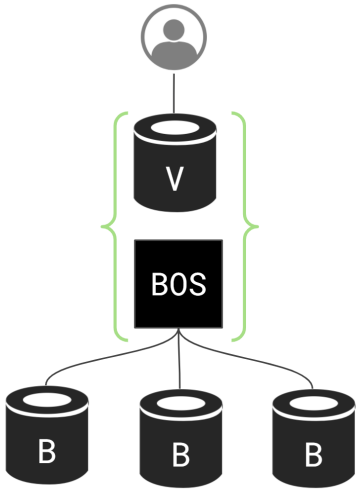
The Battery Abstraction Layer takes a different approach: distributed energy resources have a single, common, standard interface that is general enough to use in the majority of use cases. By having a single, common interface, energy resources are not siloed into a particular use case and can be easily repurposed across them. A factory, for example, if operating well below capacity, could apply its energy storage to a larger arbitrage or demand-response market, because those applications and operators would use the same interface. Towards this end, the BAL draws inspiration from computer systems, where the operating system (OS) provides a common programming abstraction of hardware resources, while hardware manufacturers build devices with software drivers that provide that abstraction. This makes it possible for third parties to build applications on the OS which are agnostic to the underlying hardware. In line with these principles, we propose the Battery Operating System (BOS).

The key resource that BOS introduces and manages is a *logical battery.* There are two kinds of logical batteries: physical and virtual. A **physical battery** is what we think of today when we think of distributed energy resources. It represents physical energy storage, controlled and managed by a networked battery management system (BMS). The BMS provides information and control on the state and activity of the battery.

A **virtual battery** is the key new concept in BOS. Virtual batteries allows BOS to aggregate multiple batteries into a single, larger virtual one, or to partition the resources of a single battery into multiple virtual batteries of different sizes, which in sum are equal to the original battery. An analogy for virtual batteries is hard disk management in computer systems: multiple physical drives can be aggregated (via RAID) into a single logical disk. A single logical disk can be partitioned into multiple logical volumes that users can like a regular hard drive. Just as a computer OS manages the virtualization of disks, systems using the BAL can manage virtual batteries, physical batteries, and what they can do.

Virtualization allows a system to allocate and manage resources according to each user's needs. There are two types of virtual batteries: aggregate and partitioned. An **aggregate battery** takes several batteries and presents them as a single, larger battery. A system may, for example, aggregate several smaller batteries as one large battery for demand response purposes. A **partitioned battery** takes a single battery and splits it into multiple smaller ones. For example, a home-owner's association may use

BAL to pool funds to acquire one large battery array, aggregate it, then partition it for the individual homes according to their fund contribution. As shown in Figure 1, virtualization allows systems using BAL to treat a battery the same without needing to know if it is actually one battery, part of a larger battery or many smaller batteries.



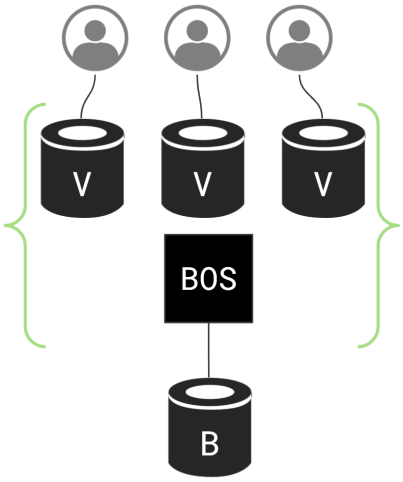
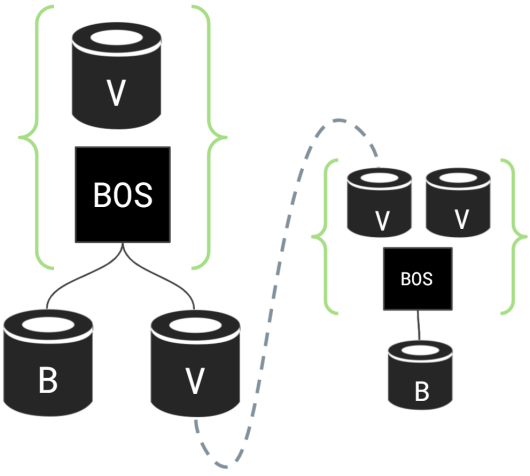


Figure 1a: Aggregate Battery *Figure 1b: Partitioned Battery*

Figure 1: Virtual batteries allow a Battery Operating System (BOS) to aggregate multiple batteries into a single logical battery or partition a battery into multiple logical batteries. These source batteries B can themselves be logical batteries, or physical ones.



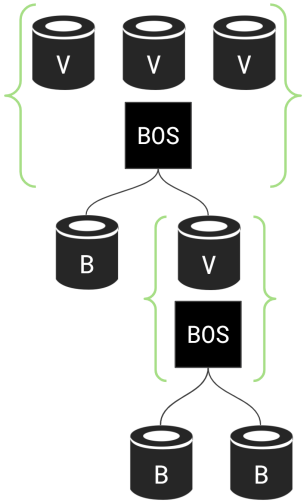


Figure 2a *Figure 2b*

Figure 2a shows an example of a networked virtual battery that allows BOS to interact with a battery using the BAL API over a network, e.g., to a logical battery on another BOS node. Figure 2b demonstrates the ability to combine the two concepts.

3. Aggregate Batteries

An aggregate battery takes multiple batteries (logical or physical) and composes them into a single, larger battery. Three major questions arise in aggregate batteries:

1. What current ranges and capacity should an aggregate battery have, based on its constituent batteries?
2. If constituent batteries have unbalanced states of charge, how should the aggregate battery report this?
3. How does an aggregate battery advertise its location or connection point?

We examine each in turn.

3.1 Reporting Discharge and Capacity

Consider the following example: we have two batteries, A and B, with the following properties. We use ampere-hours as the units of capacity to simplify explanation of the issues that arise.

Battery	Discharge	Capacity	State of Charge
A	60A	110Ah	100%
B	40A	40Ah	100%

Table 1: Example battery configuration

If we combine these two batteries into a single, larger battery C, what values should BAL report? One simple solution is to simply sum the values:

Battery	Discharge	Capacity	State of Charge
A	60A	110Ah	100%
B	40A	40Ah	100%
C (A+B)	100A	150Ah	100%

Table 2: Aggregate battery reporting a simple sum of values. The report is inaccurate: battery C cannot discharge at 100A for 1.5 hours.

This aggregate battery C reports having a maximum current of 100A and a capacity of 150Ah. This means that it has a C-Rate of 0.67 (100A/150Ah). However, if battery C is instructed to discharge at 100A, this will cause both A and B to discharge at the maximum discharge rates: A will discharge at 60A and B will discharge at 40A. After one hour, B will fully discharge. The maximum current then drops to 60A and 50Ah remains. Therefore, reporting a discharge rate of 100A is misleading: the aggregate battery can discharge at 100A for only one hour, at which point its maximum discharge drops to 60A.

However, this discontinuity is dependent on the discharge rate. Smaller discharge rates can operate continuously over the entire charge. For example, if battery C is instructed to discharge at 30A, then it can discharge for 5 hours at 30A: battery A discharges 22A and battery B discharges 8A.

One approach to solve this problem is to report a current curve, based on capacity or some other property. For example, battery C could report several discharge curves based on a number of discharge

rates. As Section 6 discusses, such complex data models are not easily supported in existing standards. Therefore, BAL takes a simpler approach, of reporting a scalar value for the maximum discharge rate.

The discharge rate for an aggregate battery is derived from the C-Rate of its constituent batteries. The C-Rate of an aggregate battery is the minimum C-Rate of its constituent batteries. Its capacity is the sum of its constituent batteries, and its maximum discharge rate is derived from the C-Rate. In our example, this means that battery C has a maximum discharge of 81A. This comes from the fact that the maximum C-Rate of battery A is 0.56 (60A/110Ah). If battery A discharges at its maximum rate, it will discharge in just under two hours. Therefore, the aggregate battery limits battery B such that it will discharge in the same amount of time, at 21A.

Battery	Discharge	Capacity	State of Charge	C-Rate
A	60A	110Ah	100%	0.56
B	40A	40Ah	100%	1
C (A+B)	81A	150Ah	100%	0.56

Table 3: Aggregate battery reporting based on C-Rate. The report is accurate: battery C can discharge at 81A for 110 minutes.

3.2 Reporting with Unbalanced States of Charge

The second challenge that arises in aggregate batteries: what values should BAL report if their charges are not balanced? In the steady state and regular operation, BAL keeps the state of charges of constituent batteries balanced, because doing so allows an aggregate battery to provide a consistent current over its charge. However, when an aggregate battery is first created, the constituent batteries may not have identical states of charge. Also, if one of the constituent batteries loses network connectivity (see below), it may cease charging and discharging with the others and so have its state of charge diverge.

We consider three options:

1. **Variable current:**Use the minimum C-rate computation to compute the current of each battery. The advantage of this approach is that it is simple and adds no further logic. The disadvantage is that the discharge and charge currents of an aggregate battery change significantly as its charge equalizes. The state of charge of the aggregate battery is the weighted sum of the constituent batteries based on their maximum charge.
2. **Offline:** Take an aggregate battery offline when its state of charge is not balanced and charge or shift charge to balance it. The advantages of this approach are that it is simple and requires no logic, and the reported properties of a battery do not shift significantly over time. The disadvantage is that failures of constituent batteries can cascade into larger failures, limiting aggregation and reliability.
3. **Variable charge:** Fix the discharge current of the aggregate battery and derive a reported aggregate state of charge based on how long the system could sustain this discharge. This has the advantage that the discharge of an aggregate battery is constant. It has the disadvantage that the state of charge does not reflect the actual state of charge of the constituent batteries: charging the aggregate battery requires less energy than what the state of charge and capacity indicate.

The BAL uses the variable current approach. The offline approach is too fragile to network failures, which can cause cascading failures in large battery topologies. The variable charge approach leads to much greater swings and uncertainty than variable current, leading to variable current being a better choice. Consider the following example, using variable current:

Battery	Discharge	Capacity	State of Charge	C-Rate
A	60A	110Ah	9%	0.56
B	40A	40Ah	100%	1
C (A+B)	50A	150Ah	33%	1

Table 4: Unbalanced aggregate battery reporting using variable current. Because A's state of charge is 9%, battery C can only provide 50A, rather than the 81A in Table 3.

Battery C can report a higher C-Rate than before, because battery A is no longer limiting the discharge of the battery. However, its current is capped to 50A so that battery A can discharge its 10Ah over an hour.

With variable charge, the batteries would report a state of charge of 13% because, with a fixed maximum discharge of 81A, the aggregate battery will last longest with 40A from battery B and 41A from battery A. If 41A come from battery A, it can discharge for just under 15 minutes (10Ah). If battery B discharges that long, it can discharge 9.75Ah, for a total of 19.75Ah, or 13% of 150Ah.

Battery	Discharge	Capacity	State of Charge	C-Rate
A	60A	110Ah	9%	0.56
B	40A	40Ah	100%	1
C (A+B)	81A	150Ah	13%	0.56

Table 5: Unbalanced aggregate battery reporting using variable charge. Because A's SoC is 9%, to maintain an 81A discharge battery C can only report 13% SoC.

Because variable current is more stable (the current of the matched batteries remains stable), BAL uses variable current.

To address the uncertainty that variable current introduces, the BAL reports two sets of values for a virtual battery: the *expected values* and the *current values.* The expected values report the expected charge, discharge and capacity if the virtual battery is operating optimally. The current values report potentially lower values if constituent batteries are offline or charge is not balanced.

3.3 Reporting Location/Connection Point

The final issue that arises in aggregate batteries is how to report the location and tolerances of an aggregate battery connection point. This is ongoing work.

Location can be an important consideration when to keep grid elements within voltage bounds or maintain other safety limits. In addition to location, information about the connection point include the nominal values and bounds for VAR, voltage, frequency, and wattage.

Reporting location is fundamentally in tension with virtualization: the goal of virtualization is to abstract away physical properties and break brittle assumptions on them. Location, however, is

fundamentally physical and it represents a point in space. If the constituent batteries of an aggregate battery are highly distributed, they cannot be easily summarized as a single location.

We propose that the BAL reports aggregate location as the center point of all of the constituent connection points, computed with the Floyd-Warshall algorithm.[1] This approach means that virtual batteries which aggregate geographically dispersed batteries will appear to be in the “center” of the grid topology, combined with an indication that this is a center position (i.e., the top of a connection point hierarchy).

We are currently exploring and evaluating different approaches for reporting bounds and tolerances. Our current approach involves a “down-up” query model, where values propagate down from an aggregate to its constituent batteries, then back up to report actual values.

4. Partitioned Batteries

A partitioned battery takes a single logical battery and divides it into multiple, smaller batteries. Because of normal variations and because the underlying battery may be an aggregate battery, the values of the underlying battery can change over time even in the absence of charging and discharging. When this battery is partitioned, this raises the question and policy of how these variations are reflected in the partitions. Furthermore, when some partitions request charging and others request discharging, the partitioner needs to map these requests onto the underlying battery and dynamically account for changes in charge.

4.1 Partitioning Policies

Partitioned batteries are defined in terms of fractions of the expected value of a battery. For example, if a battery has an expected maximum current of 80A, this can be partitioned into 25% and 75% as two batteries with currents of 20A and 60A. This proportional splitting is uniform across the properties. The BAL provides three policies for partitioned batteries:

1. In a **proportional** battery, the partitions are strict fractions of the underlying battery. Any variations in the values of the underlying battery are proportionally reflected in the partitions.
2. In a **tranched** battery, the partitioned batteries are placed in an ordering of tranches: there is a top (level A) tranche, a second tranche (level B), and more tranches as the battery is more finely partitioned. When values of the current values of a battery go *above* the expected value, these increases are first given to the highest tranche. However, when the current values of a battery go *below* the expected value, these reductions are first taken from the lowest tranche.
3. In a **reserved** battery, the partitions are placed in an ordering, as with a tranched battery. However, increases and decreases both affect the bottom partition first. In a reserved battery, the higher partitions have more reliable and dependable values.

Table 6 shows the difference between these three partitioning policies, with capacity as the variable of interest. A single battery A is partitioned into P1, P2, and P3. P1 is the top partition and P3 is the bottom partition.

[1] https://dl.acm.org/doi/10.1145/367766.368168

Battery	Charge	Fraction	Proportional	Tranched	Reserved
A: Expected	100				
A: Actual	90	90%			
P1	50	50%	45	50	50
P2	30	30%	27	30	30
P3	20	20%	18	10	10

Table 6: If the actual charge of a partitioned battery is lower than its expected charge, the type of partition determines how this deficit appears: in both the tranched and reserved batteries the deficit is borne by the bottom battery P3.

Battery	Capacity	Fraction	Proportional	Tranched	Reserved
A: Expected	100				
A: Actual	110	110%			
P1	50	50%	55	60	50
P2	30	30%	33	30	30
P3	20	20%	22	20	30

Table 7: If the actual charge of a partitioned battery is higher than its expected charge, the type of partition determines how this surplus appears. In the reserved battery the surplus is taken by the bottom battery P3, while in the tranched battery the surplus is taken by the top battery P1.

Applied surpluses and deficits are bounded by the 0 and the maximum values of the partitions. For example, suppose, in Table 7, the total maximum charge (capacity) is 125. P3 is 20% of this, so its maximum charge (capacity) is 25. If the actual charge of A is 110, then the surplus will be allocated so:

Battery	Capacity	Fraction	Maximum	Reserved
A: Expected	100			
A: Actual	110	110%		
P1	50	50%	62.5	50
P2	30	30%	37.5	35
P3	20	20%	25	25

Table 8: Surplus charge on partitioned batteries cannot go above the maximum charge of a partition. In this example, the excess 10 charge units are first applied to P3, raising it to 25. The remaining 5 surplus units are applied to P2.

The tranched and reserved policies can be useful in different economic models and markets. Furthermore, these policies can compose. One can construct complex topologies of batteries, with tranches of aggregates of tranches.

4.2 Partition Accounting

A second issue that arises in partitioned batteries: partitions can request discharging and charging simultaneously. Consider, for example, the following partitioned battery A, partitioned into P1 and P2. P1 is discharging at 40A and P2 is charging at 60A.

Battery	Capacity	State of Charge	Charge	Discharge
P1	80Ah	100%		40A
P2	120Ah	16%	50A	
A	200Ah	50%	10A	

Table 9: Behavior of Battery A when its two partitions (P1 and P2) request both charging and discharging. The underlying battery is the sum of the charge and discharge, while State of Charge flows from P1 to P2.

In this example, the underlying battery A charges at 10A (the sum of the charge and discharge requests). The 40A of discharge from P1 is effectively absorbed by P2's request. This means that the underlying battery only requests 10A of charging. However, the state of charge on P1 and P2 need to change to reflect this virtual energy flow from P1 to P2. Note that this change in their state of charge is entirely in software: the BAL accounts for this virtual flow. In 2 hours, P1's state of charge will decrease to 0% and it will stop discharging. This 80Ah of charge

will transfer to P2. After 2 hours, P2's state of charge will be 100%, having drawn 20Ah from external charging and 80Ah from P1.

This virtual transfer of charge requires that the entity providing virtualized batteries can accordingly bill its clients. In the above example, P2's charging cost is what will pay P1 for discharging. This implies that the owner of P2 must pay the owner of P1 at least as much as P1 expected to receive from the grid. Exploring the exact billing and financial agreements necessary for these virtual charge flows is an area of future work and discussion. For example, if P1 promised to discharge as part of a day-ahead market, it is important that the owner of P1 isn't able to simply virtually transfer the charge to P2 (at no cost) and claim its responsibilities were met. One approach we are discussing is for both charges to go through the utility directly, such that a virtual transfer does involve a flow of funds through the utility directly.

5. Battery Abstraction Layer API

BOS provides a Battery Abstraction Layer API as a software programming interface for interacting with batteries. The API is identical for virtual and physical batteries, making them indistinguishable to software. The BAL follows three design principles:

Flexible: As the BAL defines how many different pieces of software will interact with batteries, it must provide a general and flexible API. A flexible API that allows many different use cases will be highly reusable and provide a common basis for more complex services. It should provide complete-yet-simple functionalities of a physical battery management system, which includes retrieving basic information like voltage, current, state of charge etc, as well as controlling the discharging/charging of a battery. This flexible API should be implementable for both physical and virtual batteries.

Simple: The API must be simple. Complex APIs are hard to implement and difficult to write software for. If the API is simple, it is also easier for it to be general, as its core abstractions do not depend on particular features or capabilities.

Atomicity: To support correct and stable services being built on top of it, the BAL must allow software to atomically (as a single, indivisible action) request multiple values at once. This is critical so that services do not try to (incorrectly) compute on values obtained at different moments in time. For example, software must be able to atomically request both current and voltage simultaneously. If the voltage and current are measured at different times, the power calculation can be incorrect. For example, suppose after the voltage is sampled there is a huge spike in draw, leading to a high current and a voltage drop. Multiplying the high current with the high (previous) voltage would indicate the power output is higher than it is.

The Battery Abstraction Layer contains the following two functions:

1. `GetStatus()`: retrieves a snapshot of the logical battery status, which returns a structure containing a snapshot of the following information.
    - voltage,
    - current,
    - maximum current,
    - state of charge,
    - maximum charge,
    - the timestamp of the snapshot, represented as an interval.
2. `SetCharge()`/`SetDischarge()`: sets the amount of current flow into (charge) or out from (discharge) the battery.

This `GetStatus()` function satisfies the atomicity principle, as all of those values are returned on one call of GetStatus. The timestamp interval is to tackle with different stalenesses of data from different batteries when the virtual battery is an aggregate of multiple underlying logical batteries. Each physical battery request is an atomic set of results, but virtual batteries may aggregate the results of more than one physical battery.

For physical batteries, BOS includes driver software that converts these calls into messages or commands to the physical battery, for example over a serial port or network connection to a battery management system (BMS). Virtual batteries are entirely software.The next two sections describe how virtual batteries set charge or discharge rates.

5.1 Charge/Discharge on Aggregate Batteries

BOS maps operations on virtual aggregate batteries into their constituent batteries. The charge and discharge amounts are distributed across the constituent batteries based on their contribution to the aggregate value. For example, consider the batteries in Table 3 (repeated here):

Battery	Discharge	Capacity	State of Charge	C-Rate
A	60A	110Ah	100%	0.56
B	40A	40Ah	100%	1
C	81A	150Ah	100%	0.56

Table 10: Aggregate battery reporting based on C-Rate. The report is accurate: battery C can discharge at 81A for 110 minutes.

Battery C's discharge of 81A consists of 60A from A and 21A from B. Therefore, if software calls SetDisharge(50) on battery C, BOS will tell battery A to discharge at 50A x (60/81) and battery B to 50A x (21/81).

5.2 Charge/Discharge on Partitioned Batteries

BOS combines change and discharge operations on partitioned batteries using a simple sum. The resulting value is passed to the underlying battery. For example, suppose we have the following partitioned battery:

Battery	Fraction	Capacity	Charge	Discharge
Source		200Ah	80A	100A
P1	50%	100Ah	40A	50A
P2	30%	60Ah	24A	30A
P3	20%	40Ah	16A	20A

Table 11: Example proportional battery.

If software tells P1 and P2 to both discharge at 20A, BOS calls SetDischarge on the source battery with 40A. However, if software tells P1 to charge at 20A and P2 to discharge at 20A, then BOS

stops charging/discharging the source. Instead, over time, BOS changes its accounting of charge for P1 and P2, shifting charge from P2 to P1.

This implies an economic exchange. E.g., P2 is discharging to earn income by providing power, while P1 is expecting to pay for the power it draws. The entity running BOS must interact with the local utility as a virtual meter, informing the utility that P1 is drawing 20A whole P2 is discharging 20A.

5.3 Battery Topology Configuration

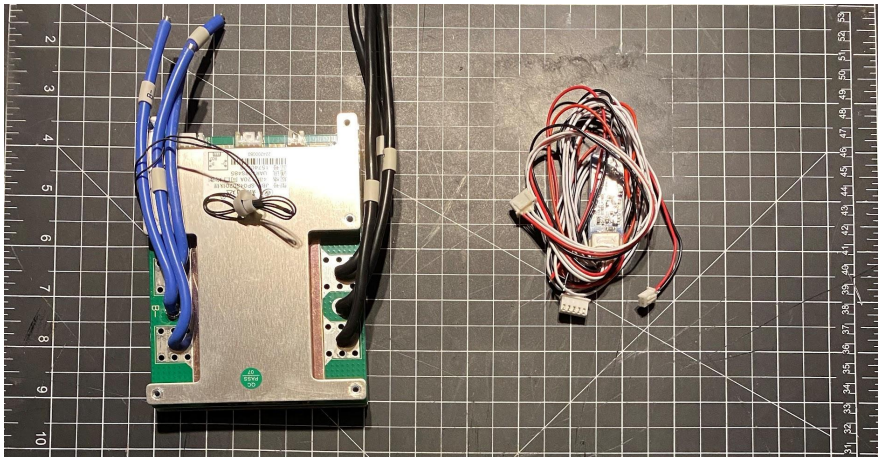
The BOS function family `make_{battery, aggregator, splitter}()` provides the mechanism for constructing a topology of batteries that all use a common Battery Abstraction Layer (BAL).

- `make_battery(name, kind, interface)` -- create a new battery with the given name on the given interface. The interface can be local or remote (UART, BLE, TCP, etc.); the battery can be virtual or physical.
- `make_aggregator(name, [batteries...])` -- aggregate a list of batteries into one virtual battery with the given name.
- `make_splitter([(name, info)...], name)` -- split a battery with the given name into *n* different batteries.

5.3 Example Physical Battery Driver: JBD BMS

To show how the BAL is implemented on top of a physical battery, we consider the example of a battery management system (BMS) manufactured by JBD, the SP04S020. This BMS can be controlled and queried over either a UART or BLE interface. The BMS provides a rich interface, with many controls and data.

To implement GetStatus(), our implementation reads the Basic Info register of the BMS, which contains a large set of data values, including voltage, current, state of charge, pack cells, FET status, and maximum charge. It translates a subset of the data in this data structure into the BAL's data structure and returns the values.



To implement SetCurrent(current), our implementation controls the MOSFET on/off status of the BMS by writing to the FET status register -- if the target current is positive, turn on the discharging MOSFET and shut off the charging MOSFET, and vice versa. If the current is zero, the implementation disables the FET. The BMS does not have a current regulator. We are working to extend it to have one.

5.4 Example BOS Software

We have an initial prototype implementation of the BAL API as described above. The API operates in Python. It allows software to construct battery topologies. In addition to virtual and physical batteries, this implementation adds the abstraction of a *networked battery*, which is a way to control a BAL device over a network. This allows a BOS instance to construct virtual batteries from logical batteries that exist on other BOS nodes. For example, one BOS node that has three BMSes which it combines into a single aggregate battery can provide access to this aggregate battery over a network. Another BOS node can create a networked battery to take control of this aggregate battery, and partition it into multiple batteries. Networked batteries allow BOS to provide distributed battery resources.

6. Representing a BAL Device in IEC 61850

International Electrotechnical Commission (IEC) 61850 is an international standard defining communication protocols for intelligent electronic devices at electrical substations. IEC 61850 defines object models, data models, and mappings to communication protocols such as HTTP. The Battery Abstraction Layer is a software API: it defines methods and data structures that software running on a microprocessor or microcontroller can use. This section describes how the BAL API can be mapped to IEC 61850 data models: it outlines how a software service can respond to and generate IEC 61850 messages by interacting with a BAL device.

In IEC 61850, the standard battery energy storage object is a ZBAT node, defined in IEC 61850 7-420, Section 8.2.2. A ZBAT node has 4 Mandatory and 23 Optional data objects. This table shows how a software service can represent a BAL device as an IEC 61850 ZBAT device:

ZBAT	Description	Mandatory / Optional	BAL
	LN shall inherit all mandatory data from common logical node class.	Mandatory	Implemented by service and BAL namespace.
BatSt	Battery status ON/OFF	Mandatory	A switch of enabling / disabling the battery discharging
BatTyp	Type of battery	Mandatory	99 (other)
Vol	External battery voltage	Mandatory	The measured external voltage of the logical battery (returned by the BAL API)
BatVHi	Battery voltage is high or overcharged	Optional	Start signalling when the state-of-charge reaches the maximum capacity of the logical battery.
BatVLo	Battery voltage low or undercharged	Optional	Start signalling when the state-of-charge reaches 0.
AhrRtg	Amp-hour capacity rating	Optional	The maximum capacity of the logical battery.
MinAhrRtg	Minimum capacity allowed	Optional	0, or configurable with BAL management
BatVNorm	The nominal voltage of the battery	Optional	The nominal voltage of the logical battery, from BAL's voltage field.
MaxBatA	Maximum battery discharge current	Optional	The maximum discharging current of the logical battery, from BAL's maximum current field.
Amp	Battery drain current	Optional	The current of the logical battery, from BAL's current field.

Table 12: Mapping BAL to ZBAT.

Notice that ZBAT does not have a data object to report the current state-of-charge, but there are two signalling fields indicating whether the battery is full or empty. Instead, its design assumes that an IEC 61850 client calculates this my measuring the voltage of the battery (with the required Vol data object) and combines this with the discharge curve (with the optional DisChaCrv data object) to determine the discharge from the voltage. Our current mapping does not provide the DisChaCrv object; future improvements may include it, to allow clients to compute the state of charge.

The ZBTC (battery charger) node defines the interface for charging a battery. This is how ZBTC maps to a BAL device:

ZBTC	Description	Mandatory / Optional	BAL
BatChaSt	Battery charger charging mode status -- 1 = off, 2 = operational mode	Mandatory	Whether the battery is charging (2) or not (1)
BatChaTyp	Type of battery charger	Optional	1 (constant voltage)
BatChaMod	Battery charger mode setting	Mandatory	An option to set the battery to charging or not.
ChaV	Charging voltage	Optional	The voltage returned by BAL API when the battery is charging.
ChaA	Charging current	Optional	The current returned by BAL API when the battery is charging.

Table 13: Mapping BAL to ZBTC.

7. Open Questions

There are three major open questions in the design and implementation of the Battery Abstraction Layer: location, IEC profiles, and higher-layer services.

1. **Is the proposed approach for reporting location effective and useful?** Location is an important consideration for energy resources. An aggregate battery current describes its location (its electrical connection point/ECP) as a midpoint of its constituent batteries. This design decision assumes that actors which need location-specific energy storage will request logical batteries which are tightly localized within the grid topology. That is, virtual batteries which combine batteries from Massachusetts and Oregon are not likely to be very useful, so energy storage providers will not do so. We expect that aggregators can manage large pools of physical batteries (or tightly localized aggregate batteries) and generate aggregate batteries on demand that meet the needs of client requests. Exploring the tradeoffs of this approach and its implications to existing grid economic and business models is an important next step.
2. **How should a BAL-to-61850 service handle state of charge?** Section 6 described our current mapping of the BAL to IEC 61850. One important edge case in this mapping is how state of charge is communicated. The ZBAT logical node provides this information through a layer of indirection, reporting voltage and a curve that maps voltage to charge. This raises the question of whether the BAL-to-ZBAT service should report a static curve and slightly tweak the reported voltage to reflect the state of charge, or instead report the exact voltage and adjust the curve to reflect the state of charge. We believe that answering this question requires talking with energy storage providers and operators to understand what assumptions their software systems make on values returned from ZBAT. E.g., if they assume that the curve does not change (or read it once), we cannot adjust the curve.
3. **What benefits does the BAL bring to higher-layer services and algorithms?** The larger TrustDER project introduces higher-layer services and data sharing approaches. We believe that

integrating these use cases into BAL services could lead to significant benefits. For example, our work in distributed coordination has shown that by exchanging conservative bounds, energy storage systems and grid operators can efficiently coordinate storage and time-shifting with day-ahead scheduling. One approach to use these bounds is that a BAL logical node responds with bounds directly, based on the state of its underlying batteries. Another is that a request to a logical node actually travels down the entire battery topology underneath it. E.g., if a grid operator requests certain bounds from a logical battery that aggregates several physical batteries, the logical battery decides how to partition those bounds across its physical batteries and passes the requests down to them; BAL answers with the bounds each can provide, which the aggregate battery then combines in its response to the operator. Similarly, currently our BAL prototype does not authenticate networked batteries; incorporating SecureID, another sub-project in TrustDER, into network battery authentication will demonstrate how these parts compose into a larger, more complete system.

8. Conclusion

This document describes the design of the Battery Abstraction Layer (BAL), a software abstraction that allows energy storage systems to virtualize energy storage. Using BAL, a software system can *aggregate* multiple batteries into a single one, or *partition* a battery into multiple batteries. This allows software to build robust, distributed topologies of energy storage.

The key research questions in the design of BAL is what values these aggregate and partitioned batteries should report. Edge cases arise when the constituent batteries of an aggregate battery have different states of charge, or how changes in an underlying battery are reflected in partitions. This document describes how the BAL handles these edge cases in order to provide a clear, robust abstraction for management and software.