TBOS SOFTWARE ARCHITECTURE

WHERE (AND HOW) THE MAGIC HAPPENS

tanktwo.com

Our software-defined battery (SDB) solutions supported by the Tanktwo Battery Operating System (TBOS) offer a range of battery management capabilities that aren't available through any existing battery solution.

This white paper delves into the inner workings of TBOS to show how the various functions come together to deliver an innovative battery architecture that can overcome the limits of today's battery solutions to make electrification sustainable and profitable.

THE FOUR KEY COMPONENTS OF TBOS

TBOS is an intricate system connecting software with hardware (i.e., battery cells) and involves four main components:

T-BLOCKS

A T-Block is a physical unit consisting of cells in a predetermined arrangement — it's the basic building block of a typical TBOS-driven battery system. It consists of the following:

- Electrochemical cells of any age and chemistry mix.
- Hardware that controls the cells using predetermined logic.
- Hardware for monitoring the cells and collecting telemetry.
- Software to control the hardware via a TCP/IP network.

TBOS CONTROL UNIT

A computer in the TBOS control unit communicates with other components via a TCP/IP network to monitor and control the battery system. The control unit performs functions including:

- Detect T-Blocks in the system.
- Provide a control interface for operators to interact with the system.
- Collect and analyze telemetry from T-Blocks.

TCP/IP NETWORK

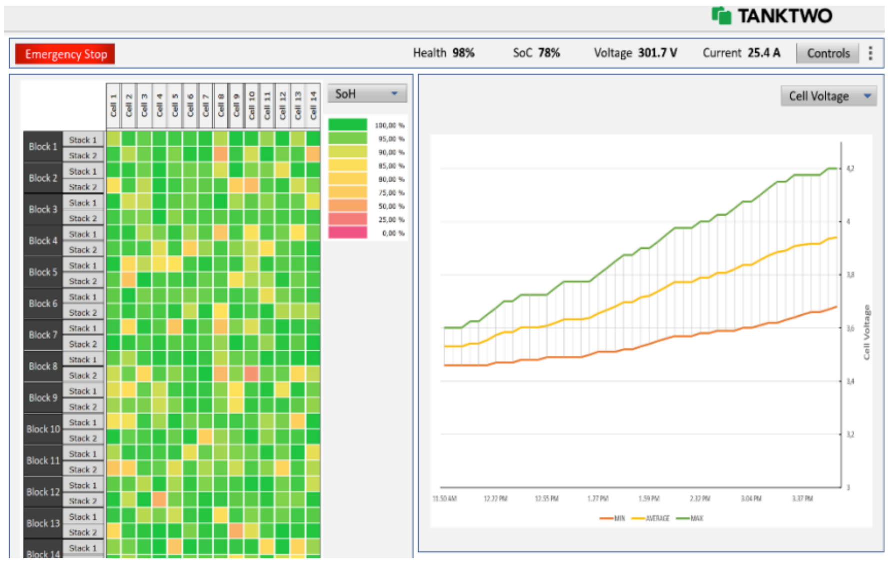
The framework provides communication protocols to transmit data over the internet. In a TBOS-driven battery system, it performs the following functions:

- Hosts the DHCP server to assign IP addresses for all devices in the TBOS system.
- Provide connectivity between T-Blocks and the control unit.
- Connect the battery solution to external networks for system control.

USER DEVICES

A TBOS-driven battery system can connect with any number of user devices (or none at all) so operators can interact with it in real-time by:

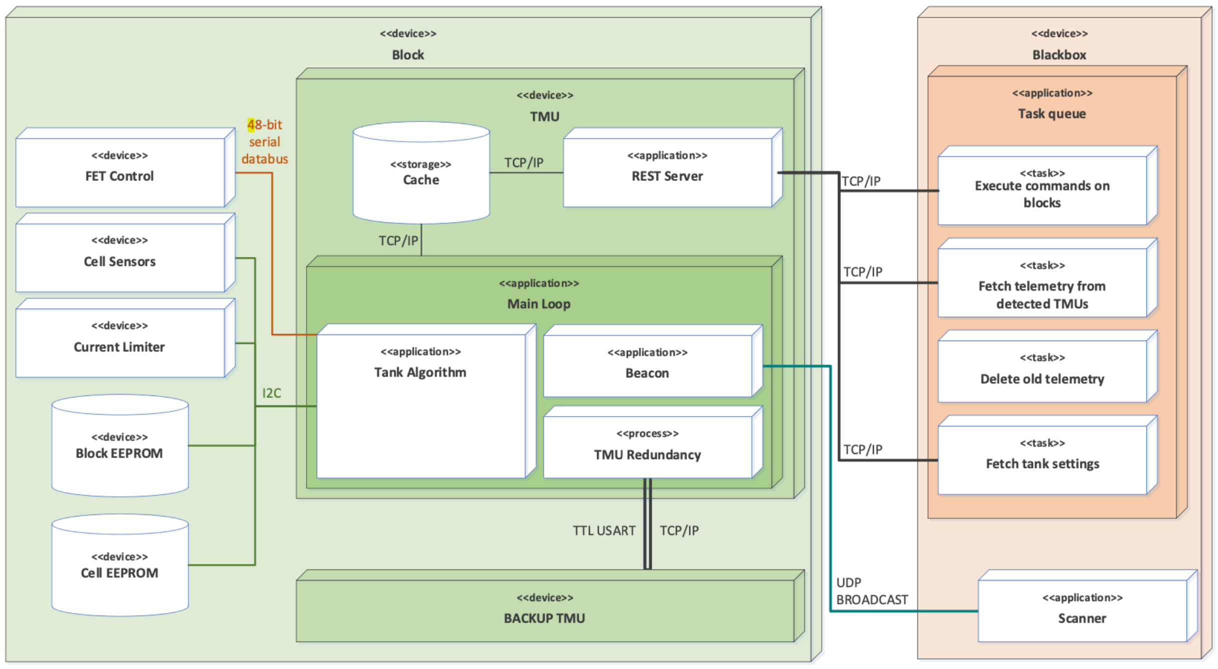
- Connecting to the graphic user interface (UI) in the control unit.
- Visualizing telemetry data.
- Offering a UI for system control and monitoring.


Health 98% SoC 78% Voltage 301.7 V Current 25.4 A

Software components within the T-Blocks and control unit power TBOS's innovative capabilities. Here's a closer look at these two components:

T-BLOCKS

The software component in a T-Block acts as an intermediary between the cells and the control unit. It allows the system to monitor the state of health (SoH) and assert granular control over each cell to achieve the many features unique to SDB systems.



1. TBOS MANAGEMENT UNIT (TMU)

The TMU is the "brain" of each T-Block. It consists of multiple software components, typically implemented on top of Linux OS. Each T-Block comes with two identical TMUs for redundancy and reliability.

2. MAIN LOOP

The main loop is the high-level logic that makes the battery system work — then rinse and repeat.

Main Loop Component 1: TBOS Algorithm

The TBOS algorithm is the control logic that identifies cells and T-Blocks using non-volatile memory (or EEPROMs). It reads cell telemetry, performs cell selection routines, monitors cell safety, and controls cell field effect transistors (FETs), master FETs, and the current limiter. Additionally, it stores the system state with cell telemetry and executes the desired charging and discharging voltages using information from the cache.

The TBOS algorithm has four elements:

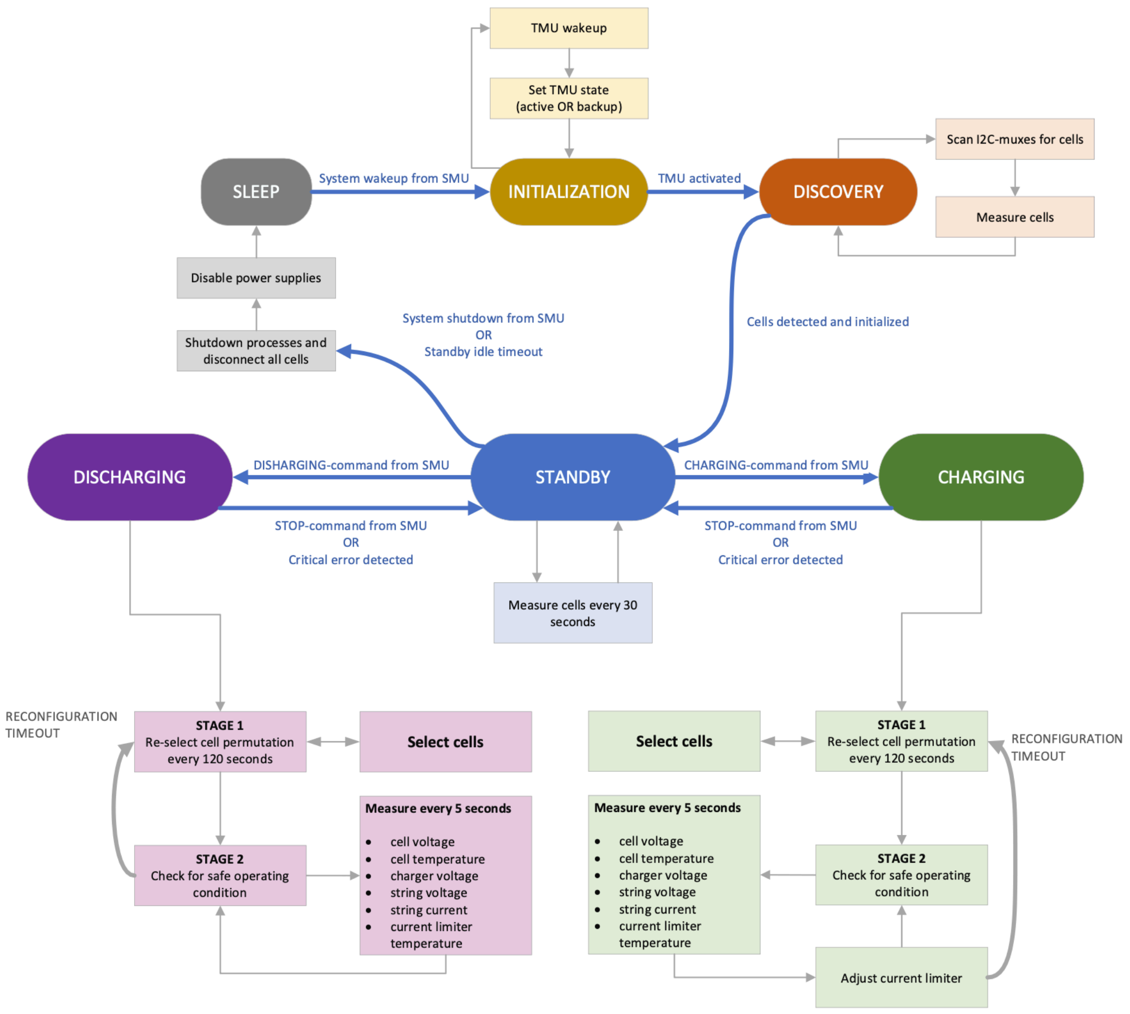
- TBOS API uses the cache to transmit data between the algorithm and the REST server.
- The TBOS cell management feature maintains the logical structure, allowing the software to work with any hardware (i.e., battery packs.)
- TBOS drivers function as an abstraction layer for sensors, allowing builders to implement TBOS on any hardware.
- TBOS config defines connected peripherals and settings for specific hardware.

The TMU monitors cell and system health in the standby state. Once it receives commands from the system controller, it executes the algorithm and switches to a charging or discharging state.

Charging starts with configuration imports to define the operating conditions for the algorithm. Then, the software performs a verification loop to ensure the measured charging voltage matches the predefined value.

The charging algorithm controls cell selection and monitoring. Cell selection is the outer loop of the algorithm and continues to operate until a stop command is issued or a critical exception occurs. Cell monitoring occurs in the inner loop, and continues until the charging cycle times out or an exception occurs. When the monitoring loop breaks, the algorithm determines whether the software should select new cells or stop charging.

Like the charging algorithm, the discharging algorithm controls cell selection and monitoring. The process starts by ensuring all cells are usable and within the safe operating window. The algorithm then sorts and adds cells to the string. Next, it determines the maximum power that can be transferred to or from each string based on each cell's internal resistance, string length, and desired voltage.


TMU wakeup
Set TMU state (active OR backup)
SLEEP
System wakeup from SMU
INITIALIZATION
TMU activated
DISCOVERY
Scan I2C-muxes for cells
Measure cells
Disable power supplies
Shutdown processes and disconnect all cells
System shutdown from SMU OR Standby idle timeout
Cells detected and initialized
DISCHARGING
DISHARGING-command from SMU
STANDBY
CHARGING-command from SMU
CHARGING
STOP-command from SMU OR Critical error detected
STOP-command from SMU OR Critical error detected
Measure cells every 30 seconds
RECONFIGURATION TIMEOUT
STAGE 1 Re-select cell permutation every 120 seconds
Select cells
STAGE 2 Check for safe operating condition
Measure every 5 seconds
cell voltage
cell temperature
charger voltage
string voltage
string current
current limiter temperature
Select cells
STAGE 1 Re-select cell permutation every 120 seconds
RECONFIGURATION TIMEOUT
Measure every 5 seconds
cell voltage
cell temperature
charger voltage
string voltage
string current
current limiter temperature
STAGE 2 Check for safe operating condition
Adjust current limiter

Main Loop Component 2: Beacon

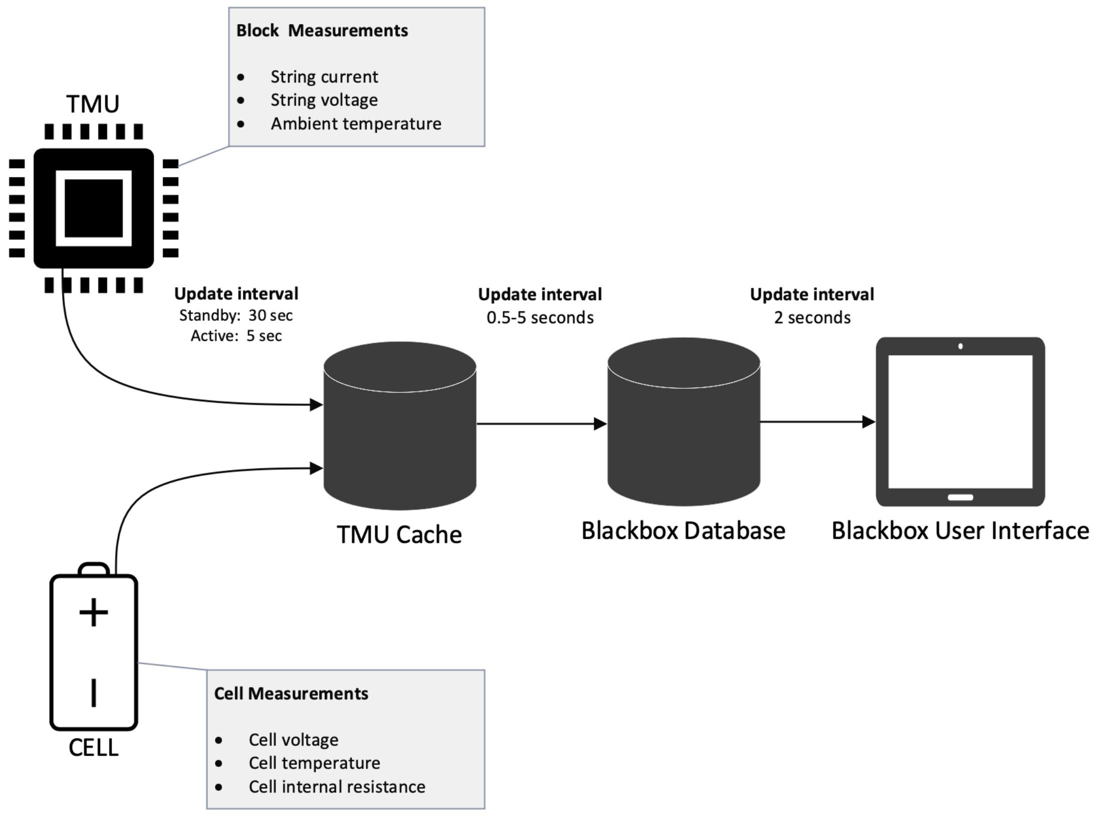
The beacon runs parallel to the TBOS algorithm and sends universal data protocol (UDP) broadcast packets to the local network to announce the TMU's ID, IP address, and state. The control unit uses this information to detect usable TMUs in the network.

Main Loop Component 3: TMU Redundancy

The TMU redundancy software runs parallel to the TBOS algorithm. It manages the two parallel and identical TMUs inside each T-Block. It communicates via serial and TCP/IP with the TMUs, determines the controlling unit, monitors the TMU in control, and ensures safe switchover if the controlling unit experiences an issue.

3. CACHE

The cache is an information exchange service. It allows software components running in parallel threads to share data. The one in TBOS stores settings and hardware reservations using Redis, an in-memory data store that supports various abstract data structures.



Telemetry data is first stored in the cache and then transferred to the control unit's database (which could cause the cell measurement displayed through the control unit UI to have a theoretical delay of up to 12 seconds.)

4. REST SERVER

This HTTP server uses the cache to relay telemetry and commands between T-Blocks and the TMU algorithms.

5. FET CONTROL

Our system uses gallium nitride (GaN) metal oxide semiconductor FET, a critical component in the Dynamic Current Routing Matrix (Dycromax^{TM}) that supports TBOS's unique features like variable output voltage and yellow flagging.

6. CELL SENSORS

These measurement instruments collect telemetry from the electrochemical cells, including cell temperature, cell voltage, string current, and string voltage.

7. Current Limiter

This component is essential for constant voltage charging — a unique TBOS capability. The hardware feedback loop measures string current and limits it to the desired level. It also includes over-temperature protection for the current-limiting semiconductors.

The TBOS algorithm controls the digital-to-analog converter (DAC), which includes a proportional integration differential (PID) loop to predict the current limiter’s dissipated power. When the temperature in the semiconductor increases, the software PID loop reduces the dissipated power before the hardware’s one-time programming (OTP) circuit disables the current flow completely to protect the cells from overheating.

8. Block and Cell EEPROM

The EEPROMs in T-Blocks and cells store unique identifiers, which can be read by the TBOS algorithm and included in the collected telemetry to make all measurements traceable.

The memory can also include device-specific settings and usage limits for the TBOS algorithm to enforce specific rules. For example, we can use it as a safety control to ensure that the device’s maximum current limit isn’t surpassed, even if software components receive higher values via external commands or software updates.

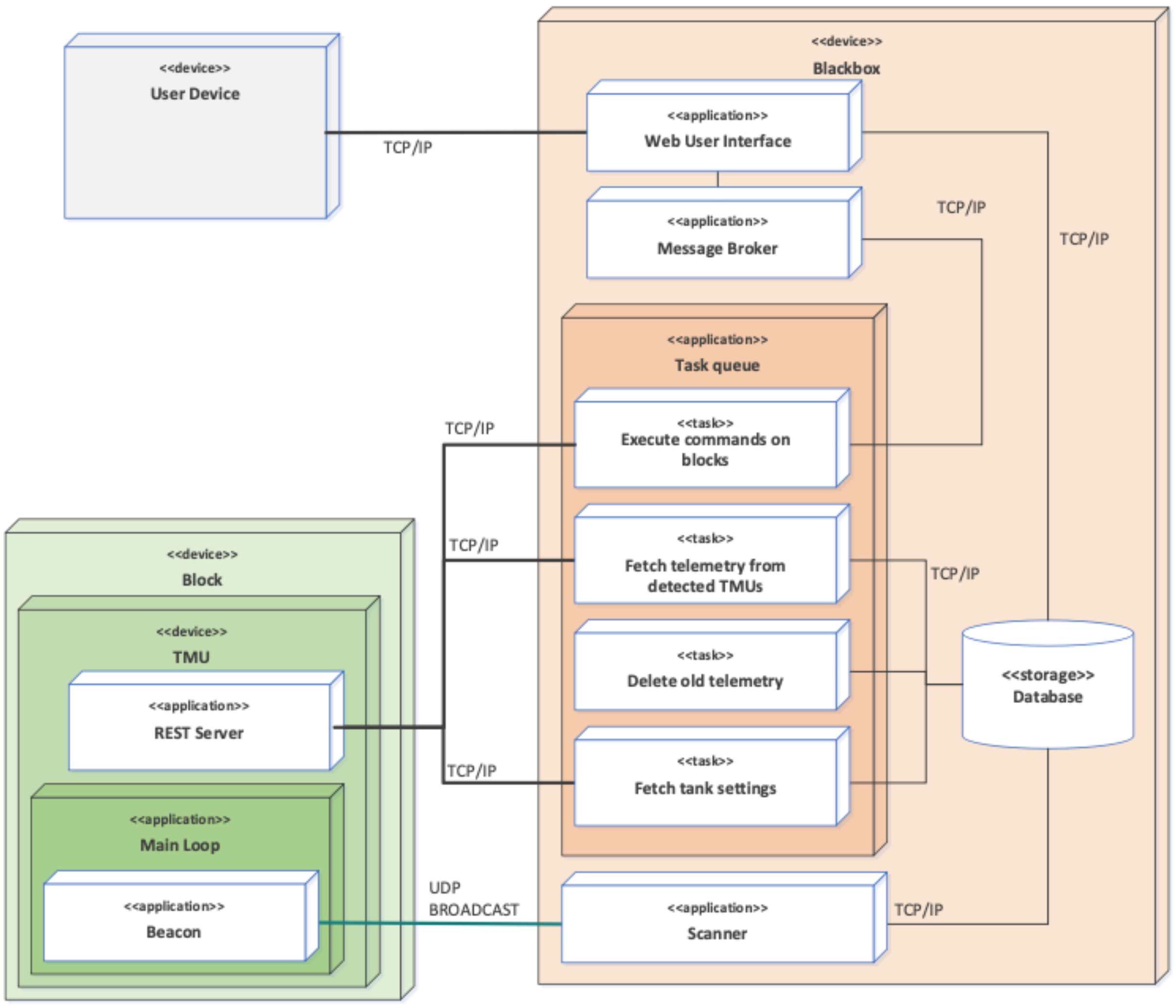
TBOS CONTROL UNIT

The control unit automatically detects all T-Blocks in the network, collects telemetry, and stores the data in its database. The information contains time series telemetry data and log information for debugging purposes.

The unit has a scanner that listens for broadcast messages from the beacon in all T-Blocks. It controls the T-Blocks, e.g., by defining the voltages and operation mode and issuing commands via the web user interface. The message broker then relays the messages from the web user interface to the command execution task.

Meanwhile, the task queue in the control unit fetches TBOS settings from the TMU and telemetry from the TMU REST server endpoint, executes commands on T-Blocks, and deletes old telemetry no longer required.

To create the redundancy required for maximum reliability, we put multiple instances of the software into each system — all of which have the command line to control the entire system. We also have an arbitrage logic to ensure the integrity of the decision-making process in case there are conflicting commands from different control units.



<<device>>
User Device
TCP/IP
<<device>>
Blackbox
<<application>>
Web User Interface
<<application>>
Message Broker
TCP/IP
TCP/IP
<<application>>
Task queue
<<task>>
Execute commands on blocks
<<task>>
Fetch telemetry from detected TMUs
<<task>>
Delete old telemetry
<<task>>
Fetch tank settings
TCP/IP
TCP/IP
TCP/IP
TCP/IP
<<storage>>
Database
<<device>>
Block
<<device>>
TMU
<<application>>
REST Server
<<application>>
Main Loop
<<application>>
Beacon
UDP
BROADCAST
<<application>>
Scanner
TCP/IP

TBOS SOFTWARE IN ACTION

These use cases show how the components in the TBOS software architecture work together.

SYSTEM WAKEUP

An external signal triggers the wake-up sequence in the TMUs and starts the software components.

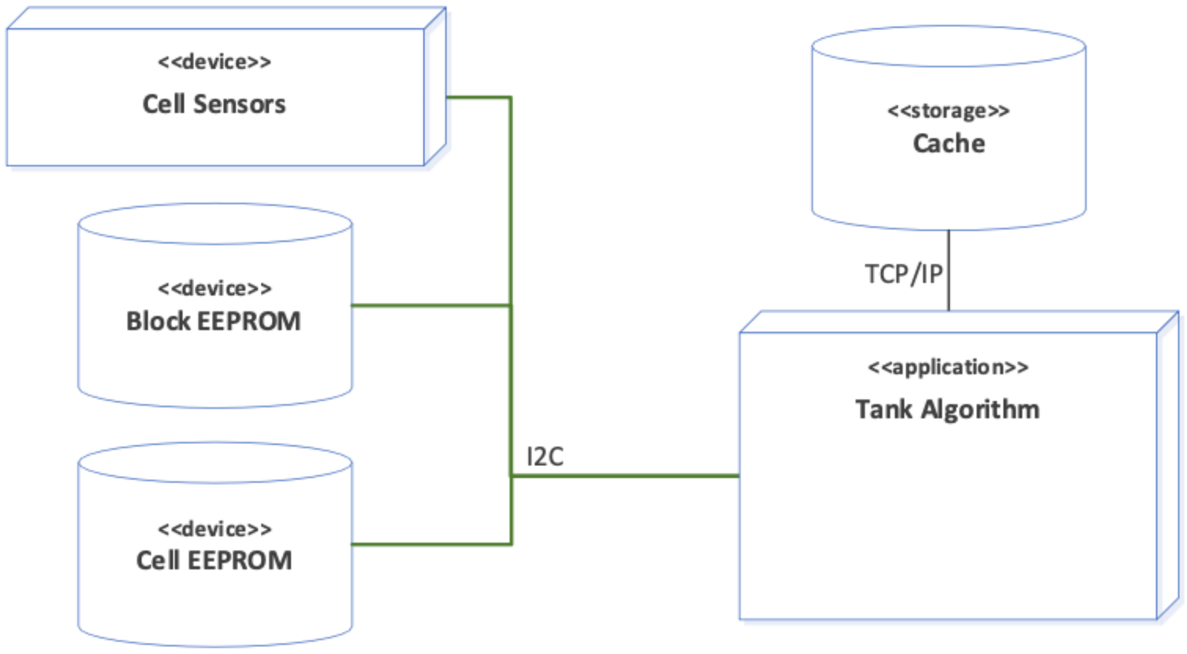
Step 1: Hierarchy and Redundancy

After the TMU loads the OS and starts the main loop process, the TMU redundancy software kicks in to ensure that only one of the two TMUs in a T-Block takes control of the unit. It runs a handshake sequence to confirm that the two TMUs it connects with are in the same T-Block and that the TCP/IP network extends to all T-Blocks in the system.

If the TMU redundancy software finds another TMU, the algorithm selects the optimal TMU and allows it to take control. If it doesn't detect a second TMU, the first one will assume control. This process prevents two TMUs from controlling the unit simultaneously.

Step 2: Cell Discovery

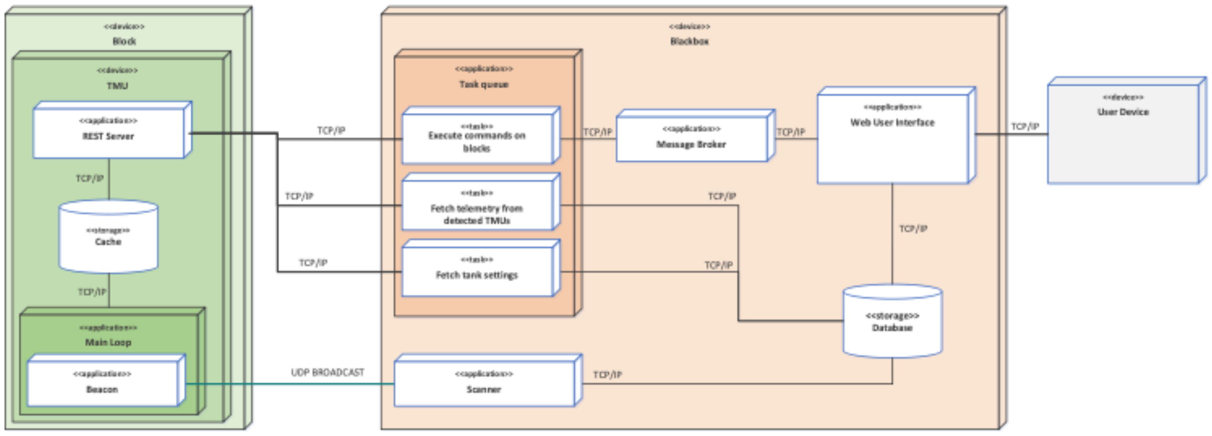
After gaining control of the T-Block, the TMU runs a discovery process. Block and cell EEPROMs detect the type of hardware, and cell sensors provide information on all available cells in the system. Then, the TMU cache stores the resulting list of cells, including their state and location in the system.



Step 3: T-Block Discovery

The control unit runs the scanner application, which listens to UDP broadcast packets containing the TMU state and IP address. After discovering an active TMU, scheduled tasks use the endpoint at the TMU REST server to read the TMU state, settings, and telemetry. Then, they store the values in the control unit's database.

The web user interface fetches information from the database and visualizes the state of the active T-Blocks via the control unit UI — allowing the operator to set desired voltage levels for the selected T-Blocks and control the system's state using the user device(s).



SYSTEM CHARGING AND DISCHARGING

An operator selects an operation mode in the user device's graphic interface, and the external command activates the system. (Alternatively, a machine-to-machine interface using a REST API may send commands to the control unit.)

The message broker in the control unit receives the commands and relays them to the appropriate task in the task queue. Then, the command execution task issues the commands to the selected T-Blocks via their TMU REST server endpoints.

When managing system-level commands, the control unit runs an internal logic to determine the system topology and match the desired voltage levels. Let's say, a system consists of 4 T-Blocks, 2 in series and 2 in parallel (2S2P). When a user sets the system output voltage at 200V, the web user interface assesses the system topology, dividing the desired output voltage by the number of T-Blocks in series. The control unit then gives the message broker a command to issue 100V output voltage for all T-Blocks in the system.

The TMU gathers data from the cells using the cell sensors and stores this data in the cache during charging and discharging. The control unit queries the information, iterates through all the detected T-Blocks, and stores the latest telemetry values in its database to automate processes such as cell balancing, determining SoH, yellow-flagging, and more.

The web user interface then processes this data and visualizes it via the user device — allowing the operator to make informed decisions and adjust the system's behaviors on the fly.

TL;DR

Software on top of a generic block of battery cells receives commands from the control unit and makes them do very cool things that even industry veterans deem impossible.

The control unit oversees all the activities within the system — it takes commands from the user device, makes the calculations, and tells each T-Block what to do. It also provides real-time data visualization through the user device to support decision-making on a dime.

A TCP/IP network connects all these devices and gets everything to talk to everything to make any battery nerd's dreams come true.

But, of course, there are a lot of nuances. The software makes the magic happen by allowing us to program individual battery cells to behave and work together in ways that are out of reach for traditional battery systems — providing features and capabilities that will make TBOS and SDBs the backbone of any sustainable and profitable electrification solutions.