BLE Mesh DC Monitor — Implementation Documentation

System Overview

A BLE Mesh network for DC load monitoring and control. The ESP32-C6 mesh nodes read INA260 sensors directly via I2C and control load PWM, eliminating the original Pico 2W UART bridge.

```
Pi 5 (gateway.py — Python TUI/CLI)
  |  BLE GATT (Service 0xDC01)
  v
ESP32-C6 GATT Gateway (ESP_GATT_BLE_Gateway)
  |  BLE Mesh (Vendor Model, CID 0x02E5)
  v
ESP32-C6 Mesh Node(s) (ESP-Mesh-Node-sensor-test)
  |  I2C (INA260) + LEDC PWM (load control)
  v
INA260 sensor + 2N2222/MOSFET load circuit
```

A separate **ESP32-C6 Provisioner** auto-discovers and configures all mesh devices.

Components

Component	Firmware	Role
ESP32-C6 Provisioner	`ESP/ESP-Provisioner`	Auto-provisions mesh nodes (UUID prefix `0xdd`)
ESP32-C6 GATT Gateway	`ESP/ESP_GATT_BLE_Gateway`	Pi 5 ↔ Mesh bridge (GATT + vendor client)
ESP32-C6 Mesh Node(s)	`ESP/ESP-Mesh-Node-sensor-test`	Direct I2C sensor reading + PWM load control
Raspberry Pi 5	`gateway-pi5/gateway.py`	Python TUI/CLI with PowerManager

Network Topology

The system operates as a **hybrid mesh/star topology**:

- **Star (current):** The GATT Gateway sends unicast commands to each node individually. ALL commands are expanded into sequential sends with stagger delays.
- **Mesh (relay-capable):** All nodes have relay enabled with TTL=7. If a node is out of direct radio range of the gateway, the mesh stack automatically relays through intermediate nodes. No code changes needed for multi-hop relay.
- **Group addressing (not yet implemented):** True mesh broadcast using group addresses would allow a single send to reach all nodes simultaneously. See Network Topology section for details.

Architecture Evolution

v1: Point-to-Point GATT

```
Pi5 → GATT → ESP32-C6 → UART → Pico 2W → INA260
```

Single ESP32-C6 acting as a GATT-to-UART bridge. No mesh, no multi-node support.

v2: BLE Mesh with Pico UART Bridge

```
Pi5 → GATT → Gateway → Mesh → Node → UART → Pico → sensor → UART → Node → Mesh →
Gateway → Pi5
```

7-hop path with async UART response queue. Introduced BLE Mesh for multi-node support but the UART bridge was unreliable — wrong responses paired with wrong commands, 500ms–2s variable latency, `vnd_send_in_progress` flags getting stuck.

v3: BLE Mesh with Direct I2C Sensor (Current)

```
Pi5 → GATT → Gateway → Mesh → Node → I2C read (~2ms) → Mesh → Gateway → Pi5
```

4-hop path. ESP32-C6 reads INA260 directly via I2C and controls PWM via LEDC. Commands processed synchronously in the mesh callback — no async queue, no UART, no Pico dependency. See `ESP/ESP-Mesh-Node-sensor-test/CHANGES.md` for the full refactor details.

Vendor Model Transport

Custom vendor model replaces the original 1-bit Generic OnOff model with arbitrary text command/response transport.

Shared Definitions (all three ESP devices)

```
#define CID_ESP              0x02E5
#define VND_MODEL_ID_CLIENT  0x0000   // On GATT Gateway
#define VND_MODEL_ID_SERVER  0x0001   // On Mesh Node(s)
#define VND_OP_SEND          ESP_BLE_MESH_MODEL_OP_3(0x00, CID_ESP)  // Gateway ->
Node
#define VND_OP_STATUS        ESP_BLE_MESH_MODEL_OP_3(0x01, CID_ESP)  // Node ->
Gateway
```

Message Flow

1. Pi 5 writes command to GATT characteristic 0xDC03 (e.g., `1:DUTY:50`)
2. Gateway parses command, maps to node-native format (`duty:50`)
3. Gateway sends via `esp_ble_mesh_client_model_send_msg()` with `VND_OP_SEND`
4. Mesh Node receives `VND_OP_SEND` in `custom_model_cb()`
5. Node reads INA260 via I2C (~2ms) and formats response
6. Node sends response via `esp_ble_mesh_server_model_send_msg()` with `VND_OP_STATUS`
7. Gateway receives `VND_OP_STATUS`, formats as `NODE<id>:DATA:<payload>`
8. Gateway notifies Pi 5 via GATT characteristic 0xDC02 (chunked if >20 bytes)

Command Mapping (Gateway)

Pi 5 Command	Node Command	Description
`RAMP`	`r`	Start ramp test (0→25→50→75→100%)
`STOP` / `OFF`	`s`	Stop load (duty 0%)
`ON`	`r`	Start ramp (alias)
`DUTY:<n>`	`duty:<n>`	Set duty cycle 0-100%
`STATUS` / `READ`	`read`	Single sensor reading
`MONITOR`	*(gateway polls)*	Continuous monitoring

Response Format

All sensor responses use the same text format:

```
D:50%,V:12.003V,I:250.00mA,P:3000.8mW
```

ESP32-C6 GATT Gateway (`ESP_GATT_BLE_Gateway/main/main.c`)

Role

Bridges Pi 5 GATT client to BLE Mesh network. Provisioned by the mesh provisioner (UUID prefix `0xdd`).

GATT Service

UUID	Type	Description
`0xDC01`	Service	DC Monitor Service
`0xDC02`	Characteristic	Sensor Data (Read/Notify)
`0xDC03`	Characteristic	Command (Write)

Command Serialization

Mesh commands are serialized to prevent overlapping radio transmissions:

```
static bool vnd_send_busy;              // True while waiting for response
static uint16_t vnd_send_target_addr;   // Node we're waiting to hear from
#define VND_SEND_TIMEOUT_MS 5000        // Auto-clear if no response
```

- Before sending: wait loop checks `vnd_send_busy` with 5s timeout
- After sending: `vnd_send_busy = true`, `vnd_send_target_addr = target`
- Only a `VND_OP_STATUS` from the matching target address clears `vnd_send_busy`
- `SEND_COMP` (mesh send complete) does NOT clear busy — it only means the message left the local radio, not that it was delivered

ALL Command Handling

```
if (is_all) {
    for (int i = 0; i < known_node_count; i++) {
        send_vendor_command(known_nodes[i], cmd, len);
        vTaskDelay(pdMS_TO_TICKS(500));  // Inter-send stagger
    }
    // Probe for undiscovered nodes (unless discovery_complete)
    if (!discovery_complete) {
        uint16_t probe_addr = NODE_BASE_ADDR + known_node_count + 1;
        send_vendor_command(probe_addr, cmd, len);
    }
}
```

- `known_nodes[]` tracks nodes that have responded with STATUS messages
- `discovery_complete` flag prevents redundant probe timeouts after all nodes found
- `SENT:` notification sent to Pi 5 before the send loop starts

NVS Persistence

Gateway state survives power cycles:

```
struct gateway_state {
    uint16_t net_idx;        // Mesh NetKey index
    uint16_t app_idx;        // Mesh AppKey index
    uint16_t addr;           // Unicast address (e.g., 0x0005)
    uint8_t  vnd_bound_flag; // Vendor model bound to AppKey
};
```

GATT Notification Chunking

BLE Mesh GATT Proxy hard-limits MTU to 23 bytes (20 bytes usable). Sensor data strings (~48 bytes) are split:

- Continuation chunks: `+` prefix byte + 19 bytes data
- Final chunk: no prefix, just remaining data

- Pi 5 reassembles by buffering `+` prefixed chunks

Monitor Mode

Gateway-side polling via FreeRTOS timer:

```
static TimerHandle_t monitor_timer;           // Fires every 1000ms
static uint16_t monitor_target_addr;          // Node being polled
static bool monitor_waiting_response;         // Prevents overlapping requests
```

- Timer callback sends `read` via vendor model
- Vendor STATUS response clears `waiting_response` for next poll
- Any non-MONITOR command automatically stops monitoring
- Timeouts during monitor mode are suppressed

OnOff Fallback

If vendor model not yet bound (`vnd_bound = false`), commands fall through to Generic OnOff:

- `STOP`/`OFF` → OnOff SET 0
- Everything else → OnOff SET 1

ESP32-C6 Mesh Node (`ESP-Mesh-Node-sensor-test/main/main.c`)

Role

Self-contained sensing node — reads INA260 power sensor via I2C and controls load PWM directly. No external microcontroller needed.

I2C Sensor (INA260)

Parameter	Value
SDA	GPIO6
SCL	GPIO7
Address	0x45
Speed	400 kHz
Averaging	1024 samples (config register 0x0F49)

Reads:

- **Voltage** (register 0x02): LSB = 1.25 mV
- **Current** (register 0x01): LSB = 1.25 mA
- **Power**: calculated as V × I

I2C bus scan runs at startup and prints all detected device addresses.

LEDC PWM Load Control

Parameter	Value
GPIO	5
Frequency	1 kHz
Resolution	13-bit (8192 steps)
Logic	Inverted (100% duty → GPIO LOW → transistor OFF)

The 2N2222 transistor + MOSFET circuit inverts the signal: high GPIO → transistor ON → MOSFET gate pulled low → load OFF. PWM duty is inverted in firmware to compensate.

Vendor Model Server

Commands processed **synchronously** in `custom_model_cb()`:

```
case ESP_BLE_MESH_MODEL_OPERATION_EVT:
    if (opcode == VND_OP_SEND) {
        // 1. Parse command (read, duty:50, r, s)
        // 2. Execute immediately (I2C read or PWM set)
        // 3. Format response: "D:50%,V:12.003V,I:250.00mA,P:3000.8mW"
        // 4. Send VND_OP_STATUS back to sender
    }
```

No async queue, no UART wait, no pending flags. The response is generated and sent in the same callback context.

Serial Console Task

For local testing via `idf.py monitor`:

Command	Action
`read`	Read INA260 voltage/current/power
`duty:50`	Set PWM to 50%
`50`	Same as duty:50
`r`	Ramp test (0→25→50→75→100%)
`s`	Stop (duty 0%)
`scan`	Full I2C bus scan

Wiring

Signal	GPIO	Connected To

Signal	GPIO	Connected To
INA260 SDA	GPIO6	INA260 SDA pin
INA260 SCL	GPIO7	INA260 SCL pin
PWM out	GPIO5	2N2222 base / MOSFET gate
INA260 VCC	3V3	INA260 VCC pin
GND	GND	Common ground

NVS Persistence

```
struct mesh_node_state {
    uint16_t net_idx;
    uint16_t app_idx;
    uint16_t addr;
    uint8_t  onoff;
    uint8_t  tid;
};
```

Stored under key `"mesh_node"`, restored on boot.

ESP32-C6 Provisioner (`ESP-Provisioner/main/main.c`)

Role

Auto-discovers and provisions all mesh devices with UUID prefix `0xdd`. Distributes keys and binds all models.

Node Info Tracking

```
typedef struct {
    uint8_t  uuid[16];
    uint16_t unicast;
    uint8_t  elem_num;
    char     name[16];       // "NODE-0", "NODE-1", etc.
    bool     has_onoff_srv;
    bool     has_onoff_cli;
    bool     has_vnd_srv;    // Vendor Server detected
    bool     has_vnd_cli;    // Vendor Client detected
    bool     srv_bound;
    bool     cli_bound;
    bool     vnd_srv_bound;  // Vendor Server bound to AppKey
    bool     vnd_cli_bound;  // Vendor Client bound to AppKey
} mesh_node_info_t;
```

Composition Data Parsing

After provisioning a device, the provisioner requests Composition Data and parses both SIG and vendor models:

- `0x1000` → Generic OnOff Server
- `0x1001` → Generic OnOff Client
- `0x02E5 + 0x0001` → Vendor Server (mesh node)
- `0x02E5 + 0x0000` → Vendor Client (gateway)

Model Binding Chain

After AppKey is added to a node, models are bound in priority order via `bind_next_model()`:

```
1. OnOff Server   (company_id = 0xFFFF)
2. OnOff Client   (company_id = 0xFFFF)
3. Vendor Server  (company_id = 0x02E5)
4. Vendor Client  (company_id = 0x02E5)
5. → "FULLY CONFIGURED"
```

On bind failure, the chain continues to the next model instead of stopping.

Note: The provisioner does NOT send `CONFIG_MODEL_SUB_ADD` — nodes have no group address subscriptions. This is needed for future group messaging support.

Provisioning Sequence

```
1. Device with 0xdd UUID prefix discovered
2. PB-ADV provisioning initiated
3. Provisioning complete → request Composition Data
4. Parse composition → add AppKey
5. AppKey added → bind models in chain
6. All models bound → "FULLY CONFIGURED"
```

Address Assignment

Device	Address
Provisioner	0x0001
First provisioned node	0x0005 (NODE-0)
Second provisioned node	0x0006 (NODE-1)
Additional nodes	Sequential

Pi 5 Gateway (`gateway-pi5/gateway.py`)

Role

Python BLE central with Textual TUI (or plain CLI fallback). Connects to ESP32-C6 GATT Gateway via BLE, sends commands to mesh nodes, and runs the PowerManager for automated power balancing.

Dependencies

```
pip install bleak textual
```

- `bleak` — BLE library (required)
- `textual` — Terminal UI framework (optional, falls back to plain CLI)

Architecture

```
# gateway.py structure (~1700 lines):
NodeState (dataclass)       # Per-node sensor state + target/commanded duty tracking
PowerManager (class)        # Equilibrium power balancer (poll → evaluate → nudge)
DCMonitorGateway (class)    # BLE engine (bleak GATT client, notification handler)
BleThread (class)           # Dedicated asyncio thread for bleak operations
MeshGatewayApp (App)        # Textual TUI (only when textual installed)
```

BleThread: Runs bleak in its own asyncio event loop on a dedicated thread. This prevents D-Bus signal handler orphaning on Linux and keeps the TUI responsive during BLE operations. All BLE calls go through `BleThread.submit()`.

Device Discovery

Scans for devices matching:

- Name prefix: `"Mesh-Gateway"` (pre-provisioning) or `"ESP-BLE-MESH"` (post-provisioning)
- Service UUID: `0xDC01`
- Specific MAC address (via `--address` flag)

GATT Chunk Reassembly

```
if decoded.startswith('+'):
    self._chunk_buf += decoded[1:]  # Accumulate continuation
    return
if self._chunk_buf:
    decoded = self._chunk_buf + decoded  # Combine with final
    self._chunk_buf = ""
# Process complete message...
```

Chunk buffer is cleared on disconnect to prevent stale data corruption.

TUI Layout

```
+--------------------------+------------------------------+
| STATUS PANEL (sidebar)   | LOG (scrollable)             |
|                          |                              |
| Connected                | [12:01:01] NODE1 >> D:100%.. |
| ESP-BLE-MESH             | [12:01:03] [POWER] ▲ UP ...  |
|                          | [12:01:03] [POWER] Balancing |
| Target: ALL              |   N1:60->100%, N2:60->100%   |
|                          |                              |
| Power Mgmt               |                              |
| Threshold: 10000mW       |                              |
| Budget:     9500mW       |                              |
| Priority: none           |                              |
| Total: 9188mW            |                              |
| Headroom: 812mW          |                              |
| Nodes: 2/2               |                              |
+--------------------------+------------------------------+
| NODES TABLE                                             |
| ID | Duty | Target | Voltage | Current | Power | Status  |
|  1 |  87% |  100%  | 11.79V  | 407mA   | 4803  | ok      |
|  2 | 100% |  100%  | 11.73V  | 374mA   | 4385  | ok      |
+---------------------------------------------------------+
| > [command input]                                       |
+---------------------------------------------------------+
| F2 Debug  F3 Clear  Esc Input                  [Footer] |
+---------------------------------------------------------+
```

Commands

Node Control:

Command	Description
`node <id>`	Switch target node (1-9 or ALL)
`duty <0-100>`	Set duty cycle on target node
`ramp` / `r`	Start ramp test
`stop` / `s`	Stop load
`read`	Single sensor reading
`monitor` / `m`	Continuous monitoring
`raw <cmd>`	Send raw command string

Power Management:

Command	Description
`threshold <mW>`	Set total power limit (starts PM)
`threshold off`	Disable PM, restore original duties

Command	Description
`priority <id>`	Set priority node (gets 2x power share)
`priority off`	Clear priority
`power`	Show PM status

Debug:

Command / Key	Description
`debug` / `d` / F2	Toggle debug mode (shows `[PM]` and `SENT:` messages)
F3	Clear log
Esc	Focus input box

Output Format

```
[HH:MM:SS] -> SENT:DUTY                              # Command sent (debug only)
[HH:MM:SS] NODE1 >> D:50%,V:11.95V,I:240mA,P:2868mW  # Sensor data
[HH:MM:SS] !! ERROR:MESH_TIMEOUT                      # Error
[HH:MM:SS] [POWER] ▲ UP: 6000/9500mW, nodes: ['1','2']  # PM action
[HH:MM:SS] [POWER] Balancing ... N1:60->100%, N2:60->97%  # PM changes
```

PowerManager Design

Equilibrium-based power balancer. Maintains total power near `(threshold - headroom)` by nudging node duty cycles up or down each poll cycle.

Constants

Parameter	Value	Purpose
`POLL_INTERVAL`	3.0s	Seconds between poll cycles
`READ_STAGGER`	2.5s	Seconds between individual READ commands
`STALE_TIMEOUT`	45.0s	Mark node unresponsive after this
`COOLDOWN`	5.0s	Minimum seconds between adjustments
`HEADROOM_MW`	500.0	Buffer below threshold
`PRIORITY_WEIGHT`	2.0	Priority node gets 2x power share

State Model (Initialization → PM → Initialization)

1. **Setup phase:** User sends `duty`, `ramp`, `monitor` etc. These are direct commands to nodes.

2. **PM phase:** `threshold <mW>` freezes `target_duty` at whatever each node's current duty is. PM then adjusts duty cycles to stay within budget. User can change `threshold` value, set/clear `priority`.
3. **Back to setup:** `threshold off` restores all nodes to their frozen `target_duty` and exits PM.

Duty Tracking (Three Values Per Node)

Field	Set By	Purpose
`duty`	Sensor data (INA260)	Actual measured duty from node
`target_duty`	User commands / PM freeze	User's ceiling — PM never exceeds this
`commanded_duty`	PM nudge (on confirmed delivery)	Last duty PM sent that was confirmed

Equal Share Mode (No Priority)

```
budget = threshold - HEADROOM
share = budget / N  (N = number of responsive nodes)
```

Each node gets `budget/N` milliwatts. PM calculates ideal duty from `share / mw_per_pct` and sends the command. Scales dynamically — 2, 3, 5 nodes all work the same.

Priority Mode

```
total_shares = PRIORITY_WEIGHT + (N - 1)
priority_share = budget * (PRIORITY_WEIGHT / total_shares)
other_share = budget * (1 / total_shares)
```

With 2 nodes: priority gets 67%, other gets 33%. With 3 nodes: priority gets 50%, others get 25% each. With 5 nodes: priority gets 33%, others get 17% each.

If priority node can't use its full share (limited by `target_duty` ceiling), surplus redistributes to non-priority nodes.

Poll Cycle

```
poll_loop():
  while threshold is not None:
    _poll_all_nodes()         # Send READ to each node individually
    _wait_for_responses()     # Wait for all to report (4s timeout)
    _mark_stale_nodes()       # Mark unresponsive after STALE_TIMEOUT
    sleep(1.0)                # Relay breathing gap
    _evaluate_and_adjust()    # Calculate and send duty changes
    sleep(POLL_INTERVAL)      # Wait before next cycle
```

Key Behaviors

- **Bidirectional:** Increases duty when under budget, decreases when over
- **Direct jump:** Calculates ideal duty and sends it (no step-size limiting) — converges in 1-2 cycles
- **Dead band:** 5% of budget tolerance prevents constant jitter
- **Forced eval:** Priority/threshold changes bypass cooldown and dead band
- **target_duty frozen on first enable:** Only snapshots on `threshold_mw` going from None to a value. Changing threshold while PM is active does NOT re-snapshot (prevents capturing PM-reduced values as ceiling)
- **commanded_duty confirmation:** Only updated after `_wait_node_response()` returns True. If node didn't confirm, PM retries on next cycle with accurate state
- **Sync check:** If `commanded_duty` differs from sensor `duty` by >2%, PM detects the mismatch and retries instead of skipping
- **Debug logging:** Verbose `[PM]` messages (per-node state, skip reasons, nudge calculations) are hidden by default. Enable with F2 or `debug` command. `[POWER]` action messages always visible.

Network Topology

Current: Hybrid Mesh/Star

- **Addressing:** Unicast to each node individually (addresses 0x0005, 0x0006, etc.)
- **Relay:** Enabled on all nodes with TTL=7
- **Net/Relay transmit:** 5 retransmissions, 20ms interval

If all nodes are in direct radio range of the gateway (~10-30m indoor for ESP32-C6 with PCB antenna), commands go directly. If a node is out of range but within range of another node, the mesh stack automatically relays. No code changes needed for multi-hop.

ESP32-C6 BLE Range

Scenario	Range
Indoor, PCB antenna	~10-30 meters
Indoor, optimized TX power	~30-50 meters
Outdoor, line-of-sight	~50-100 meters
Outdoor, Coded PHY (long range)	~200-400 meters

Future: Group Addressing

Currently ALL commands are expanded into N sequential unicast sends. True mesh group addressing would:

1. **Provisioner** — send `CONFIG_MODEL_SUB_ADD` to subscribe each node's vendor server to group address 0xC000
2. **ESP Gateway** — replace the ALL send loop with a single `send_vendor_command(0xC000, cmd)`
3. **Mesh Nodes** — no code changes (ESP-IDF handles group delivery automatically)
4. **Python Gateway** — new `_wait_all_responses()` method, single send + collect N responses

What becomes group: ALL:READ (polling), ALL:DUTY (initialization), ALL:RAMP **What stays unicast:** PM nudge (different duty per node), per-node commands, priority rebalancing

Benefit: Poll cycle drops from O(N) to O(1). With 5 nodes: ~15s → ~3s.

Issues Resolved

1. GATT Connection Handle Not Captured

Problem: `BLE_GAP_EVENT_CONNECT` never fires through mesh GATT proxy. **Fix:** Capture connection handle from GATT access callbacks (`command_access_cb`, `sensor_data_access_cb`) and subscribe events.

2. Vendor Model Not Persisted Across Reboots

Problem: `vnd_bound` was false after power cycle even though provisioner had bound it. **Fix:** Added `vnd_bound_flag` to NVS-persisted `gateway_state`. Also infer binding from valid `app_idx` for backward compatibility.

3. ALL Command Flooding Non-Existent Nodes

Problem: ALL sent to 10 sequential addresses, causing 9 timeouts. **Fix:** Track `known_nodes[]` from vendor STATUS responses. ALL only targets confirmed nodes.

4. GATT Notification Data Truncation

Problem: MTU limited to 23 bytes, sensor data is ~48 bytes. **Fix:** Chunked notification protocol with `+` continuation prefix.

5. Multi-Line UART Data Inflating Payload

Problem: Pico sent multi-line responses, inflating mesh payload beyond segmentation limits. **Fix:** Strip to first line in receive task. (Historical — UART removed in v3.)

6. Monitor Mode Only Returning One Reading

Problem: `esp_ble_mesh_server_model_send_msg()` context becomes stale after first response. **Fix:** Gateway-side polling with FreeRTOS timer. Each poll is a fresh request-response cycle.

7. SeqAuth Collisions During Rapid Sends

Problem: Sending new mesh message before previous completes causes SeqAuth errors. **Fix:** `vnd_send_in_progress` flag with timeout guard. (Simplified in v3 — synchronous responses eliminate the race.)

8. CONFIG_BLE_MESH_SETTINGS Not Applied to Build

Problem: `sdkconfig.defaults` was ignored because `sdkconfig` was generated before mesh defaults were added. **Fix:** Delete `sdkconfig`, `sdkconfig.old`, and `build/` directory, then run `idf.py set-target esp32c6`.

9. Gateway/Node cached_app_idx Lost on Reboot

Problem: Application-level `cached_app_idx` initialized to `0xFFFF` on every boot. **Fix:** Added `save_gw_state()` / `restore_gw_state()` with NVS persistence.

10. Command Serialization (Gateway)

Problem: Overlapping mesh sends caused delivery failures. `SEND_COMP` was incorrectly clearing `vnd_send_busy`. **Fix:** `vnd_send_busy` + `vnd_send_target_addr` serialization. Only matching STATUS response clears busy. `SEND_COMP` only clears on error, not success.

11. Pico UART Bridge Eliminated

Problem: 7-hop async UART bridge caused unreliable command delivery — wrong responses, stuck flags, variable latency. **Fix:** Complete rewrite of mesh node firmware (`ESP-Mesh-Node-sensor-test`). ESP32-C6 reads INA260 directly via I2C and controls PWM via LEDC. See `ESP/ESP-Mesh-Node-sensor-test/CHANGES.md`.

12. ALL Command Discovery + Stagger

Problem: ALL command probed non-existent addresses, causing timeouts reported to TUI. **Fix:** `discovery_complete` flag prevents redundant probes. `SENT:` notification sent before send loop. Inter-command stagger prevents radio contention.

13. PM commanded_duty Corruption

Problem: `commanded_duty` was set optimistically before confirming node received the command. Failed sends corrupted `mw_per_pct` estimates, causing PM to get stuck. **Fix:** Only update `commanded_duty` after `_wait_node_response()` returns True. Added sync check comparing sensor duty vs commanded duty.

14. PM target_duty Not Frozen on Re-Threshold

Problem: `set_threshold()` re-snapshotted `target_duty` from sensor data every time it was called. When PM had already reduced duty to 60%, changing threshold captured 60% as the ceiling — preventing ramp-up. **Fix:** Only freeze `target_duty` on first enable (`threshold_mw` going from None to a value). Subsequent threshold changes while PM is active do not re-snapshot.

15. Poll Loop Race Condition

Problem: TUI's `@work(exclusive=True)` cancels old power_poll worker when threshold changes. New `poll_loop()` sees `_polling=True` (old loop hasn't processed cancellation yet) and returns immediately. Result: no poll loop running. **Fix:** `poll_loop()` now waits up to 1s for old loop to clear `_polling` flag before giving up.

16. Debug Log Filtering

Problem: Verbose `[PM]` diagnostic messages (per-node state, skip reasons, nudge calculations) cluttered the TUI log panel. **Fix:** Added `_debug` parameter to `gateway.log()`. When `_debug=True`, message is only shown if `debug_mode` is on (F2 toggle). All `[PM]` lines tagged as debug; `[POWER]` action lines always visible.

Build & Flash

ESP32-C6 Devices

Each device is built independently:

```
# Provisioner
cd ESP/ESP-Provisioner
idf.py build flash monitor

# GATT Gateway
cd ESP/ESP_GATT_BLE_Gateway
idf.py build flash monitor

# Mesh Node (sensor-test — replaces old ESP-Mesh-Node)
cd ESP/ESP-Mesh-Node-sensor-test
idf.py build flash monitor
```

Erase flash before first build after adding vendor models:

```
idf.py erase-flash
```

If `sdkconfig.defaults` is modified, delete old config and rebuild:

```
del sdkconfig
del sdkconfig.old
rmdir /s /q build
idf.py set-target esp32c6
idf.py build
```

Flash Order (for fresh provisioning)

1. Provisioner first (waits for devices)
2. Mesh Node(s) second (get provisioned, models bound)
3. GATT Gateway third (gets provisioned, models bound)

Pi 5 Gateway

```
cd gateway-pi5
pip install bleak textual
python gateway.py           # TUI mode (default)
python gateway.py --no-tui  # Plain CLI mode (legacy)
```

---

sdkconfig.defaults

GATT Gateway

```
CONFIG_BT_ENABLED=y
CONFIG_BT_BTU_TASK_STACK_SIZE=4512
CONFIG_PARTITION_TABLE_SINGLE_APP_LARGE=y
CONFIG_BLE_MESH=y
CONFIG_BLE_MESH_NODE=y
CONFIG_BLE_MESH_RELAY=y
CONFIG_BLE_MESH_PB_GATT=y
CONFIG_BLE_MESH_TX_SEG_MSG_COUNT=10
CONFIG_BLE_MESH_RX_SEG_MSG_COUNT=10
CONFIG_BLE_MESH_GENERIC_ONOFF_CLI=y
CONFIG_BLE_MESH_SETTINGS=y
CONFIG_BT_NIMBLE_ENABLED=y
CONFIG_BT_NIMBLE_ATT_PREFERRED_MTU=185
```

Mesh Node (sensor-test)

```
CONFIG_BT_ENABLED=y
CONFIG_BT_BTU_TASK_STACK_SIZE=4512
CONFIG_PARTITION_TABLE_SINGLE_APP_LARGE=y
CONFIG_BLE_MESH=y
CONFIG_BLE_MESH_NODE=y
CONFIG_BLE_MESH_RELAY=y
CONFIG_BLE_MESH_PB_GATT=y
CONFIG_BLE_MESH_GATT_PROXY_SERVER=y
CONFIG_BLE_MESH_TX_SEG_MSG_COUNT=10
CONFIG_BLE_MESH_RX_SEG_MSG_COUNT=10
CONFIG_BLE_MESH_GENERIC_ONOFF_CLI=y
CONFIG_BLE_MESH_SETTINGS=y
```

Project Structure

```
BLE-mesh-networking/
├── ESP/
│   ├── ESP-Provisioner/                    # Auto-provisions mesh nodes
│   │   ├── main/main.c
│   │   └── documentation/
│   │       └── NETKEY_FIX.md
│   │
│   ├── ESP_GATT_BLE_Gateway/              # Pi 5 ↔ Mesh bridge
│   │   ├── main/main.c
│   │   └── documentation/
│   │       └── GATT_GATEWAY_FIXES.md
│   │
│   ├── ESP-Mesh-Node-sensor-test/         # ☑ ACTIVE — Direct I2C + PWM nodes
│   │   ├── main/main.c
```

```
│   │   └── CHANGES.md                    # Refactor documentation
│   │
│   └── ESP-Mesh-Node/                    # ❌ ARCHIVED — Old UART bridge node
│       └── main/main.c
│
├── gateway-pi5/
│   └── gateway.py                        # Pi 5 Python gateway + TUI +
PowerManager
│
├── Pico2w-power-sensing/                 # ❌ ARCHIVED — Pico sensor nodes
(replaced by sensor-test)
│   ├── DC-Monitoring-pico2w-1/
│   └── DC-Monitoring-pico2w-2/
│
├── Documentation/
│   └── pi5-GATT-ESP-pico2w-documentation/ # Historical point-to-point docs
│
├── v0.3.0-core-mesh-docs/
│   └── BLE_MESH_PROGRESS_REPORT.md       # Progress report (Feb 2026)
│
└── MESH_IMPLEMENTATION.md                # ← This file
```

Reference Files

File	Purpose
ESP/ESP-Provisioner/documentation/NETKEY_FIX.md	NetKey/msg_role bug fixes
ESP/ESP_GATT_BLE_Gateway/documentation/GATT_GATEWAY_FIXES.md	GATT init & provisioner bind fixes
ESP/ESP-Mesh-Node-sensor-test/CHANGES.md	UART→I2C refactor details
Documentation/pi5-GATT-ESP-pico2w-documentation/GATT_Point_to_Point_Documentation.md	Original 3-tier system (historical)