Memfault

How to Monitor IoT Devices at Scale

Tyler Hoffman, Co-Founder & Engineer

December 3, 2020

The Speaker

Tyler Hoffman
Co-Founder & Engineer, Memfault

- Passion: developer tools and infrastructure for embedded engineers and companies
- Previously a Firmware Engineer @ Pebble & Fitbit
- Split time between writing firmware and building internal services to help monitor millions of devices
- Can find my thoughts and content on Memfault's Interrupt blog (interrupt.memfault.com)

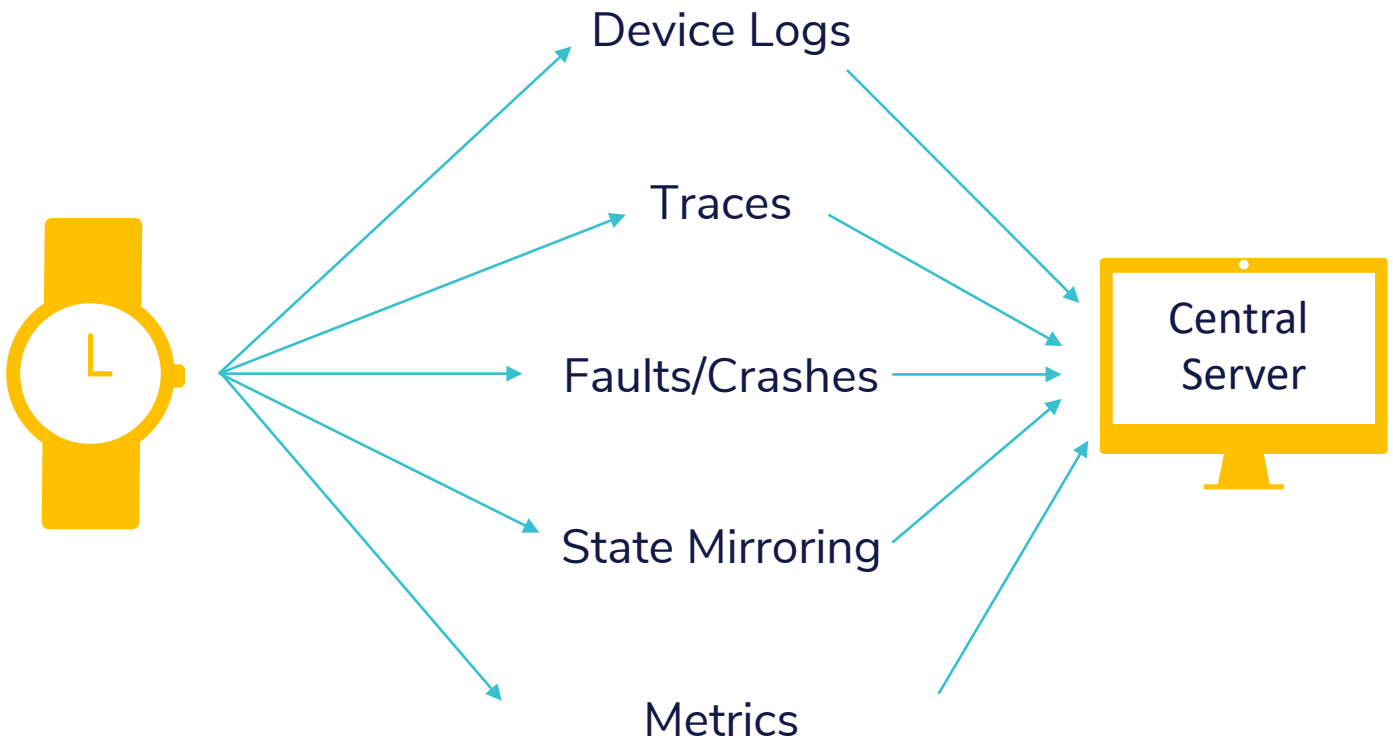
pebble fitbit Memfault

Agenda

1. **Monitoring Overview**
2. What are Metrics?
3. Implementing Metrics
4. Collecting Metrics
5. Making Decisions with Metrics

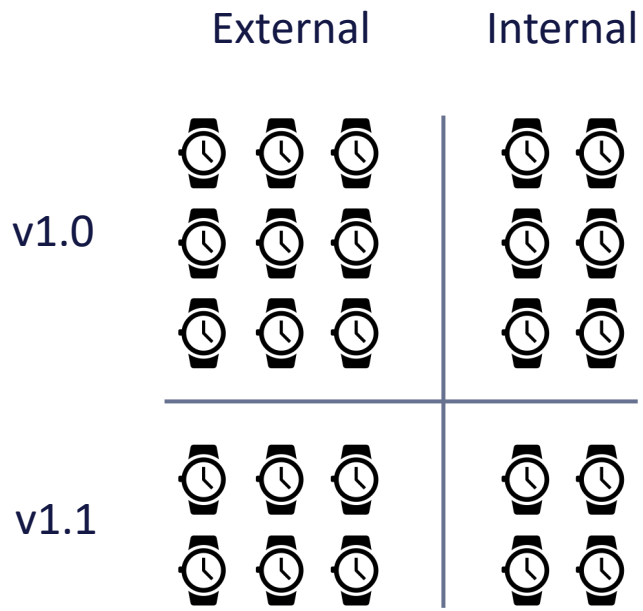
1 Monitoring Overview

Monitoring Strategies



Embedded Device Monitoring Criteria

1. **Ease of implementation**
2. **Measure "health" across dimensions**
3. Bandwidth and connectivity
4. Processing complexity
5. Scalability
6. Clock time not required (bonus)



Monitoring Types Compared

	Logging	Tracing	Fault & Crash Analysis	State Mirroring	Metrics
Easy to Implement	✓	✓	■	■	✓
Fleet & Version Health	■	■	■	✗	✓
Minimal Bandwidth	■	■	■	✗	✓
Cheap Infrastructure	✗	■	■	✗	✓
Can debug single device issues	✓	✓	✓	✗	■

Where ✓ is a benefit, ■ is neutral, and ✗ is not a benefit.

Agenda

1 Monitoring Overview

2 **What are Metrics?**

3 Implementing Metrics

4 Collecting Metrics

5 Making Decisions with Metrics

2 What are Metrics?

Metrics

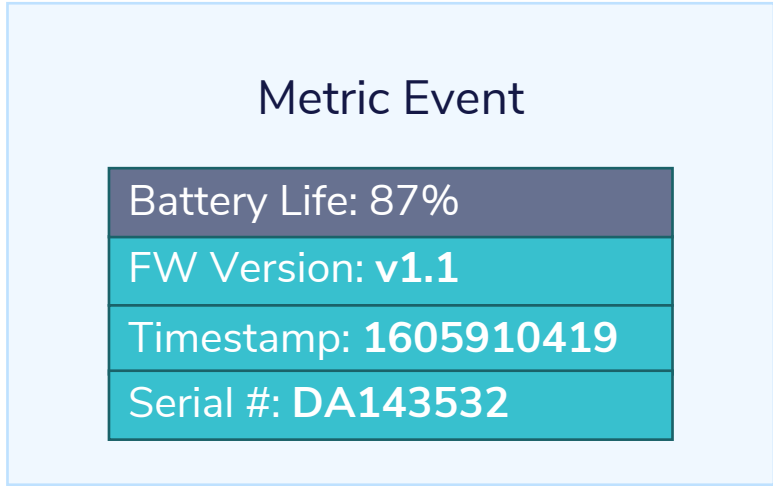
A **metric** is a measurement captured at runtime

Combing large numbers of metrics and calculating statistics is called an **aggregation**

Metrics

Proven methods for monitoring using metrics

- StatsD, Prometheus, OpenTelemetry
- Integers or floats
- Tags or Labels - firmware version and device serial
- Months and years of data
- Personally Identifiable Information (PII) unlikely



Metrics

Uncover trends on anything that is numerical

Common metrics are:

- Task runtimes
- Heap information
- Connectivity statistics
- Frequency of errors
- Peripheral utilization and power states

```
Example App Booting
Metric ID: 1 -- Value: 15000  # elapsed ticks
Metric ID: 2 -- Value: 8531   # Main Task ticks
Metric ID: 3 -- Value: 8223   # Timer Task ticks
Metric ID: 4 -- Value: 14     # Timer Task count
Metric ID: 5 -- Value: 3593   # "sensor on" ticks
Metric ID: 6 -- Value: 4696   # heap-free low watermark
```

Pinpoint power, connectivity, and performance issues and regressions

Aggregate Metrics

Per Device

Counts
Averages
Min/Max

0110...

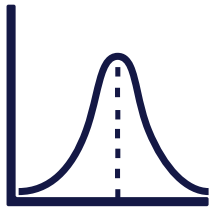
0001...

1110...

Central Server

Entire Fleet

Counts
Averages
Min/Max



7.2 days battery life

1.7 reboots/day

87% connected

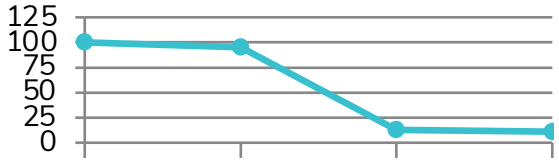
Product, Engineering, & Business Decisions

Metric Uses

Metrics serve three purposes:

1. View device's vitals over time

Battery Life %:



2. Assessing the collective health of all devices

% Time Connected per Hour: 95% -> 77%

3. Enable comparing metrics between firmware versions

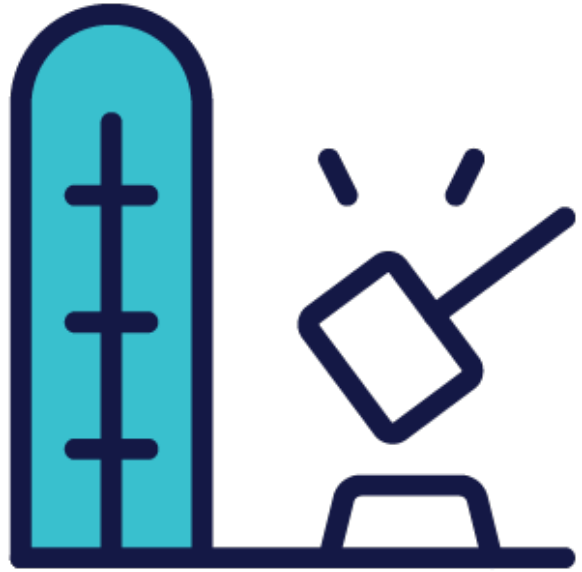
Average Battery Life:

v1.0 - 5.5 days

v1.1 - 4.1 days

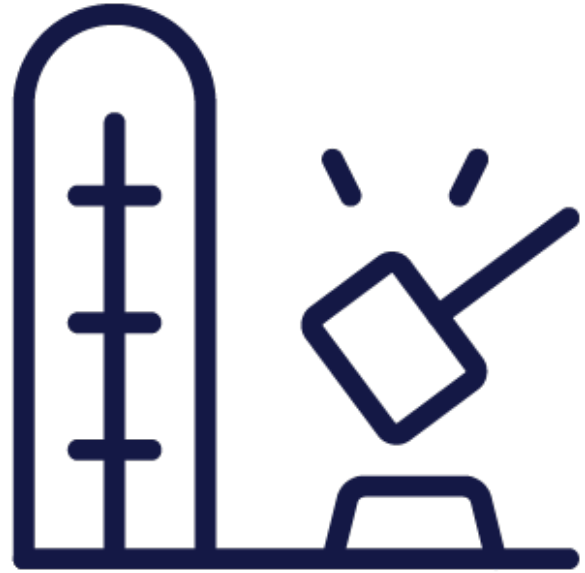
Metric Strengths

- Compare health across all devices and firmware releases
- Battery life, connectivity, and performance issues
- Highly compressed
- Negligible performance impact
- Simple instrumentation on device



Metric Weaknesses

- Awareness of potential issues
- SQL knowledge required
- Paradigm shift for embedded developers
- Supplemental monitoring solutions still required



Agenda

1 Monitoring Overview

2 What are Metrics?

3 **Implementing Metrics**

4 Collecting Metrics

5 Making Decisions with Metrics

3 Implementing Metrics

Three Common Types of Metrics

Counter	Accumulated value for the heartbeat duration
Timer	Total time in a particular state for the heartbeat duration
Gauge	Instantaneous values, set at beginning or end of the heartbeat

*These are the three metric types from StatsD

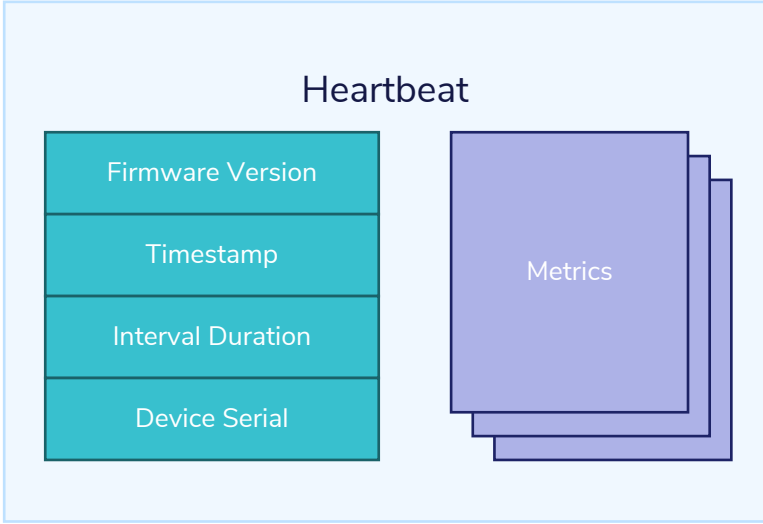
Device Hearbeats

Periodic “pulse” sent from device to a monitoring service

- Is a device functional?
- Vitals and metrics
- Metadata of the running firmware
- Packaged and batched up on a device

Constraints?

Send a heartbeat every **minute, hour, or day**



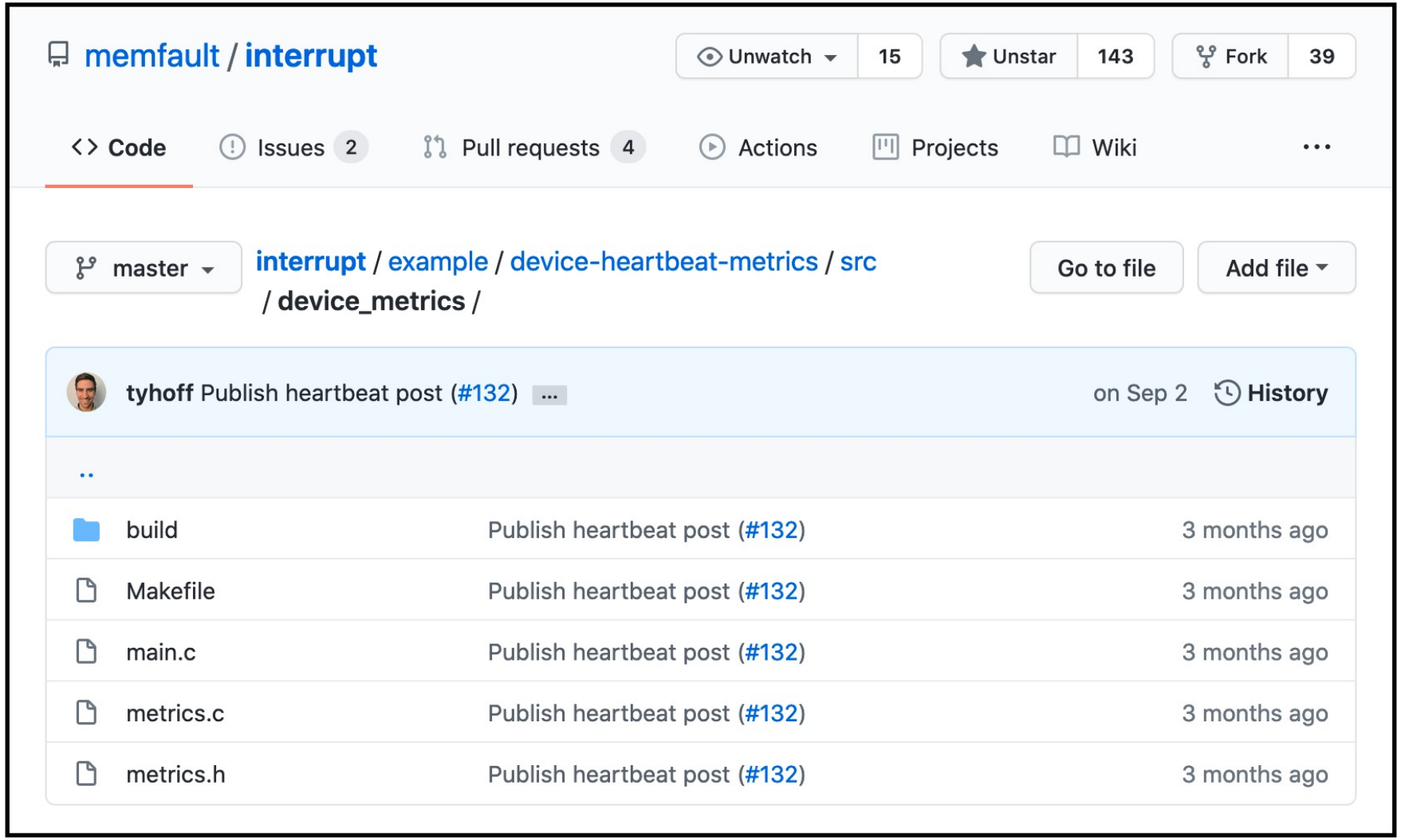
Implementing Heartbeats & Metrics

Building out a heartbeat and each metric type by providing a few examples

Using source code from a simple device heartbeat library I wrote for the Interrupt post on the topic.

https://interrupt.memfault.com/blog/device-heartbeat-metrics

Implementing Heartbeats & Metrics



Implementing Device Heartbeats

Metric ID Definitions

```
typedef enum {
    kDeviceMetricId_INVALID = 0,
    kDeviceMetricId_ElapsedTime = 1,
    kDeviceMetricId_MainTaskTime = 2,
    kDeviceMetricId_TimerTaskTime = 3,
    ...
} eDeviceMetricId;
```

Metric ID's - unsigned integers (**1-4 bytes)**

Metric values - integers or floats **(4 bytes)**

Source code: https://interrupt.memfault.com/blog/device-heartbeat-metrics

Implementing Device Heartbeats

Device Metrics API

```
// Counters
void device_metrics_incr(eDeviceMetricId metric_id);
void device_metrics_incr_by(eDeviceMetricId metric_id,
                            int32_t n);

// Timers
void device_metrics_timer_start(uint32_t *start);
void device_metrics_timer_end(eDeviceMetricId metric_id,
                              const uint32_t *tick_buf);

// Gauges
void device_metrics_set(eDeviceMetricId metric_id,
                        int32_t value);

// Hourly Flush
void device_metrics_flush(void);
```

Source code: https://interrupt.memfault.com/blog/device-heartbeat-metrics

Agenda

1 Monitoring Overview

2 What are Metrics?

3 Implementing Metrics

4 **Collecting Metrics**

5 Making Decisions with Metrics

4 Collecting Metrics

Counter Metric

Accumulated value for heartbeat duration

Useful for:

Detecting behavioral differences of firmware and devices between releases and over time

Examples:

- bytes sent/received
- connectivity events
- flash operations
- mutex_lock failures
- # times a feature was used

Counter Metric

Adding counters is easy...just add a single line!

```
void flash_sector_erase(uint16_t sector) {
    ...
    device_metrics_incr(kDeviceMetric_FlashSectorErases);
}

void flash_write_bytes(uint32_t addr, void *buf, size_t n) {
    ...
    device_metrics_incr_by(kDeviceMetric_FlashWriteBytes, n);
}
```

Source code: https://interrupt.memfault.com/blog/device-heartbeat-metrics

Counter Metric

RAM-backed value that is incremented

```
void device_metrics_incr_by(eDeviceMetricId metric_id,
                            int32_t n) {
    int32_t *val = prv_get_value_ptr(metric_id);
    *val += n;
}
```

Source code: https://interrupt.memfault.com/blog/device-heartbeat-metrics

Timer Metric

Stores amount of time spent in a particular state

Useful For:

Detecting regressions in performance and user-experience, and battery life.

Examples:

- CPU utilization & sleep time
- Connectivity radio utilization
- Sensor & peripheral utilization
- Task utilization
- Time spent in syscalls

Timer Metric

Two lines of code

```
static uint32_t s_wifi_ticks;

void wifi_connected_callback(void) {
    // Start timer when Wi-Fi connects
    device_metrics_timer_start(&s_wifi_ticks);
}

void wifi_disconnected_callback(void) {
    // Stop timer when Wi-Fi connects
    device_metrics_timer_end(kDeviceMetricWifiConnectedDuration
                             &s_wifi_ticks);
}
```

Source code: https://interrupt.memfault.com/blog/device-heartbeat-metrics

Timer Metric

Measure Task Utilization

```
static void prvAltTask(void *pvParameters) {
    uint32_t task_tick;
    while (1) {
        unsigned long received;
        xQueueReceive(xQueue, &received, portMAX_DELAY);

        device_metrics_timer_start(&task_tick) {

        // Do work

        device_metrics_timer_stop(kDeviceMetricId_AltTaskTime,
                                  &task_tick) {
    }
}
```

Source code: https://interrupt.memfault.com/blog/device-heartbeat-metrics

Gauge Metric

Instantaneous values from beginning or end of interval

Useful for:

Obtaining a sampling of the system state and calculating prevalence of an issue across a fleet.

Examples:

- Battery Life
- Battery drain during interval
- Current heap bytes used/free
- Heap & stack high-water marks

Gauge Metric

Captured at flush time.

```
void device_metrics_flush(void) {
    ...
    static int32_t s_prev_battery_pct;

    const int32_t current_battery_pct = battery_get_pct();
    const int32_t battery_delta =
        current_battery_pct - s_prev_battery_pct;

    device_metrics_set(kDeviceMetricId_BatteryLifeDrain,
                       battery_delta);
}
```

Source code: https://interrupt.memfault.com/blog/device-heartbeat-metrics

Heartbeat Interval Flush

A timer can be used to flush metrics every hour

```
TimerHandle_t metrics_flush_timer =
    xTimerCreate("heartbeatFlush",
                 1000 * 60 * 60, /* interval */
                 pdTRUE,
                 (void*)0,
                 prv_metrics_flush);

xTimerStart(metrics_flush_timer, 0);
```

At flush time:

- Set gauge values
- Metric values are flushed to persistent storage
- Reset metrics

Source code: https://interrupt.memfault.com/blog/device-heartbeat-metrics

Memfault Heartbeats

Simple Heartbeat Library

https://interrupt.memfault.com/blog/device-heartbeat-metrics

Memfault Heartbeat Library

https://github.com/memfault/memfault-firmware-sdk

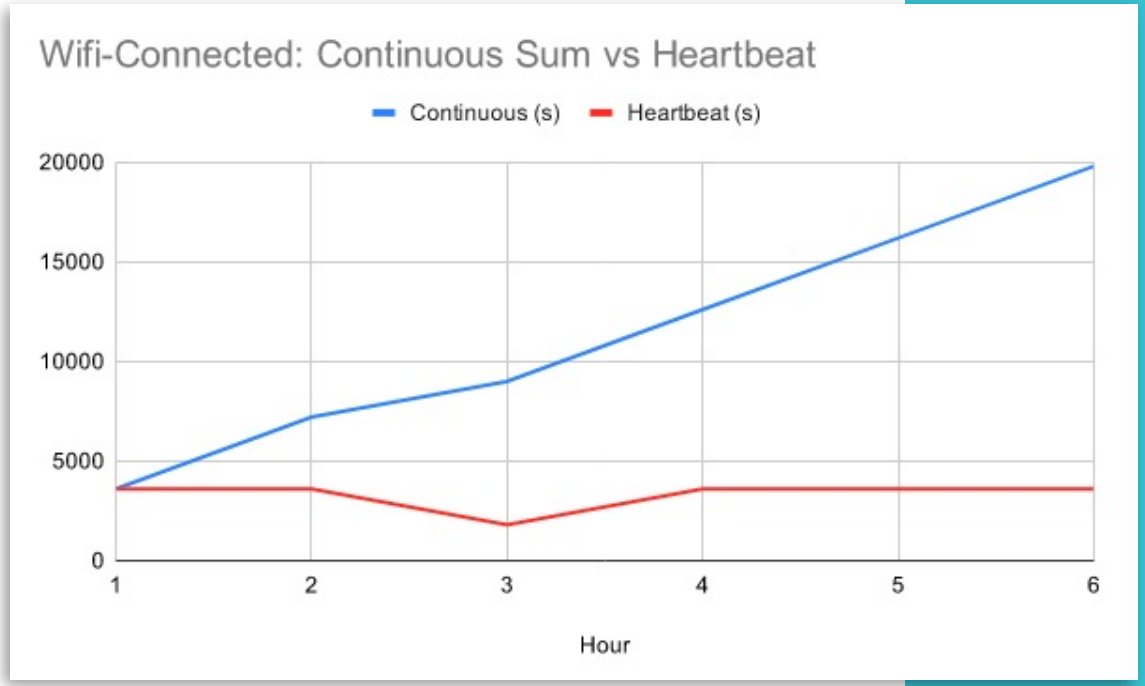
Heartbeat Best Practices

1. Reset data after each heartbeat
2. Include metadata with heartbeats
3. Be creative

Source code: https://interrupt.memfault.com/blog/device-heartbeat-metrics

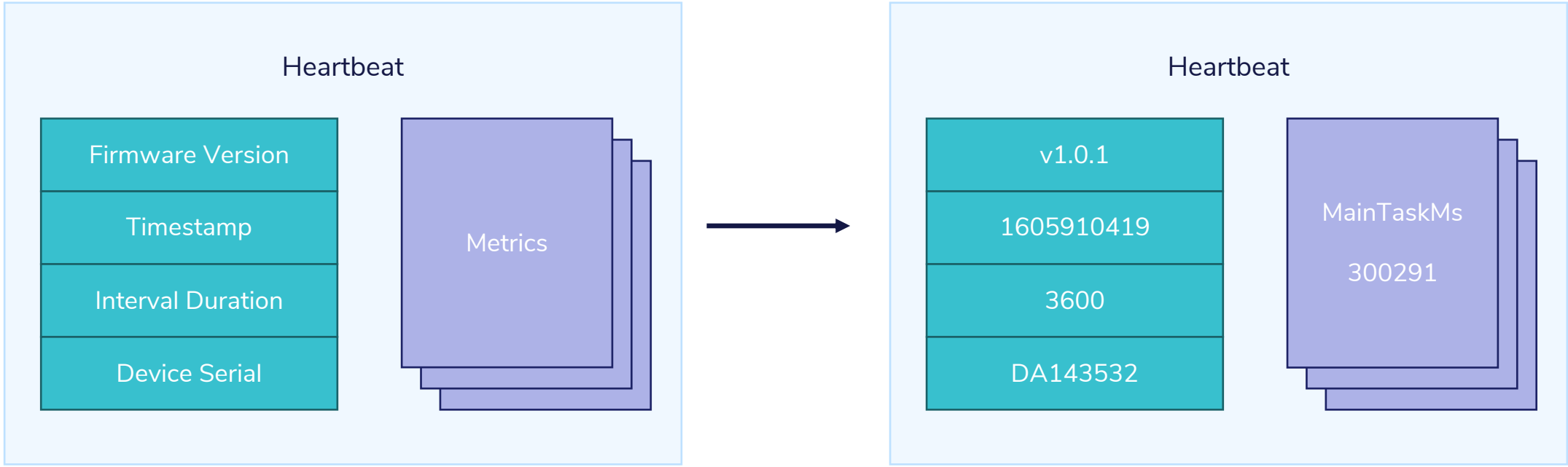
Resetting Counters

- Easily spot issues and trends
- Resilient against resets and data loss
- Simpler math



Source code: https://interrupt.memfault.com/blog/device-heartbeat-metrics

Include Metadata with Heartbeats



Be Creative

An int32_t can store **a lot of data**!

- Register Values
- Hardware Readings
- 4 ASCII chars
- Bitfields
- Device Configuration
- 32 booleans

What about logs?

Transform logs into metrics?

```
[I][1605794400] Wi-Fi connected
[W][1605796200] Wi-Fi disconnected, reason: -2
[I][1605796500] Wi-Fi connected
[W][1605797100] Wi-Fi disconnected, reason: -2
[I][1605798000] Battery status: 67%, 3574 mV
```

Not really...

Transforming Logs

```
[I][1605794400] Wi-Fi connected
[W][1605796200] Wi-Fi disconnected, reason: -2
[I][1605796500] Wi-Fi connected
[W][1605797100] Wi-Fi disconnected, reason: -2
[I][1605798000] Battery status: 67%, 3574 mV
```

```
wifi_connected_s: 144000
battery_life_pct: 67
```

Transforming Logs

What if a log is missed? Or the device hiccups?

```
[I][1605794400] Wi-Fi connected
-------------   MISSING LOG   ----------------
[I][1605796500] Wi-Fi connected
[W][1605797100] Wi-Fi disconnected, reason: -2
[I][1605798000] Battery status: 67%, 3574 mV
```

```
wifi_connected_s: ??????
battery_life_pct: 67
```

Transforming Logs

Dropped logs = we can't **reliably** calculate metrics

Also:

- Complex log processing
- Developers update logs constantly
- Higher bandwidth requirements

From my experiences of monitoring 2M+ devices:

Generating metrics directly is easier

Agenda

1. Monitoring Overview
2. What are Metrics?
3. Implementing Metrics
4. Collecting Metrics
5. **Making Decisions with Metrics**

5 Making Decisions with Metrics

Go / No-Go Metrics

Are we good to ship this firmware?

Battery Life	Projected Days of Battery Life
Connectivity	% Connected per Hour
Stability	% Crash-Free Hours

*Do **not** ship firmware **without** hitting your metrics.*

Projected Battery Life

$$\left(100 \div \frac{Total\ Battery\ \%\ Drained\ Over\ All\ Heartbeats}{\#\ of\ Heartbeats}\right)$$

$$= Projected\ Hours\ Battery\ Life$$

Projected Battery Life

$$\left(100 \div \frac{Total\ Battery\ \%\ Drained\ Over\ All\ Heartbeats}{\#\ of\ Heartbeats}\right)$$

Average percent battery drain = 2%

$$100 \div 2 = \boldsymbol{50}\ Projected\ Hours\ Battery\ Life$$

% Time Connected per Hour

$$\left(\frac{Average\ Seconds\ Connected\ Over\ All\ Heartbeats}{3600\ seconds}\right) \times 100$$

$$= \%\ Time\ Connected\ per\ Hour$$

% Time Connected per Hour

$$\left(\frac{Average\ Seconds\ Connected\ Over\ All\ Heartbeats}{3600\ Seconds}\right) \times 100$$

Average time connected is 3500 (out of 3600) seconds for all devices:

$$\left(\frac{3500}{3600}\right) \times 100 = 97.2\%\ Time\ Connected\ per\ Hour$$

% Crash-Free Hours

By recording whether a device crashed during a heartbeat interval

$$\left(1 - \frac{\#\ Crashing\ hours}{\#\ Total\ Hours}\right) \times 100 = \ \% \ Crash{-}Free\ Hours$$

% Crash-Free Hours

$$\left(1 - \frac{\#\ Crashing\ hours}{\#\ Total\ Hours}\right) \times 100 = \ \%\ Crash{-}Free\ Hours$$

2000 hours out of 24000 had a crash:

$$\left(1 - \frac{2000}{24000}\right) \times 100 = 91.6\%\ Frash{-}Free\ hours$$

7 day average between crashes => **99.5%** crash-free hours

Agenda

1. Monitoring Overview
2. What are Metrics?
3. Implementing Metrics
4. Collecting Metrics
5. Making Decisions with Metrics

So we built...

Memfault

Brief Demo

Memfault & Metrics

- Open-source firmware SDK
- Heartbeat module included
- Turn-key integrations
- We deal with scaling

Trusted By

logitech

proxy

willow

ŌURA

WHOOP®

skip

THANKS!

@ty_hoff

https://www.linkedin.com/in/tyhoff/

tyler@memfault.com

https://interrupt.memfault.com