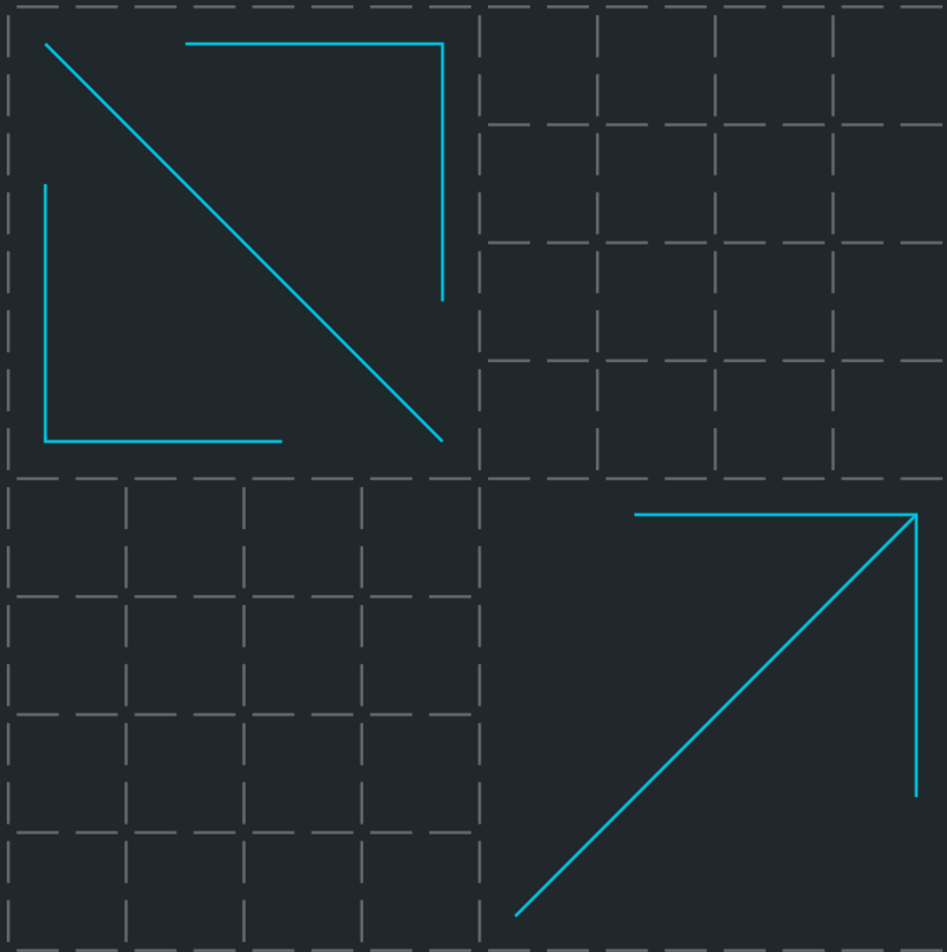


Automated Failure Analysis of Remote Edge Devices at Scale

Tyler Hoffman



Speaker

Tyler Hoffman, Co-founder @ Memfault



Previously a Firmware Engineer @ Pebble & Fitbit

Split time between writing & debugging 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

What are we trying to do today?

- Talk about firmware on embedded MCU systems
- Speed from **development** to **testing** to **mass production** and **scaling**
- Find and fix (quickly) 1 in 10,000 hour bugs in production
- Prevent issues from being released to 100% of users
- Establish measurable stats to track progress on goals

One of my favorite quotes

“These techniques are not necessary because experienced firmware engineers do not introduce bugs in their code”

- A (real) Firmware Engineer

Agenda

1

State of the union

2

Quantifying current issues

3

Analyzing the issues

4

Preventing future issues

5

Best Practices



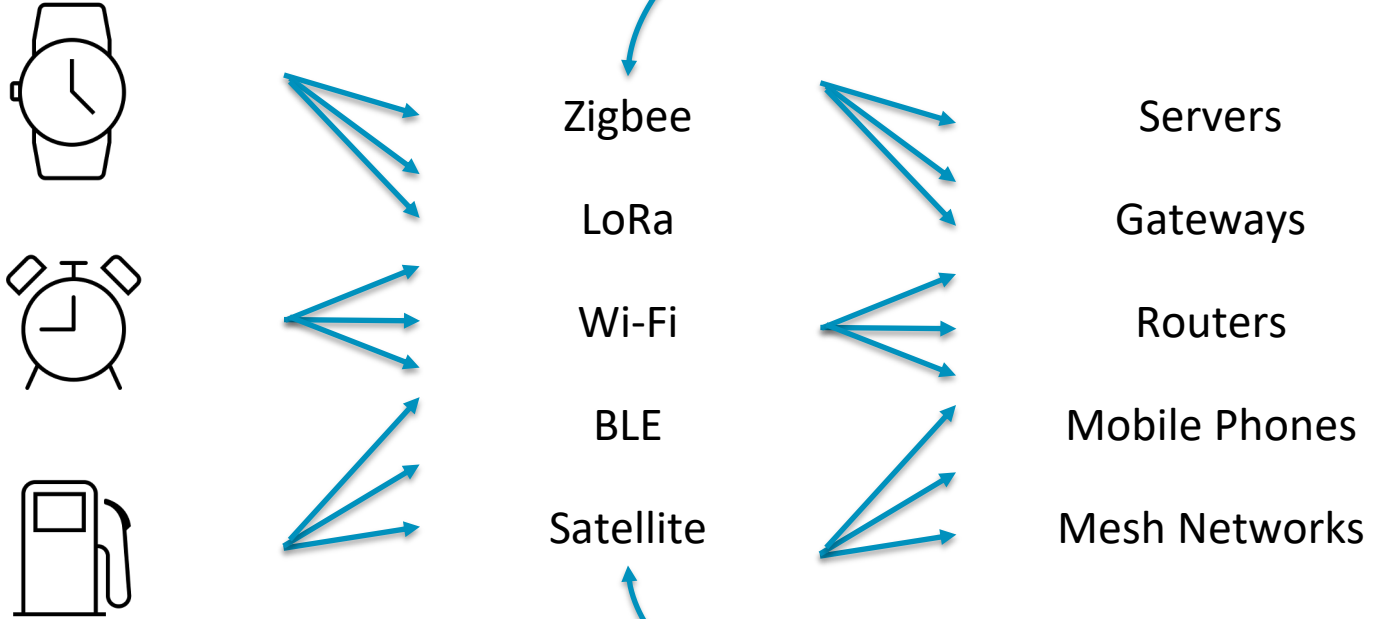
State of the union for firmware

Firmware is *pervasive*, but difficult

- In Q4 2020, 4.4 billion Cortex M's shipped
- Tons and tons of firmware running on these devices
- Very few tools to help debug these devices
- Software is becoming more complex - leads to bugs
- Software issues lead to:
 - Bricked devices, RMA's, and security exploits

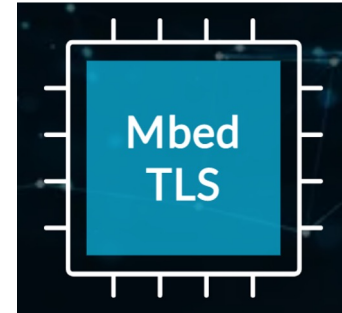
We need proper tools to help us build quality products

Increasingly complex topologies



Increasingly complex software

- Firmware no longer is complete isolated
 - Must communicate to mobile phones & gateways
 - Protocols, devices, and security issues constantly changing
 - Libraries are *massively complicated* (e.g. mbedTLS)
- Expectations are growing
 - Everything connects to Internet
 - Must enable firmware updates
 - Competition is growing



Rudimentary Debugging Tools

***** MPU FAULT *****

Instruction Access Violation

***** Hardware exception *****

Current thread ID = 0x20000074

Faulting instruction address = 0xe0000000

Fatal fault in thread 0x20000074! Aborting.

```
[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
```

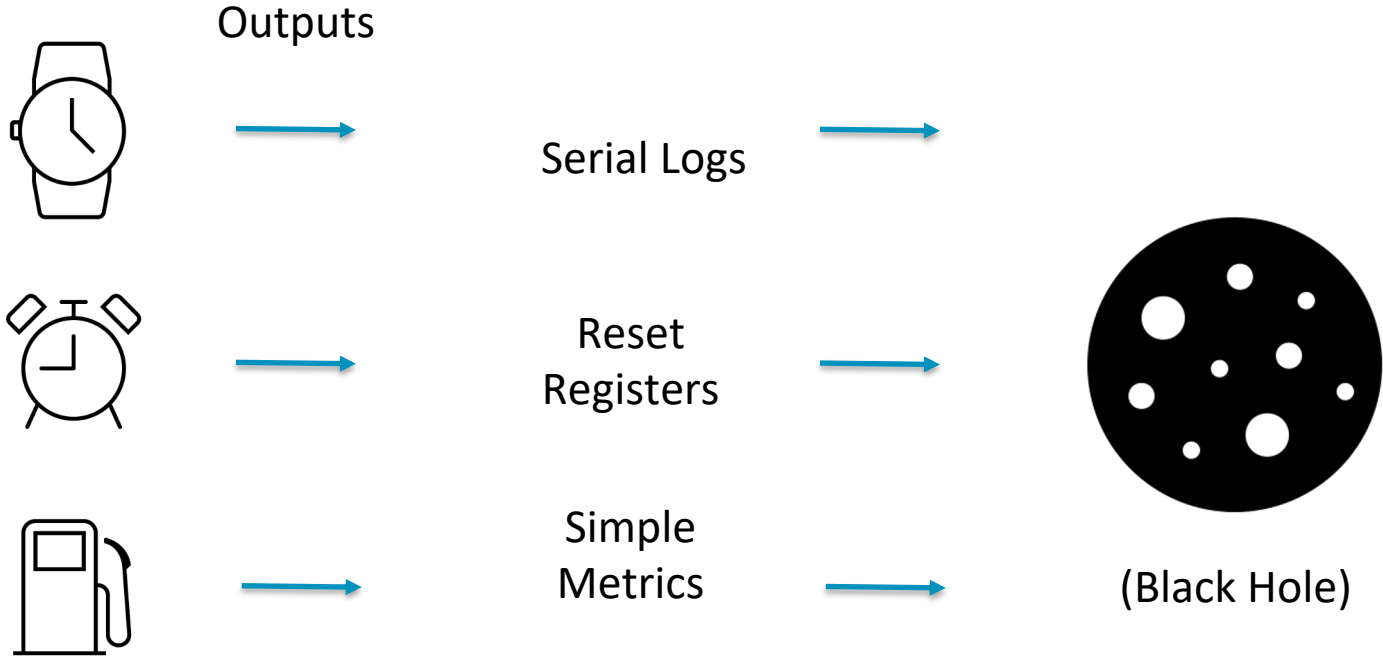
Tasks					
ID	Priority	Name	Stack Info (Used / Size)	Status	Run Count
40	113	IP_Task	564 / 1280 @ 0x20006708	Waiting (event), with Timeout	2943
→ 198	108	GUI_Task	1324 / 4096 @ 0x200060F0	Running	4170
97	106	USBMSDTask	408 / 4096 @ 0x20008C20	Suspended, with Timeout	468
140	105	IP_WebServer	672 / 2048 @ 0x20006E78	Waiting (event object)	2
244	101	IP_FTP_Server	572 / 2072 @ 0x20007AF0	Waiting (event object)	0
		Idle	Idle		

Call Stack				
Function	Stack Info	PC	Return Address	
→ _RadialMenu	0 @ 2000 61E8	0000 27A4	0000 2A0E	
GUIDEMO_RadialMenu	8 @ 2000 61E8	0000 2A0A	0000 1162	
_Main	8 @ 2000 61F0	0000 1160	0000 1508	
GUIDEMO_Main	64 @ 2000 61F8	0000 1504	0000 B3B0	
OS_StartTask	0 @ 2000 6238	0000 B3AE	N/A	

Local Data @ _RadialMenu				
Name	Value			
aItemInfo				
Para				
hMotion	160			
hDraw	37	2000 61C4	4	long
pPara	0x80	2000 61BC	4	struct PARA*
xSizeWin	160	2000 61D0	4	int
ySizeWin	4 995	2000 61CC	4	int
xSize	518 248	2000 61D8	4	int
ySize	5	2000 61D4	4	int
xPos	0	2000 61C8	4	int
i	-859 045 884	2000 61DC	4	int

The existing tools are decent, **but only when connected using JTAG**

Debugging Data Exported to ???



Not much exists to help us

- Existing software solutions won't work
 - We aren't monitoring 10-100 servers, rather **10k – 1m+ devices**
 - We don't have MB's and GB's of RAM lying around, **we have KB's**
 - Assembly, C, C++, Rust (?)
 - Symbol files (.elf's) required for typical debugging experiences
 - AWS IoT,
- Few existing solutions that can help embedded devices
- Firmware is not an open-source ecosystem or sharing community

Whatever we need, we have to build ourselves

Some issues will only happen in the field

“a third of all software faults take more than 5000 execution-years to manifest themselves.”

*Source: <http://www.ganssle.com/tem/tem417.html>

Agenda

1

State of the union

4

Preventing future issues

2

Quantifying current issues

5

Best Practices

3

Analyzing the issues



Detecting Current Issues

Do we have issues in the field...

They're likely having issues

Detecting current issues

Fundamentally, we need to know if and how many times devices are experiencing issues.

Once we have this information

- Can assess whether further work is needed.
- Can make assumptions about whether a firmware update improved stability
- Can guide business & product decisions
- Leaders know whether it's time to fix bugs and tech debt or build new features

Detecting current issues

```
void buggy_function(void) {  
    *(uint32_t *)0xbadcafe = 0x0;  
}
```



```
void HardFault_Handler(void) {  
    // ... fault handling code ...  
    NVIC_SystemReset();  
}
```

Detecting current issues

```
::::: LOGS :::::  
ERROR: Connection lost - reason -7
```

Detecting current issues

- Sending & Parsing logs
 - Log important device events
 - Log resets and registers
 - **Hard and doesn't scale well**



- Tracking simple metrics
 - # device resets
 - # devices alive and well
 - Average uptime of devices
 - **Scalable, but less information**



Requirements:

1. Path to the Internet
2. Central repository for device data

Detecting current issues



Reset

```
{  
  "reason": "watchdog",  
}
```

```
"uptime": 86345
```

Hourly
Heartbeat

```
"ble_connected_s": 3212
```

```
"main_stack_min": 16
```

Aggregate!

Watchdogs: 67
Hardfaults: 23
Shutdowns: 17

Number devices: 890
Average Uptime: 90,234 seconds

Average BLE Connected per hour: 3212 seconds
% Connected per hour: 89%

Stack low watermark is 16 bytes on some devices. Yikes!

Agenda

1

State of the union

4

Preventing future issues

2

Quantifying current issues

5

Best Practices

3

Analyzing the issues



Analyzing the Issues

We have issues. How do we fix them?

Debugging in Production

- We don't have access to our normal toolset.
 - No UART & serial logs
 - No JTAG and quick re-flashing
 - No IDE, GDB, or step-through debugging
 - RMA's take weeks. Devices must persist the data.
- We need tools specifically for debugging in production
 - Build these tools **as early as possible**
 - Having them early will pay off **10x** (Speaking from experience)
 - If you have no interest in building them, check out **Memfault**

Using Logs to Analyze Issues

```
[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
```

2 Wi-Fi disconnect events

1 Battery level event

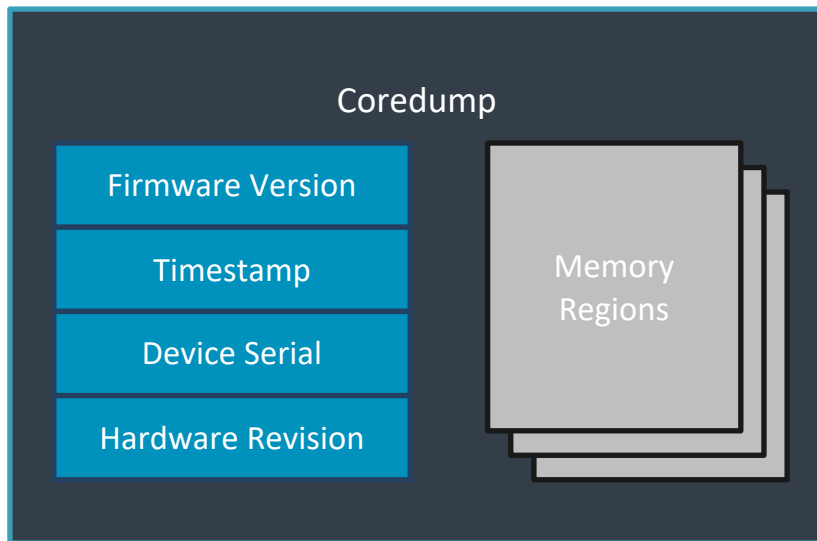
```
***** MPU FAULT *****  
Instruction Access Violation  
***** Hardware exception *****  
Current thread ID = 0x20000074  
Faulting instruction address = 0xe0000000  
Fatal fault in thread 0x20000074! Aborting.
```

1 crash due to MPU violation

Program Counter:
0xe0000000

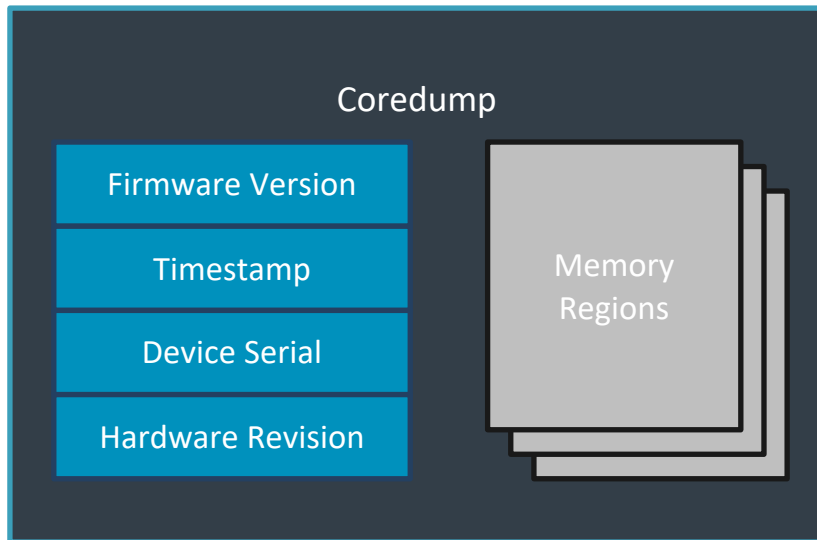
Using Coredumps to Analyze Issues

- What is a coredump
 - A snapshot of RAM regions at a specific moment in time.
 - Can be captured on command, or at time of a fault, such as a hardfault or ASSERT.
 - Helps with post-mortem debugging
 - Requires symbol file & clever parsing
- What RTOS's support coredumps natively?
 - ESP32, Zephyr RTOS, MyNewt RTOS
 - Any RTOS with Memfault



Using Coredumps to Analyze Issues

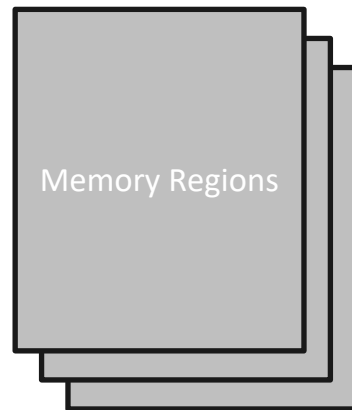
```
void HardFault_Handler(void) {  
    // ... fault handling code ...  
    NVIC_SystemReset();  
  
    // Don't just crash...  
    capture_coredump();  
}
```



Using Coredumps to Analyze Issues

A coredump is comprised of raw memory regions

- From coredumps, one can recover:
 - Threads and stacktraces
 - Heap allocations and statistics
 - Global variables
 - Local variables and function arguments
 - System and peripheral registers
 - Anything else you want to collect (and parse)



Using Coredumps to Analyze Issues

The screenshot displays a debugger interface with two main panels. The left panel lists threads and their backtraces, while the right panel shows the current register values.

Thread Information:

- accel-workq (2)** (STACK OVERFLOW RUNNING)
 - 0 compute_fft in .../src/fft.c at line 10
 - 1 sleep_algo_compute_sleep_time in .../src/sleep_algo.c at line 12
 - 2 process_accel_data_worker_task in .../src/accel_data.c at line 106
 - 3 z_work_q_main in .../zephyr/lib/os/work_q.c at line 32
 - 4 z_thread_entry in .../lib/os/thread_entry.c at line 29
 - 5 0xaaaaaaaa
- Thread 3** (SUSPENDED)
- idle (4)** (READY)
- logging (5)** (SUSPENDED)
- net_mgmt (6)** (BLOCKED)
- rx_workq (7)** (BLOCKED)
- shell_uart (8)** (BLOCKED)
- sysworkq (9)** (BLOCKED)
- tx_workq (10)** (BLOCKED)
- workqueue (11)** (BLOCKED)

Register Values:

- A** dft_out = 0x2000a900 <my_stack_area+1344>
- L** i = 400
- A** num_samples = 536912536
- A** raw_samples = 0x3128115f
- L** tmp = {1, 222, 7, 84}
- R** \$r0 = long 536912536 (0x2000a298)
- R** \$r1 = long 1372324912 (0x51cc0430)
- R** \$r2 = long 1372324919 (0x51cc0437)
- R** \$r3 = long 536912832 (0x2000a3c0)
- R** \$r4 = long 536912508 (0x2000a27c)
- R** \$r5 = long 536914136 (0x2000a8d8)
- R** \$r6 = long 0 (0x00000000)
- R** \$r7 = long 536912488 (0x2000a268)
- R** \$r8 = long 0 (0x00000000)
- R** \$r9 = long 0 (0x00000000)
- R** \$r10 = long 0 (0x00000000)

Threads, backtraces, local variables, and registers

Using CoreDumps to Analyze Issues

Analysis

Memory management fault detected at 0x2000a3c0

Memory management fault on a data access

Fault Register	Value	Hex Value
CFSR	130	0x00000082
HFSR	0	0x00000000
SHCSR	458753	0x00070001

Parse and interpret system registers

Using CoreDumps to Analyze Issues

The screenshot displays a memory dump analysis tool interface. On the left, a list of global variables is shown with their types and values. In the center, a column lists the memory addresses for each variable. On the right, a detailed view of the memory dump shows the raw data and its hexadecimal representation.

Variable	Memory Location	Raw Data
heap_sz = unsigned int 0	@ 0x20002378	0x20009f58 ec 31 00 20 00 00 00 00
▶ _mbedtls_heap = unsigned char[28000] {...}	@ 0x200031ec	0x20009f60 00 00 00 00 00 00 00 00
▼ heap = buffer_alloc_ctx {...}	@ 0x20009f4c	0x20009f68 00 00 00 00 00 00 00 00
▶ buf = unsigned char* {...}	@ 0x20009f4c	0x20009f70 00 00 00 00 00 00 00 00
len = size_t 28000	@ 0x20009f50	0x20009f78 00 00 00 00 00 00 00 00
▶ first = memory_header* {...}	@ 0x20009f54	0x20009f80 00 00 00 00 00 00 00 00
▼ first_free = memory_header* {...}	@ 0x20009f58	0x20009f88 01 00 0f 00 00 00 00 00
▼ * = memory_header {...}	@ 0x200031ec	0x20009f90 00 00 00 00 00 00 00 00
magic1 = size_t 4278233685	@ 0x200031ec	0x20009f98 00 00 00 00 00 00 00 00
size = size_t 27968	@ 0x200031f0	0x20009fa0 00 00 00 00 ad c2 02 08
alloc = size_t 0	@ 0x200031f4	0x20009fa8 00 00 00 00 00 00 00 00
▶ prev = memory_header* {...}	@ 0x200031f8	0x20009fb0 00 00 00 00 00 00 00 00
▶ next = memory_header* {...}	@ 0x200031fc	0x20009fb8 00 00 00 00 00 00 00 00
▶ prev_free = memory_header* {...}	@ 0x20003200	0x20009fc0 00 00 00 00 48 e2 00 20
▶ next_free = memory_header* {...}	@ 0x20003204	0x20009fc8 00 00 00 00 00 00 00 00
magic2 = size_t 3994130790	@ 0x20003208	0x20009fd0 ad c2 02 08 00 00 00 00
verify = int 0	@ 0x20009f5c	0x20009fd8 00 00 00 00 00 00 00 00
▶ heap_sem = sys_sem {...}	@ 0x2001166c	0x20009fe0 00 00 00 00 69 64 6c 65

View all global variables at time of coredump

Coredumps – What can I debug?

- Crashes and other items worth extra investigation
 - Hardfaults, Memfaults, Asserts, etc.
 - Deadlocks, system stalls
 - Memory corruption issues
 - Heap out-of-memory
 - Stack overflows
 - Undefined behavior
("SHOULD NEVER HAPPEN" 😊)

```
static void prv_handle_new_state(eWifiState state) {  
    switch (state) {  
        case kWifiState_Off:  
            do_something();  
            break;  
        case kWifiState_Connecting:  
            do_something();  
            break;  
        case kWifiState_Connected:  
            do_something();  
            break;  
        // !! Should never get here. !!  
        case kWifiState_UNDEFINED:  
            TRIGGER_COREDUMP();  
    }  
}
```

<https://embeddedartistry.com/blog/2021/01/11/hard-fault-debugging/>

When Coredumps Aren't Enough

- Coredumps aren't the perfect solution for all issues
 - Behavioral bugs require more of a timeline than a snapshot (if this & this, then that)
 - Devices with storage or bandwidth constraints
 - Issues in interpreted languages (MicroPython, JerryScript)
 - Determining trends
- When you need context before a crash...
 - Use logging or simple tracing
 - Store log lines in a RAM buffer, send it with coredump!

State Logs	
Search	
Lvl	Message
I	Initializing Accel Subsystem
I	Launching sleep tracking app
I	Wifi Connected.
I	Analyzing Raw Algo Data (50B)
I	Wifi Unavailable. Retrying in 5 min
I	sleep tracking app closed
I	Launching sleep tracking app
E	Temp Sensor I2C transaction timeout, rv=-8
I	Wifi Connected.
I	Analyzing Raw Algo Data (100B)

Agenda

1

State of the union

2

Quantifying current issues

3

Analyzing the issues

4

Preventing future issues

5

Best Practices



Preventing Issues

We fixed the issues. How do we don't introduce more?

Internal Testing

- You must be testing the devices internally
 - Catch and fix as many bugs before production as possible
 - Firmware team, QA, engineers, and other employees
 - Ask biggest fans of your product to be beta testers
 - Encourage users to report bugs
 - Less concerns around privacy & data collection
- Build and deploy nightly or regular firmware updates to devices
 - Constantly be fixing and verifying fixes
 - Perform experiments!

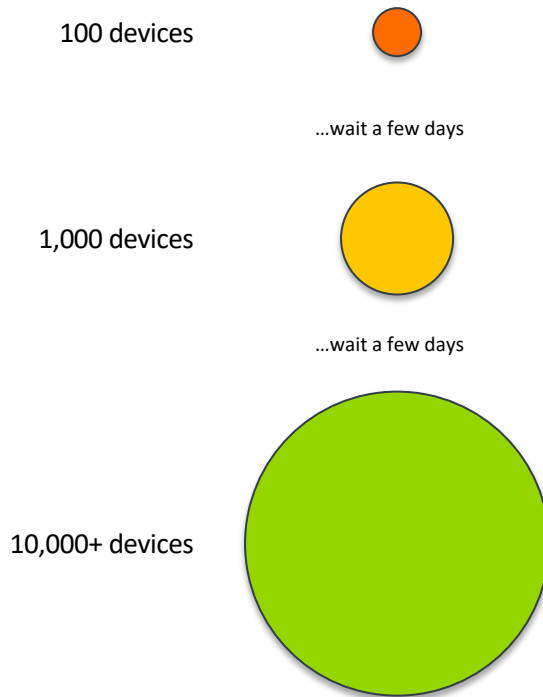
+

Your device just
reset. Please submit
a support ticket!

Unfortunately, some bugs will only be caught in production.

Staged Rollouts

- Send out firmware updates in a controlled manner
 - Send it out to 5%, then 10%, then 20%, etc.
 - At each step, assess quality and performance
 - New issues? – pull the firmware update!
- How to implement staged rollouts?
 - Hosted firmware update solutions have this feature
 - At the very least, can set up devices to poll once every 24 hours (random interval) for new firmware update



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

Metrics

```
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);  
}
```

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

Metrics

Per Device

Counts
Averages
Min/Max



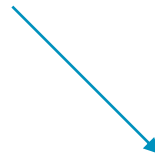
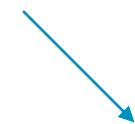
0110...



0001...

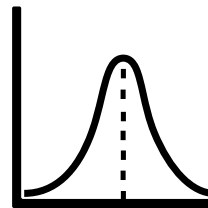


1110...



Entire Fleet

Counts
Averages
Min/Max



7.2 days battery life

1.7 reboots/day

87% connected

39

Metrics

We can compare metrics between firmware versions

	v1.0.0	v1.1.0	Improvement?
CPU sleep time	83%	96%	
Average Uptime	1.3 days	1.32 days	-
Stack bytes free – low water mark	150 bytes	72 bytes	

Agenda

1

State of the union

2

Quantifying current issues

3

Analyzing the issues

4

Preventing future issues

5

Best Practices



~~Best Practices~~ My Favorite Practices

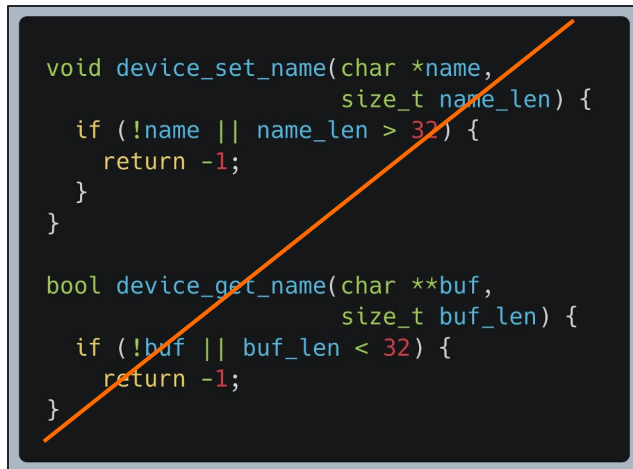
Sharing my tips & tricks monitoring millions of devices
with proper tooling in place

Life With Debugging Infrastructure

If you have logs & coredumps being sent from devices, you can employ what I call **offensive programming**.

This is the proper way to debug “in production”

- During runtime, **raise errors immediately!**
 - Bring bugs front and center
 - Undefined behavior – reset to a known state
 - Assume that if bugs are raised early and often, they can be fixed before shipping
 - With a handful of devices, the 1 in 10,000 hour bugs can be caught



```
void device_set_name(char *name,
                    size_t name_len) {
    if (!name || name_len > 32) {
        return -1;
    }
}

bool device_get_name(char **buf,
                   size_t buf_len) {
    if (!buf || buf_len < 32) {
        return -1;
    }
}
```

BAD BAD BAD BAD

<https://interrupt.memfault.com/blog/defensive-and-offensive-programming>

Use ASSERT() Liberally

```
void device_set_name(char *name,  
                    size_t name_len) {  
    ASSERT(name && name_len <= 32);  
    ...  
}  
  
bool device_get_name(char **buf,  
                    size_t buf_len) {  
    ASSERT(buf && buf_len >= 32);  
    ...  
}
```

Argument validation

Use ASSERT() Liberally

```
void *malloc_assert(size_t n) {  
    void *p = malloc(n)  
    ASSERT(p);  
    return p;  
    ...  
}
```

Assert on heap exhaustion

Use ASSERT() Liberally

```
void timing_sensitive_task(void) {  
    const bool success =  
        mutex_lock(&s_mutex, 1000 /* 1 second */);  
    ASSERT(success);  
    {  
        ...  
    }  
}
```

Easily capture and diagnose timing issues

Use ASSERT() Liberally

```
void on_commit(eState prev_state) {  
    ASSERT(prev_state == kState_Flushing  
           || prev_state == kState_Idle);  
}
```

State machine transition violations

Erase after free

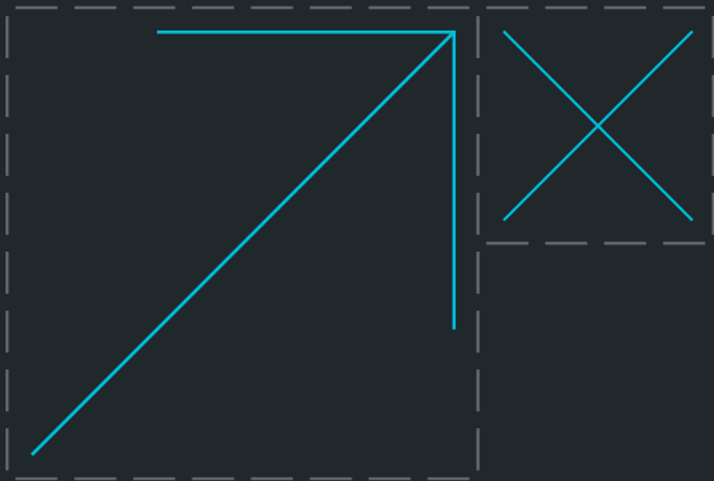
```
void free_and_erase(void *ptr) {  
    HeapInfo *info = heap_get_info(ptr);  
    memset(info->ptr, '0', info->len);  
    free(ptr);  
}
```

Zero out allocations after free

*Now standard in iOS 16.1 and macOS Ventura***

The End

- Happy to connect!
- <https://www.linkedin.com/in/tyhoff/>
- <https://memfault.com/webinars/>
- <https://interrupt.memfault.com>
- Email: tyler@memfault.com



Thank You

Danke

Gracias

Grazie

谢谢

ありがとう

Asante

Merci

감사합니다

धन्यवाद

Kiitos

شكراً

ধন্যবাদ

תודה

The Arm trademarks featured in this presentation are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.

www.arm.com/company/policies/trademarks

