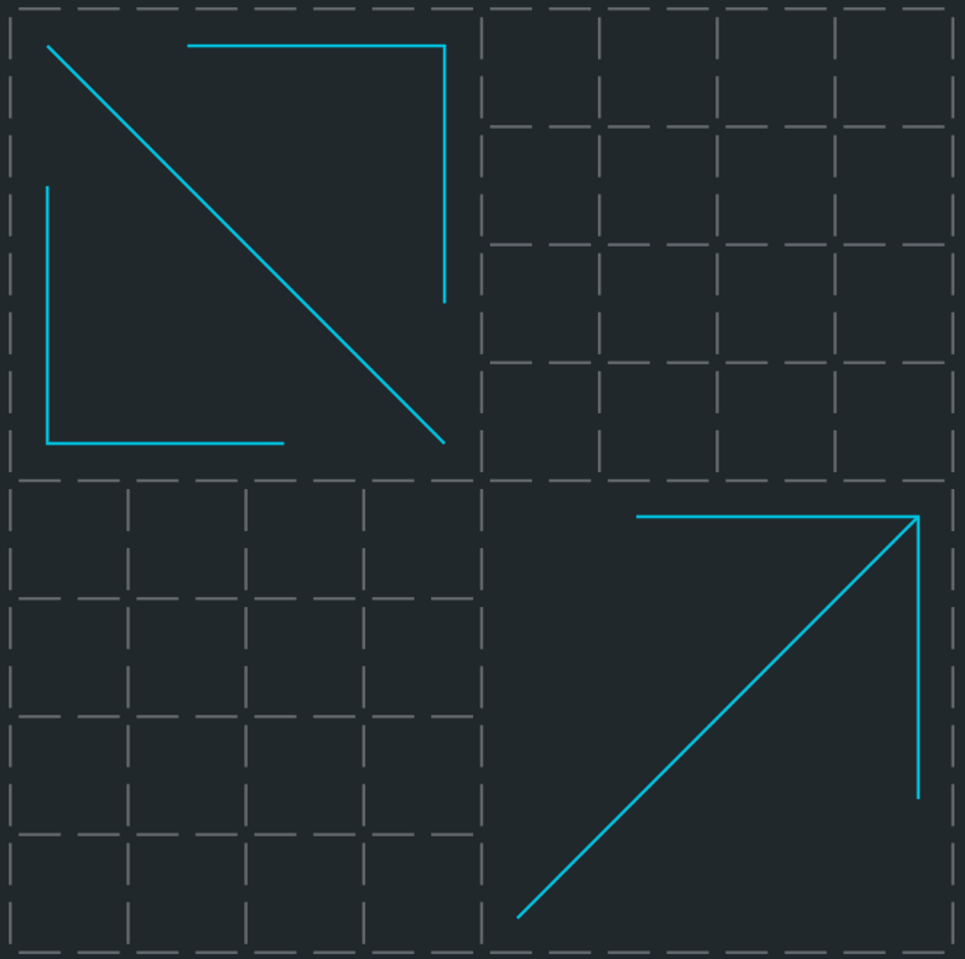
arm
DEVSUMMIT

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

2

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

3

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

4

Agenda

1. **State of the union**
2. Quantifying current issues
3. Analyzing the issues
4. Preventing future issues
5. Best Practices

5

State of the union for firmware

6

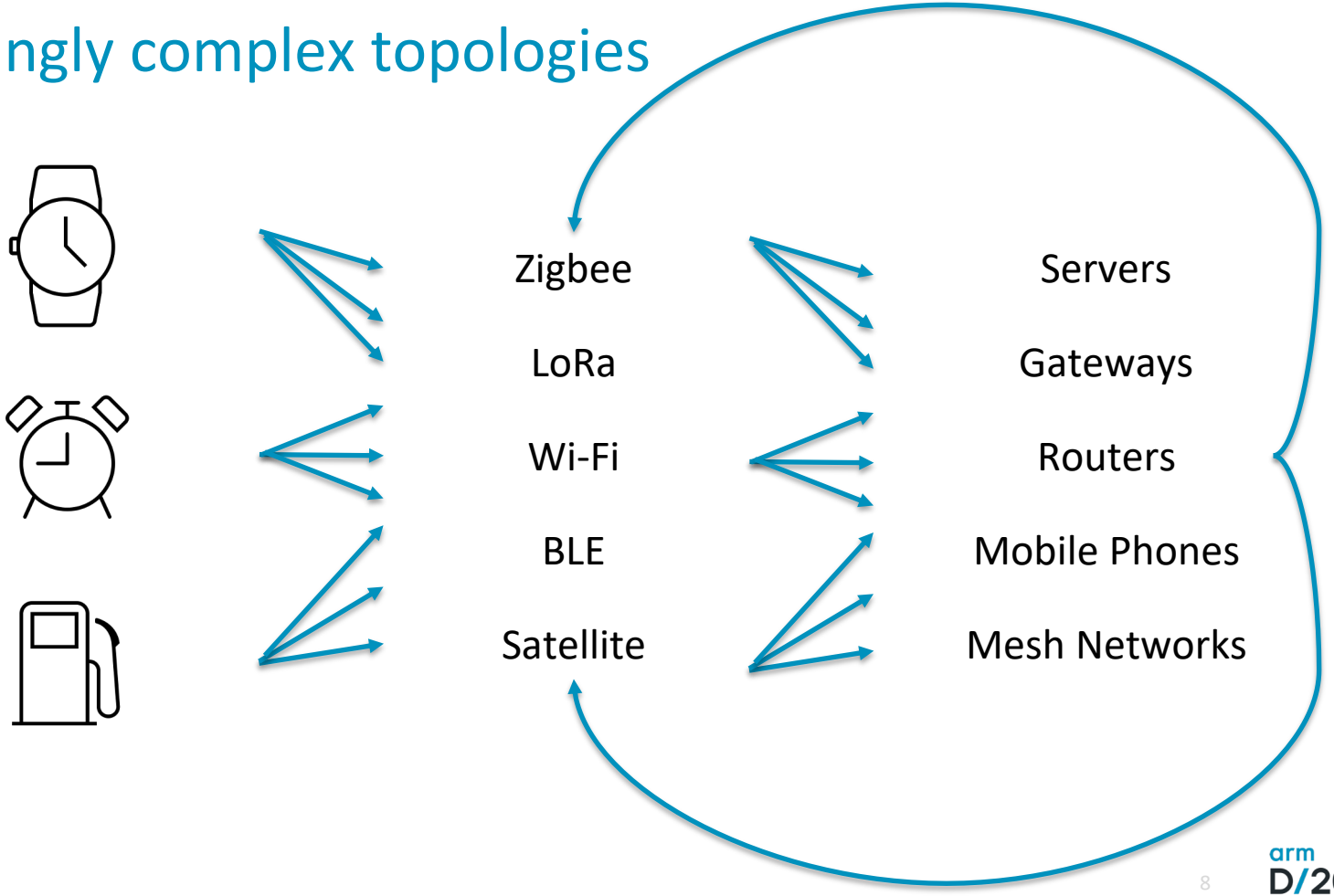
Firmware is *pervasive*, but difficult

- 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

7

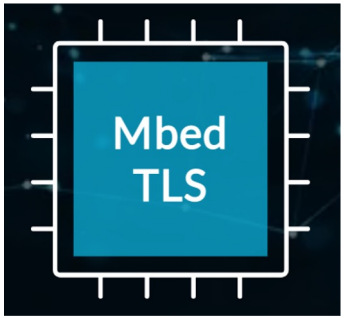
Increasingly complex topologies



8

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



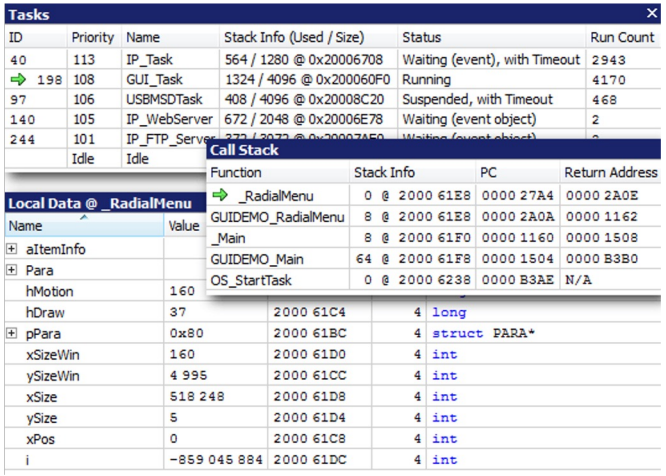
wolfSSL

9

Rudimentary Debugging Tools

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

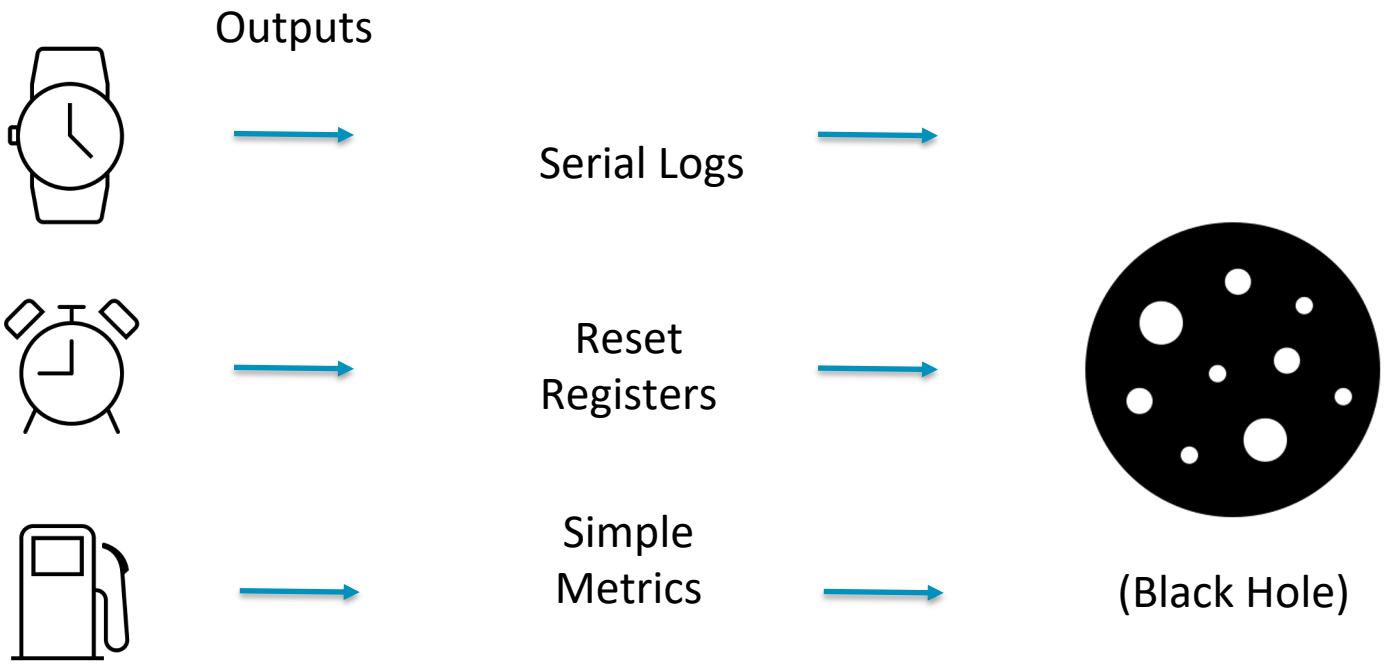
```
[I][1605794400] Wi-Fi connected
[W][1605796200] Wi-Fi disconnected, reason: -2
[I][1605796500] Wi-Fi connected
[W][1605797100] Wi-Fi disconnected, reason: -2
[I][1605798000] Battery status: 67%, 3574 mV
```



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

10

Debugging Data Exported to ???



11

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

12

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

13

Agenda

1. State of the union
2. **Quantifying current issues**
3. Analyzing the issues
4. Preventing future issues
5. Best Practices

14

Detecting Current Issues

Do we have issues in the field…

They’re likely having issues

15

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

16

Detecting current issues

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

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


17

Detecting current issues

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


18

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

19

Detecting current issues

Aggregate!

Reset

```
{
    “reason”: “watchdog”,
}
```

Watchdogs: 67
Hardfaults: 23
Shutdowns: 17

```
“uptime”: 86345
```

Number devices: 890
Average Uptime: 90,234 seconds

Hourly Heartbeat

```
“ble_connected_s”: 3212
```

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

```
“main_stack_min”: 16
```

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

20

Agenda

1. State of the union
2. Quantifying current issues
3. **Analyzing the issues**
4. Preventing future issues
5. Best Practices

21

Analyzing the Issues

We have issues. How do we fix them?

22

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**

23

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.
```

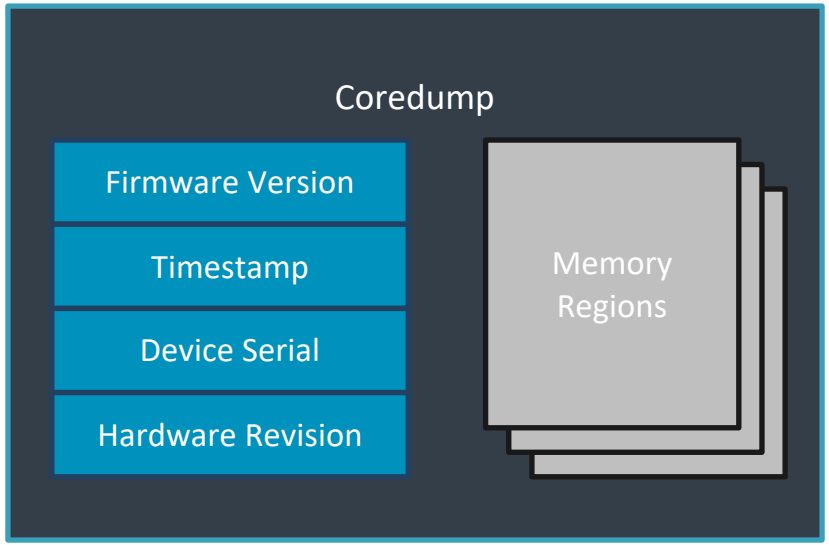
1 crash due to MPU violation

Program Counter:
0xe0000000

24

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

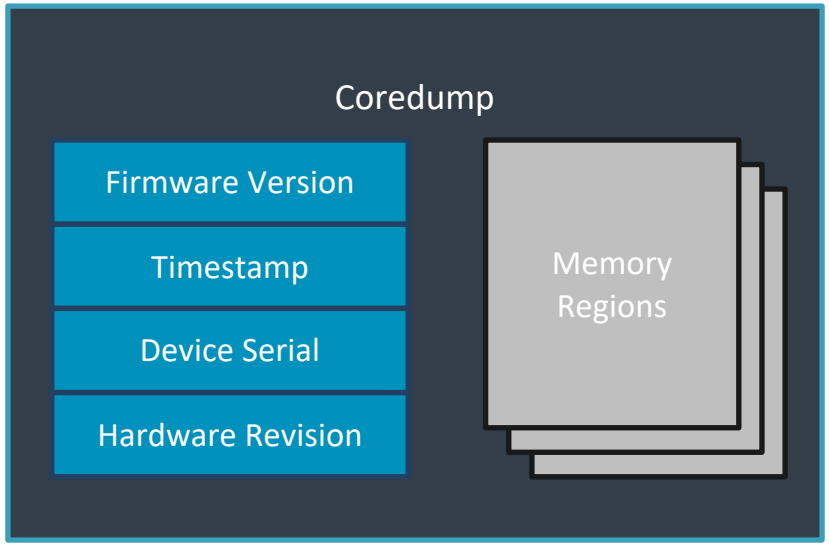


25

Using Coredumps to Analyze Issues

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

   // Don’t just crash…
   capture_coredump();
}
```

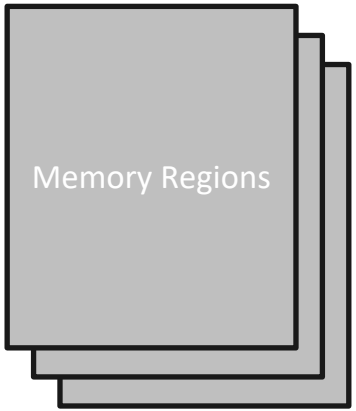


26

Using Coredumps to Analyze Issues

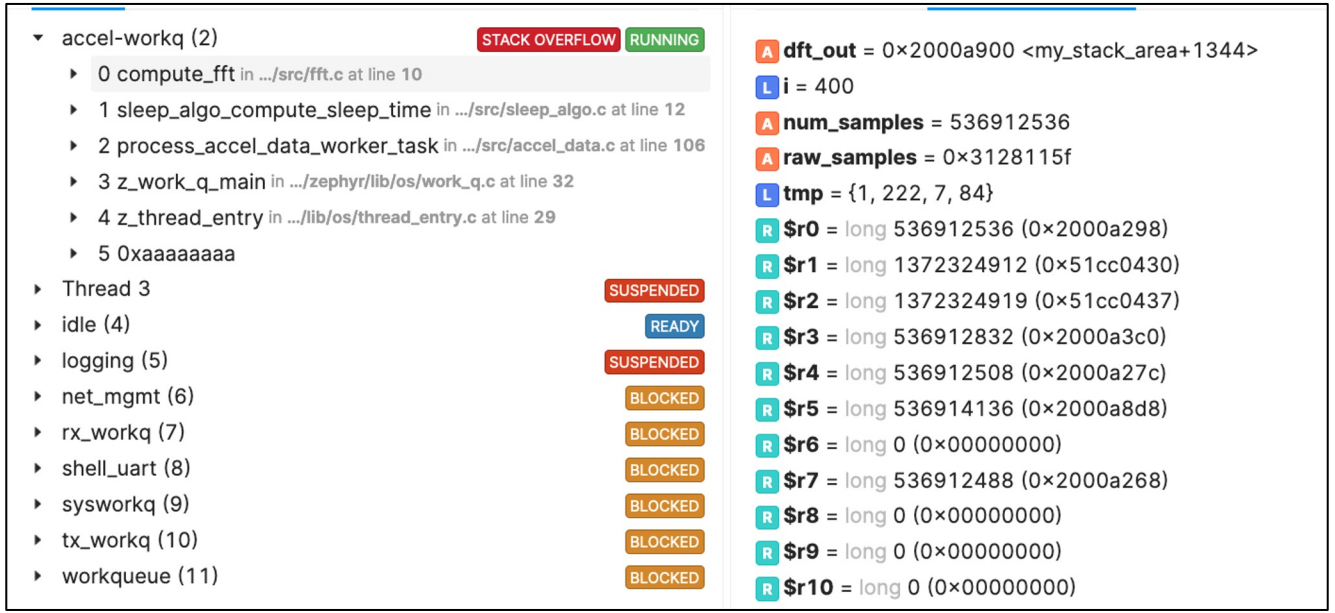
A coredump is comprised of raw memory regions

- 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)



27

Using Coredumps to Analyze Issues



Threads, backtraces, local variables, and registers

28

Using Coredumps to Analyze Issues

Analysis

Memory management fault detected at 0×2000a3c0

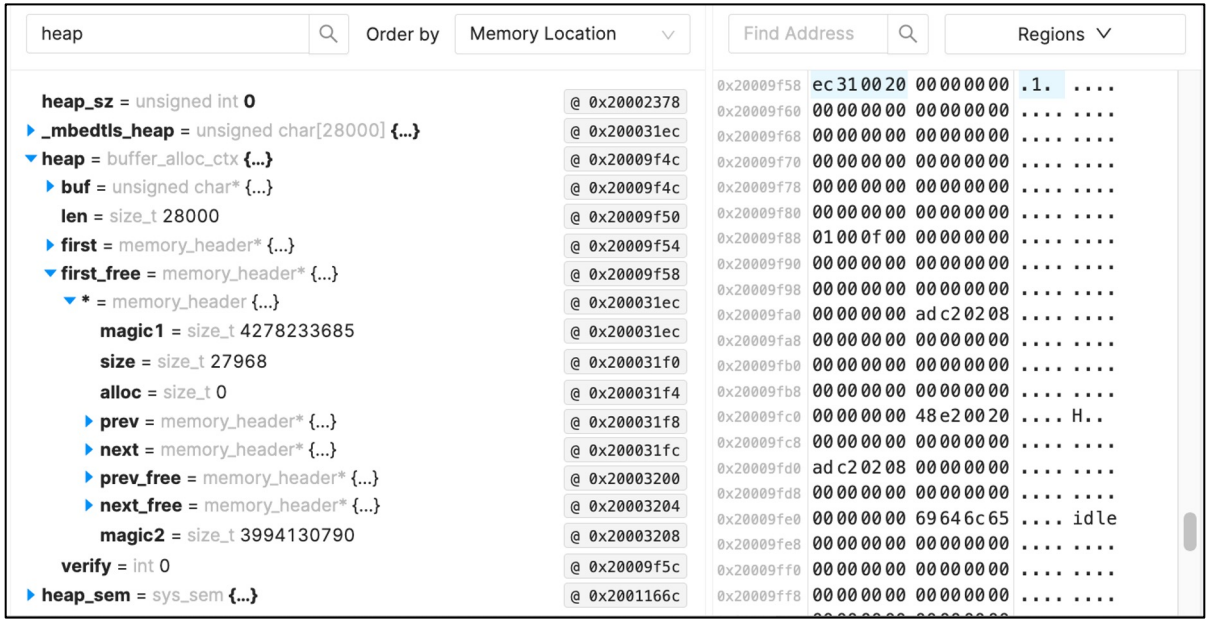
Memory management fault on a data access

Fault Register	Value	Hex Value
CFSR	130	0×00000082
HFSR	0	0×00000000
SHCSR	458753	0×00070001

Parse and interpret system registers

29

Using Coredumps to Analyze Issues



View all global variables at time of coredump

30

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” 😉)

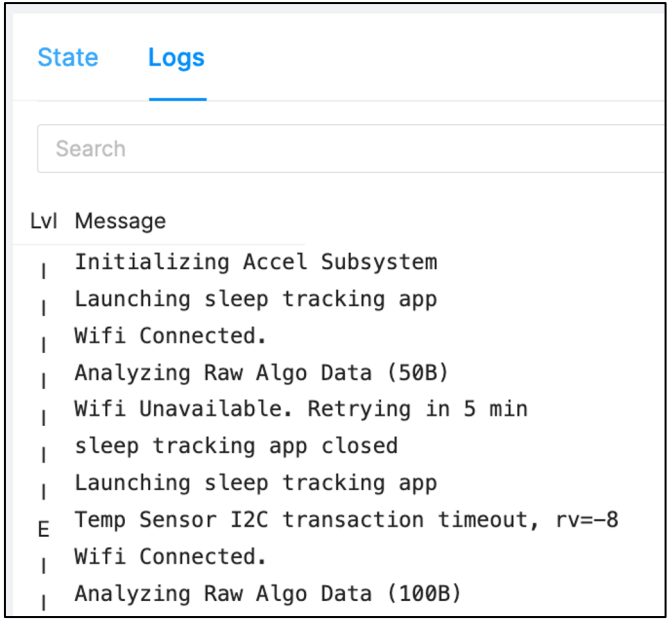
```
static void prv_handle_new_state(eWifiState state) {
    switch (state) {
        case kWifiState_Off:
            do_something();
            break;
        case kWifiState_Connecting:
            do_something();
            break;
        case kWifiState_Connected:
            do_something();
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

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

31

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!



32

Agenda

1. State of the union
2. Quantifying current issues
3. Analyzing the issues
4. **Preventing future issues**
5. Best Practices

33

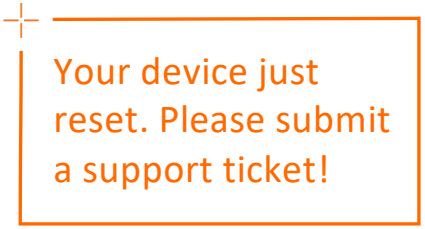
Preventing Issues

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

34

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!

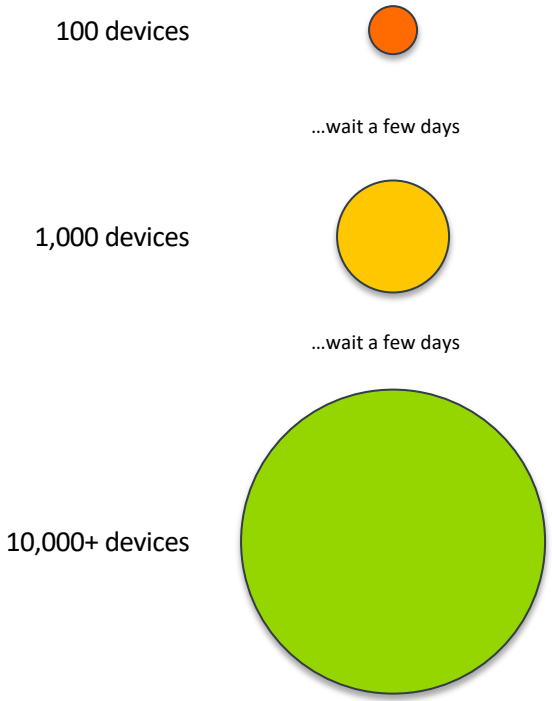


Unfortunately, some bugs will only be caught in production.

35

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



36

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

37

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

38

Metrics

Per Device

Counts
Averages
Min/Max

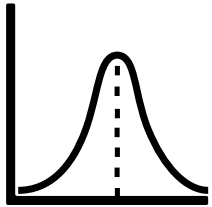
0110...

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	

40

Agenda

1. State of the union
2. Quantifying current issues
3. Analyzing the issues
4. Preventing future issues
5. **Best Practices**

41

~~Best Practices~~
My Favorite Practices

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

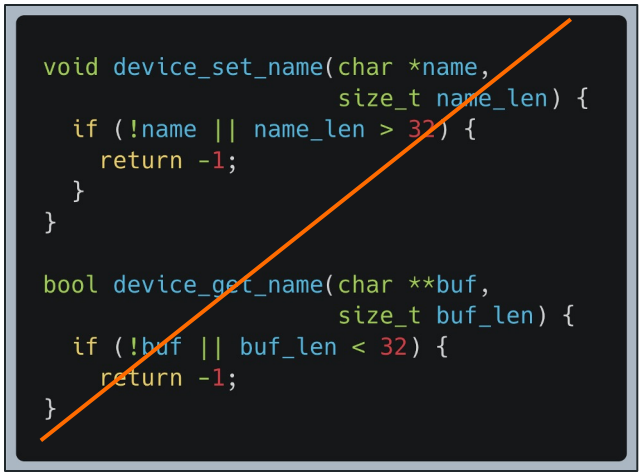
42

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



BAD BAD BAD BAD

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

43

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

44

Use ASSERT() Liberally

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

Assert on heap exhaustion

45

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

46

Use ASSERT() Liberally

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

State machine transition violations

47

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***

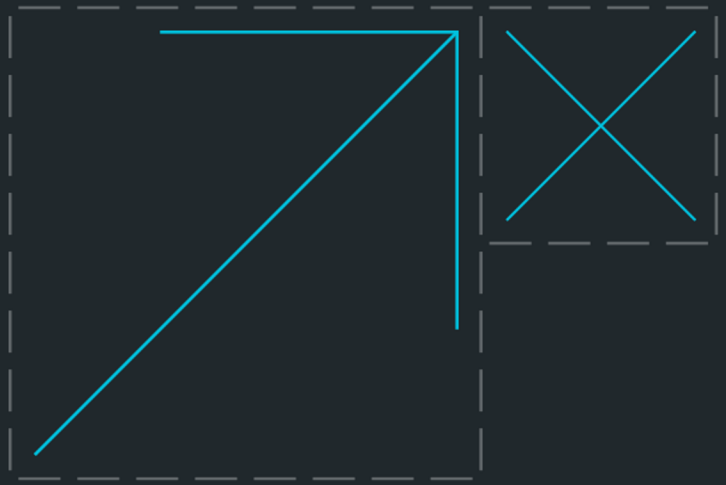
48

The End

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

49

arm
DEVSUMMIT



Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה

arm
DEVSUMMIT