

You Should Probably Just Write A Wireshark Dissector

(...And Then, We Should All Dream Bigger)

Joshua Wise

April 2026

MTVRE

@joshua@social.emarhaviil.com

Accelerated **Tech**

Thesis:

- I spend a lot of time sifting through log files of structured communications
- Writing bespoke analyzers is a bad use of time!

Thesis:



Mike Primavera

@primawesome



My neighbor told me `bespoke analysis tools` keep parsing his `event traces` so I asked how many `tools` he has and he said he just goes to the `text editor` and gets a new `script` afterwards so I said it sounds like he's just feeding `billable hours` to `grep and sed` and then his `customer` started crying.

7:03 AM · Sep 30, 2019

Thesis:

- I spend a lot of time sifting through log files of structured communications
- Writing bespoke analyzers is a bad use of time!
- Therefore, if you have to write analyzer / parsers more than a few times, consider using a better tool for the job

Thesis:

- I spend a lot of time sifting through log files of structured communications
- Writing bespoke analyzers is a bad use of time!
- Therefore, if you have to write analyzer / parsers more than a few times, consider using a better tool for the job
 - Like Wireshark

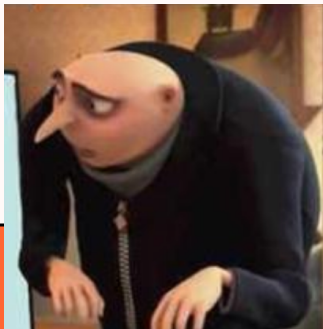
Thesis:

- I spend a lot of time sifting through log files of structured communications
- Writing bespoke analyzers is a bad use of time!
- Therefore, if you have to write analyzer / parsers more than a few times, consider using a better tool for the job
 - Like Wireshark



Thesis:

- I spend a lot of time sifting through log files of structured communications
- Writing bespoke analyzers is a bad use of time!
- Therefore, if you have to write analyzer / parsers more than a few times, consider using a better tool for the job
 - Like Wireshark

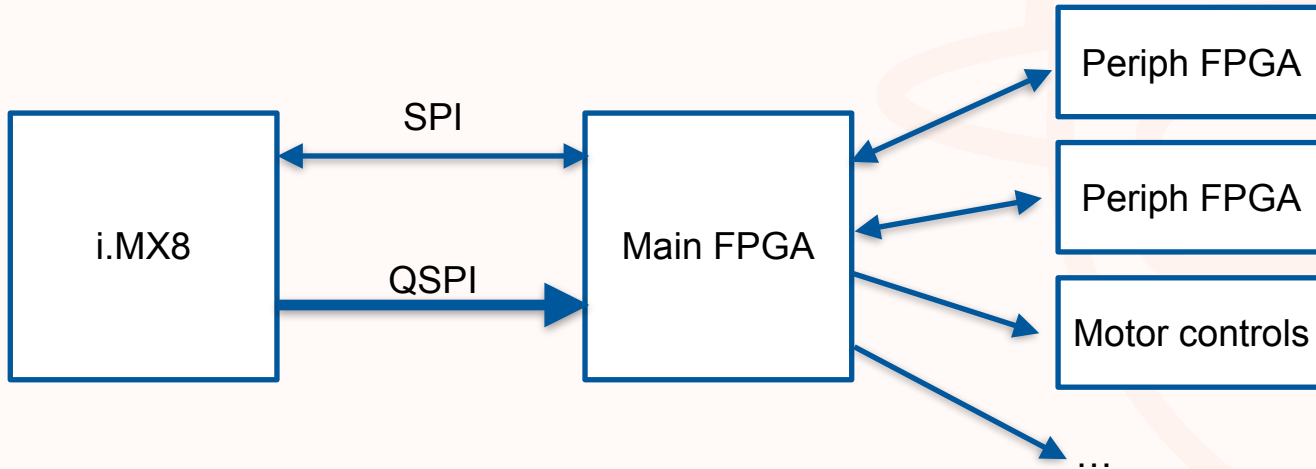


Part 1: You should probably just write a Wireshark dissector

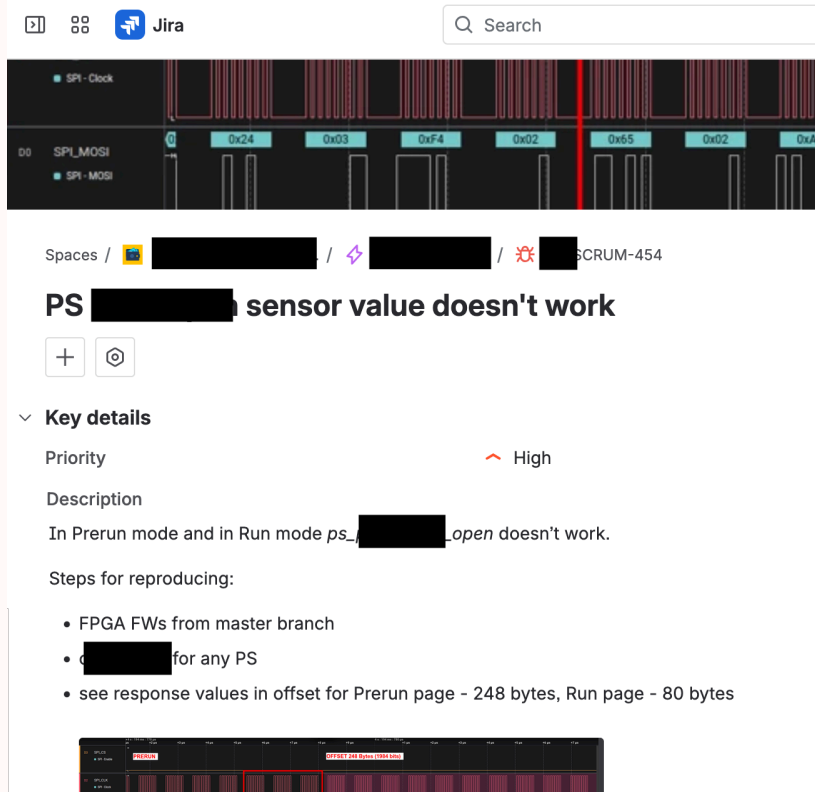
Accelerated Tech

Problem statement

- “SNIAZ” is a connected industrial device with a few operating modes
 - OS runs on Linux SoC, peripherals controlled by an FPGA
- Users submit bug reports to me with logic analyzer traces



Problem statement



The screenshot shows a Jira issue titled "PS sensor value doesn't work". The issue is in the "Key details" section, with a priority of "High". The description states: "In Prerun mode and in Run mode ps_..._open doesn't work." The steps for reproducing the issue are:

- FPGA FWs from master branch
- ... for any PS
- see response values in offset for Prerun page - 248 bytes, Run page - 80 bytes

At the bottom of the screenshot, there is a small image showing a hex dump with a red box highlighting a specific data offset.

Attachments 5

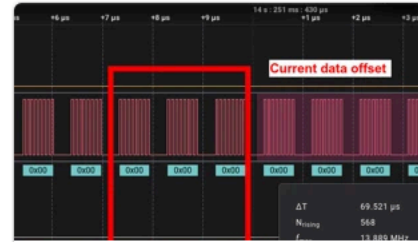



image-202408... 308.png
13 Aug 2024, 01:36 AM



Roller sensor.sal
13 Aug 2024, 01:36 AM

Assignee

 Joshua Wise

Problem statement



Joshua Wise [🔗](#)

August 13, 2024 at 8:33 PM

I decoded the trace that you show there. I see that the trace starts with sequence number 797, for which `ps_open=(0, 0, 0)`; at sequence number 2834, I see that PS2 opens: `ps_open=(0, 1, 0)`

I checked in my decoder to the `utils` directory of the `sniaz-fpga` repository. To run it:

- [some other steps to get into a JSON trace of packets...]
- decode the trace and look for changes in the `ps_open` field:

```
joshua@samskara:~/work/sniaz/jira/454$ python3 ../../sniaz-fpga/utils/decode_spi_transactions_from_json_and_look_for_sensor_changes.py spi_trace.json
```
- or if you want to fully decode every transaction:

```
joshua@samskara:~/work/sniaz/jira/454$ python3 ../../sniaz-fpga/utils/decode_spi_transactions_from_json.py spi_trace.json | less
```

One obvious solution

```
from data_pages.decode_data_page import *

last_open = None

j = json.load(open(sys.argv[1], "r"))
for txn in j:
    if 'mosi' in txn and 'miso' in txn:
        miso = binascii.unhexlify(txn['miso'])
        mosi = binascii.unhexlify(txn['mosi'])
        if len(miso) == 0:
            continue
        mosi_dec = decode_data_page_request(mosi[6:])
        miso_dec = decode_data_page_response(miso[6:])

        if last_open != miso_dec.ps_open:
            print(txn['t'])
            print(f"> {mosi_dec}")
            print(f"< {miso_dec}")
            print("")
            last_open = miso_dec.ps_open
```

Refining it

- OK, but this produces a wall of text — the contents of every transaction that matches
- Hard to get what you want out of each transaction, visually

```
if type(mosi_pkt) == PreprintModePageToFpga:
    #mosi_txt = f" cutter_seek_home {mosi_pkt.cutter_seek_home}"
    mosi_pkt.sequence_number = 0
    mosi_pkt.crc16 = 0
    mosi_txt = f" {str(mosi_pkt)}"
if type(miso_pkt) == PreprintModePageFromFpga:
    #miso_txt = f" git commit {miso_pkt.main_git_commit:x}, {miso_pkt.main_version_major}.{miso_pkt.main_ver
    miso_pkt.sequence_number = 0
    miso_pkt.crc16 = 0
    miso_txt = f" {str(miso_pkt)}"
if type(mosi_pkt) == RunModePageToFpga or type(mosi_pkt) == IdleModePageToFpga:
    mosi_txt = f" cutter_enabled {mosi_pkt.cutter_enabled}, next_cut_line {mosi_pkt.next_cut_line}, " +
        f"lines_to_append {mosi_pkt.feed_number_of_print_lines_to_append}, " +
        f"target_step_value {mosi_pkt.feed_target_step_value}"
if type(miso_pkt) == RunModePageFromFpga or type(miso_pkt) == IdleModePageFromFpga:
    miso_txt = f" cutter_ready {miso_pkt.cutter_ready}, cut_is_done {miso_pkt.cut_is_done}, " +
        f"last_cut {miso_pkt.last_cut_line}, current_line_number {miso_pkt.current_line_number}, " +
        f"remaining_lines {miso_pkt.remaining_lines}, " +
        f"current_step_value {miso_pkt.feed_current_step_value}"
```

Refining it

- OK, but how do you know which duplicates matter, how many duplicates there are, ...?

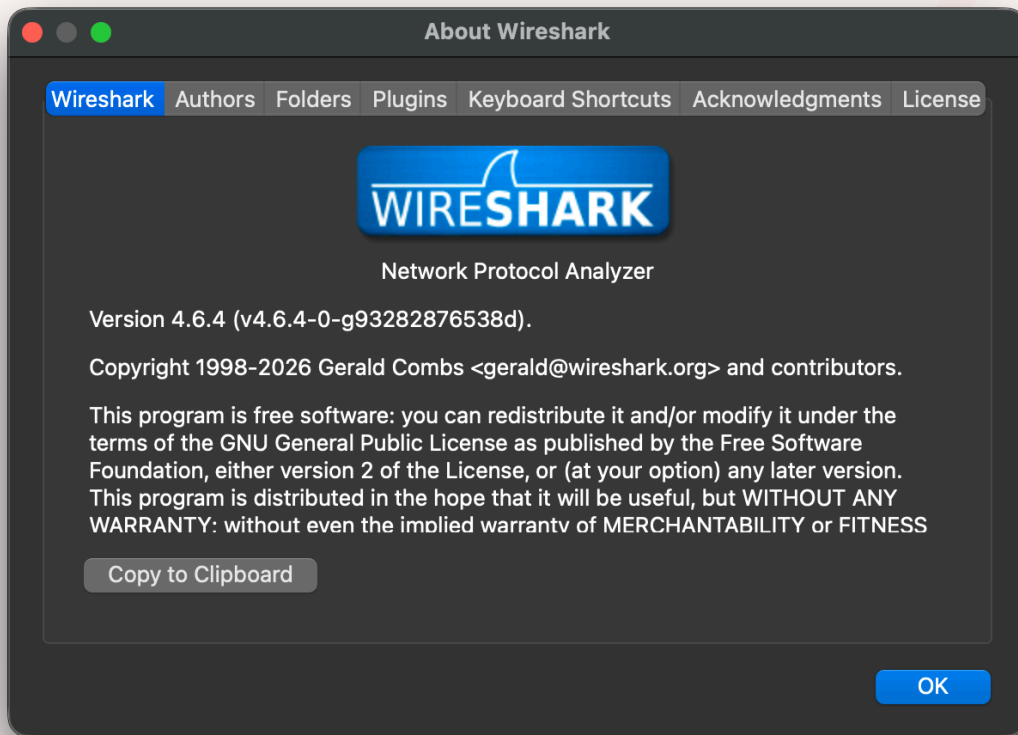
```
if mosi_txt == last_mosi_txt and miso_txt == last_miso_txt:
    n_dups += 1
else:
    if n_dups != 0:
        print(f" [ {n_dups} duplicates ]")
        n_dups = 0
    print(f"{now * 1e-9:0.5f} {type(mosi_pkt).__name__} {type(miso_pkt).__name__}")
    print(mosi_txt)
    print(miso_txt)

last_mosi_txt, last_miso_txt = mosi_txt, miso_txt
```

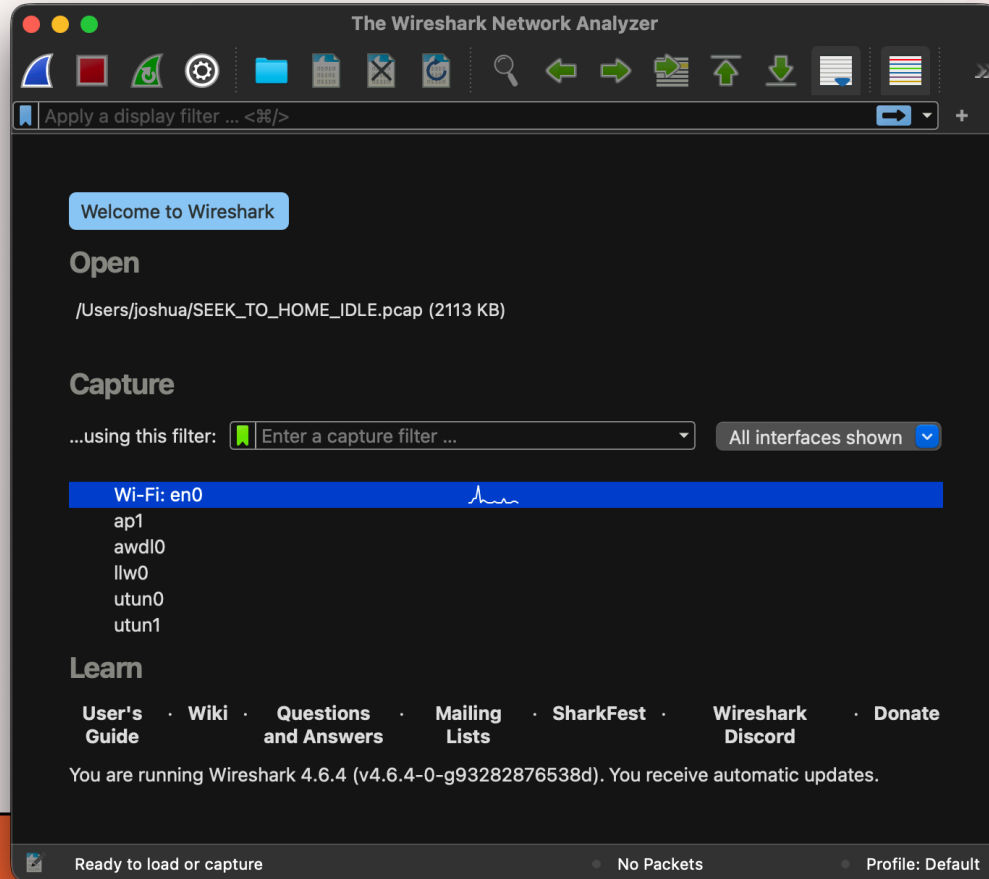
Beyond refinement

```
joshua@samskara:~/work/sniaz$ ls */*.py
main-fpga/saleae_to_vcd.py
'Logic Capture/print_spi_transactions_fan.py'
'Logic Capture/print_spi_transactions_ps.py'
'Logic Capture/print_spi_transactions.py'
'Logic Capture/saleae_to_vcd.py'
jira-1100/print spi transactions cutter.py
jira-1195/print_spi_transactions_cutter.py
jira-1195/print_spi_transactions_from_trace.py
jira-1195/vcd_to_trace.py
jira-500/print_spi_transactions_fan.py
jira-760/print_spi_transactions_fan.py
jira-823/print spi transactions fan.py
```

Isn't there a tool for packet-structured data?



Wait, but I thought...?



PCAP files are for everyone

<https://www.ietf.org/archive/id/draft-gharris-opsawg-pcap-01.html>

4. File Header

The File Header has the following format:

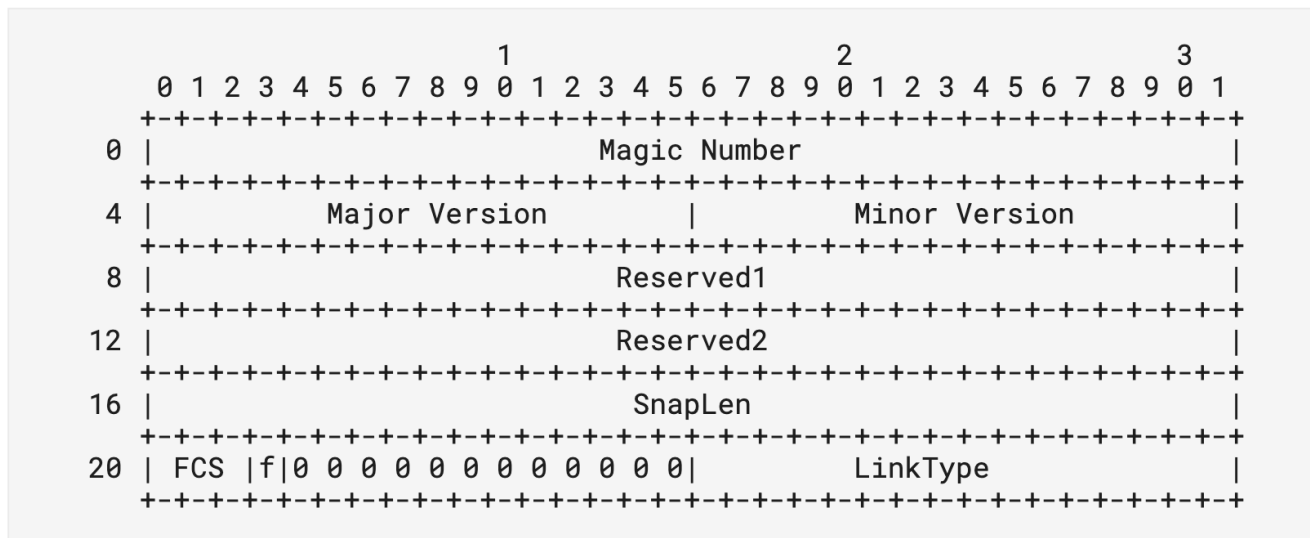


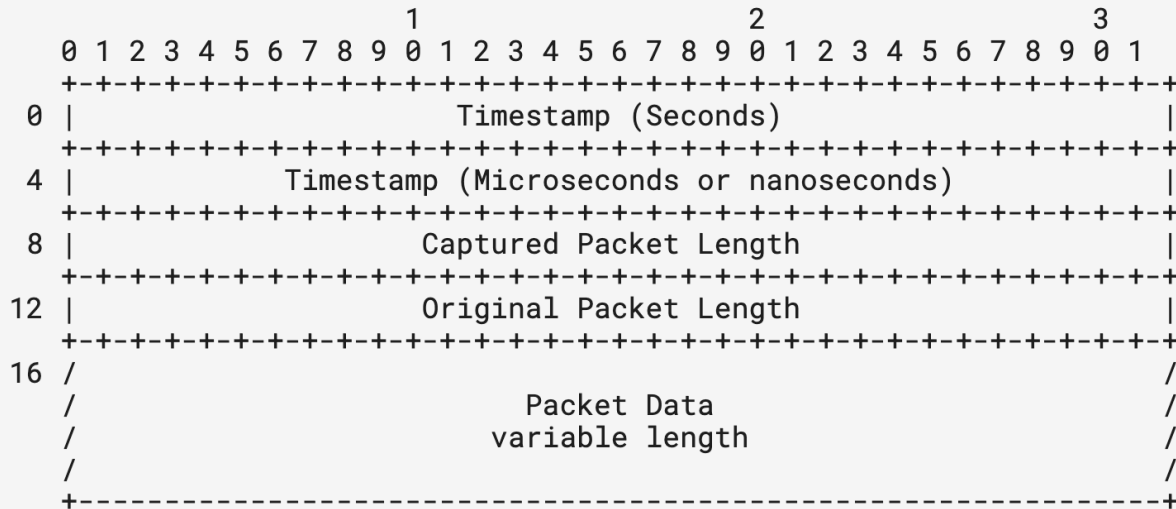
Figure 1: File Header

PCAP files are for everyone

<https://www.ietf.org/archive/id/draft-gharris-opsawg-pcap-01.html>

5. Packet Record

A Packet Record is the standard container for storing the packets coming from the network.



So easy, even an electrical engineer can do it

- OK, you probably want to write a “Wireshark upper PDU” packet rather than registering your own pcap datatype with the IETF

```
def write_header(f):
    # LinkType = wireshark upper PDU
    f.write(struct.pack("<LHHLLLL", 0xA1B2C3D4, 2, 4, 0, 0, 4096, 252))

def write_event(f, timestamp, event, dissector_name='sniaztrace'):
    sharkhdr = struct.pack('>HH', 12, len(dissector_name)) + \
                dissector_name.encode() # DISSECTOR_NAME
    sharkhdr += struct.pack('>HH', 0, 0) # END_OF_OPT

    buf = sharkhdr + event

    pcaphdr = struct.pack("<LLLL",
                          int(timestamp), int((timestamp % 1) * 1e6),
                          len(buf), len(buf))

    f.write(pcaphdr + buf)
```

So easy, even an electrical engineer can do it

SEEK_TO_HOME_IDLE-SNIAZ.pcap

Apply a display filter ... <⌘/>

No.	Time	Protocol	Length	Info
1	0.000000	Exported PDU	74	
2	0.001538	Exported PDU	850	
3	0.005537	Exported PDU	850	
4	0.009541	Exported PDU	850	
5	0.013543	Exported PDU	850	
6	0.017545	Exported PDU	850	
7	0.021596	Exported PDU	850	
8	0.025534	Exported PDU	850	
9	0.029539	Exported PDU	850	
10	0.033536	Exported PDU	850	
11	0.037534	Exported PDU	850	
12	0.041534	Exported PDU	850	

> Frame 5: Packet, 850 bytes on wire (6800 bits), 850 bytes captured (6800 bits) on interface 0

- EXPORTED_PDU
 - Tag: PDU content dissector name (12)
 - Length: 10
 - Protocol Name: sniaztrace
 - Tag: End-of-options (0)
 - Exported PDU data**

```
0010 00 00 73 70 69 5f 69 6d 78 38 5f 6d 66 70 67 61  ..spi_im x8_mfpga
0020 00 00 6d 6f 73 69 00 00 00 00 00 00 00 00 00 00  ..mosi..
0030 00 00 84 01 00 00 00 02 00 00 7c 01 46 9f 10 3a  ..|·F·
0040 06 00 2c 00 01 00 ff ff 00 00 00 00 00 00 00 00  ..,.....
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..
00a0 00 00 00 00 00 00 00 00 00 00 01 01 00 10 0e 00  ..
00b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 40  ..
00c0 26 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..&
00d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..
00e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..
```

Exported PDU data (exported_pdu.exported_pdu), 832 bytes

Packets: 11008

Profile: Default

You Should Write A Wireshark Dissector

The screenshot shows a Wireshark window titled "SEEK_TO_HOME_IDLE-SNIAZ.pcap". The main pane displays a list of packets with the following columns: No., Time, Protocol, Length, and Info. The packets are filtered to show only "sniaztrace.spi_imx8_mfpga" events.

No.	Time	Protocol	Length	Info
10200	4.674929	CODEPAGE	850	[MOSI: run_mode_page #0x3e6c] [MISO: run_mode_page #0x3e6b]
10227	4.679315	CODEPAGE	850	[MOSI: run_mode_page #0x3e6d] [MISO: run_mode_page #0x3e6c]
10253	4.682980	CODEPAGE	850	[MOSI: run_mode_page #0x3e6e] [MISO: run_mode_page #0x3e6d]
10279	4.686980	CODEPAGE	850	[MOSI: run_mode_page #0x3e6f] [MISO: run_mode_page #0x3e6e]
10352	4.698716	CODEPAGE	850	[MOSI: run_mode_page #0x3e70] [MISO: run_mode_page #0x3e6f]
10370	4.701937	CODEPAGE	850	[MOSI: idle_mode_page #0x3e71] [MISO: run_mode_page #0x3e70]
10391	4.705930	CODEPAGE	850	[MOSI: idle_mode_page #0x3e72] [MISO: idle_mode_page #0x3e71]
10408	4.709922	CODEPAGE	850	[MOSI: idle_mode_page #0x3e73] [MISO: idle_mode_page #0x3e72]
10423	4.713920	CODEPAGE	850	[MOSI: idle_mode_page #0x3e74] [MISO: idle_mode_page #0x3e73]
10434	4.717920	CODEPAGE	850	[MOSI: idle_mode_page #0x3e75] [MISO: idle_mode_page #0x3e74]
10442	4.721487	CODEPAGE	850	[MOSI: idle_mode_page #0x3e76] [MISO: idle_mode_page #0x3e75]

The packet details pane shows the following structure for the selected packet (No. 10442):

- SNIAZ trace event
 - Event type: spi_imx8_mfpga
 - i.MX8 <-> MFPGA: Codepage
 - Opcode: Codepage (0x0200)
 - Parameter: 0x0000
 - Length: 380
 - MOSI: run_mode_page (seq 3e70)
 - CRC16: 0xf008
 - Sequence number: 0x3e70
 - Data page number: run_mode_page (0x0001)
 - Data page version: 0x002c
 - rotary_cutter_enabled: True

The packet bytes pane shows the raw data for the selected packet, with the first 110 bytes displayed. The data is shown in hexadecimal and ASCII format.

0040 01 00 2c 00 01 00 d9 02 00 00 00 00 32 38 00 03
0050 02 00 13 2a 00 01 03 00 40 1f 00 00 00 00 00 37
0060 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00a0 00 ba 00 00 01 f3 00 00 00 00 01 01 00 10 0e 00
00b0 ff ff ff ff 00 00 00 00 00 00 00 00 e8 03 00 40
00c0 26 00 01 00 5d 00 60 00 46 00 6a 00 24 00 08 00
00d0 01 00 e8 03 e1 1f 00 00 00 00 00 00 00 00 00 00
00e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

At the bottom of the window, the status bar shows: "Data page number (sniaztrace.mfpga.codepage.mosi.data_page_number), 2 bytes", "Packets: 11008 · Displayed: 1687 (15.3%)", and "Profile: Default".

exercise for the reader

- OK writing a pcap file is easy, but your first dissector is not
- Dissectors are Lua scripts
 - That are not really written in very idiomatic Lua
 - OK, also C
- The rest of this talk does NOT discuss how to do it
 - Instead: why to do it, things to consider, ...
 - But I provide a sample
- There is a lot of documentation on it
 - Unfortunately it is not well-organized
- My best suggestion: do it once, prepare to cut and paste

SCRUM-1541: "The cutter did not complete at idle"

- Filter *quickly* to find exactly what time it croaked at

The screenshot shows a network traffic analysis tool window titled "SEEK_TO_HOME_IDLE-SNIAZ.pcap". The filter bar contains the expression "sniaztrace.mfpga.codepage.miso.cutter_ready & 0x80 == 0x80". The main display shows a list of filtered packets with the following columns: No., Time, Protocol, Length, and Info.

No.	Time	Protocol	Length	Info
244	0.944020	CODEPAGE	850	[MOSI: idle_mode_page #0x3aff] [MISO: idle_mode_page #0x3afe]
245	0.948026	CODEPAGE	850	[MOSI: idle_mode_page #0x3b00] [MISO: idle_mode_page #0x3aff]
246	0.951519	CODEPAGE	850	[MOSI: idle_mode_page #0x3b01] [MISO: idle_mode_page #0x3b00]
247	0.955518	CODEPAGE	850	[MOSI: idle_mode_page #0x3b02] [MISO: idle_mode_page #0x3b01]
248	0.959173	CODEPAGE	850	[MOSI: idle_mode_page #0x3b03] [MISO: idle_mode_page #0x3b02]
249	0.962623	CODEPAGE	1674	[MOSI: preprint_mode_page #0x3b04] [MISO: idle_mode_page #0x3b03]
10454	4.733420	CODEPAGE	850	[MOSI: idle_mode_page #0x3e79] [MISO: idle_mode_page #0x3e78]
10455	4.737453	CODEPAGE	850	[MOSI: idle_mode_page #0x3e7a] [MISO: idle_mode_page #0x3e79]
10456	4.741170	CODEPAGE	850	[MOSI: idle_mode_page #0x3e7b] [MISO: idle_mode_page #0x3e7a]
10457	4.745169	CODEPAGE	850	[MOSI: idle_mode_page #0x3e7c] [MISO: idle_mode_page #0x3e7b]
10458	4.749227	CODEPAGE	850	[MOSI: idle_mode_page #0x3e7d] [MISO: idle_mode_page #0x3e7c]
10459	4.752916	CODEPAGE	850	[MOSI: idle_mode_page #0x3e7e] [MISO: idle_mode_page #0x3e7d]
10460	4.756917	CODEPAGE	850	[MOSI: idle_mode_page #0x3e7f] [MISO: idle_mode_page #0x3e7e]
10461	4.760919	CODEPAGE	850	[MOSI: idle_mode_page #0x3e80] [MISO: idle_mode_page #0x3e7f]
10462	4.764917	CODEPAGE	850	[MOSI: idle_mode_page #0x3e81] [MISO: idle_mode_page #0x3e80]

The selected packet (No. 10462) is expanded to show the following data:

```
media_encoder_angle: -4085
cutter_ready: 0x83
reserved0: 0x00
ps_unwind_motor_abspos
ps_rewind_motor_abspos
ps_unwind_motor_speed
```

The hex dump for the selected packet is as follows:

```
01e0 0b f0 83 00 7c 9d ff ff c2 71 00 00 00 00 00 00
01f0 fe 7b ff ff 60 50 00 00 00 00 00 00 00 6e ff 51 02
0200 00 00 35 00 e0 01 00 00 00 00 00 00 00 00 f1 fe
0210 00 00 00 00 00 01 18 18 00 00 00 00 00 00 00 00
0220 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0230 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0240 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

SCRUM-1541: "The cutter did not complete at idle"

- Add columns to see what it was doing at the time

The screenshot shows a network traffic analysis tool window titled "SEEK_TO_HOME_IDLE-SNIAZ.pcap". The address bar displays "sniaztrace.mfpga.codepage.miso.crc16". The main pane shows a list of network packets with the following columns: No., Time, Protocol, Length, cutter_ready, cutter_phase_index, and Info. The packets are all CODEPAGE, 850 bytes long, and have a cutter_ready value of 0x03 or 0x83. The Info column shows MOSI and MISO run and idle mode pages.

No.	Time	Protocol	Length	cutter_ready	cutter_phase_index	Info
10253	4.682980	CODEPAGE	850	0x03	0x03	[MOSI: run_mode_page #0x3e6e] [MISO: run_mode_page #0x3e6e]
10279	4.686980	CODEPAGE	850	0x03	0x03	[MOSI: run_mode_page #0x3e6f] [MISO: run_mode_page #0x3e6f]
10352	4.698716	CODEPAGE	850	0x03	0x04	[MOSI: run_mode_page #0x3e70] [MISO: run_mode_page #0x3e70]
10370	4.701937	CODEPAGE	850	0x03	0x05	[MOSI: idle_mode_page #0x3e71] [MISO: run_mode_page #0x3e71]
10391	4.705930	CODEPAGE	850	0x03	0x05	[MOSI: idle_mode_page #0x3e72] [MISO: idle_mode_page #0x3e72]
10408	4.709922	CODEPAGE	850	0x03	0x05	[MOSI: idle_mode_page #0x3e73] [MISO: idle_mode_page #0x3e73]
10423	4.713920	CODEPAGE	850	0x03	0x05	[MOSI: idle_mode_page #0x3e74] [MISO: idle_mode_page #0x3e74]
10434	4.717920	CODEPAGE	850	0x03	0x05	[MOSI: idle_mode_page #0x3e75] [MISO: idle_mode_page #0x3e75]
10442	4.721487	CODEPAGE	850	0x03	0x05	[MOSI: idle_mode_page #0x3e76] [MISO: idle_mode_page #0x3e76]
10448	4.725420	CODEPAGE	850	0x03	0x05	[MOSI: idle_mode_page #0x3e77] [MISO: idle_mode_page #0x3e77]
10453	4.729427	CODEPAGE	850	0x03	0x05	[MOSI: idle_mode_page #0x3e78] [MISO: idle_mode_page #0x3e78]
10454	4.733420	CODEPAGE	850	0x83	0xff	[MOSI: idle_mode_page #0x3e79] [MISO: idle_mode_page #0x3e79]
10455	4.737453	CODEPAGE	850	0x83	0xff	[MOSI: idle_mode_page #0x3e7a] [MISO: idle_mode_page #0x3e7a]
10456	4.741170	CODEPAGE	850	0x83	0xff	[MOSI: idle_mode_page #0x3e7b] [MISO: idle_mode_page #0x3e7b]
10457	4.745169	CODEPAGE	850	0x83	0xff	[MOSI: idle_mode_page #0x3e7c] [MISO: idle_mode_page #0x3e7c]

Below the packet list, there is a hex dump and a metadata section. The metadata section shows:

```
Data page version: 0x002c
remaining_print_lines: 0x00002d9e
media_encoder_angle: -4085
cutter_ready: 0x83
reserved0: 0x00
ps_unwind_motor_abspos
```

The hex dump shows the raw data of the selected packet, with the value 83 highlighted in red in the first byte of the first line.

SCRUM-1541: “The cutter did not complete at idle”

- Found preceding transition to idle — compare parameters

The screenshot shows a network traffic analysis tool window titled "SEEK_TO_HOME_IDLE-SNIAZ.pcap". The address bar displays "sniaztrace.mfpga.codepage.miso.crc16". The main window contains a table of network packets with the following columns: No., Time, Protocol, Length, cutter_ready, cutter_phase_index, and Info. The selected packet (No. 10352) is highlighted in blue. Below the table, a detailed view of the selected packet is shown, including fields like cutter_cmd, cut_trq, cut_free_spin_trq, reserved1, cut_phase_indexes, aux_cutter_enabled, and MISO. The packet data is displayed in hexadecimal and ASCII format.

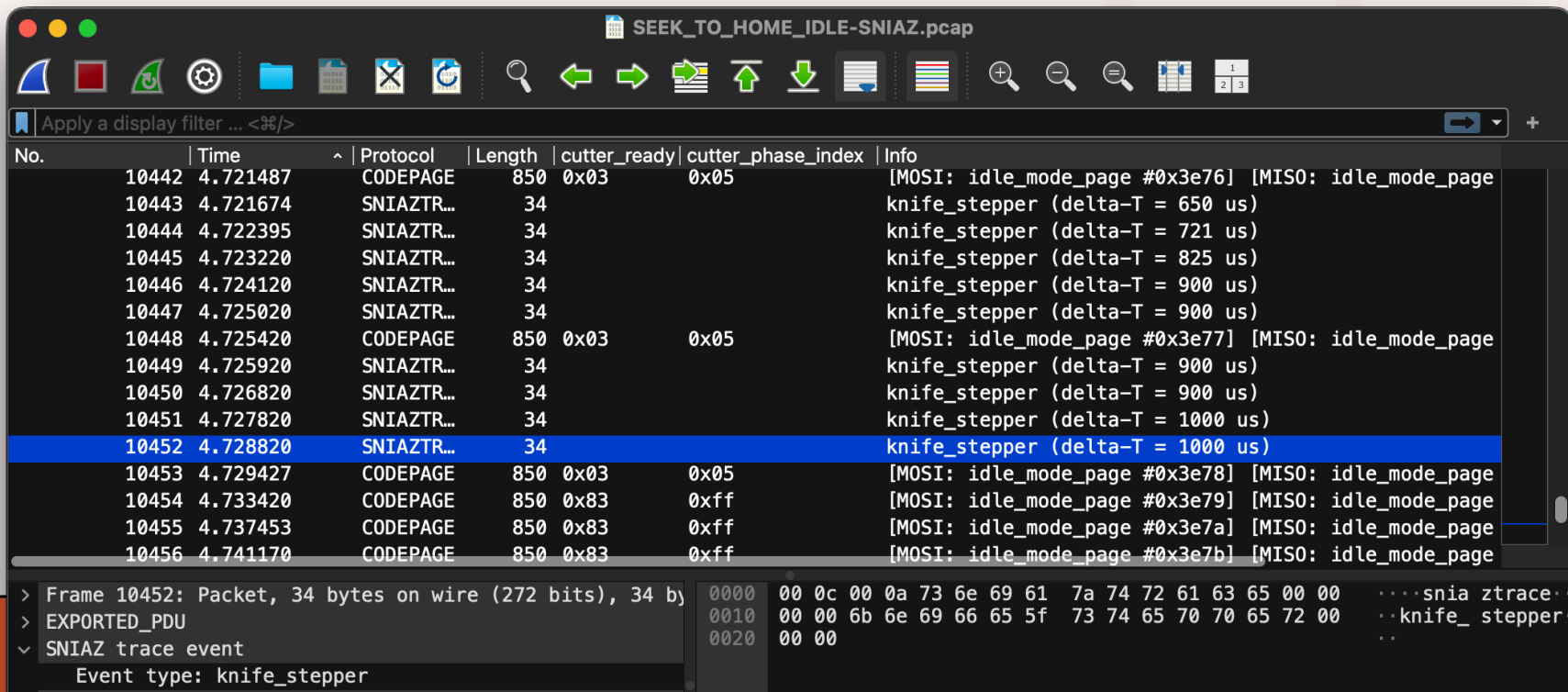
No.	Time	Protocol	Length	cutter_ready	cutter_phase_index	Info
10227	4.679315	CODEPAGE	850	0x03	0x03	[MOSI: run_mode_page #0x3e6d] [MISO: run_mode_page #0x3e6d]
10253	4.682980	CODEPAGE	850	0x03	0x03	[MOSI: run_mode_page #0x3e6e] [MISO: run_mode_page #0x3e6e]
10279	4.686980	CODEPAGE	850	0x03	0x03	[MOSI: run_mode_page #0x3e6f] [MISO: run_mode_page #0x3e6f]
10352	4.698716	CODEPAGE	850	0x03	0x04	[MOSI: run_mode_page #0x3e70] [MISO: run_mode_page #0x3e70]
10370	4.701937	CODEPAGE	850	0x03	0x05	[MOSI: idle_mode_page #0x3e71] [MISO: run_mode_page #0x3e71]
10391	4.705930	CODEPAGE	850	0x03	0x05	[MOSI: idle_mode_page #0x3e72] [MISO: idle_mode_page #0x3e72]
10408	4.709922	CODEPAGE	850	0x03	0x05	[MOSI: idle_mode_page #0x3e73] [MISO: idle_mode_page #0x3e73]
10423	4.713920	CODEPAGE	850	0x03	0x05	[MOSI: idle_mode_page #0x3e74] [MISO: idle_mode_page #0x3e74]
10434	4.717920	CODEPAGE	850	0x03	0x05	[MOSI: idle_mode_page #0x3e75] [MISO: idle_mode_page #0x3e75]
10442	4.721487	CODEPAGE	850	0x03	0x05	[MOSI: idle_mode_page #0x3e76] [MISO: idle_mode_page #0x3e76]
10448	4.725420	CODEPAGE	850	0x03	0x05	[MOSI: idle_mode_page #0x3e77] [MISO: idle_mode_page #0x3e77]
10453	4.729427	CODEPAGE	850	0x03	0x05	[MOSI: idle_mode_page #0x3e78] [MISO: idle_mode_page #0x3e78]
10454	4.733420	CODEPAGE	850	0x83	0xff	[MOSI: idle_mode_page #0x3e79] [MISO: idle_mode_page #0x3e79]
10455	4.737453	CODEPAGE	850	0x83	0xff	[MOSI: idle_mode_page #0x3e7a] [MISO: idle_mode_page #0x3e7a]
10456	4.741170	CODEPAGE	850	0x83	0xff	[MOSI: idle_mode_page #0x3e7b] [MISO: idle_mode_page #0x3e7b]

Packet details for No. 10352:

```
cutter_cmd: 0x00
cut_trq: 0x40
cut_free_spin_trq: 0x26
reserved1: 0x00
cut_phase_indexes: 00c0 26 00 01 00 5d 00 60 00 46 00 6a 00 24 00 08 00
aux_cutter_enabled: True
MISO: run_mode_page #0x3e70
```


You can have heterogeneous events

- Analyze multiple things at once — add other types of events to your pcap (and dissector!) to get better understanding of fault conditions



The screenshot shows the Wireshark interface for a packet capture file named 'SEEK_TO_HOME_IDLE-SNIAZ.pcap'. The main display area shows a list of packets with columns for No., Time, Protocol, Length, cutter_ready, cutter_phase_index, and Info. Packet 10452 is highlighted in blue. Below the list, the packet details pane shows the SNIATR trace event structure, and the packet bytes pane shows the raw data.

No.	Time	Protocol	Length	cutter_ready	cutter_phase_index	Info
10442	4.721487	CODEPAGE	850	0x03	0x05	[MOSI: idle_mode_page #0x3e76] [MISO: idle_mode_page
10443	4.721674	SNIAZTR...	34			knife_stepper (delta-T = 650 us)
10444	4.722395	SNIAZTR...	34			knife_stepper (delta-T = 721 us)
10445	4.723220	SNIAZTR...	34			knife_stepper (delta-T = 825 us)
10446	4.724120	SNIAZTR...	34			knife_stepper (delta-T = 900 us)
10447	4.725020	SNIAZTR...	34			knife_stepper (delta-T = 900 us)
10448	4.725420	CODEPAGE	850	0x03	0x05	[MOSI: idle_mode_page #0x3e77] [MISO: idle_mode_page
10449	4.725920	SNIAZTR...	34			knife_stepper (delta-T = 900 us)
10450	4.726820	SNIAZTR...	34			knife_stepper (delta-T = 900 us)
10451	4.727820	SNIAZTR...	34			knife_stepper (delta-T = 1000 us)
10452	4.728820	SNIAZTR...	34			knife_stepper (delta-T = 1000 us)
10453	4.729427	CODEPAGE	850	0x03	0x05	[MOSI: idle_mode_page #0x3e78] [MISO: idle_mode_page
10454	4.733420	CODEPAGE	850	0x83	0xff	[MOSI: idle_mode_page #0x3e79] [MISO: idle_mode_page
10455	4.737453	CODEPAGE	850	0x83	0xff	[MOSI: idle_mode_page #0x3e7a] [MISO: idle_mode_page
10456	4.741170	CODEPAGE	850	0x83	0xff	[MOSI: idle_mode_page #0x3e7b] [MISO: idle_mode_page

```
> Frame 10452: Packet, 34 bytes on wire (272 bits), 34 by
> EXPORTED_PDU
  0000 00 0c 00 0a 73 6e 69 61 7a 74 72 61 63 65 00 00  ...snia ztrace...
  0010 00 00 6b 6e 69 66 65 5f 73 74 65 70 70 65 72 00  ..knife_stepper...
  0020 00 00
```

Event type: knife_stepper

Very quick debug cycle!

- Previous scripts reparse whole file each time I change anything...
 - Filtering in Wireshark is *fast*
- Being able to link bytes on wire to field indices is very useful to share with SW team
- Set up any fields you want to visualize!
- MSB: **debugging is about exploration; use Wireshark as tool for exploring event-structured data and understanding it better**

The other thing where you want to explore

- PCAP-formatted data is also useful for getting an understanding of *unknown* data on the wire
- Wireshark dissector may become “documentation of record” for proto that you are reverse engineering!
 - Ghidra structure editor only lets you show so much
- Invaluable while visualizing Bambu Lab MC / AP protocol

set-ams-color-with-fix.pcap

not mcproto.publish_dds_cmn_recorder and not mcproto.pack_get_part_info

No.	Time	Source	Destination	Protocol	Length	Info
149	0.554446	AP	AMS	MCPRTO	19	[#9] ams_tray_info_read
→ 3262	16.590119	AP	AMS	MCPRTO	50	[REQ #138] ams_tray_info_write
4090	20.906510	AP	AMS	MCPRTO	50	[REQ #174] ams_tray_info_write
2365	12.026929	AP	AMS	MCPRTO	50	[REQ #99] ams_tray_info_write
45	0.065187	MC	AP	MCPRTO	28	[RSP #0] ping
2450	12.414588	AP	MC	MCPRTO	24	[RSP #103] sync_timestamp
325	1.388415	AMS	AP	MCPRTO	147	[RSP #10] ams_tray_info_read
2635	13.427166	AP	MC	MCPRTO	24	[RSP #110] sync_timestamp
2658	13.466144	MC	AP	MCPRTO	32	[RSP #111] link_ams_tray_consumption_ack2
2660	13.466251	MC	AP	MCPRTO	32	[RSP #112] link_ams_tray_consumption_ack2

> Frame 3262: Packet, 50 bytes on wire (400 bits), 50 bytes captured (400 bits) on interface 0
 DLT: 147, Payload: mcproto (Bambu MC wire protocol)

▼ Bambu bus protocol data

- Capture source: Serial port write from forward (1)
- > Flags: 0x05
- Sequence number: 138
- Response: 3300
- Destination: AMS (7)
- Source: AP (6)
- Command: 0x0218
- ▼ ams_tray_info_write
- Payload: 0101474647393900000046a8f2ffdc000e015045!

```

0000  01 3d 05 8a 00 31 00 21 00 07 00 06 18 02 01 01  .=. . . 1 ! . . . . .
0010  47 46 47 39 39 00 00 00 46 a8 f2 ff dc 00 0e 01  GFG99 . . . F . . . . .
0020  50 45 54 47 00 00 00 00 00 00 00 00 00 00 00 00  PETG . . . . .
0030  99 a0
  
```

Payload (mcproto.ams_tray_info_write.payload), 34 bytes Packets: 7048 · Displayed: 300 (4.3%) Profile: Default

set-ams-color-with-fix.pcap

not mcproto.publish_dds_cmh_recorder and not mcproto.pack_get_part_info

No.	Time	Source	Destination	Protocol	Length	Info
2828	14.415474	AP	MC	MCPROTO	24	[RSP #122] sync_timestamp
3032	15.420518	AP	MC	MCPROTO	24	[RSP #128] sync_timestamp
187	0.707678	22630	AP	MCPROTO	152	[RSP #12] ams_tray_info_read
3220	16.421264	AP	MC	MCPROTO	24	[RSP #136] sync_timestamp
← 3300	16.817612	AMS	AP	MCPROTO	19	[RSP #138] ams_tray_info_write
3406	17.439293	AP	MC	MCPROTO	24	[RSP #143] sync_timestamp
3611	18.432188	AP	MC	MCPROTO	24	[RSP #151] sync_timestamp
3796	19.419452	AP	MC	MCPROTO	24	[RSP #157] sync_timestamp
3955	20.245842	MC	AP	MCPROTO	32	[RSP #162] link_ams_tray_consumption_ack2

> Frame 3300: Packet, 19 bytes on wire (152 bits), 19 bytes captured (152 bits) on interface 0
 DLT: 147, Payload: mcproto (Bambu MC wire protocol)

▼ Bambu bus protocol data

- Capture source: Serial port read from forward (2)
- > Flags: 0x00
- Sequence number: 138
- [Request: 3262](#)
- Destination: AP (6)
- Source: AMS (7)
- Command: 0x0218
- ▼ ams_tray_info_write
 - Payload: 010100

0000 02 3d 00 8a 00 12 00 d5 00 06 00 07 18 02 01 01 ..=.....
 0010 00 96 02

pack_get_part_info: Protocol Packets: 7048 · Displayed: 300 (4.3%) Profile: Default

bug-report.pcap

not mcproto.publish_dds_cm_n_recorder and not mcproto.pack_get_part_info

No.	Time	Source	Destination	Protocol	Length	Info
39449	190.244423	MC	AP	MCPR0T0	56	[#938] mcu_display_message: "[BMC]GL AP lock! Pre=idle"
39450	190.244528	MC	AP	MCPR0T0	66	[#939] mcu_display_message: "[BMC]g29_before_print_flag set to 0"
39451	190.244942	AP	MC	MCPR0T0	95	[#1658] gcode_line_handle #16: M1009 L1 01\nM1002 set_flag:extru
39452	190.245228	AP	MC	MCPR0T0	101	[#1659] gcode_line_handle #17: M1009 L1 01\nM1002 set_flag:xy_me
39453	190.245308			X1PLUS ...	102	CProtocal_send_mcu_packet_override: found M620 E0, patching to M
39454	190.245406	AP	MC	MCPR0T0	68	[#1660] gcode_line_handle #18: M1009 L1 01\nM620 E1\n\nM1009 L0
39455	190.245585	AP	MC	MCPR0T0	99	[#1661] gcode_line_handle #19: M1009 L1 01\nM1002 set_flag:time
39456	190.256353	MC	AP	MCPR0T0	56	[#940] mcu_display_message: "[BMC]GL AP unlock! Pre=AP"
39458	190.256877	MC	AP	MCPR0T0	111	[#941] mcu_display_message: "[LINK]GL s=16 [60] [M1009 L1 01^M100
39459	190.257080	MC	AP	MCPR0T0	36	[RSP #1658] gcode_line_handle #16 -> 0

> Frame 39453: Packet, 102 bytes on wire (816 bits), 102 b

> EXPORTED_PDU

> X1Plus log event

Log type: Log from forward (0)

Message: CProtocal_send_mcu_packet_override: found M62

```

0000 00 0c 00 0a 78 31 70 6c 75 73 5f 6c 6f 67 00 00  ...x1pl us_log..
0010 00 00 00 43 50 72 6f 74 6f 63 61 6c 5f 73 65 6e  ...CProtocal sen
0020 64 5f 6d 63 75 5f 70 61 63 6b 65 74 5f 6f 76 65  d_mcu_pa cket_ove
0030 72 72 69 64 65 3a 20 66 6f 75 6e 64 20 4d 36 32  rride: f ound M62
0040 30 20 45 30 2c 20 70 61 74 63 68 69 6e 67 20 74  0 E0, pa tching t
0050 6f 20 4d 36 32 30 20 45 31 3b 20 75 73 65 5f 61  o M620 E 1; use_a
0060 6d 73 20 3d 20 30  ms = 0
  
```

Message (x1plus_log.message), 83 bytes

Packets: 120304 · Displayed: 16495 (13.7%)

Profile: Default

Writing dissectors

- Dissector Lua API is arcane, probably useful to have wrappers to hide some of the worst of it and focus on what you are writing

Now for some commands:

```
function mk_cmd(cmdset, cmdid, name, fields, dis)
    local myname = "mcproto." .. name
    local proto = Proto(myname, name)
    proto.fields = {}
    for _,v in pairs(fields) do table.insert(proto.fields,v) end
    proto.dissector = function (buf, pinfo, root) return dis(proto, fields, buf, pinfo, root) end
    mcommands:set(cmdset * 256 + cmdid, proto)
    return proto
end
```

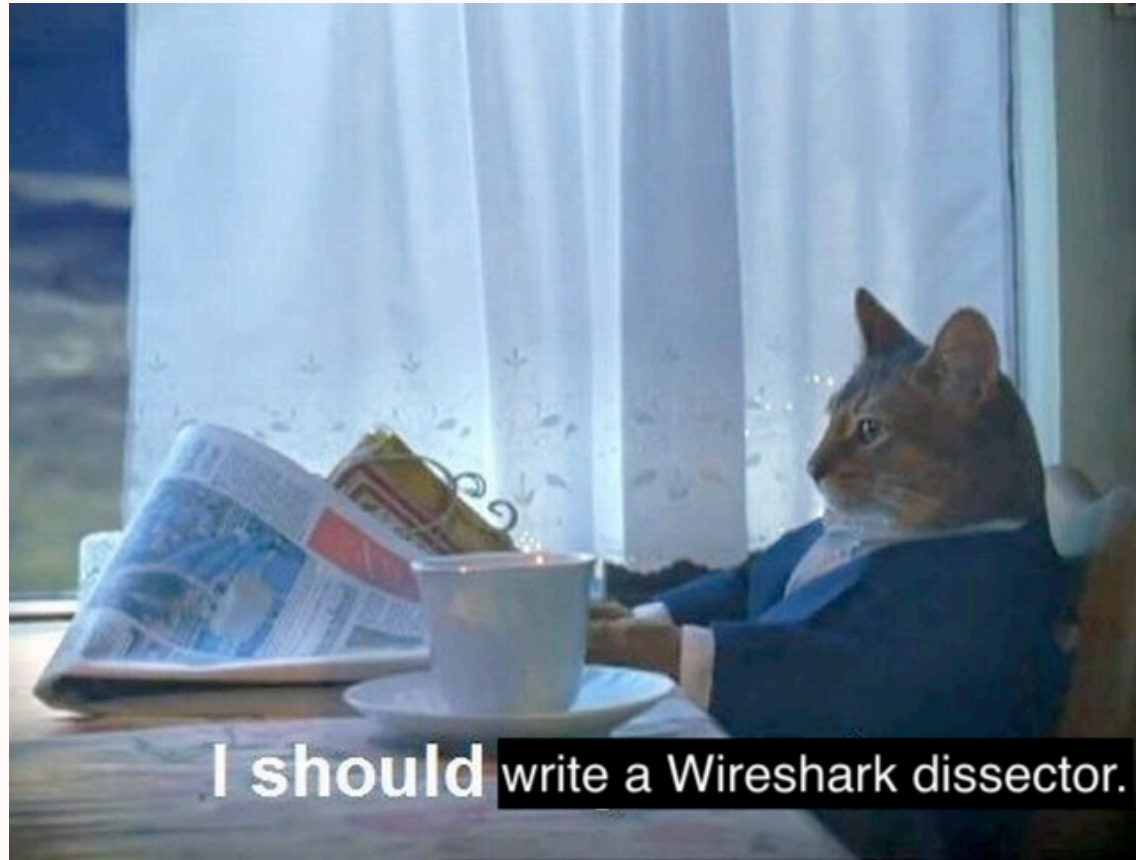
Writing dissectors

- Dissector Lua API is arcane, probably useful to have wrappers to hide some of the worst of it and focus on what you are writing

```
mk_cmd(2, 9, "mcu_display_message",
  { message = ProtoField.string("mcproto.mcu_display_message.message", "Message") },
  function (proto, fields, buf, pinfo, root)
    local len = buf:len()
    local tree = root:add(proto, buf:range(0, len), "mcu_display_message")
    tree:add(fields.message, buf:range(0, len))
    pinfo.cols.info:append("mcu_display_message: \"\"..buf:range(0, len):string()..\"\"")
  end
)
```

Writing dissectors

- Dissector Lua API is arcane, probably useful to have wrappers to hide some of the worst of it and focus on what you are writing
- Just hold your nose and do it



I should write a Wireshark dissector.

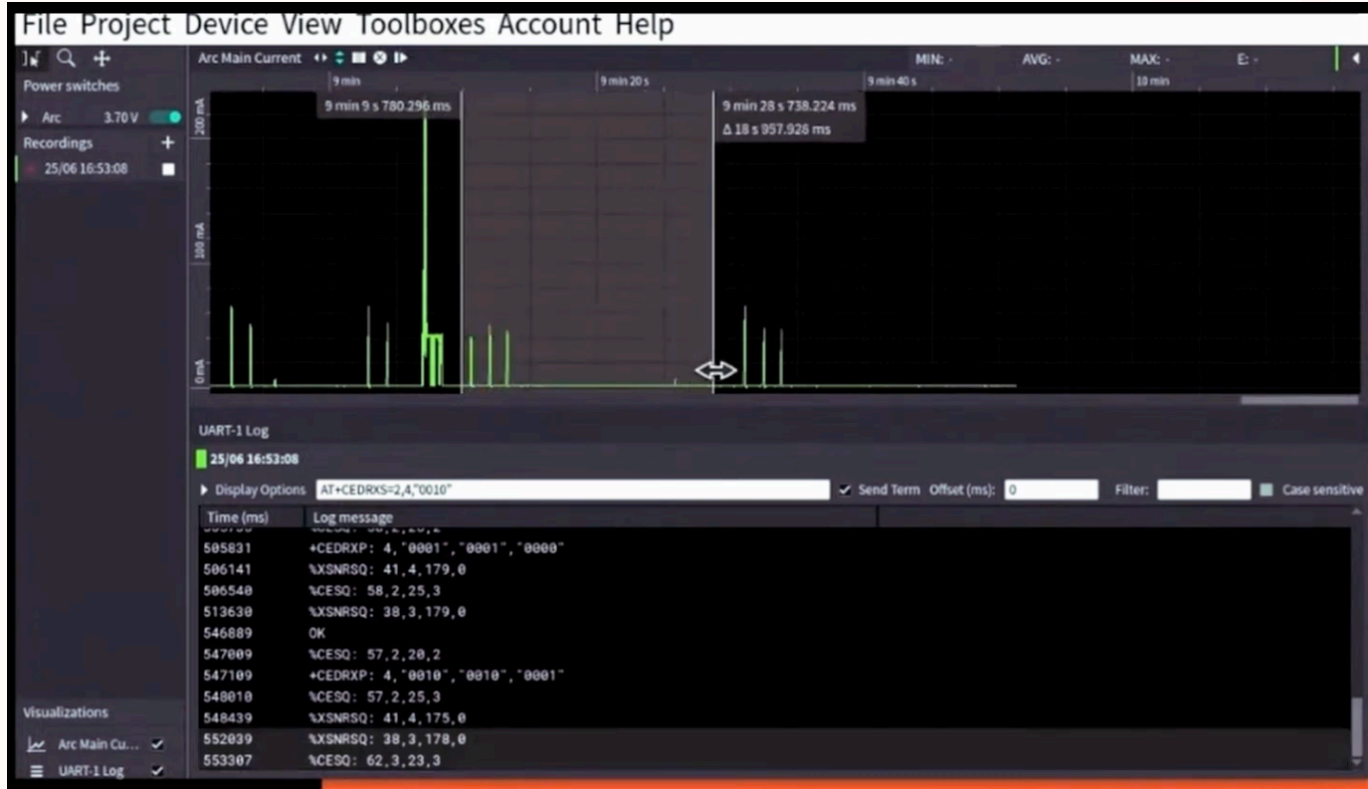
**Part 2: ... and then, we should all
dream bigger**

Accelerated **Tech**

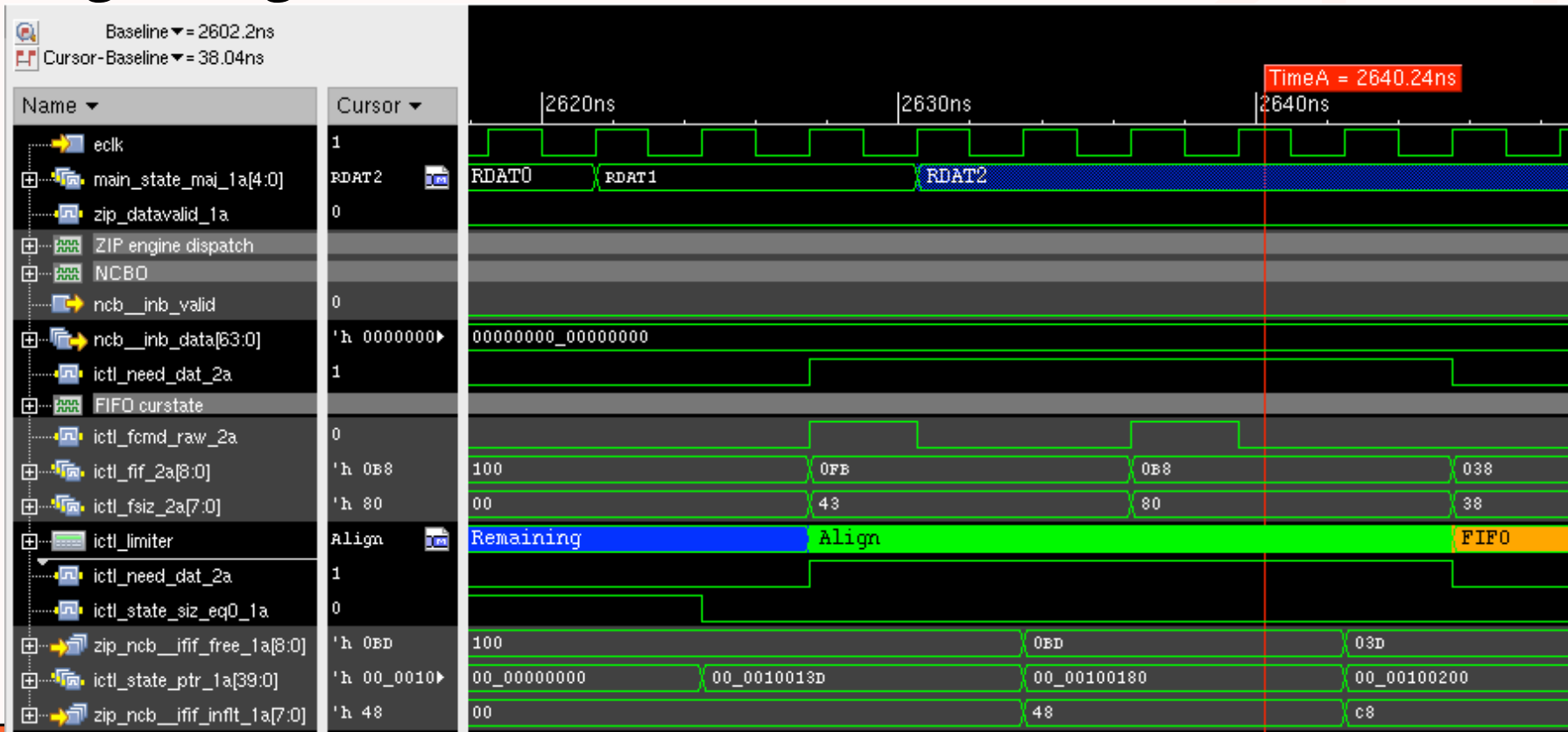
Wireshark is ok, but...

- There are many ways of debugging using time-series data
- Wireshark / pcap tools are only one of them
- Here are a few

Analog data, synced with UART (Otii Arc)



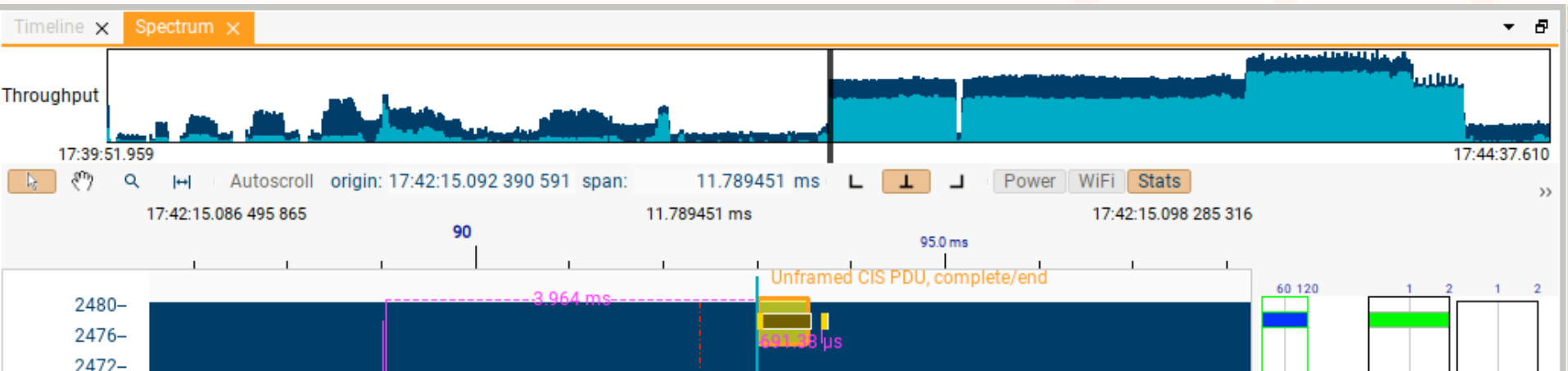
Digital logic waveform viewers (Cadence Simvision, 2009!)



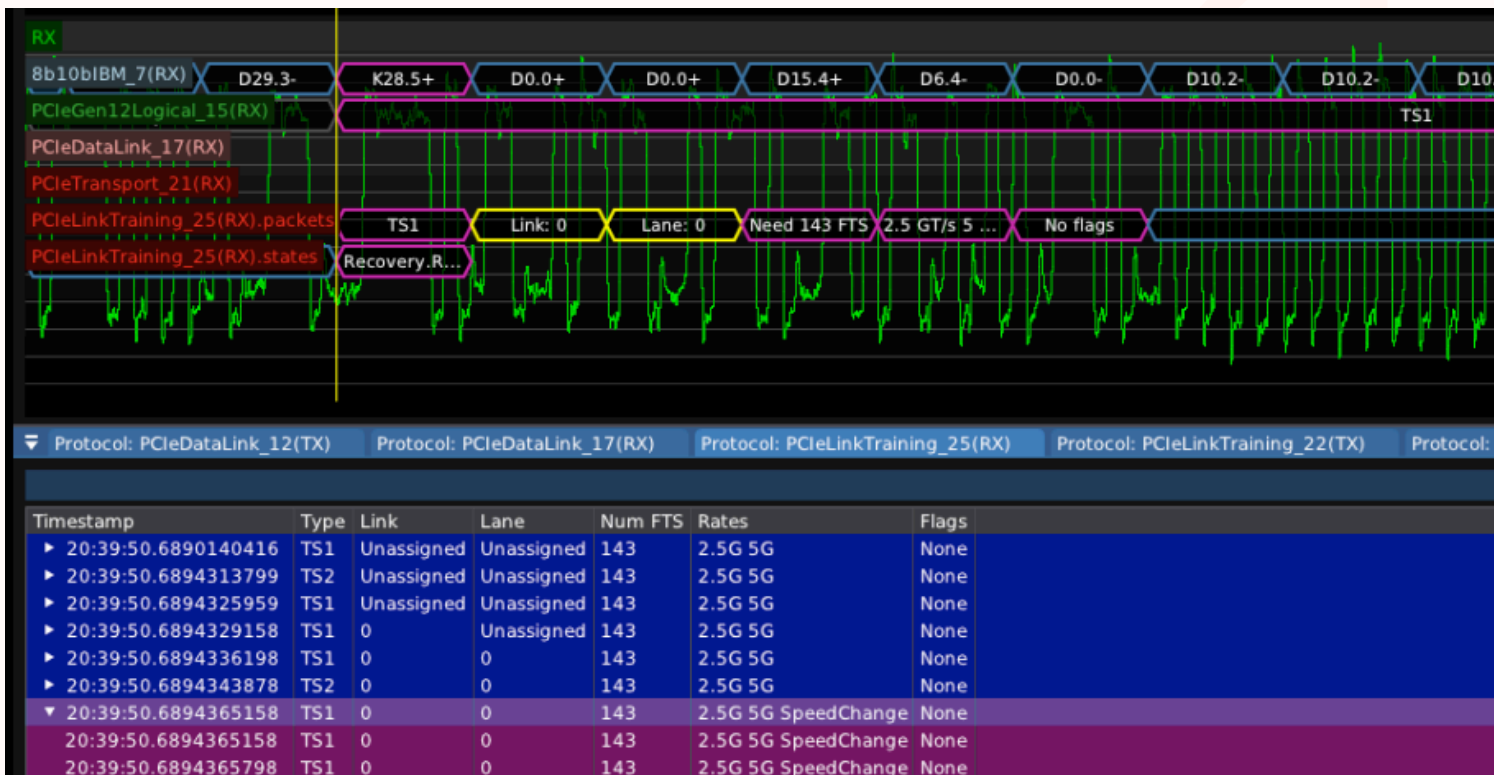
Transaction-level analysis (RFcreations blueSPY)

Icons	Access Code	Summary	Status	Payload	Dur
BR Q 10889	08:5E:5F	SDP Service Attribute Transaction (IDs #25, #26)		4 PDUs: 15 B, 44 B, 17 ...	
BR Q 10934	08:5E:5F	SDP Service Attribute Transaction (IDs #27, #28)		4 PDUs: 15 B, 44 B, 17 ...	
BR Q 10986	08:5E:5F	SDP Service Attribute Transaction (IDs #29, #30)		4 PDUs: 15 B, 44 B, 17 ...	
BR Q 10986	08:5E:5F	Trans #29, SDP SERVICE ATTR REQ	OK	23 bytes: 13 00 13 04 ...	
BR Q 10986	08:5E:5F	ACL (ACL-U), EDR-2: 2-DH1, SDP SERVICE ATTR REQ	OK	27 bytes: BE 00 13 00 ...	
BR Q 11022	08:5E:5F	Trans #29, SDP SERVICE ATTR RSP	OK	52 bytes: 30 00 13 11 ...	
BR Q 11038	08:5E:5F	Trans #30, SDP SERVICE ATTR REQ; 16 ReTXed packets	ReTXed packets	25 bytes: 15 00 13 04 ...	
BR Q 11069	08:5E:5F	Trans #30, SDP SERVICE ATTR RSP	OK	45 bytes: 29 00 13 11 ...	
BR Q 11081	08:5E:5F	SDP Service Search Transaction (ID #31)		2 PDUs: 9 B, 6 B	
BR + 11111	08:5E:5F	Proc #11: L2CAP DISCONNECTION, SDP (0x413, 0x1113)	OK	REQ; RSP	
BR + 11149	08:5E:5F	CQDDR request (Peripheral initiated)	OK	1 PDU: LMP PREFERRE...	
BR + 11153	08:5E:5F	Proc #14: L2CAP CONFIGURATION, AVDTP (0x3CD, 0x1012)	OK	REQ; RSP	
BR + 11161	08:5E:5F	Proc #12: L2CAP CONFIGURATION, AVDTP (0x3CD, 0x1012)	OK	REQ; RSP	
BR + 11186	08:5E:5F	AVDTP Transaction #8: AVDTP DISCOVER, Command + Response Accept			
Q + 11202	0xD50AD9BF	Proc #23: L2CAP CONNECTION, 0xFEFD (0x454, 0x8D4)	OK	REQ; RSP; RSP	
Q + 11220	0xD50AD9BF	Proc #9: L2CAP CONFIGURATION, 0xFEFD (0x454, 0x8D4)	OK	REQ; RSP	
BR + 11236	08:5E:5F	AVDTP Transaction #8: AVDTP GET ALL CAPABILITIES, Command + Response Accept			
Q + 11243	0xD50AD9BF	Proc #24: L2CAP CONFIGURATION, 0xFEFD (0x454, 0x8D4)	OK	REQ; RSP	
Q + 11279	0xD50AD9BF	Proc #25: L2CAP CONNECTION, 0xFEFB (0x495, 0x915)	OK	REQ; RSP; RSP; RSP	
BR + 11295	08:5E:5F	AVDTP Transaction #8: AVDTP GET ALL CAPABILITIES, Command + Response Accept			

Spectrum analysis (RFcreations blueSPY)



Transaction + analog analysis (ngscopeclient)



Program values: good old gdb

```
Breakpoint 1, main () at each-loop.c:3
3      usleep(i);
(gdb) print i
$1 = 0
(gdb) c
Continuing.

Breakpoint 1, main () at each-loop.c:3
3      usleep(i);
(gdb) print i
$2 = 1
(gdb) c
Continuing.

Breakpoint 1, main () at each-loop.c:3
3      usleep(i);
(gdb) print i
$3 = 2
(gdb) c
Continuing.
```

Program values: good old gdb

Q: what about more variables?

Q: what about if we ran this in rr,
and could step forward and back?

Q: what if all expressions had
timestamps associated with them?

```
Breakpoint 1, main () at each-loop.c:3
3          usleep(i);
(gdb) print i
$1 = 0
(gdb) c
Continuing.
```

```
Breakpoint 1, main () at each-loop.c:3
3          usleep(i);
(gdb) print i
$2 = 1
(gdb) c
Continuing.
```

```
Breakpoint 1, main () at each-loop.c:3
3          usleep(i);
(gdb) print i
$3 = 2
(gdb) c
Continuing.
```

And, of course, Wireshark

set-ams-color-with-fix.pcap

not mcproto.publish_dds_cmh_recorder and not mcproto.pack_get_part_info

No.	Time	Source	Destination	Protocol	Length	Info
2828	14.415474	AP	MC	MCPROTO	24	[RSP #122] sync_timestamp
3032	15.420518	AP	MC	MCPROTO	24	[RSP #128] sync_timestamp
187	0.707678	22630	AP	MCPROTO	152	[RSP #12] ams_tray_info_read
3220	16.421264	AP	MC	MCPROTO	24	[RSP #136] sync_timestamp
← 3300	16.817612	AMS	AP	MCPROTO	19	[RSP #138] ams_tray_info_write
3406	17.439293	AP	MC	MCPROTO	24	[RSP #143] sync_timestamp
3611	18.432188	AP	MC	MCPROTO	24	[RSP #151] sync_timestamp
3796	19.419452	AP	MC	MCPROTO	24	[RSP #157] sync_timestamp
3955	20.245842	MC	AP	MCPROTO	32	[RSP #162] link_ams_tray_consumption_ack2

> Frame 3300: Packet, 19 bytes on wire (152 bits), 19 bytes captured (152 bits) on interface 0, Ethernet II, Src: Bambu (08:00:00:00:00:00), DLT: 147, Payload: mcproto (Bambu MC wire protocol)

- ▼ Bambu bus protocol data
 - Capture source: Serial port read from forward (2)
 - > Flags: 0x00
 - Sequence number: 138
 - [Request: 3262](#)
 - Destination: AP (6)
 - Source: AMS (7)
 - Command: 0x0218
 - ▼ ams_tray_info_write
 - Payload: 010100

```
0000 02 3d 00 8a 00 12 00 d5 00 06 00 07 18 02 01 01 ..=.....
0010 00 96 02 .....
```

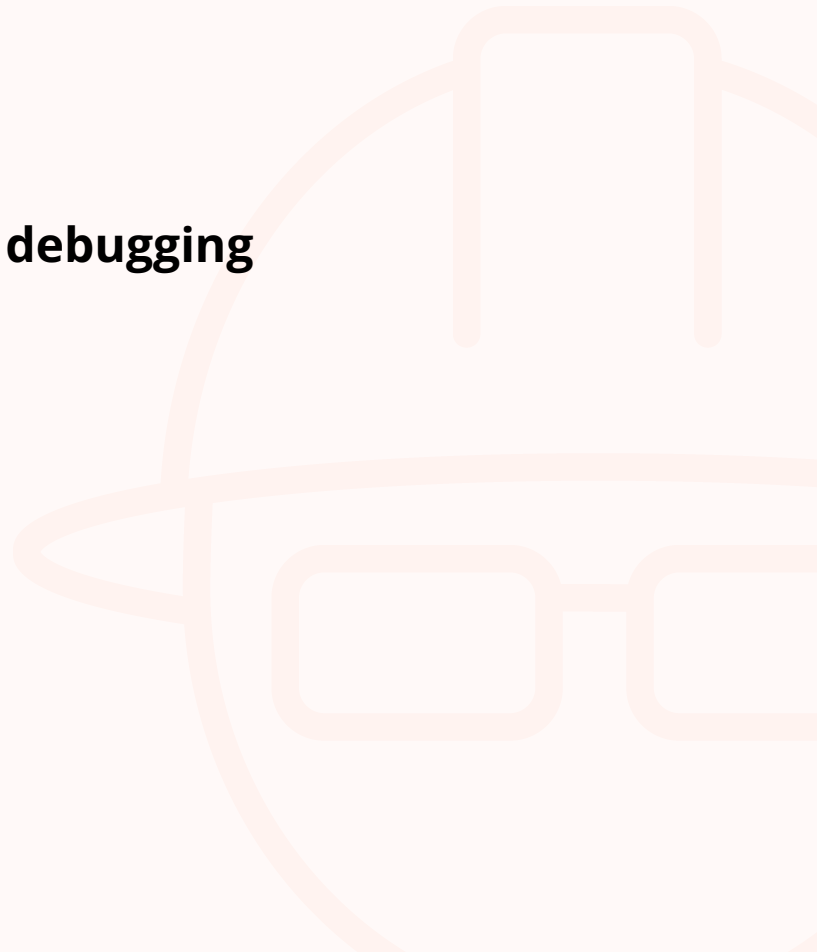
pack_get_part_info: Protocol

Packets: 7048 · Displayed: 300 (4.3%)

Profile: Default

Time series debugging

- All of these are ways to do time series debugging



Time series debugging

- **All of these are ways to do time series debugging**
- None of them are on speaking terms with each other
 - That is a god-damn shame
- I do not know the answer to this

Dream bigger

- It seems clear to me that we are handicapped by tools
- I have often wanted combinations of these
 - scope + ARM swtrace! flame chart plus UART! for the love of everything good, I want memory variables in a waveform viewer!

Dream bigger

- It seems clear to me that we are handicapped by tools
- I have often wanted combinations of these
 - scope + ARM swtrace! flame chart plus UART! for the love of everything good, I want memory variables in a waveform viewer!
- You should build a Wireshark dissector, and experiment with these other modalities too. You deserve better than grep and sed.
- But we all deserve better than what we have. Time for us to get dreaming.

Joshua Wise

April, 2026

Web: <https://joshuawise.com/>

Work: <https://accelerated.tech/>

E-mail: joshua@accelerated.tech

Fediverse: [@joshua@social.emarhivil.com](https://social.emarhivil.com/@joshua)

Accelerated **Tech**

