If a checksum fails, the diagnostic flag is set to 0, and we won't be able to read the optical and temperature monitoring data. To restore functionality, the EEPROMs need to be corrected to pass the checksum.

Firstly, we need to proceed with shutting down the HAL service, as reading and writing to EEPROMs is done through the concurrent I2C bus. Before killing the HAL service, stop the monitoring that checks its status and brings it up if it's not running:

1. /etc/init.d/monit stop
2. ps aux | grep Hal
3. kill <pid>

At this point, we're ready to operate safely, but first, some things to know about the structure of the EEPROM, as directly reported from the internal code of WWRS:

struct shw\_sfp\_header {

uint8\_t id=0x80;

uint8\_t ext\_id=0x04;

uint8\_t connector=0x80;

uint8\_t transciever[8]={0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00};

uint8\_t encoding=0x01;

uint8\_t br\_nom=0x0A;

uint8\_t reserved1=0x00;

uint8\_t length1=0x0A; /\* Link length supported for 9/125 mm fiber (km) \*/

uint8\_t length2=0x64; /\* Link length supported for 9/125 mm fiber (100m) \*/

uint8\_t length3=0x00; /\* Link length supported for 50/125 mm fiber (10m) \*/

uint8\_t length4=0x00; /\* Link length supported for 62.5/125 mm fiber (10m) \*/

uint8\_t length5=0x00; /\* Link length supported for copper (1m) \*/

uint8\_t length6=0x00; /\* Link length supported on OM3 (1m) \*/

uint8\_t vendor\_name[16]={0x47, 0x4C, 0x45, 0x4E, 0x41, 0x49, 0x52, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20};

uint8\_t reserved3=0x00; /\* This is now a field named transceiver \*/

uint8\_t vendor\_oui[3]={0x00, 0x00, 0x00};

uint8\_t vendor\_pn[16]={0x30, 0x35, 0x37, 0x2D, 0x30, 0x32, 0x33, 0x31, 0x2D, 0x32, 0x37, 0x2D, 0x31, 0x47, 0x20, 0x20};

uint8\_t vendor\_rev[4]={0x20, 0x20, 0x20, 0x20};

uint8\_t tx\_wavelength[2]={0x04, 0xF6};

uint8\_t reserved4=0x00;

uint8\_t cc\_base=0x25;

/\* extended ID fields start here \*/

uint8\_t options[2]={0x00, 0x1A};

uint8\_t br\_max=0x00

uint8\_t br\_min=0x00

uint8\_t vendor\_serial[16]={0x30, 0x30, 0x31, 0x31, 0x31, 0x31, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20};

uint8\_t date\_code[8]={0x32, 0x31, 0x30, 0x34, 0x32, 0x31, 0x00, 0x00};

uint8\_t diagnostic\_monitoring\_type=0x68;

uint8\_t enhanced\_options=0xB0;

uint8\_t sff\_8472\_compliance=0x05;

uint8\_t cc\_ext=0xD3;

} \_\_attribute\_\_ ((packed));

What we need to know about the structure is that the checksum comparison fields are CC\_BASE and CC\_EXT. The checksum is simply the cumulative sum of the bytes from the first to the 63rd (reserved4) byte. Then, a bitwise AND 0xFF is performed on the result to extract only the last 8 bits to compare the resulting byte with the CC\_BASE byte. If they are equal, the first checksum will pass positively, and we will proceed with the second. The second checksum involves comparing the cumulative sum from the 64th to the 95th byte. Then, a bitwise AND 0xFF is performed on the result, and it's compared to the 96th byte CC\_EXT. The only variable fields found in this structure are CC\_EXT, VENDOR\_SERIAL (starting from the 69th byte), and DATE\_CODE (starting from the 85th byte). Since there are no variable fields in the first 63 bytes, CC\_BASE is always the same for all (0x25).

Scenario 1:

Checksum failed, EEPROM readable and writable without errors. In this scenario, the EEPROM allows reading without errors and writing. In this case, we just need to copy the hexadecimal structure of the header from another functioning SFP with /wr/bin/wrs\_sfp\_dump -I -x -p <sfp>. Copy the hexadecimal and prepare to rewrite it all:

*echo -n -e "\x80\x04\x80\x00\x00\x00\x02\x00\x00\x00\x00\x01\x0a\x00\x0a\x64\x00\x00\x00\x00\x47\x4c\x45\x4e\x41\x49\x52\x20\x20\x20\x20\x20\x20\x20\x20\x20\x00\x00\x00\x00\x30\x35\x37\x2d\x30\x32\x33\x31\x2d\x32\x37\x2d\x31\x47\x20\x20\x20\x20\x20\x20\x04\xf6\x00$CC\_BASE\x00\x1a\x00\x00$Vendor\_Serial\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20$Date\_Code\x68\xb0\x05$CC\_EXT" > header*

(Pay attention at the fields that you have to substitute with corrispondent hexadecimal, $CC\_BASE, $Vendor\_Serial, $Date\_Code and $CC\_EXT) If everything has gone correctly, a file of exactly 96 bytes will be generated, which we can write to EEPROM with /wr/bin/wrs\_sfp\_dump -I -a WRITE -f <header> -p <sfp>. Reboot the WWRS, and everything should be restored. We can check the diagnostic flag with /wr/bin/wrs\_dump\_shmem | grep "Hal.port.<sfp>.has\_sfp\_diag:".

Scenario 2:

In this scenario, we find ourselves with an EEPROM that no longer allows the overwriting of some bytes, which are permanently corrupted. In this case, we need to proceed in reverse to retrieve the various CC\_BASE and CC\_EXT values. If the CC\_BASE and CC\_EXT fields are still writable, we can simply extract the corrupted header by adding the -v command, then /wr/bin/wrs\_sfp\_dump -I -v -x -p <sfp>. Calculate the two cumulative sums (we can also perform some writing tests first to check which bytes we can restore, just to maintain the highest possible data consistency and similarity to functioning SFPs) and generate the header file with the same command as before, replacing CC\_BASE (0x25) and CC\_EXT with our results.

Scenario 3:

The worst-case scenario... Even the CC\_BASE and CC\_EXT fields are corrupted and no longer writable. Proceed with verbose reading (it would also be possible to use /wr/bin/wrs\_sfp\_dump -I -a READ -f <output\_file> -p <sfp>, but then we would have to convert it to bytes) with /wr/bin/wrs\_sfp\_dump -I -v -x -p <sfp>. At this point, test which byte closest to 0x00 (for ease of calculation) is still writable and use it to make the cumulative sum equal to the corresponding CC by modifying its value ad hoc. This way, we can overcome the various checksums, but we won't have a header completely consistent with the original data, yet it will still be fully functional!