OpenFWWF
RX & TX data paths

A glimpse into the Linux Kernel Wireless Code

Part 3

Firmware in brief

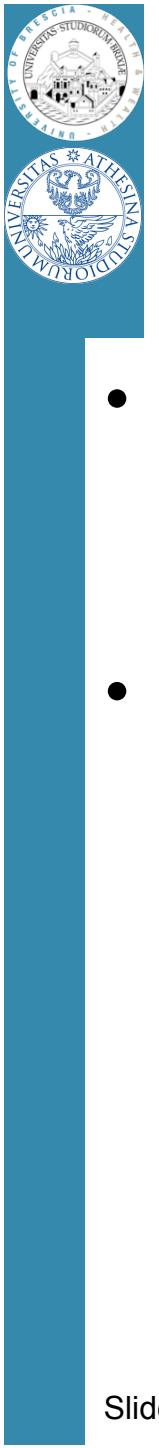
- Firmware seems really complex to understand ☹
  - Assembly language
    - CPU registers: 64 registers [r0, r1, …, r63]
    - SHM memory: 4KB of 16bits words addressable as [0x000] -> [0x7FF]
    - HW registers: spr000, spr001, …, spr1FF
  - Use `#define` macro to ease understanding
    - `#define CUR_CONTENTION_WIN r8`
    - `#define SPR_RXE_FRAMELEN spr00c`
    - `#define SHM_RXHDR SHM(0xA88)`
      - `SHM(.)` is a macro as well that divides by 2
  - Assignments:
    - Immediate `mov 0xABBA, r0; // load 0xABBA in r0`
    - Memory direct `mov [0x0013], r0; // load 16bit @ 0x0026 (LE!)`

Firmware in brief/2

- Value manipulation:
  - Arithmetic:
    - Sum: `add r1, r2, r3;` // r3 = r1 + r2
    - Subtraction: `sub r2, r1, r3;` // r3 = r2 - r1
  - Logical:
    - Xor: `xor r1, r2, r3;` // r3 = r1 ^ r2
  - Shift:
    - Shift left: `sl r1, 0x3, r3;` // r3 = r1 << 3
- Pay attention:
  - In 3 operands instruction, immediate value in range [0..0x7FF]
  - Value is sign extended to 16bits

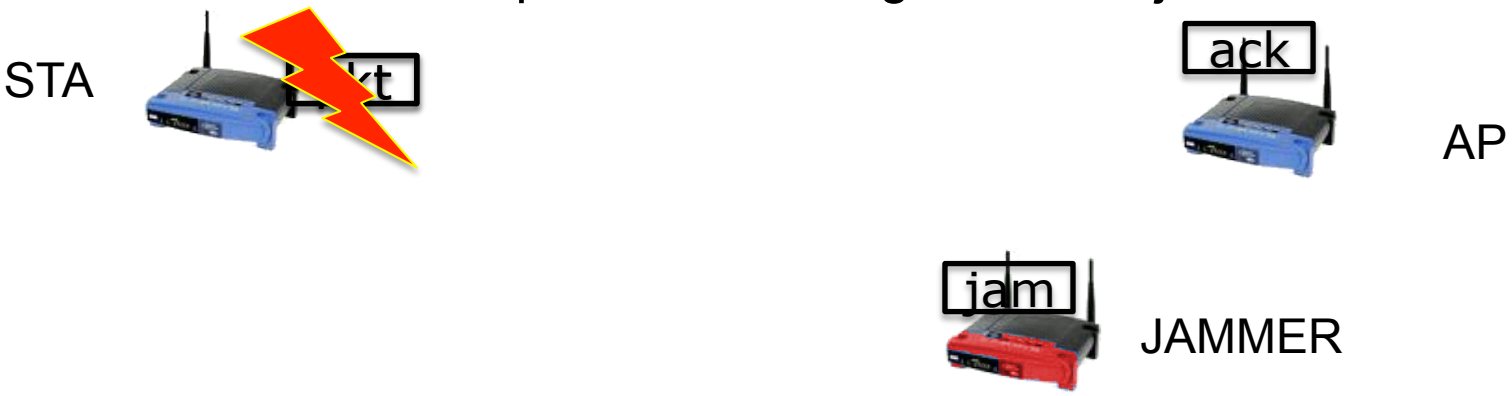
Firmware in brief/3

- Code flow execution controlled by using jumps
  - Simple jumps, comparisons
    - Jump if equal: `je r2, r5, loop;` // jump if r2 == r5
    - Jump if less: `jl r2, r5, exit;` // jump if r2 < r5 (unsigned)
  - Condition register jumps: jump on selected CR (condition registers)
    - on plcp end: `jext COND_RX_PLCP, rx_plcp;`
    - on rx end: `jext COND_RX_COMPLETE, rx_complete;`
    - on good frame: `jext COND_RX_FCS_GOOD, frame_ok;`
    - unconditionally: `jext COND_TRUE, loop;`
  - A check can also clean a condition, e.g.,
    - `jext EOI(COND_RX_PLCP), rx_plcp;` // clean CR bit before jump
  - Call a code subsection, save return value in link-registers (lr):
    - `call lr0, push_frame;` // return with `ret lr0, lr0;`



Firmware in brief/4

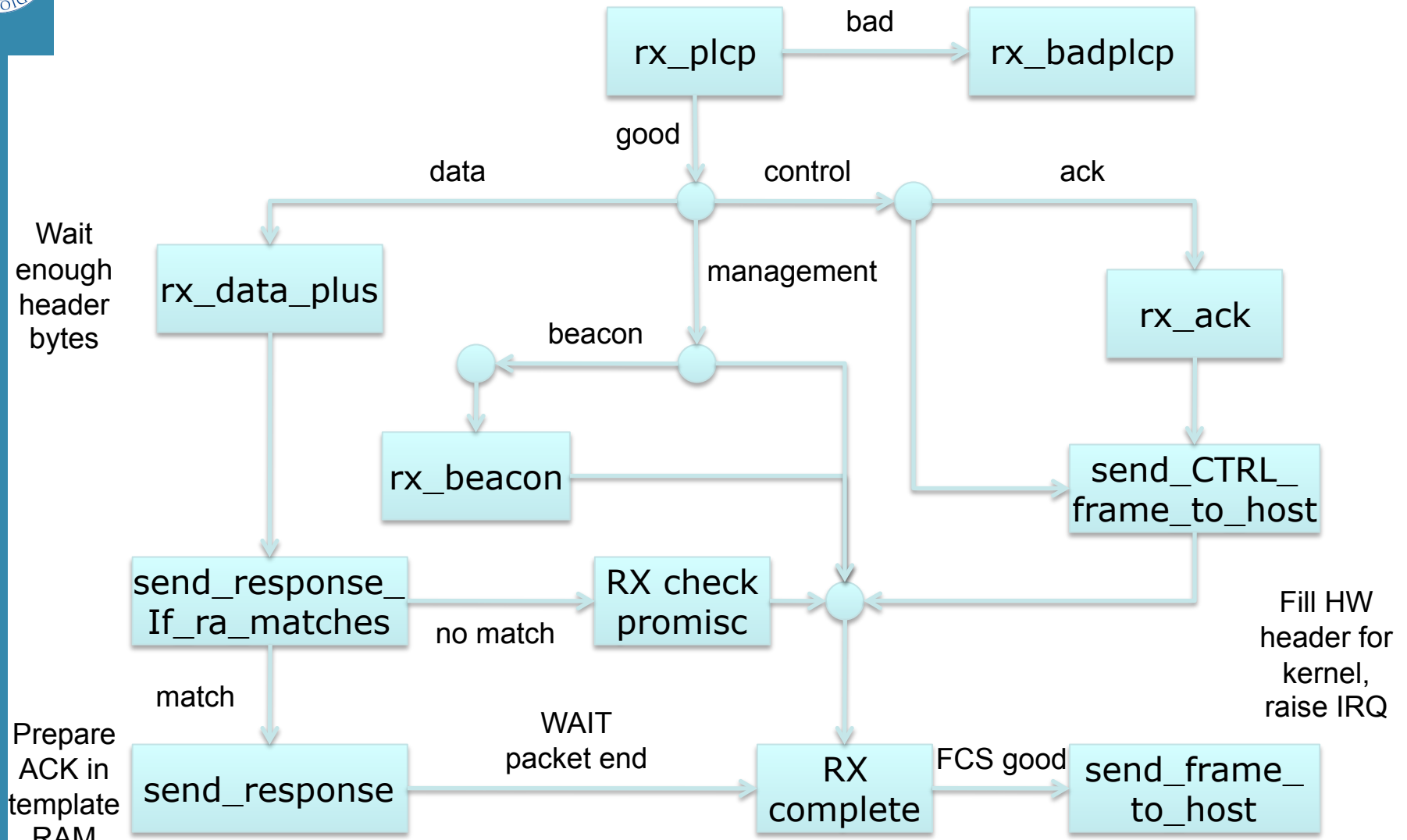
- OpenFWWF is today ~ 1000 lines of code
  - Not possible to analyze in a single lesson
  - We will analyze only some parts
- A simple exercise:
  - Analyze quickly the receiver section
  - Propose changes to implement a jammer
    - When receives packets from a given STA, jams noise!



RX code made easy

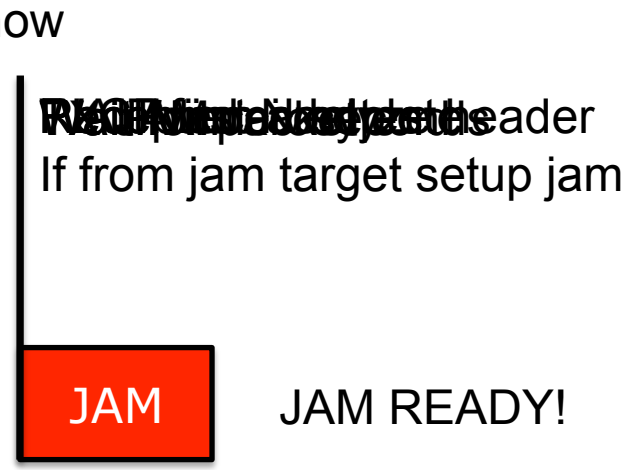
- During reception
  - CR RX_PLCP set when PLCP is completely received
  - CR COND_RX_BADPLCP set if PLCP CRC went bad
  - SPR_RXE_FRAMELEN hold the number of already received bytes
  - First 64B of packet are copied starting at `SHM_RXHEADER = SHM(0xA08)`
    - First 6B hold the PLCP
  - CR COND_RX_COMPLETE set when packet is ready
- We can have a look at the code flow for a data packet
  - rx_plcp: checks it's a data packet
  - rx_data_plus: checks packet is longer than 0x1C = 6(PLCP)B + 22(MAC)B
  - send_response: copy src mac address to ACK addr1, set state to TX_ACK
  - rx_complete: schedule ACK transmission

RX code path


rx_plcp
bad
rx_badplcp
good
data
control
ack
Wait enough header bytes
rx_data_plus
management
rx_ack
beacon
rx_beacon
send_CTRL_ frame_to_host
send_response_ If_ra_matches
no match
RX check promisc
Fill HW header for kernel, raise IRQ
match
Prepare ACK in template RAM
WAIT packet end
send_response
RX complete
FCS good
send_frame_ to_host

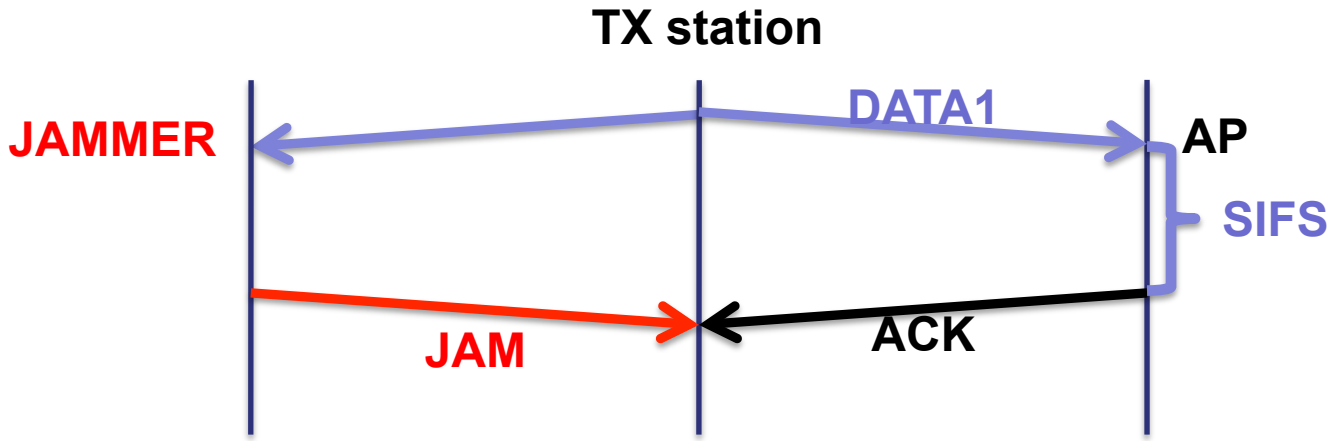
Let's hack and do jamming

- During reception CPU keeps on running
  - Detect end of PLCP
  - May wait for a given number of bytes received
  - May prepare a response frame (ACK)
  - Wait for end of reception
  - May schedule response frame transmission after a while



Let's hack and do jamming/2

- Disturbing a station when sending data
  - Jammer recognizes tx'ed data and sends fake ACK
- Maybe (for testing) jamming all packets is too much
  - Selected packets?



Let's hack and do jamming/3

- If first byte of a packet are copied to SHM
- If we have ways of displaying SHM
  - Could we find evidence of received packets?
- Useful tool
  - `$: readshm`
  - Display shared memory
- Run this experiment: run traffic from the STA to AP
  - On AP dump the SHM: locate the UDP packet
  - Fix the rate on STA: how do the first 6 bytes change?

Let's hack and do jamming/4

- Shared memory appears like this

```
0x0A00:  0000 0000 0000 0000 CCBF 0200 0000 0801   ................
0x0A10:  0400 0014 A442 958D 0014 A442 958D 0013   .....B.....B....
0x0A20:  D4BB 2CBF C006 AAAA 0300 0000 0800 4500   ..,...........E.
0x0A30:  05DA 3E7E 4000 4011 751B C0A8 0028 C0A8   ..>~@.@.u....(..
0x0A40:  0001 CB86 0BB8 05C6 0F6E 0000 459E 531C   .........n..E.S.
0x0A50:  ADA9 0000 84FD 0000 0000 0000 0001 0000   ................
0x0A60:  0BB8 0000 0000 0337 F980 FFFE 7960 3637   .......7....y`67
0x0A70:  3839 3031 3233 3435 3637 3839 3031 3233   8901234567890123
0x0A80:  3435 3637 3839 3031 5100 0000 0600 2A50   45678901Q.....*P
0x0A90:  E54F 0000 0000 0000 B4FB A202 0000 0000   .O..............
```

Let's hack and do jamming/4

- Shared memory appears like this

```
0x0A00:  0000 0000 0000 0000 CCBF 0200 0000 0801   ................
0x0A10:  0400 0014 A442 958D 0014 A442 958D 0013   .....B.....B....
0x0A20:  D4BB 2CBF C006 AAAA 0300 0000 0800 4500   ..,...........E.
0x0A30:  05DA 3E7E 4000 4011 751B C0A8 0028 C0A8   ..>~@.@.u....(..
0x0A40:  0001 CB86 0BB8 05C6 0F6E 0000 459E 531C   .........n..E.S.
0x0A50:  ADA9 0000 84FD 0000 0000 0000 0001 0000   ................
0x0A60:  0BB8 0000 0000 0337 F980 FFFE 7960 3637   .......7....y`67
0x0A70:  3839 3031 3233 3435 3637 3839 3031 3233   8901234567890123
0x0A80:  3435 3637 3839 3031 5100 0000 0600 2A50   45678901Q.....*P
0x0A90:  E54F 0000 0000 0000 B4FB A202 0000 0000   .O..............
```

- What should we check if we want to jam only UDP frame to port 3000?
- We have also to wait for at least …. Bytes have been received, right?

Let's hack and do jamming/5

- Legacy rx_data_plus:

```
rx_data_plus:
        jext    COND_RX_COMPLETE, end_rx_data_plus
        jl      SPR_RXE_FRAMELEN, 0x01C,rx_data_plus
end_rx_data_plus:
        jl      SPR_RXE_FRAMELEN, 0x01C, rx_check_promisc
        jnext   COND_RX_RAMATCH, rx_ra_dont_match
        jext    COND_TRUE, send_response
```

- What we change?
  - Change the frame length
  - Add filter
  - If frame match filter, then "send_response" and remember somewhere!

Let's hack and do jamming/6

- Legacy rx_complete

```
rx_complete:
   [cut]
frame_successfully_received:
        jext    COND_RX_FIFOFULL, rx_fifo_overflow
        jnext   COND_NEED_RESPONSEFR, check_frame_subtype
need_regular_ack:
        je      [SHM_CURMOD], 0x001, ofdm_modulation
```

- What we change?
  - If we had remembered somewhere this is to jam
    - JAM IT!, schedule the frame anyway

JAM code

- To switch to a different firmware
  - Look at /lib/firmware
  - Link the desired firmware release as “b43”
  - Remove b43 module, reload and bring back the network up

```
$: rmmod b43 . . .
```

- How to test JAM code? “iperf” performance tool
- On AP run in server mode (receiver)

```
$: iperf -s -u -p 3000 -i 1
```

- On STA run in client mode (transmit)

```
$: iperf -c IP_OF_AP -u -p 3000 -i 1 -t 10
```

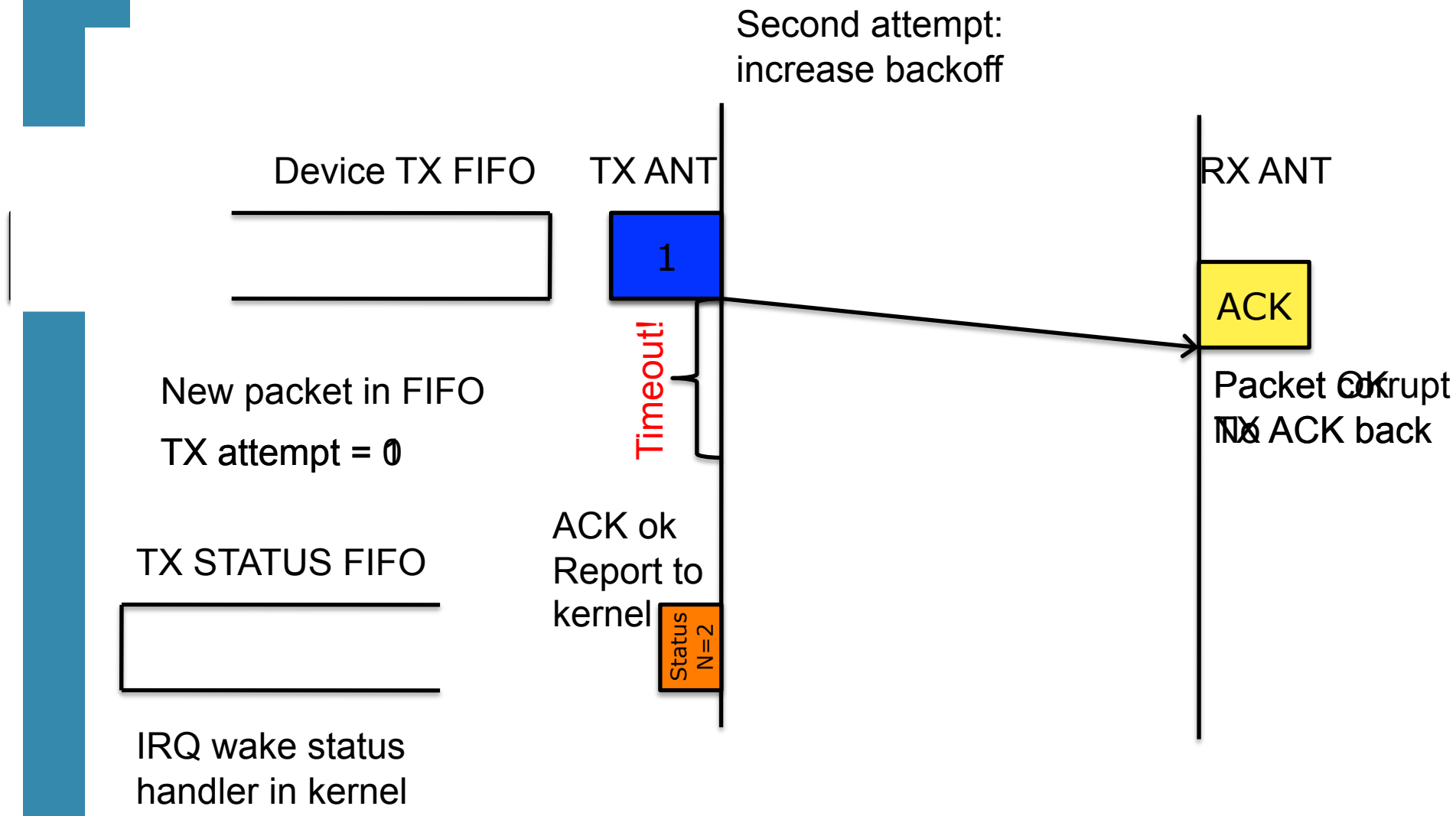
TX made easy

- Packets are prepared by the kernel
  - Fill all packet bytes (e.g., 802.11 header)
  - Choose hw agnostic device properties
    - Tx power to avoid energy wasting
    - Packet rate: rate control algorithm (minstrel)
  - A driver translates everything into hw specific
    - b43: rate encoded in PLCP (first 6B)
    - b43: append a fw-header at packet head
      - Firmware will setup hw according to these values

TX made easy/2

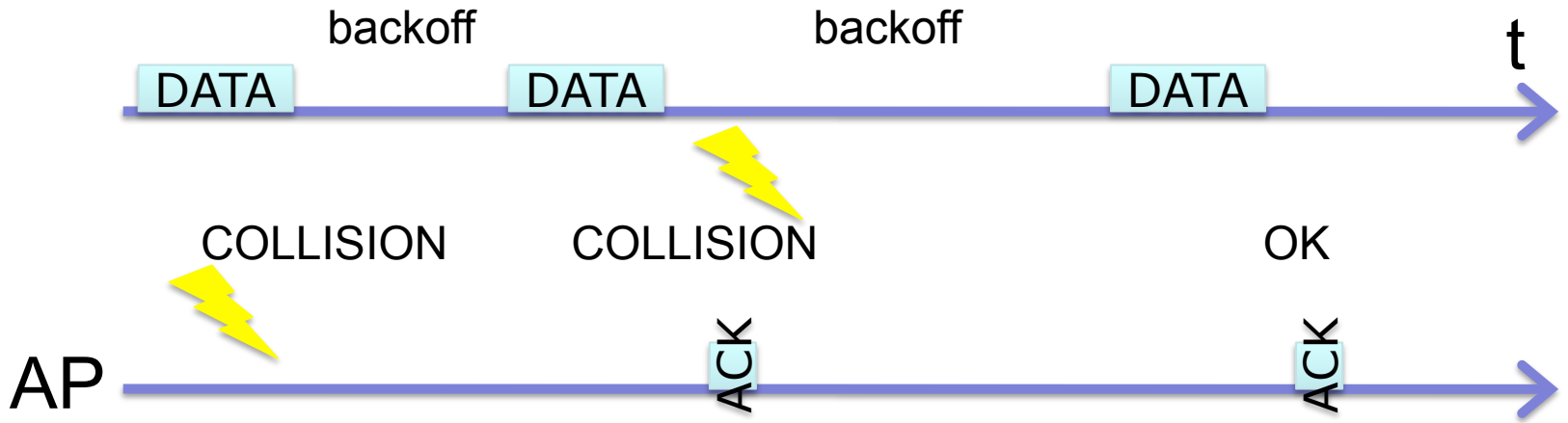
- Kernel (follows)
  - b43: send packet data (+hw info) through DMA
- firmware:
  - Continuous loop, when no receiving
    - If IDLE, check if packet in FIFO (comes from DMA)
    - If packet does not need ACK, TX,report and exit
    - If packet needs ACK, wait ACK timeout
    - If ACK timeout expired:
      - if ACK RXed, report to kernel, exit
      - If ACK not RXed, setup backoff, try again
      - If too much TX attempts
        - » remove packet from FIFO, report to kernel, exit

TX made easy/3



TX made easy/4

- Summary



- FW reports to kernel the number of attemps
  - Kernel feeds the rate control algo
  - A rate for the next packet is chosen

TX made easy/5

- Currently “minstrel” is the default RC algo
  - At random intervals tries all rates
  - Builds a tables with success “rate” for each “rate”
  - In the short term it selects the best rate
  - How to checks this table from userspace?
    - DEBUGFS ☺
    - Take a look at folder

      `/sys/kernel/debug/ieee80211/phyN/`

TX made easy: exercise

- Firmware: backoff entered if ack is not rx
  - Simple experiment
    - Two STAs joined to the same BSS
    - iperf on both STAs to the AP
    - They should share the channel
  - What happen if we hack one station fw?
  - Let's try…
    - TX path really complex, skip
    - But at source top we have a few "_CW" values