## *Exercise: playing with jammers*

### *1. Goals*

Students must implement a perfectly working jammer that is able to transmit some data over a frame that is being received and that satisfies some specific rules. To this end they have to

1) add to the firmware the code that stops the current reception and schedules immediate transmission of a new packet;
2) deploy a tx/rx infrastructure with proper counters so that they can validate and measure the jam efficiency;
3) run measurement experiments and determine the jam efficiency with respect to specific parameters.

### *2. Tutorial steps*

1) **Stopping current reception** Basic building block for the jam system is the filtering code that determines in real time and in a "at earliest" fashion whether the frame is "To Be Jammed" (TBJ) or not. This code should be placed in `rx_plcp` handler after the first two instructions. Given that we want to jam at earliest, we should simply filter the source MAC address. Given also that we want to measure the efficiency of the jammer it is much safer to avoid disconnection of the TBJ client from its Access Point, so we should jam only data frames and leave out control and management (like probe, probe response and their acknowledgment). Another possibility for debugging purposes is to jam only UDP packets to a specific destination port: refer to previous exercises for the filtering code.
   Once the filter detects a TBJ frame (e.g., UDP to port 12345) we should a) stop immediately the current reception and reset the state machine, and b) schedule the jam signal for immediate transmission. To this end we should

   a. *Stop immediately the current reception* with the following code

```
orx    7, 8, 0x000, 0x004, SPR_RXE_FIFOCTL1 // stop the receiver
or     SPR_RXE_FIFOCTL1, 0x000, r63         // flush the receiver
mov    0, SPR_BRC                           // reset the state machine
```

   A side effect of this simple snippet is that even if we stopped the current reception, soon or later the `rx_complete` handler will be executed the same, as it is not actually connected with the decoding activity but it's an event that will be triggered when the reception should have been completed. For this reason we have to remember somewhere that reception that was started with this PLCP (hence with the execution of the current `rx_plcp`) was stopped: this is really important otherwise the firmware will pass to the kernel inconsistent data. Add a variable `RX_STOPPED` that is cleaned each time `rx_plcp` is executed, set when jam is transmitted, and checked by handler `rx_complete`. In `rx_complete` add label `force_no_rx` before the following two instructions

```
force_no_rx:
        orxh    RX_ERROR, SPR_BRC & ~ RX_ERROR, SPR_BRC
        orxh    0x0000, SPR_BRC & ~ NEED_RESPONSEFR, SPR_BRC
```

and jump there when the reception was stopped because of jamming (`RX_STOPPED != 0`). Remember also to clean that variable.

NOTE: remember to jam only the specific frames that passed the filtering code and add some instruction to allow the user to enable/disable jamming from the user space (add a variable `JAMMING_ENABLED` in shared memory that is checked by the jamming firmware before jamming and that you can modify from the userspace using `writeshm` tool).

*b.* *Schedule the jam signal for immediate transmission* We should be already able to do this: it was the topic of a previous lab tutorial (for immediate transmission refer to step 5 of document "2015-trento-lab5.pdf"). Place the code in `rx_plcp` after that added at the previous step: in this piece of code remember also to setup the transmitter hardware (take a look at TME section in document "2015-trento-lab4.pdf" for any doubt about how to set up the plcp for the selected jam signal datarate and length). Make the length of the jamming signal fixed to `JAM_LENGTH = 20` bytes and datarate to 1Mb/s by properly setting `SPR_TXE0_PHY_CTL`, `SPR_TME_VAL0`, `SPR_TME_VAL2`, `SPR_TME_MASK0` and `SPR_TME_MASK2`. Remember in a variable `DO_JAM` that the transmission scheduled right now is a jam frame: this variable should be checked in section `tx_frame_now` right after this block of instructions:

```
no_param_update_needed:
        orxh    FRAME_BURST, SPR_BRC & ~ FRAME_BURST, SPR_BRC
        orxh    0x0010, SPR_IFS_CTL & ~0x0010, SPR_IFS_CTL
        orxh    0x0000, SPR_BRWK0 & ~0x0006, SPR_BRWK0
        orx     7, 8, 0x000, 0x000, SPR_TXE0_WM0
        orx     7, 8, 0x000, 0x000, SPR_TXE0_WM1

        [check DO_JAM!!]
```

If `DO_JAM` is set we should further configure the system by telling the tx engine to use the values previously set in the TME (`SPR_TXE0_VAL/MASK` registers) and transmit a frame of the required length `JAM_LENGTH`, i.e.,

```
        mov     0x3, SPR_TXE0_WM0
        mov     0, DO_JAM
        mov     0, SPR_TXE0_SELECT
        mov     0, SPR_TXE0_Template_TX_Pointer
        add     JAM_LENGTH, 2, SPR_TXE0_TX_COUNT
        mov     0x826, SPR_TXE0_SELECT
        jext    COND_TRUE, complete_tx
```

Remember to change the end of the `tx_complete` code for avoiding firmware crash (again refer to "2015-trento-lab5.pdf", specifically to the `tx_really_done` hack). Finally, add counters for counting the number of TBJ frames that have been filtered/found and the number of times the added section of `tx_frame_now` handler has been executed for the proper transmission of the jam: this should help you debugging the code.

2) **Deploy the tx/rx infrastructure** As the goal of this exercise is to characterize the efficiency of the jam process, we have to customize also the TBJ sender and its receiver (e.g., a station and its Access Point). We refer in the following to the configuration reported in Figure 1. Here we want to count how many of the TBJ frames transmitted by the TBJ station are received by the AP. To

this end we have to add a 16 bit counter `SENT_FRAMES` at the TBJ station that counts ONLY DATA frames, and similar counters `RECEIVED_FRAMES_OK` and `RECEIVED_FRAMES_KO` at the AP that count, respectively, the number of DATA frames correctly received and affected by errors.

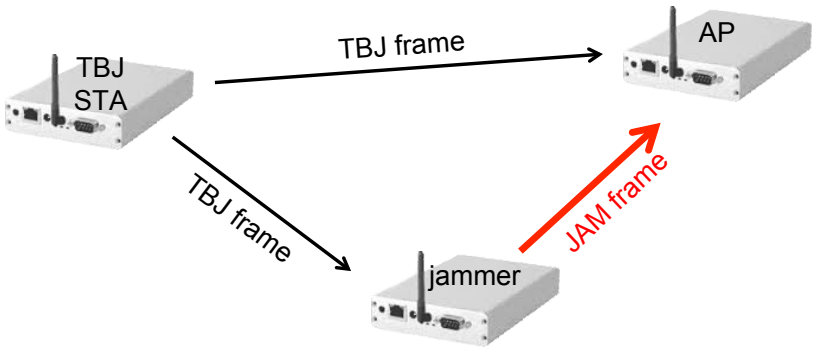


***Figure 1*** Deployment scenario for measuring jam efficiency

At the jammer node we similarly need a `JAMMED_FRAMES` counter. As we might need to clear such counters, implement them in shared memory so that we can easily reset them to zero using `writeshm` tool.

At the TBJ STA sender the counter must be placed in the `tx_data` handler and it should check the value of `[TXHDR_FCTL,off0]` that holds the first two bytes of the outgoing packet for checking the frame is data (look at the standard!): to avoid counting unnecessary frames, we should also set manually (and permanently) on the TBJ STA and AP the layer 2 address of the other peer, so that no ARP traffic is ever exchanged. To this end run on the TBJ STA:

```
$: sudo arp -s WLAN_IP_ADDRESS_OF_AP WLAN_MAC_ADDRESS_OF_AP
```

Run the same command on the AP (change addresses with those of the TBJ STA).

As other data frames can be transmitted in the lab building, the counter at the AP should be implemented with complex filters on the received frame by checking not only the frame control (must be data) but also the source MAC address that must be the one of the STA sender. This filtering code must be implemented at `rx_complete`, checking hence the frame control (data) and the source MAC address and the `COND_RX_FCS_GOOD` condition to understand if the frame is correct or not.

Similarly to TBJ STA and AP, we need a `JAMMED_FRAMES` counter also on the jammer node: we should add such counter right after the code that schedules a transmission.

3) **Measure the jam efficiency** Once the jamming system is implemented and we are sure the TBJ station and its corresponding AP are counting correctly, we are ready for starting the measurement experiments. We have first to validate that at least in one condition where the jammer as some advantage over the TBJ sender, jamming is closed to 100%. To this end simply try lowering the transmission power at the TBJ STA sender and check what happens.
We can now focus on quickly checking the jamming efficiency for different datarates of the TBJ STA sender. To this end use `iwconfig` to fix the datarate at the TBJ STA. It should turn out that

jamming DSSS at 1Mb/s is harder 11Mb/s, and similarly that OFDM 6Mb/s is harder than higher OFDM datarates. Try also what happens for increasing length of the jamming signal. Try different parameters and talk with the facilitator.