Exercise: Deploy a Frequency-Hopping Basic Service Set (FH-BSS)

1. Goals

Students must extend the standard DCF firmware and deploy FH-enabled nodes that keep jumping all together on the same sequence of channel. To this end they need to develop the following points:

1) add a mechanism in all nodes for computing the current channel from the current time;
2) add a simple soft-switch (e.g., a variable in shared memory) so the FH-BSS behavior can be enabled from the userspace;
3) define a methodology for measuring the impact of the channel hopping over the traffic transmitted in the BSS;
4) check how to use a third node for capturing the traffic exchanged by a couple of FH-BSS STA and AP.

Before continue *please note* that in the following we assume both the FH-AP and all the FH-stations are not hopping till the BSS setup is complete. For this reason we have to use the soft-switch as in point 2. Please also note that by embedding the FH-BSS capabilities in the firmware without any signaling between firmware and kernel, the kernel always consider as "current" channel the one initially set up with the BSS: we will not add any mechanism for letting the kernel know that the firmware is changing the channel.

2. Assignment steps

1) **Add a mechanism in all nodes for computing the current channel from the current time**
Whether or not hopping is active, we need first to define a mechanism so that all nodes in a BSS (including the AP and all STAs) can compute the same channel from their internal clock value. In a BSS, in fact, the clock of all stations matches that of the AP, distributed by the beacons. By creating a function chan = f(clock) we can make all nodes always tuned to the same channel. To this end students should first add a piece of code after the label `state_machine_start` that reads the current time, e.g.,

```
mov    SPR_TSF_WORD0, r62
mov    SPR_TSF_WORD1, r63
```

Now the 32 least significant bits of the BSS clock are in registers `r62` and `r63` and can be analyzed. The best approach to avoid nodes jumping to different channels, is to drop the N least significant bits, and use the 32 - N upper significant bits for assigning the channel: this obviously makes the hopping time a power of two number of microseconds but also makes the algorithm error proof.

Once the value of the channel (a number between 1 and 13, we avoid channel 14 for this experiment, why?) has been determined, it's time to compute the value of the frequency register. To this end remember the formula

```
freq_register_value = 0x8000 + (freq_in_ghz) - 2400
```

which reads

```
freq_register_value = 0x8000 + (2407 + 5 * channel) - 2400
```

Remember that 1) it is not possible to perform products but only additions/subtractions in the firmware 2) the formula cannot be embedded in the firmware "as is", some instructions should be used to compute the value.

Once the value has been computed it could be simply written in the hardware by executing these three instructions

```
mov    0x0008, REG34
mov    freq_register_value, REG35
call   lr0, write_phy_reg
```

If the channel is not changed since the last time it was written, the algorithm will simply overwrite the old value. This should be the snippet of new code:

```
state_machine_start:
        mov    SPR_TSF_WORD0, r62
        mov    SPR_TSF_WORD1, r63
        [code for computing the channel from r62, r63]
        [code for writing the value into the hardware]
```

NOTE: Remember to comment instruction `nap` at the very beginning of the code.

2) **Add a simple soft-swith for enabling/disabling FH functionalities from user-space** Before nodes in the FH-BSS can start hopping, they need to be joined to the FH-AP. To this end the last part of the code developed in the previous step (writing the channel value inside the hardware) should be executed only if enabled from user-space. To this end, simply define a variable in shared memory, so that its value would condition the execution of the channel refresh, e.g.,

```
#define FH_ENABLED [SHM(0xF00)]
        je     FH_ENABLED, 0, skip_fh
        [code for writing the value into the hardware]
skip_fh:
```

Then use the tool `writeshm` (execute it without arguments for getting instructions on how to use it) to set a value different than zero and enable the FH code.

Note: after the initialization, all SHM is set to zero. This means that the FH code will be disabled by default.

3) **Measuring the impact of the channel hopping mechanisms** Once everything is tested and working we have to quantify the amount of time that the system is losing because of the switching: during a switch, in fact, both the transmitter and the receiver are not usable (actually, they will try to tx/rx but the RF engine is tuning the main frequency, so they are rx/txing *rubbish* ☺. As channel switching is not instantaneous, this could lead to some losses. Another

inefficiency is when we switch to a very busy channel, as we keep doing look-around switch. This is mandatory but the total throughput can drop because of many neighbors.

4) **Check how to use a third node for capturing the traffic exchanged by a couple of FH-BSS STA and AP** A third node, external to the BSS is not going to copy the AP clock inside its own. In this way the same code developed for the AP and the STAs would not work for tuning the channel. Try to think to a solution to this problem (without explicitly joining the sniffer to the BSS!). Try discussing with the facilitator.