Internet of Things – A.Y. 2020/2021

Home Challenge #4 – Diego Savoia (Personal Code: 10535515)

To implement the code requested for this challenge, I used the "sendAckC" template and the example seen during the laboratory session ("RadioToss") as a reference.

In the header file ("sendAck.h") I defined the message structure, which contains the message type (nx_uint8_t: 1 if REQ and 2 if RESP), the counter (filled by mote #1 in each REQ and then by mote #2 in the RESP) and the fake sensor value (filled by mote #2 in the RESP).

The "sendAckAppC.nc" file contains all the useful components and the wiring. Apart from the standard *MainC* and *sendAckC* (renamed as *App* for simplicity) components, I also introduced the *Timer* component (to define the timer of mote #1), the usual ones for communication (*AMSenderC*, *AMReceiverC*, *ActiveMessageC*) and the one to generate the value from the fake sensor (*FakeSensorC*). The wiring is done as usual: in particular, I wired the *App.Read* interface to the *FakeSensorC* component, and the *PacketAcknowledgements* interface to the *AMSenderC* component.

Then, I implemented all the useful functions in the "sendAckAppC.nc" file, following the instructions in the template. After the motes boot, they turn their radio on, and only mote #1 starts its timer (with periodicity 1 second). Every time this timer fires, the function *sendReq* is called, in which mote #1 creates a new message, fills it with its current counter and the type REQ (=1) and sends it to mote #2 requesting an acknowledgement. Then, it increments its counter.

Whenever the sending is done, if the message was acknowledged, then mote #1 stops its timer; otherwise, it prints out "Packet not acknowledged" and goes on, sending again a message the next time the timer fires (one second later).

On the other hand, mote #2 has no timer itself. This is because it waits for a message from mote #1. Indeed, when mote #2 receives a message from mote #1 (and it is not malformed), it reads it (printing the content in the debug window) and, if the message type is REQ (=1), mote #2 saves the received counter in a variable "counter" and the function *sendResp* is executed.

This *sendResp* function calls the *read* function of the *Read* interface of the fake sensor. Therefore, a value is read from the fake sensor component, and the *readDone* function is executed. In this function, mote #2 creates a new message, fills it with the previously saved counter (the one received by mote #1), the type RESP (=2) and the value read from the fake sensor. Then, it sends the packet to mote #1, requesting an acknowledgement. The simulation is done.

Furthermore, I applied some additional modifications to the "RunSimulationScript.py" file in order to: save the simulation log in a "simulation.txt" file; add the proper debug channels I needed; set the starting time of mote #2 at 5 seconds (line 52). As requested, mote #1 starts instead at time 0.

The source files are: sendAck.h, sendAckC.nc, sendAckAppC.nc for the implementation of the motes; FakeSensorC.nc, FakeSensorP.nc for the implementation of the fake sensor (I did not modify these files); meyer-heavy.txt, topology.txt for the simulation (I did not modify these files); Makefile and RunSimulationScript.py for the compiling and the simulation. The complete log of the simulation (simulation.txt) is included in the .zip file.

The final value that mote #2 sends to mote #1 is 50421 (printed with %u; it would be 245 if printed with %hhu).