HOME ASSIGNMENT #2

IoT 19-20

Name Surname	Person Code	Id Number
Andrea Crivellin	10491856	928320
Gabriele Guelfi	10491169	916207

Link repo git:
https://github.com/GabrieleGuelfi/IoT_assignments2020.git

IMPLEMENTATION

We started from *sendAck.h*, defining the message struct that includes:

- **msg_type**: used to distinguish between a request(REQ) and a response(RESP)
- **msg_counter**:
  - in REQ it is a incremental counter for the number of message sent
  - in RESP it is the same counter to which it is replying
- **value**:
  - in REQ it is an empty field
  - in RESP it is the data provided by the sensor

In the *sendAckAppC.nc* file we defined and linked the components we needed:

- AMSenderC, AMReceiverC and ActiveMessageC for communication
- TimerMilliC to start the periodic timer of mote #1
- FakeSensorC to retrieve data from the sensor

We want to highlight that here we wired the PacketAcknowledgements interface to AMSenderC component in order to ack the message.

Now we'll describe the *sendAckC.nc* and the most relevant functions and events:

- **SplitControl.startDone**: mote #1 starts its periodic timer(1Hz)
- **MilliTimer.fired**: every time the timer fires, mote #1 send a request calling the function sendReq()
- **sendReq**: first it increases the counter and prepares the message, setting the *requestAck* flag for the packet, that means that the packet will use synchronous acknowledgement
- **AMSend.sendDone**: it checks if the packet is acked or not, if it *wasAcked* the mote #1 turn off its own timer(stops to send REQ, it waits for the RESP); mote

#2 instead does nothing, an acked packet means that RESP was received and simulation is done

- **Receive.receive**: the payload of the received packet is read: if it is a RESP it's just printed, if it is a REQ it saves the counter in the *rec_id* global variable and calls the *sendResp()* function
- **sendResp**: it just calls the *Read* interface of the sensor
- **Read.readDone**: it creates the messages, setting as *msg_counter* the *rec_id* previously saved and as *value* the data retrieved by the sensor. As in *sendReq* the *requestAck* flag is set

We also did error management, for example we check if a message is sent correctly, if not and it was a REQ we display the error, if it was a RESP we try to send it again.

SIMULATION

An entire log of the simulation can be found in the *log.txt* file.
We used a lot of debug statements to clearly understand the behaviour of the motes. First it can be observed the mote #1 is booted after zero seconds while mote #2 after five seconds, as requested by the specifications; this implies that the first five messages are not received, then the sixth, if no error occurred, is received by mote #2 that replies to it, ending the simulation.