CHALLENGE 4

Federico Romeo - 10566536
Simone Sarti - 10580595

TinyOS and TOSSIM

The personal code we used is 10566536, so our X is 7 and Y is 66.

The TOSSIM RunSimulationScript.py was modified by changing the activation time of mote2 to Y [sec] and adding “message” (showing how messages are put together) and “timer” debug channels. We used the given topology.txt and meyer-heavy.txt files for our topology and noises. Also the fake sensors were not touched. Our message structure, which is composed of the fields **counter**, **value** and **msg_type** is defined in the ch4.h file. Components are wired together in file ch4AppC.nc, the Makefile was adapted to use it. Here is a description of what the various functions do in the ch4C.nc file:

- **sendPacket()**: a function that is used to create a packet given as parameters the message type and the value to send, used to avoid code repetition in sendReq() and sendResp(), both call this function. All packets created by this function are marked with the requestAck flag.The unicast address is simply the moteID of the other node.
- **sendReq()**: calls sendPacket with value 0 and REQ type message
- **sendResp()**: calls Read.read(), which gets a value from the fake sensor
- **Read.readDone()**: triggered when the value from the fake senor has been read, calls sendPacket with message of type RESP and the value read from the fake sensor
- **Boot.booted()**: boots the mote at the time specified in the python script, and turns on its radio
- **SplitControl.startDone()**: triggered when the radio is started, in case we are mote1 we start a timer of 1 second. Mote2 just replies to requests, it doesn’t need the timer.
- **SplitControl.stopDone()**: triggered when the radio is turned off, does nothing
- **MilliTimer.fired()**: works only on mote1, when the timer fires we increment the counter and then invoke sendReq, unless the radio is still locked from the previously sent message
- **AMSend.sendDone()**: checks that the message was correctly sent, and in case unlocks the radio. Then it checks if the ack was received. If no, we re-send the packet (for mote1 we wait for the next timer fire event). If yes, and if we are on mote1, we increment the counter of acks received, and when we receive the Xth we stop the timer, so no more requests will be done.
- **Receive.receive()**: event triggered when a mote receives a packet. If the message is a REQ message, we set the mote counter to the received counter value and call sendResp to return a proper response message.

The result of the simulation can be found in the simulation.txt file. Motes seem to behave as expected, also when the ack is not received and a retransmission is needed. The TinyOS timer.fires() event fires a little bit earlier than it should, therefore mote2 receives Y+2 (68) as first counter instead of Y (66). After 7 steps, we receive the last sensor read at step 74 (68→ 74)