#04: TinyOS

Andrea Calici, C. Persona 10490117, Matricola 944717

Starting from the *SendAck* template, I implemented this version for the challenge. In the following list I'll describe the main components.

sendAck.h: the message contains 3 main fields; the **type** represents the typology of the message (REQ, RESP), the **counter** represents the number of the message sent by mote 1 and the **value** represents the data read by mote 2 and sent to mote 1.

sendAckAppC.h: this element allows to connect components and interfaces. The new links added that are differents from the previous examples seen are the one between `PacketAcknowledgements` and `AMSenderC`, and the one between `Read` and `FakeSensorC`.

sendAckC.h: this element implements the code of the application. When the application is booted the Radio is activated and on mote 1 a timer starts to send messages every 1s. Each message contains the type == REQ, an incremental counter that represents the number of the message sent (1 for the 1st message, 2 for the 2nd message, ...) and needs an ack implemented using the `PacketAcknowledgements.requestAck` function.
When the mote 2 starts at time 5, it receives the message from mote 1, sends and ack using the method of `PacketAcknowledgements`, reads a value from the `FakeSensor` and sends a response. When mote 1 receives the first ack, it stops to send messages and wait for the response.
For each phase of the flow using the `dbg()` function I printed the messages of the 2 motes that can be read in the log.

FakeSensorC.nc: it represents the fake sensor from which the mote 2 read a temperature to send it to node 1.

RunSimulationScript.py: starting form the basic script I added the debug channels used in my application (timer and packet) and I changed the starting time of the 2 motes (0 for the 1st and 5 for the 2nd).