Name: FEDERICO
Surname: DI GIUSTO
Matricola: 10693473
Github: https://github.com/Fededgs/FinalProject
ThingSpeak: https://thingspeak.com/channels/1119275

# 1. Approach

The general approach on the project has been to implement step-by-step each point required. Regarding the ACK mechanism, the retransmission and the duplicates detected, the topology used for testing was upgraded in time (for instance 1 node,1 gtw,1 server, then 1 node 2 gtw 1 server etc.). Notice that in the logs file the multiple topologies are tested. At the end, the part of NodeRed and Thingspeak has been implemented in a standard way.

In README.md in github, there is the explanations of every single logfile.

# 2. Fundamentals of the implementation: Struct of msg

To better understanding the implementation of the following parts, it is a must to present the fundamentals of the codes.

```
4  typedef nx_struct radio_count_msg {
5      nx_uint8_t msg_type;
6      nx_uint16_t value;
7      nx_uint16_t nodeid;
8      nx_uint16_t gateway;
9      nx_uint16_t count;
10
11 } radio_count_msg_t;
```

*Figure 1: Struct composition of the message exchanged*

In the figure is reported the structure of the unique message type exchanged in the project. This message is used for two type of packet (*msg_type*): DATA and ACK type. DATA type is the message sent by the sensor nodes setting the fields: *value*, *nodeid*, *count* (and *msg_type*). The DATA message is forwarded by the gateway. The gateway set the field *gateway*, that is need by the Server node to send back the ACK message. The ACK message created and sent by the Server (with all fields except of *value*, to simulate a 'real' ACK that must be light in bytes). The field *count* is needed by the Server to check possible duplication of the same Packet.

# 3. Implementing the main parts

## a. Topology

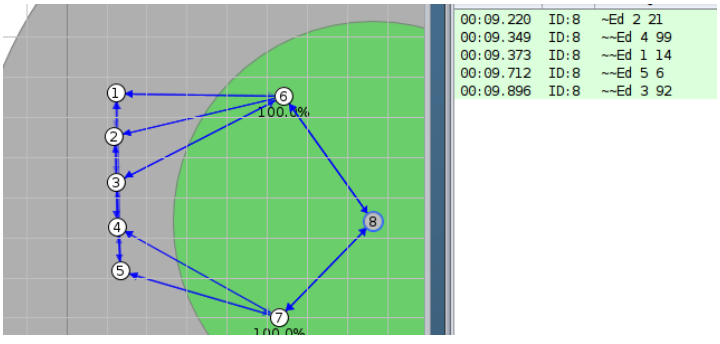



*Figure 2: Topology of the Network. On the top right, the printf used by NodeRed*

The 5 sensor nodes, on the left, send messages in a broadcasting way (using *AM_BROADCAST_ADDR*), instead the GATEWAY and the SERVER use only a one-to-one sending type. The sensor nodes can receive only Packet of type ACK. The sensor nodes use the Random interface to generate the value of their fictious sensor.

In the simulation run by the script of python, this topology is reported in the file *topology.txt* with the correspondent power dbm. Multiple topologies were tested, not only the one reported to the project explanation.

## b. Retransmission by sensor node

For implementing the retransmission of the packet, if no ack was received in less then 1000ms, it is used a MilliTimer ('*MillitimerACK*') in a '*OneShot*' way. In particular, this milli timer is started when the sensor node sends the Packet for the first time ('*call MillitimerACK.startOneShot(PERIOD_ACK)*'). If the Millitimer is not stopped by the sensor node, it fires, and it re-sends the same packet stored in the pointer *&packet.*
If the Millitimer is resending, for the second time for instance, it recalls itself the '*startOneShot*' function.

## c. ACK exchange

As it said before, we use the same type of struct of message for ACK and DATA packet, but they are distinguished by the field 'msg_type' that must be set.

The first thing needed to the ACK to do the same path the package did before, is to know, obviously, the path. For this issue, the Packet stores the 'nodeid' field, referring to the sensor node source and the 'gateway' field, that is set by the gateway before sending to the Server Node.
At this point, when a Packet DATA is received by the Server Node, it creates a new PACKET of type ACK, with the same fields apart from *'value'* field (this is for a dimension of the packet reason and also because is not needed by the ACK). Then, the Server node sends it to the corresponding Gateway (*'msg->gateway'*). In turn, the gateway sends the same packet to the node ('*msg->nodeid*'). At this point, the sensor node check if 'msg->count' is the same stored locally in itself ('*counter*'). If it is, the *MillitimerACK* is stopped.

See the logfiles:

- logfile/logfile_trasmission_OK_single_node.txt
- logfile/logfile_trasmission_KO_single_node.txt.

(In this logfile, node1=sensor node, node4=gateway, node5=server)

## d. Duplicated Packet Detection

In the scenario in which both gateways receive the same Packet DATA from the same sensor node, the Server Node can receive a duplicate of a Packet.

To face this problem, the Packet must be identifiable. The solution developed, is to identify the single PACKET DATA of a sensor node, by the fields 'nodeid' and 'count'. 'nodeid' is written by the macro 'TOS_NODE_ID' instead 'count' is written by the local variable of the node (that is updated every time is sent a Packet and its ACK is received).

From the Server point of view, it must be store, for each sensor, the last 'count' received. This was implemented by storing locally to the Server, an array of length equal to 5 (see in file 'FinalProjectC.nc' in function 'receive.receive' ).

If the packet received by the Server is not a duplicate, it sends back the ACK, otherwise no action is done. In addition, if it's not a duplicate, it is run the *'printf'* so in NodeRed, the are not multiple 'printf' due to duplicates.

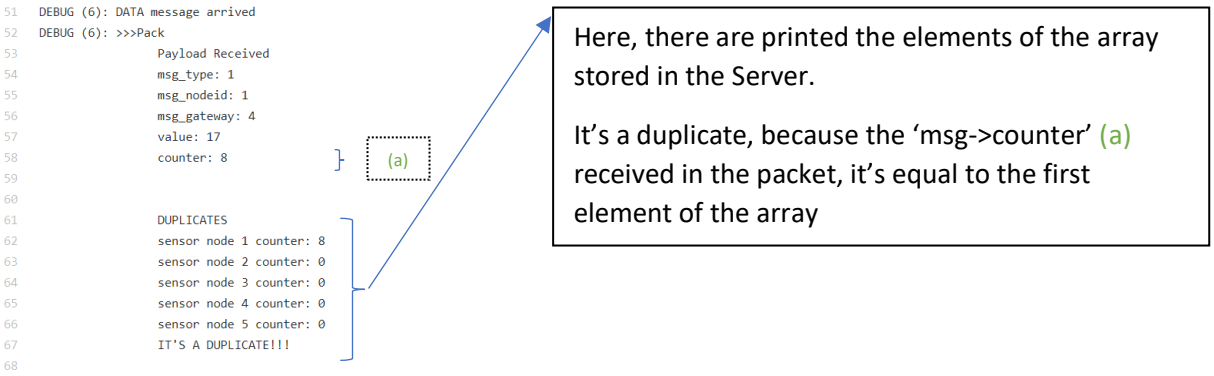


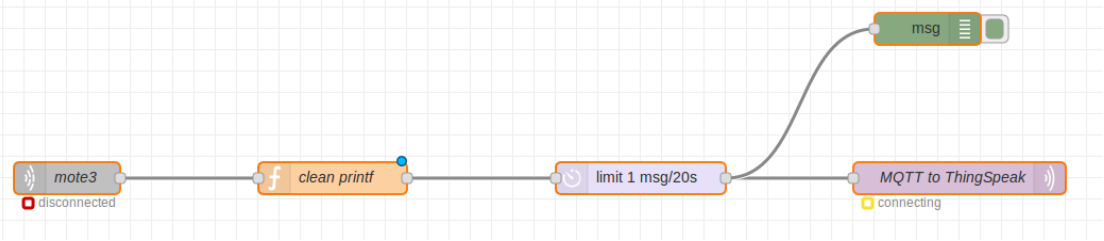

*Figure 3: Output of the array stored in the Server. It is used to check duplicates*

See the logfile:

- /logfile/ duplicate_detected.txt

## e. NodeRed and Thingspeak (through printf)

Node-red has been linked to the Server node in Cooja through tcp port, with the stream of String generated by the 'printf' of the codes. The 'printf' output can be seen in *Figure 1.*




In the script 'clean printf' we extrapolate the field 'value' and 'nodeid' from the printf of the Server Node. Then, it is set 'msg.payload' and 'msg.topic' according to the standard. The messages are limited in time according to the ThingSpeak requirement and the MQTT Connection is established according to the standards.

In ThingSpeak there are 3 charts (referred to values of nodeid=1,2,3) and a gauge linked to the second chart.

## f. Testing and debugging

For this part, it was used the script of python (see 'RunSimulationScript.py') compiling the code with 'make micaz sim'. The python script has a standard structure: there are used multiple channels for debugging and multiple topology has been tested (see the folder '/logfile').