**IoT challenge 3**

**Marco Barbieri 10684602 (leader), Matteo Bianchi 10698168**

# Message struct

To handle the messages we used a struct with 4 fields: type, sender, destination and value.

* the type field indicates wether the message is data (type=0), route request (type=1) or route reply (type=2).
* the sender field is used by data and the reply messages to indicate who sent the message.
* The destination field is used by the data message to indicate whom the message is destined to, and by route request/reply messages to indicate the requested node.
* The value field is used by the data message to indicate the payload of the message and by the route reply to indicate the cost of the route.

# Routing table

To handle the routing table we used an array of struct of type routing\_table\_entry\_t, this struct contains the next hop and the cost of the route. The destination is implicitly defined by the position in the array of the routing table. The routing table is then an array of 7 elements.

# Send message

To handle the communication we used the components activeMessageC, AMSenderC and AMReceiverC.

To send the message we just used the function send of the AMSenderC component like we saw in the lectures of the lab, we added such function in the actual\_send function provided.

# Receive message

To receive the messages we used three buffers, one for the request messages, one for the reply messages and one for the data messages. In this way we didn’t have problems when we received multiple messages of different types while elaborating the messages just received.

When we receive a message we toggle the correct led as specified in the slides and then call functions to handle the different types of messages.

* Handle\_data\_packet retrieves the destination of the packet, If we are the destination node we print a success message. If we need to relay this message, we check the entry of the routing table related to the destination of the packet and, if the next hop is 0 (default value to indicate an empty entry), we send a route request packet in broadcast and save the data message in a separate buffer.

If the next hop is different then 0, we simply relay the message as It is to the next hop of the route.

* Handle\_route\_request retrieves the destination of the packet and if we are the destination, we send a reply route message in broadcast. If we are not the destination node, we need to check our routing table, if the entry related to the destination of the packet is found, we can send a route reply in broadcast containing the cost of the entry + 1. Otherwise we relay the request route message in broadcast.
* Handle\_route\_reply checks the destination of the route reply packet and, if we are the destination we are not going to do anything. If we are not the destination, we check if we need to update our routing table and we relay route reply message with the cost of the route incremented.

to see if we need to update out routing table, we check the existence of an entry related to the destination (an entry exists if next hop != 0) and we check wether the cost indicated in the packet is less then the cost saved in the entry. In this way f the cost of the packet is lower, we found a shorter path that we prefer.

# Start

At the start of the simulation, we initialize the routing table as next hop = 0, that is because no node has id 0, so we know that 0 means and empty entry. We also set the cost of each entry to 8 because the diameter of the network is clearly less that 8 because there are 7 nodes. After the start of the radio we set timer1 to one shot after 5000ms.

When the timer1 fires, we create the data packet and because the entry related to node 7 is empty, we save this packet and send a route request in broadcast.