IoT 2023 CHALLENGE 3

(1) Name: Alessandrini Luca Person Code: 10569363

(2) Name: Renzi Flavio Person Code: 10659808

File RadioRoute.h

We opted for an `enum` in order to store the message types.
Moreover, we decided to re-use differently the same fields for accordingly to message type, hence the struct `radio_route_msg` is defined and used as follow:

		Fields of the struct			
		type	**src**	**dest**	**data**
Message Type	**Data message**	`DATA_MSG`	Sender id	Receiver id	data
	Request	`ROUTE_REQ_MSG`	0	0	Requested id
	Reply	`ROUTE_REPLY_MSG`	Sender id[1]	Requested id[2]	cost

In this file is also present the struct for the routing table (local for each node), containing a filed for the next hop, and a field for the cost.

File RadioRouteAppC.nc

In this file it is possible to find the components we used and the interfaces wiring.

File topology.txt

In this file can be found the declaration of the topology of the given network.

File RunSimulationScript.py

The file has been modified by adding the necessary debug channels and print information, plus the part for the creation of the nodes.

File RadioRouteC.nc

- `Boot.booted():` in this event we initialized the routing table of each node of the network. We chose the value 0 for the initialization of the cost for each entry of the table: we opted for this choice since every node is at least at distance 1 from every other node, and in case a

[1] The node that is broadcasting the reply. This is needed to update the next hop in the routing table

[2] The node requested in the route request that generated the reply

node receives a packet directed to itself, it does not re-route it (i.e., it does not access the routing table).

- `Timer1.fired()`: at this point, we check whether the id of the current node is the id of node 1. In the case it is, then we check if there it is an indication in the routing table for node 7: if there it is not, then we broadcast a route request message to discover the next hop, otherwise it means that our table has an indication, and we can send the data packet to the next hop. In the case we do not have the information in the routing table, we start another time the `Timer1` as `Timer1.startOneShot(250)`. This allows us to check every 250 ms if the table has been filled, and we will repeat this process. Notice that before sending, nodes will check whether they have already sent a request: this fact allows to not broadcast route requests every 250 ms.
- `Receive.receive(message_t* bufPtr, void* payload, uint8_t len)`: when a node receives a message, we perform the led toggling by calling the `play_led()` function, and then we enter a switch case according to the type of the message.
    - Data message: if the packet is for the node that receives the message, then it confirms the reception. If it is not for it, then it will check in the routing table, and send to the next hop. If it is not present an entry in the routing table for the next hop, then that packet will be discarded as a suggested simplification, since according to the structure of the challenge, this case should never happen.
    - Route request message: as specified in the delivery, if the packet is directed to the node that receives the message, then that node will broadcast a route reply (as always, if not done before). If it is for another node, of which I do not have any information in the routing table, then the node that receives the message of route request, will broadcast that route request. If the node receiving the messages has an information in the routing table for that node, it will send in broadcast that information as a route reply with the cost increased by one.
    - Route Reply message: In case a node receives a route reply, if it is not about itself and it has not that information in the routing table, or if it has that information but with an higher cost, it updates the entry and tries to broadcast the route reply with updated information about cost (increased by 1) and putting itself as source.
- `play_led()`: performs the computations requested in the delivery and toggles the correct led. At the end, we increment an index, and we perform the modulo 8 operation to consider digits of the person code in a round robin cycle.
- `actual_send(uint16_t address, message_t* packet)`: checks the `locked` bool, and if it is not setted calls the `AMSend.send([...])` and sets `locked` to true.
- `AMSend.sendDone(message_t* bufPtr, error_t error)`: checks if the message passed through, and unset the boolean variable `locked`.
- `AMControl.startDone(error_t err)`: check if the radio started successfully, and at this point, we start the radio and each node starts `Timer1` as `call Timer1.startOneShot(5000)`, in order to wait the five seconds requested in the delivery.