Challenge 3 - Report

Students

- Francesco Scandale - 10616610
- Angelo Capponcelli - 10653805

Code report

topology.txt

This file is used to define all the nodes inside the network and their connections: it was completed with all the connections of the given network using -60.0 dB. In total there were 18 connections (e.g. 1 2 -60.0, 2 1 -60.0, …).

RunSimulationScript.py

Being the actual simulation script, here many modifications were necessary:

- Both the topology file and the noise file were included
- All the debug output channels necessary for the application were added
    - init, boot, timer, led_0, led_1, led_2, radio, radio_send, radio_rec, radio_pack
- The seven nodes were created, giving 0 as starting time and setting the radio channels (using topology.txt) and adding the noise

RadioRoute.h

This file is used to define the data structures used in the program:

- radio_route_msg_t: struct used to contain the data needed in the messages
    - type: message type (0 (data), 1 (route request), 2 (route reply))
    - src: source of the message, needed in types 0 and 2
    - dst: final destination of the message, needed in all types (for types 1 and 2, it represents the field “node requested”)
    - value: data exchanged, needed in types 0 and 1 (in type 1 it represents the field “cost”)
- routing_table_entry: struct used to represent an entry of the routing table
    - dst: final destination
    - next_hop: next node to go to in order to reach dst
    - cost: cost (i.e. how many steps) needed to reach dst

RadioRouteAppC.nc

File used for the components declaration. Components used:

- MainC, RadioRouteC: components needed to activate and manage the simulation
- LedsC: used to show the receiving of messages in each node
- Timer0, Timer1: timers used in the simulation to manage different events

- ActiveMessageC, AMSenderC, AMReceiverC: components used to manage the messages, their sending and their receival.

They are then mapped to the interfaces used in the implementation file.

RadioRouteC.nc

File that actually explains how the simulation runs. Since this is an event-based paradigm, it boils down to the declaration and implementation of the needed event functions (along with some useful ones).
Initial definitions and “module” section:

- MAX_NODES for the cycles and PERSON_CODE for the leds
- Interfaces: Boot, Receive, AMSend, SpliControl (AMControl), Packet, Timer and Leds

“Implementation” section:

- Variables declaration
  - **packet**, **queued_packet**: used to build the packet to send and to actually send the packet
  - **queue_addr**: address of queue_packet
  - **time_delays**: array of times used to delay the sending of a packet
  - **route_req_sent**, **route_rep_sent**: flags used to check if the node has already sent or received a message (assumption: only 1 req and 1 reply)
  - **data_sent**: flag used to check if the data (message of type 0) has already been sent
  - **locked**: flag to lock the output radio
  - **routing_table**: array where each node stores its routing table
  - **destination**: final goal of the message
  - **person_code**: current value of the person code (used to keep track of the leds to change)
  - **j**: iteration variable used to handle the led changes
- Functions:
  - **Boot.booted**: event fired after the application is booted; launches AMControl.start to enable the radio
  - **AMControl.startDone**: if the radio is started successfully, node 1 launches Timer1 with a time of 5 seconds, otherwise re-launches AMControl.start
  - **Timer1.fired**: checks if the radio is locked or request message already sent, if not then “packet” is built and msg is retrieved as the payload. After setting the necessary fields (message type and destination), it launches generate_send.
  - **generate_send**: this function is used to trigger Timer0 (which then sends the packet). It checks if the timer is already running (in which case it aborts) and, if it isn’t, checks the type of the packet: based on the type it sets the corresponding flag (for type 0 we set it separately), starts the timer based on which node we are on and sets the necessary variables to send the packet.
  - **Timer0.fired**: when timer ends, calls the function actual_send, which performs the sending of the message (done this way for scheduling purposes).
  - **actual_send**: starts sending the packets to all the necessary addresses (usually 1 address or broadcast) and locks the radio by setting the flag.
  - **AMSend.sendDone**: if the sending of the packet was completed successfully, it unlocks the radio and shows a message. If the message was not

successfully sent, it unlocks the radio to allow the remainder of the application to keep running, but shows an error message.

- **Receive.receive**: when a packet is received, this function is launched.
    - After checking if the packet was formatted correctly (using its length), it gets the payload. Based on the message type, 3 cases are identified:
        - **0 - data message**: needs to be sent along the routing table (only the final destination doesn't and just reads it)
            - Check of the routing table: if the destination is found the search stops and a flag is updated, otherwise it stops at the first available slot in the routing table (which corresponds to the first place whose cost is 0); either way the search stops leaving "i" as the index in the routing table to be used
            - If the entry is found, a new data message is built, fields are set (type = 0, source = current node, destination = received destination, value = value from received message)
            - If the entry is not found but the current node is the destination, the message is read
        - **1 - route request message**: needs to be broadcasted (only the final destination doesn't and instead sends the first route reply)
            - If the destination is the current node start sending route replies; a new message is built, fields are set (type = 2, source = local node, destination (i.e. "node requested") = received destination, value = 1)
            - If the destination is not the current node, check if it's in the routing table (*see first bullet point of type 0 messages*)
                - If found start sending route replies; a new reply message is built and fields are set (type = 2, source = local node, destination (i.e. "node requested") = received destination, value = 1)
                - If not found, broadcast a new request message (type = 1, destination (i.e. node requested) = received destination)
        - **2 - route reply message**: needs to be broadcasted (only the final destination (i.e. node 1) doesn't and instead sends the first data message)
            - Check if it's in the routing table (*see first bullet point of type 0 messages*)
            - If not found or found and the cost in the routing table is higher than the received cost (i.e. value), set the fields in the routing table
                - If current node is node 1, the destination of the routing table is the final destination and a data message has not been sent, a new data

message is built and fields are set (type = 0, source = current node, destination = final destination, value = 5); the message is then sent to the next-hop node
        - Otherwise a new reply message is built and fields are set (type = 2, source = current node, destination = final destination, value = received value + 1)
- Leds are then updated:
    - If “person_code” is 0, j is updated with the order of magnitude of PERSON_CODE
    - Based on the current leftmost digit of person_code the leds are updated (launching on led X the corresponding Leds.ledXToggle function)
    - “person_code” is updated to delete the leftmost digit

Messages in the log

- It prints all the messages for the creations and activations, the setting of the radio channels and the noise.
- Message sending: each time a message is sent (or broadcasted) from a node, that node prints a message saying at what time the packet is sent
- Message receiving: the node prints the receival time and the received packet, as well as the new status of the leds.
- Led status messages: when a message is received it is shown which led is toggled and what is the new status.