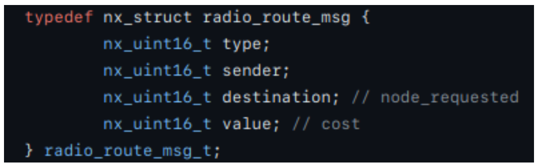
# IoT 2023 Challenge 3

Pasquale Castiglione
`10657816`

Lorenzo Campana
`10605775`

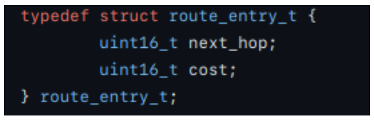
## RadioRoute.h

In the header file, the data structures for the messages and for the routing table were defined.



```
typedef nx_struct radio_route_msg {
        nx_uint16_t type;
        nx_uint16_t sender;
        nx_uint16_t destination; // node_requested
        nx_uint16_t value; // cost
} radio_route_msg_t;
```

Figure 1: Message data structure



```
typedef struct route_entry_t {
        uint16_t next_hop;
        uint16_t cost;
} route_entry_t;
```

Figure 2: Entry of the routing table

## RadioRouteC.nc

### Variables

- `waiting_packet`: Stores a packet that is waiting for a routing table update before it can be sent. It allows the code to delay the sending of the packet until a valid route is obtained.
- `waiting_address`: Stores the destination address of a packet that is waiting for a routing table update before it can be sent.

- `locked:` Controls the sending of packets to avoid overlapping transmissions. When a packet needs to be sent, the “locked” variable is checked. If it is set to “TRUE”, it means that a packet transmission is already in progress, and the sending is aborted. It is later set back to “FALSE” when the sending is completed.
- `msg_counter:` Keeps track of the number of messages received by a node.
- `codice_persona:` Array containg the codice persona of the leader.

## Functions

- `initializeRoutingTable:` Initializes the routing table.
- `getRoutingEntry:` Retrieves a routing entry from a routing table based on a given destination address
- `addRoutingEntry:` Adds or updates an entry to the routing table.
- `dbgPacketInfo:` Prints debug information about packets.

## Boot

At the boot of a node, the routing table is created and the radio module is started. When the radio module has finished the process of starting, a five seconds timer is started for the node ‘1’.

## Send Event

The sending of a message is implemented in the `actual_send` function. Before the sending of a message, the variable `locked` is evaluated in order to check if a packet is already being sent or not. If the module is not locked, it proceeds with sending the packet. It first checks the routing table to determine the next hop for the packet’s destination. If a routing entry is found, it uses the next hop as the address for sending. If no routing entry is found, it uses the provided address parameter as the destination address. In case of data message with null entry in the routing table, `waiting_packet` and `waiting_address` are set and a routing request is sent in broadcast.

### Receive Event

Received messages are handled according to their type:

- **Data Message**
  Checks if the message is intended for the current node. If not, it checks if a route to the destination is present in its routing table. If a route is present then it forwards the message to the next hop.

- **Route Request**
  If the receiver node is the intended destination of a route request or if the node has a non empty entry in the routing table, then it answers back sending a `Route Reply` broadcast message. Otherwise it broadcast again the route request message to its neighbors.

- **Route Reply**
  When a `Route Reply` is received the nodes check their routing table and they update them if the received cost for the destination is lower than the stored one. Also, the nodes check weather there is a `waiting_packet` with a `waiting_address` equals to the updated entry destination address. In this case, it proceeds with the packet forwarding to the updated next hop.

## Led Control

The LED index is calculated based on the `msg_counter` variable and the `codice_persona` array. The `msg_counter` is incremented each time a packet is received, and it is used to select the digit in a round-robin cycle from the `codice_persona` array. The digit is then used to determine the LED index by taking its modulo with 3. At this point the led status is toggled using `Leds.ledToggle()`