IoT 2023 Project 1

(1) Name: Amor Madkhour
(2) Name: Tommaso Giorgeschi

TinyOS

topology.txt

In this file we have defined the topology of the network as it is required in project 1 requirements (node 9 is the PanC).

RadioToss.h

The struct radio_route_msg is used to define the message formats with the following attributes:

- **type** indicates the type of message, that can be either:
  - 0 (Connect messages),
  - 1 (Connack messages),
  - 2 (Subscription Messages),
  - 3 (SubAck Messages),
  - 4 (Publish Messages),
- **sender** it is the sender of the message
- **destination** indicates the destination
- **topic** it is the topic of the message ,absent in the Connect and Connack message and it can be 0 for Temperature, 1 per Humidity and 2 per Luminosity
- **payload** is the value of the message used only in the pub Message.
- **acked** indicate if the connection phase is ok;

RadioTossAppC.nc

We have defined the components useful for our application: Timer0, Timer1 and Timer2 for the timers, the AMSenderC, AMReceiverC, ActiveMessageC,RandomC,SerialPrintfC and SerialStartC;
Then we have wired the components with the respective interfaces of the application.

RadioTossC.nc

We first have defined three structures used within the panC to create a routing table of the connected node, to send back the publish message to node-red and the connected node. Every node that connects and subscribes properly to the panC is inserted within this table.
Then inside implementation we have defined the whole routing table as an array of eight elements and with the function initializeRoutingTable() we have initialized the routing table so that at the beginning every node has a routing table that contains 0 as nodeID (means that the node is not inserted yet), 0 as acked field that indicate that the node is not inserted yet and 10 as topic means that the node does not subscribe to any topic.
Then we have defined all the interfaces needed for the communication.
We have defined two lists needed to queue the pub message received from the panC and then in asynchronous way send to node red and to the subscribers the messages received.
We have defined two bool variables as flags that indicate to the nodes that the subscription phase and connect are done properly.

There are two functions (addObjectToSendToNodeRed and addObjectToSendToNode) that manage the push of the pub message received from the panC inside the two lists.
Then there are two functions (sendSub and sendConnect) that manage the sending of messages of type connect and subscription.

We have defined two actual_send functions (actual_send_to_node and actual_send_to_node_red) where we have implemented the logic to perform the actual send of the packet respectively to node red and locally to the subscriber : if the lists are not empty and I'm not already sending a message we send the pub message asynchronously to node-red and to the subscribers.

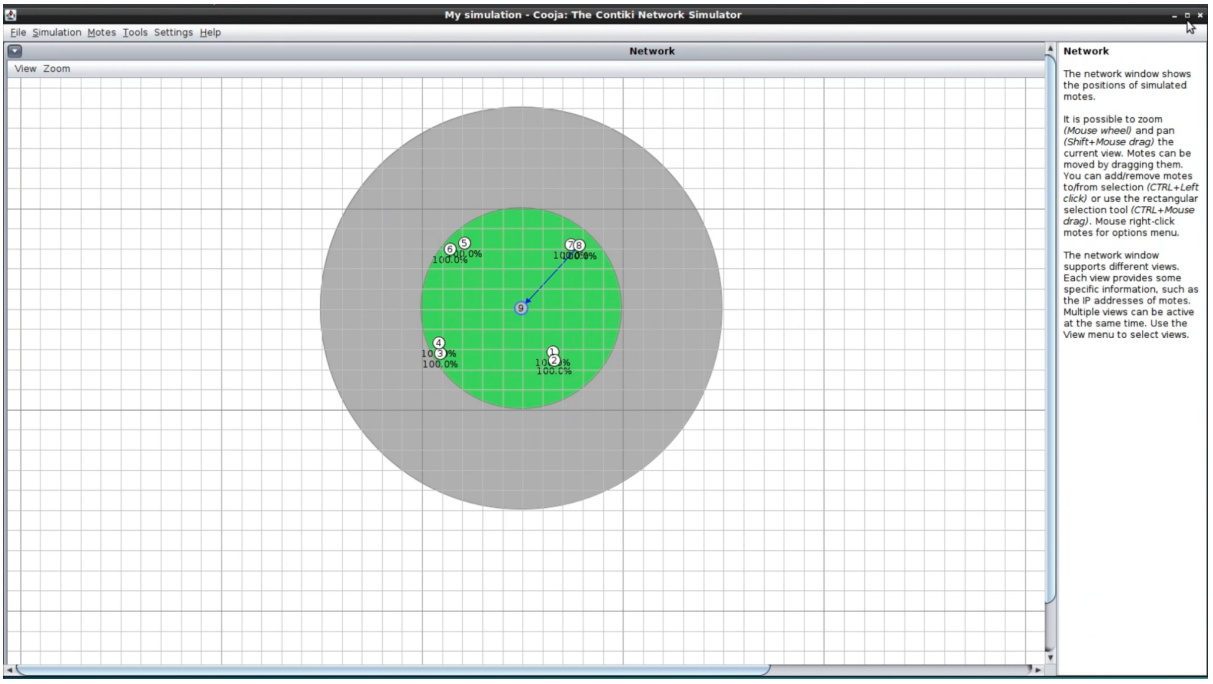
We have then made a Receive function to handle the reception of messages by the nodes:
so if the PanC receives a Connect message it first adds the node to its table and then sends back a Connack message. If the PanC receives a Sub message it first checks if the node is connected and then adds the sub info to its table and sends a Suback message back.
Until a node has received the Connack (or the Suback in case of a subscription) it performs the retransmission of the Con (or Sub) message thanks to the proper flags that indicate if the transmission of messages was correct or not.
Then if the PanC receives a pub message it adds the message to the lists for the forwarding to the nodes subscribed to that topic.
Instead if a node (that is not the PanC) receives the Connack it sends the subscription message to the PanC (we assumed that as soon as a node receives the connack it performs the subscription to a topic).
If a node receives a Suback it just sets the flag to true to stop the retransmission of sub messages.

RunSimulationScript.py

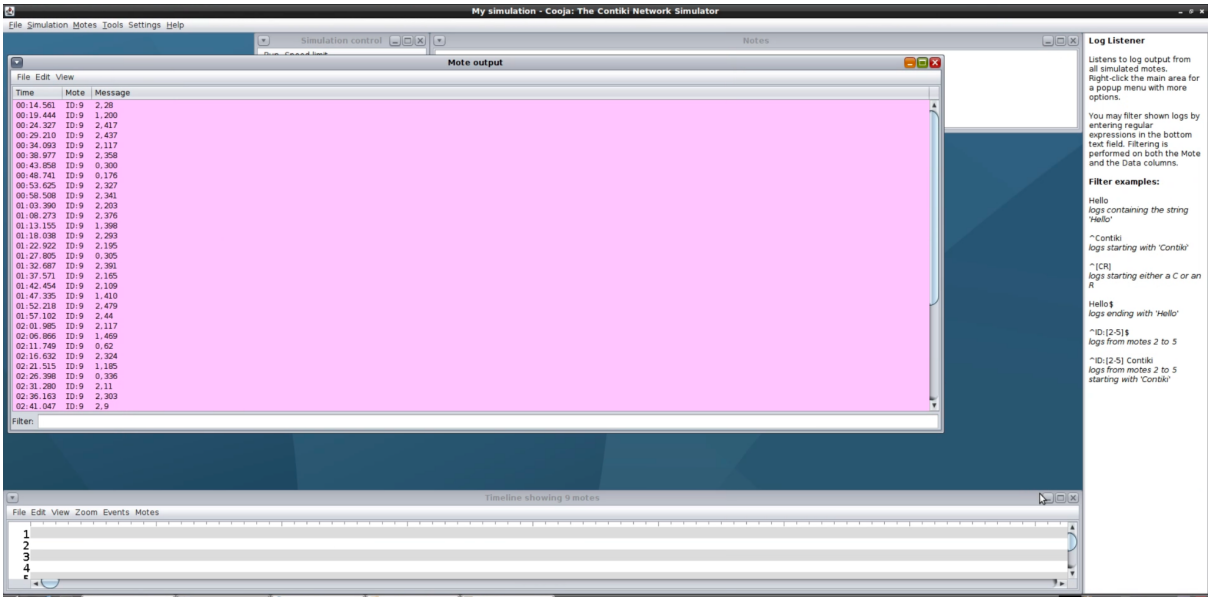
In this file we managed some useful debugging prints like the creation of every node and the table of the PanC.

COOJA

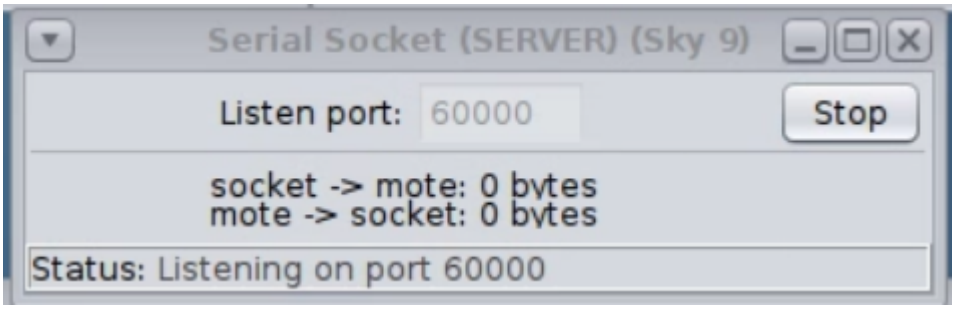
In the cooja environment we create a new simulation with nine motes and set them in a star topology. Then on the PanC(number 9) we set a serial socket (server) on port 60000 in order to send on this port the published message received. Then, after an initial phase when the node establishes a connection with the PanC, we start sending the publish message every 5 seconds to node-red.



Here we can see the topology.

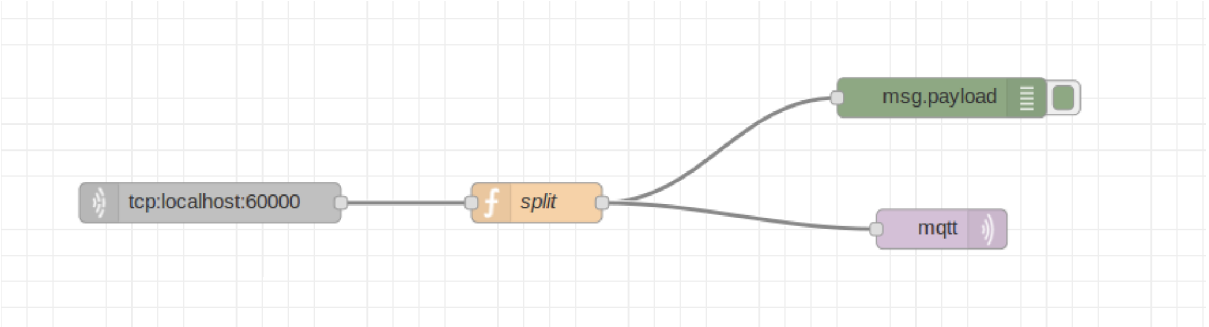


This is an example of the output we get.



This is the Socket setting.

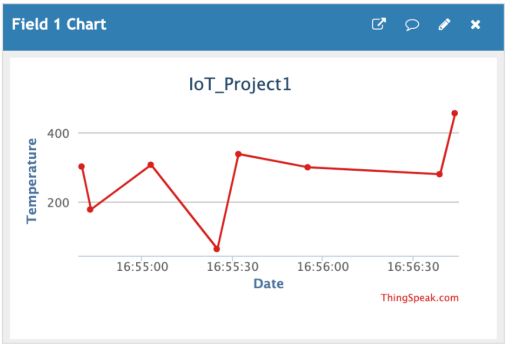
NODE-RED

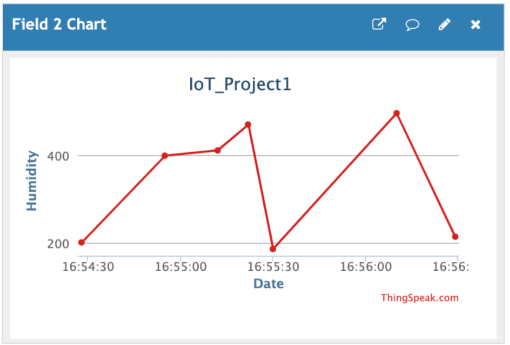


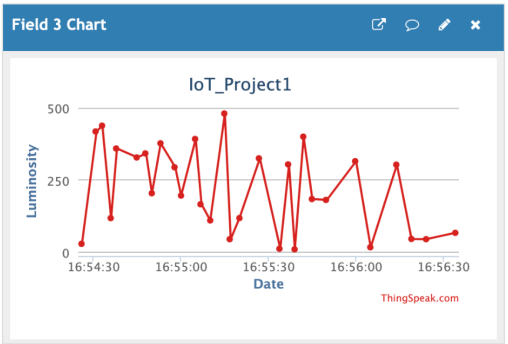
We have used a TCP node that receives messages with topic and payload from Cooja.
Then we have created a public channel on ThingSpeak with three charts, one for each kind of topic: first chart → topic 0 (temperature), second chart → topic 1 (humidity) and third chart → topic 3 (luminosity).
Then we have defined a split function that splits every message in two parts (topic and payload) and sends the payload value to one of the three ThingSpeak charts depending on the topic.

Finally we have defined a debug node for debugging and an MQTT output node to send the payload values to the ThingSpeak charts on the public channel.

A screenshot of the charts is the following:







The link to our public channel is: https://thingspeak.com/channels/2203093