**IoT Final Project**
**Project 1**
**Academic year**: 2022-2023
**Authors**: Andrea Carotti (10628385), Matteo Crugnola (10608406)

Files included:

- Node-RED code in file: `Node_Red_export.json`
- Simulation tested with TOSSIM in file: `RunSimulationScript.py`
- Topology in file: `topology.txt`
- TinyOS files:
  - `pub_sub_message.h`
  - `PubSub.nc`
  - `PubSubApp.nc`
  - Imports directory containing:
    - `LogicHandlerModule.nc`
    - `LogicHandlerModuleC.nc`
    - `OutQueueModule.nc`
    - `OutQueueModuleC.nc`
- Logging files:
  - `debug_files/simulation.log`
  - `debug_files/node_red_outfile.txt`
- Compiled micaz sim files: under simbuild folder
- Default TOSSIM files:
  - `TOSSIM.py`
  - `TOSSIM.pyc`
  - `_TOSSIMmodule.so`
  - `app.xml`
  - `meyer-heavy.txt`

Link to public Thingspeak simulation channel: https://thingspeak.com/channels/2232186

Architecture stack: TinyOS - TOSSIM - Node-Red

**pub_sub_message.h**
This file contains the content of the message sent by every node. We used the same struct for all the messages. Every message contains Type (that can be either 0, 1, 2, 3 or 4 corresponding to connect, connect-ack, subscribe, subscribe-ack and publish), senderId (the id of the node sending the message), Topic (that can be either 0,1,2 corresponding to Temperature, Humidity and Luminosity), Value (the content of the value read by the mote), and SubscribeTopic0, SubscribeTopic1, SubscribeTopic2 (these are boolean value indicating to which Topic the node is subscribing to).

With this message structure we can subscribe to more than one topic with a single subscribe message.

In this file we also define some useful constant values used in the other two files.

**PubSubApp.nc**
This file defines the wiring of the components with the interfaces used in this application.
In order to keep the code clean we created 2 custom modules:

- OutQueueModule to create a queue of outgoing messages when the radio is locked, this mainly happens when the pan coordinator needs to forward a received publish to all subscribed nodes of a topic.
- LogicHandlerModule to declare the events that will handle the received message logic, the final version of this module doesn't do much other than passing the parameters to the events defined in the PubApp.nc code.

**PubSub.nc**
This file contains the entire logic of the program.

***Main events and functions***
event void AMControl.startDone(error_t err)
If the radio was turned on correctly, the node checks whether the calling node is the pan. If it is NOT it starts the connect Timer (once triggered the node will send a connect message). If the node is the pan it starts the Periodic Update Timer (to send the latest acquired values to Node-RED)

void SendPacket(uint16_t address, message_t packet)
Tries to send the packet.It first checks whether the radio is locked. If it is, the message that the node was sending is pushed in the out queue. If it is not locked, the node locks the radio, and attempts to send the message. If any error in sending occurs, the message is pushed to the out queue.

event void AMSend.sendDone(message_t* buf, error_t error)
Once the packet is sent, the node unlocks the radio. If any error has occurred we push the message to the out queue, if we are not able to do so, the packet gets lost.

event void OutQueueModule.sendMessage(uint16_t destination_address, message_t packet)
Calls the SendPacket method provided the destination and the packet.

void sendConnectMessage()
Crafts the packet with a Connect message payload and calls the SendPacket, then it starts the checkConnectionTimer.

void sendConAckMessage(uint16_t clientId)
Crafts the packet with a ConAck message payload and calls the SendPacket.

void sendSubscribeMessage()
Crafts the packet with a Subscribe message payload and calls the SendPacket. Then it starts the checkSubscriptionTimer. With this message a node has a 60% chance to subscribe to each one of the topics.

void SendSubackMessage(uint16_t clientId)
Crafts the packet with a SubAck message payload and calls the SendPacket.

void sendPublishMessage()
Crafts the packet with a Publish message payload and calls the SendPacket. The Value of the Publish message is a random number between 0 and 99.

void forwardPublishMessage(uint16_t senderId, uint16_t topic, uint16_t value, uint16_t targetAddress)
Crafts the packet with a Publish message payload and calls the SendPacket. This method is used by the Pan Coordinator to forward the values of a received Publish to the subscribed nodes.

void updateConnectedClients(uint16_t clientId)
This function updates the list of the connected clients of the pan coordinator.

void SubscribeClientToTopic(uint16_t clientId, uint16_t topic)
This function updates the struct list that contains the subscriptions.

event void LogicHandlerModule.receivedType0Logic(uint16_t senderId)
If the node that received the message is the Pan Coordinator, it calls the updateConnectedClients function and the sendConAckMessage function.

event void LogicHandlerModule.receivedType1Logic()
If the node that received the message is a client, the connection process is completed and the connectionTimer is stopped on the client. Then the client calls the sendSubscribeMessage function and chooses at random one Topic to which it will send Publish messages. Finally the periodic publishTimer is started.

event void LogicHandlerModule.receivedType2Logic(uint16_t senderId, uint16_t subToTopic0, uint16_t subToTopic1, uint16_t subToTopic2)
If the node that received the message is the Pan Coordinator, it calls the SubscribeClientToTopic function and the sendSubAckMessage function.

event void LogicHandlerModule.receivedType3Logic()
If the node that received the message is a client, it stops the checkSubscriptionTimer to avoid sending again the subscription to the Pan Coordinator.

event void LogicHandlerModule.receivedType4Logic(uint16_t senderId, uint16_t topic, uint16_t value)
If the node that received the message is the Pan Coordinator, it updates the latestValues[i] variable keeping track of the latest value of each topic that we will send to node red when the periodic update is triggered. The pan also sends to the clients (subscribed to the topic to which the Publish is referring to) the Publish message calling the forwardPublishMessage(). Instead if the node that received the message is a client, it prints out the received values.

event message_t* Receive.receive(message_t* bufPtr, void* payload, uint8_t len)
When the receive is called and no problem on the packet occurred (length is correct), the function reads the Type of the message received and call the correct function to handle the logic (either postType0Logic, postType1Logic, postType2Logic, postType3Logic, postType4Logic).

***Timers***
ConnectTimer: Sends the first Connect message.

CheckConnectionTimer: This is triggered if the node didn't receive the ConAck, so it sends a new Connect message.

PublishTimer: This periodically sends a Publish message to the Pan Coordinator.

CheckSubscriptionTimer: This is triggered if the node didn't receive the SubAck, so it sends a new Subscribe message.

NodeRedTimer: This periodic timer is started only on the pan. When it is triggered it outputs on the node_red channel the latest values of the three topics.

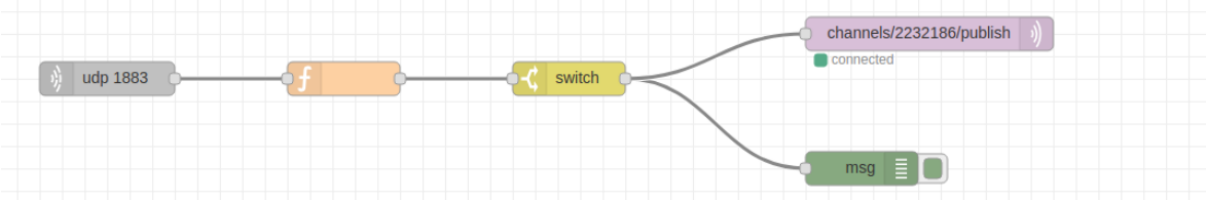
***interface OutQueueModule***
OutQueueModule.pushMessage: Adds to the out queue a message and an address, if the sendTimer is not running, it starts the sendTimer.

SendTimer.fired(): It removes the first message from the queue and calls the OutQueueModule.sendMessage. Then it checks if there are still messages in the queue. If the queue is not empty the timer is restarted.

**RunSimulationScript.py:**
Two channels were used to print the output on two different files (`simulation.log` and `node_red_outfile.txt`).
We ran a simulation of 10 000 events. After each event is run, we check whether a new line was written in the `node_red_outfile.txt` file. If it was, we send to Node-RED this new line through a socket using UDP.

To make the simulation run in real time we call the Time.sleep() after each event based on the Tossim simulation time. In this way on Thingspeak the plots would be readable. The elapsed time from the beginning of the simulation gets printed in the stdout while the `RunSimulationScript.py` is run.

**Node-Red:**




Node Red flow is the following:
When a message is received, a UDP receiver starts the flow. We then parse the message in a function block and we prepare the MQTT message to send to the server. Finally we send only the correctly parsed messages with the combination of the switch block and the MQTT out block.