Internet of Things
A.Y. 2022/2023
Project 1
Lightweight publish-subscribe application protocol

Eutizi Claudio 10812073
Perego Gabriele 10488414

28/08/2023

Contents

1 Abstract

1.1 Overview

In this project is requested to design and implement in TinyOS a lightweight publish-subscribe application protocol similar to MQTT and test it with simulations on a star-shaped network topology composed of 8 client nodes connected to a PAN coordinator, which acts as an MQTT broker (fig. 1).

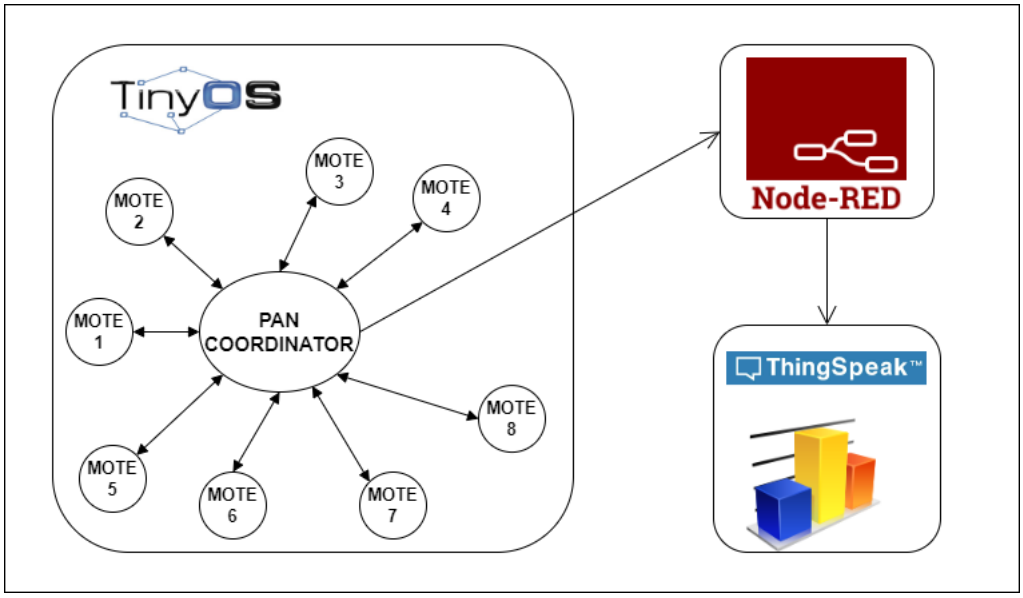

Figure 1: Project Topology

The features implemented are:

- **Connection**: upon activation, each node sends a CONNECT message to the PAN coordinator and the PAN coordinator replies with a CONNACK message. If the PAN coordinator receives messages from not yet connected nodes, such messages are ignored.
- **Subscribe**: after connection, each node can subscribe to one among these three topics: TEMPERATURE, HUMIDITY, LUMINOSITY.
  In order to subscribe, a node sends a SUBSCRIBE message to the PAN coordinator, containing its node ID and the topics it wants to subscribe to. We consider that the subscriber always use QoS=0 for subscriptions. Also, the subscribe message is acknowledged by the PANC with a SUBACK message.
- **Publish**: each node can publish data on at most one of the three aforementioned topics. The publication is performed through a PUBLISH message with the following fields: topic name, payload (assumed always QoS=0). When a node publishes a message on a topic, this is received by the PAN and forwarded to all nodes that have subscribed to a particular topic.

To test the implementation, we considered **TOSSIM** as simulation environment, with at least three nodes subscribing to more than one topic.
As payload of PUBLISH messages on all topics, we considered random number. The PAN Coordinator is connected to **Node-RED** and periodically transmit data received on the topics to **ThingSpeak** through MQTT messages. ThingSpeak shows the plot of each topic on a public channel.

1.2 Assumptions

In order to develop the project, the following assumptions have been considered: begin

- We consider node 1 as the PAN Coordinator and nodes from 2 to 9 as the other eight motes of the network.
- For temperature, humidity and luminosity values we decided to generate random values within certain ranges, in order to simulate a realistic scenario.
  For temperature we considered an interval of values between [0 ; 40] °C.
  For humidity we considered an interval of values between [30 ; 90] %.
  For luminosity we considered an interval of values between [0 ; 255] LUX.
- For emulate a periodic sending to Node-Red, at each message sent the PAN Coordinator sleeps for 10 seconds.
- Due to the sleep function mentioned previously, we set the number of events (in the *RunSimulationScript.py* file) to 10000.
- We used a TCP connection between TinyOS and Node-RED, and we used a MQTT connection between Node-Red and ThingSpeak.

2 Code Structure

In this section, we will describe the main features and the logic behind the code used to complete the project.

2.1 TinyOS

Here, we describe the TinyOS code used for the implementation of the network topology. In particular way, we are going to define at first the struct used in the header file *Project1.h*, an secondly we are going to describe the main functions used in the file *Project1.nc* for the implementation of the network.

2.1.1 Struct

Let's break down the main parts of the code structure in the header file *Project1.h*:

- **Msg** (msg_t): it is a msg_t structure that describes the format of the message. This message contains the following fields:
  - **type** : an nx_uint8_t field indicating the message type.
    Possible values could be : CONN → type = 0, CONNACK → type = 1, SUB → type = 2 , SUBACK → type = 3, PUB → type = 4.

  - **topic** : an nx_uint8_t field representing the message topic.
    Possible values could be : TEMPERATURE → topic = 0, HUMIDITY → topic = 1, LUMINOSITY → type = 2.
  - **sender** and **destination** : nx_uint8_t fields indicating the information about message sender and destination.
  - **data** : an nx_uint8_t field containing the data to be sent to NODE-RED.

- **MsgQueue** (queue_t): a queue_t structure that represents a message queue with a fixed size (MESSAGE_BUFFER_DIMENSION= 10 in our case). This queue is used to store and manage incoming and outgoing messages. The queue contains an array of msg_t, and some variables, of type nx_int16_t , like front, rear, and count used to manage message insertion and extraction from the queue.

- Queue Management Functions: main functions used to manage the operation of the queue. These functions are:
  - void **queueInit(queue_t* queue)**: initializes the message queue by setting front, rear, and count to initial values.
  - bool **enqueue(queue_t* queue, msg_t* msg)**: adds a message to the queue if there's available space.
  - bool **dequeue(queue_t* queue, msg_t* msg)**: removes a message from the queue if the queue is not empty.
  - void **printQueue(queue_t q)**: prints the current contents of the queue.

  In the *Project1.nc* file these functions are implemented.

- State Variables: array of elements used to manage the publish-subscribe operations between the PAN Coordinator and the motes of the network.
  We considered:
  - bool **connReceived**: contains booleans that store whether the PAN received the CONN message from the node.
  - bool **connAckReceived**: contains booleans that store whether the node received the CONNACK from the PAN.
  - int8_t **topicSubscriptions**: contains each node's topic subscription which is fixed, i.e. : nodes 1,4,8 are subscribed to temperature; nodes 2,3,6 are subscribed to humidity; and nodes 5,7 are subscribed to luminosity.
  - bool **subReceived**: contains booleans that store whether the PAN received the SUB message from the node.
  - bool **subAckReceived**: contains booleans that store whether the node received the SUBACK from the PAN.

2.1.2 function generateRandomValue

This function generates a random value based on the given topic (TEMPERATURE, HUMIDITY or LUMINOSITY):

- For the TEMPERATURE topic, it generates a random value in the range of [0, 40]°C.

- For the HUMIDITY topic, it generates a random value in the range of [30, 90]%.
- For the LUMINOSITY topic, it generates a random value in the range of [0, 255] lux.

2.1.3 function Boot.booted

This function initializes a queue data structure, and calls the start() function of AMControl.

2.1.4 function AMControl.startDone

This function handles the event of starting the radio communication module. If the radio start is successful, it prints a message indicating that the radio is on and sets up a periodic millisecond timer in order to trigger the node every 250 milliseconds. If the radio start fails, it logs an error message and attempts to restart the radio module.

2.1.5 function AMControl.stopDone

This function is a callback that is executed when a radio stop operation is completed. It prints a debug message indicating that the radio has been stopped.

2.1.6 function MilliTimer.fired

This function handles the activation of the MilliTimer interface "MilliTimer.fired()".
It performs the following tasks:

- Checks if a lock is active.
  if not locked, it extracts a message payload from a message;
- If the extracted message is NULL, it returns;
- If the node's ID is not 1 and it hasn't received a connection (CONN) message from the PAN Coordinator, it:
  - Populates the sender, the type (0 for CONN message), and the destination (PAN Coordinator) fields of the message.
  - Sends the message.

2.1.7 function Receive.receive

This function handles message reception, processing, and forwarding within the network. Here's a summary of what the function does:

- It takes as input a pointer to a message (bufPtr), a payload (payload), and the length of the payload (len).
- It checks if the length of the payload matches the expected size of msg_t. If the length doesn't match, the function returns. Otherwise, it proceeds with message processing.
- It casts the payload as a msg_t structure to access the received message's information.
- If the node's ID is not equal to 1 (i.e. not the PAN Coordinator node but a mote node), then:
  - If a connection acknowledgment (CONNACK) message is received (msg→type == 1) and the corresponding acknowledgment flag is not set, it:
    - Sends a subscription (SUB) message to a predefined topic based on the node's ID.
    - Sets relevant message parameters (topic, type, destination, sender).
    - Sends the message using the AMSend interface.
  - If a subscription acknowledgment (SUBACK) message is received (msg→type == 3) and the corresponding acknowledgment flag is not set, then it updates the acknowledgment flag.
- If both a connection acknowledgment and a subscription acknowledgment are received for the node:
  - Creates a new message.
  - Populates the message based on the subscribed topic.
  - Sets message parameters (topic, type, destination, sender).
  - Sends the message using the AMSend interface.
- If the node's ID is 1 (PAN Coordinator node):
  - If a connection (CONN) message is received (msg→type == 0), it sends a connection acknowledgment (CONNACK) back to the sender node.
  - If a subscription (SUB) message is received (msg→type == 2), it sends a subscription acknowledgment (SUBACK) back to the sender node.
  - If a publication (PUB) message is received (msg→type == 4), it enqueues the received message.
  - Then, it checks if any subscribed nodes match the message's topic and forwards the message to those nodes.
  - It sends finally the message over TCP/IP to the MQTT server, emulating periodic message transmission with the sleep function.

2.1.8 function AMSend.sendDone

This function is a callback handler for when a message transmission (AMSend) is completed. Its main purpose is to handle situations where packets might get lost during transmission and take appropriate actions based on the network's state. Here's a concise description of what the function does:

- It retrieves the payload of the packet that was sent (assuming the payload structure is msg_t).
- If the sent packet matches the packet pointed to by bufPtr and there was no error during transmission (error == SUCCESS): it sets a `locked` to TRUE.
- If the current node is not the PAN Coordinator (node 1) and either a connection acknowledgment (CONNACK) or a connection request (CONN) message has not been received, it indicates a potential packet loss situation for the connection message:
  - It sends again a connection (CONN) message to the PAN Coordinator.
  - Sets message parameters (type, destination, sender).
  - Sends the message using the AMSend interface.
- If the current node is not the PAN Coordinator, a connection acknowledgment (CONNACK) has been received, and either a subscription acknowledgment (SUBACK) or a subscription request (SUB) message has not been received, it indicates a potential packet loss situation for the subscription message:
  - Determines the original subscription topic based on the node's ID.
  - Sends again a subscription (SUB) message to the PAN Coordinator with the same topic.
  - Sets message parameters (topic, type, destination, sender).
  - Sends the message using the AMSend interface

2.2 Node-RED

Node-Red is composed by the following elements (fig. 2):

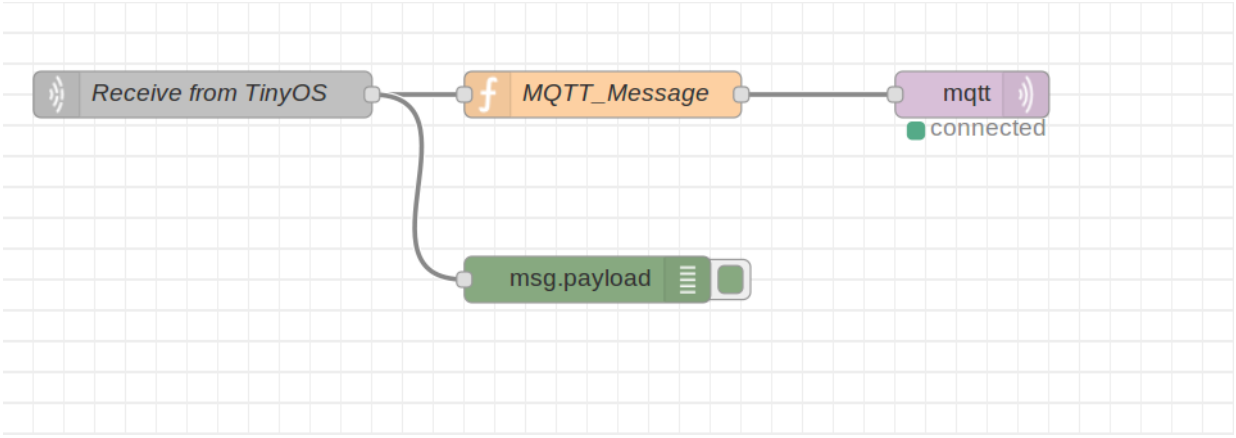

Figure 2: Node-Red Flow

- **Receive from TinyOS**: component that use TCP protocol and listens to the port "1048" that connects TinyOS and Node-Red.
- **MQTT_Message**: component that takes the message from TinyOS as an array, takes the topic and the corresponding value and, finally, sends a MQTT message with field and value to Thingspeak.
- **mqtt**: component that sends the MQTT messages, with QOS=1, to ThingSpeak. In this way, with QoS=1, the messages arrive to ThingSpeak correclty.
- **msg.payload**: component used for debugging.

2.3 ThingSpeak

All the plots of temperature, humidity and luminosity could be visualized throughout the ThingSpeak link of our public channel.

3 FINAL RESULTS

In this section we show the final results of our execution.

3.1 Make micaz sim

The log of the `make micaz sim` command can be found in the project folder as `make_micaz_sim_log.txt`.

3.2 TOSSIM Simulation

The TOSSIM simulation can be found in the project folder as `tossim_simulation_log.txt`.

3.3 ThingSpeak Simulation

The figure (fig. 3) shows the plots of temperature, humidity and luminosity as results of the previous simulation.

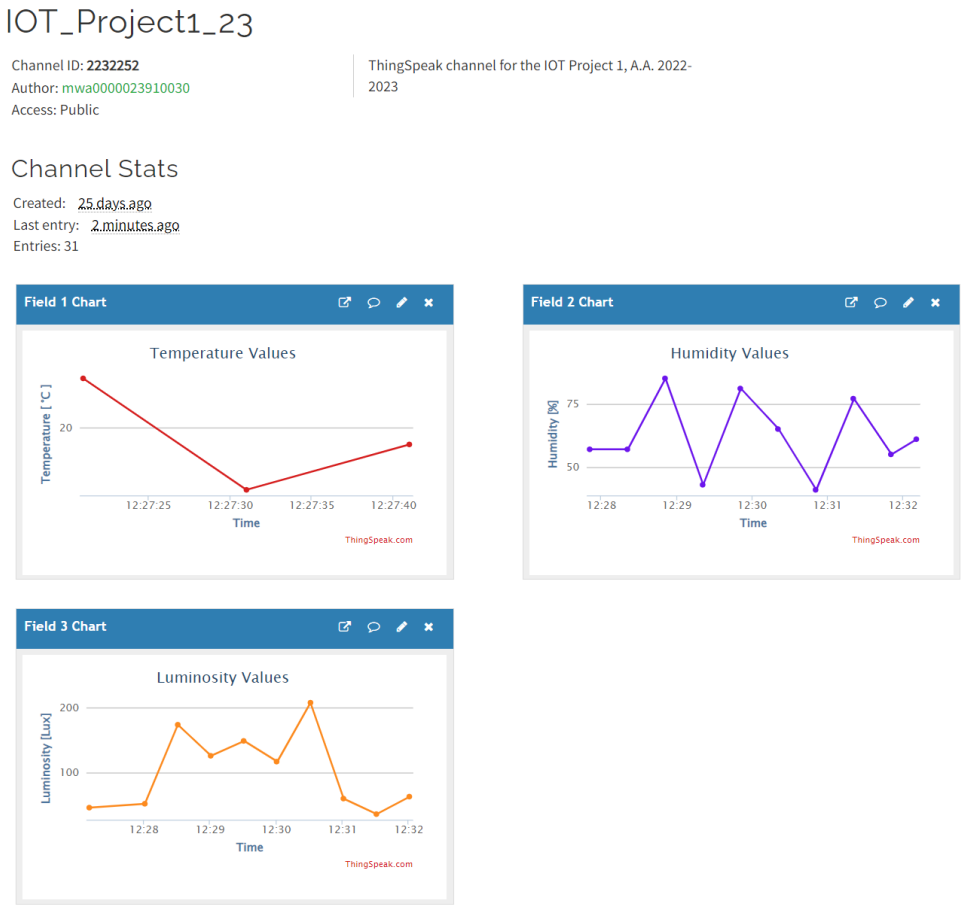

Figure 3: ThingSpeak Public View