IoT 2023 PROJECT 3

(1) Name: Fontana Nicolò Person Code: 10581197

(2) Name: Gerosa Andrea Person Code: 10583298

Sensor 1: https://wokwi.com/projects/371751839707397121

Sensor 2: https://wokwi.com/projects/372030284697277441

Actuator 1: https://wokwi.com/projects/371761136986220545

Actuator 2: https://wokwi.com/projects/372030291681843201

Assumptions

As requested, the network is deployed over a single topic (*/10581197_10583298*) of a public MQTT server (*broker.hivemq.com*, port *1883*). The PAN Coordinator (PANC) is implemented using a Node-RED flow, while the clients, both sensors and actuators, are implemented as ESP32 devices simulated in the Wokwi environment.

Every minute the PANC publishes a 1KB-payload message over a secondary topic (*/10581197_10583298_speedtest*) and receives it. The time interval needed for the entire procedure is doubled and used as the duration of the time slot. Before the first speed-test message is sent, the time slot duration is set to 100 milliseconds.

The Beacon Interval (BI) is 5 seconds (so the PANC broadcasts a beacon every 5 seconds), the frame duration is computed as 1 time slot, the Collision Free Part (CFP) duration is computed as a number of time slots equal to the number of clients registered, and the Collision Access Part (CAP) is computed as the minimum between the CFP and the remaining time (=BI duration - frame duration - CFP duration). If no client is registered, the CAP and CFP durations are put equivalent to 1 time slot.

When a field device sends an association message, the relative ACK message from the PANC is sent using as destination the MQTT ID of the field device. The MQTT ID is a 10-chars-long string randomly generated by the field device when connecting to the server and specified in the association message. When a new device joins the network, the PANC assigns it a newly sequentially generated ID.
The sequentially generated IDs from the PANC are composed as "*c_*" followed by a 2-digits number (from "*c_00*" to "*c_99*"), allowing a maximum of 100 clients to be correctly handled by the PANC; after that, the network is considered full. If the network is full, an error message is returned from the PANC to the client requesting to join the network.
When the new client joins the network, if it is an actuator, the PANC stores its ID to recover this information when will have to choose the random actuator to forward a data message to.

A client knows that its association procedure is completed when it receives the corresponding ACK message containing the new ID. After publishing the association message and if no ACK message is received, for each beacon received, the client checks if a certain amount of time (20 seconds) is passed; if that is the case, the client will try again by sending another association message in the next beacon interval. If more than 1 ACK message is received, the first one is considered and the following ones are discarded.

All the timestamps used are measured in seconds and they are relative to the moment in which the device has been booted up.

Each sensor, between every 15 and 20 milliseconds, collects a humidity percentage sample, adds a Gaussian noise (zero mean and standard deviation 15.0) to it and stores it in a global variable.
When a fully registered sensor receives a beacon, it waits for its time slot and sends the last humidity percentage value stored in its dedicated global variable.

Upon receiving a data message, the PANC chooses a random actuator using a uniform distribution over all the ones registered. After shifting the received value from the interval [0,100] to the interval [0,1], a new data message is created to forward it. The source node used for the new data message is still the sensor which originally sent the message to the PANC.

When a fully registered actuator receives a data message, it computes the equivalent level for the LED brightness and updates the LED. Every actuator allows for 8 different levels of LED brightness (from 0 to 7; 1 for every 12.5% interval of humidity sensed).

Wokwi limitations

We had to deal with some limitations of Wokwi:

- Every time an RFD had to wait a certain amount of time, we had to rely on the *delay* function because the activation of a timer made Wokwi's performances drop and did not allow us to continue the simulation.
- After a few seconds (around 20/30 seconds in real-time, which translated into 10/20 seconds in the simulation), the WiFi connection dropped, preventing further exchange of messages from and to the RFDs.
- In addition, Wokwi did not report the connection loss until after around 1 minute had passed and even then the *WiFi.status()* would have returned a *CONNECTED* status, not allowing us to restore the connection.
- The humidity samples provided by Wokwi were constant (40.0%), thus we decided to add Gaussian noise to them to add some variety and allow us to visually test also the LED updates.

Design choices

All messages have 3 common attributes:

- *type*, used to distinguish each kind of message
- *src*, representing the source node of the message
- *dst*, representing the destination node of the message

In addition, each type of message has its own specific attributes.

- Type -1 (error message): can be sent only by the PANC to a specific client in case something goes wrong, for example, a new device tries to join the network while it is full.
  Attributes:
    - *error*: contains the string of the error message

- Type 0 (beacon message): broadcasted by the PANC to communicate to the clients how the BI is split and when it is their turn to send a message during the CFP.
  Attributes:
    - *frame*: int to represent the duration of the frame (in milliseconds)
    - *cap*: int to represent the duration of the CAP (in milliseconds)
    - *cfp*: int to represent the duration of the CFP (in milliseconds)
    - *bi*: int to represent the duration of the whole BI (in milliseconds)
    - *assignments*: array of strings, where the position represents the slot of the CFP assigned to the client and the string contained in each position represents the client ID assigned to that position
- Type 1 (association message): sent by a new client to the PANC to join the network and sent upon receiving its first beacon from the PANC. As *src*, the randomly generated MQTT ID of the client is used.
  Attributes:
    - *client_type*: int to distinguish different kinds of clients. 0=sensor, 1=actuator, 2=both sensor and actuator
- Type 2 (association ACK message): sent by the PANC to a specific client which had previously requested to join the network using an association message. The attribute *dst* is the randomly generated MQTT ID used by the client as *src* for the association message.
  Attributes:
    - *id*: string containing the ID assigned to the client by the PANC. It will be used by the client and the PANC for all subsequent communication in the network
- Type 3 (data message): sent by any sensor to the PANC or forwarded by the PANC to a random actuator. It is used to pass the data sensed in the environment. When it is forwarded, the ID of the sensor which generated the data is maintained in the attribute *src* (even if the message is routed by the PANC).
  Attributes:
    - *data*: int to represent the humidity percentage value
    - *forwarded*: int used as a boolean to check whether the message is being sent from a sensor to the PANC (=0) or is being forwarded by the PANC to the chosen actuator (=1)

Wokwi code

Sensors

Both sensors have the same code with 4 functions.

- *setup*: connects the device to the WiFi network, connects to the MQTT server with a randomly generated 10-chars-long MQTT ID, subscribes to the main MQTT topic, setups the LED.
- *loop*: checks if the WiFi connection dropped and in case reconnects to it, samples the humidity, adds the Gaussian noise and stores the final value, and performs the MQTT loop to receive the messages published over the main topic.
- *callback*: deserializes the received payload and handles various cases based on the type of the message and the state of the device.
    - If type = -1 (error message): prints the error string attached to the message.
    - If type=0 (beacon message) and joined = 0 (the device has not joined the network yet) and joining=0 (the device has not sent the association message yet): prepares the association

  message, waits (using a delay) for the CAP to be started, publishes the association message, saves the current timestamps and put itself in the joining state (joining=1).
  - If type = 0 (beacon message) and joined = 0 (the device has not joined the network yet) and joining = 1 (the device is waiting for the ACK): checks whether the time interval from when the association message was sent is bigger than a given threshold. If so, exits the joining state (joining=0) to allow re-sending the association message in case there has been a collision.
  - If type = 2 (ACK message) and joining = 1 (the device has already sent the association message) and the destination matches its MQTT ID: changes state to joined (joined=1, joining=0) and stores the assigned ID.
  - If type = 0 (beacon message) and joined = 1 (the device has already joined the network): checks which slot is assigned to it, prepares the message using the last stored value of humidity, waits for its slot in the CFP and publish the data message prepared.
  - Other cases: does nothing and discards (i.e. ignores) the message.
- *gaussianNoiseGenerator*: using the current time as seed, generates the Gaussian noise (mean=0, std dev=15) to be added to the humidity value sensed in the loop.

Actuators

Both actuators have the same code with 3 functions.

- *setup*: same as the sensors.
- *loop*: checks if the WiFi connection dropped and in case reconnects to it, performs the MQTT loop to receive the messages published over the main topic.
- *callback*: deserializes the received payload and handles various cases based on the type of the message and the state of the device.
  - If type = -1 (error message): same as the sensors.
  - If type = 0 (beacon message) and joined = 0 (the device has not joined the network yet) and joining = 0 (the device has not sent the association message yet): same as the sensors.
  - If type = 0 (beacon message) and joined = 0 (the device has not joined the network yet) and joining = 1 (the device is waiting for the ACK): same as the sensors.
  - If type = 2 (ACK message) and joining = 1 (the device has already sent the association message) and the destination matches its MQTT ID: same as the sensors.
  - If type = 3 (data message) and the destination matches its MQTT ID: computes the brightness level corresponding to the received decimal percentage and updates the LED brightness.
  - Other cases: does nothing and discards (i.e. ignores) the message.

Node-RED nodes

The Node-RED flow can be semantically divided into 4 parts, each one containing some nodes carrying out a certain split of the PANC job.

- Speedtest part:
  - *speedtest clock*: marks every minute to check the speed of the MQTT connection.

    - *send test rate msg*: saves the timestamp and prepares the 1KB test message.
    - MQTT IN&OUT (*/10581197_10583298_speedtest*): publish and receive the 1KB test message.
    - *compute time slot*: saves the timestamp of the arrival of the test message and computes the time slots as double the time interval between the 2 saved timestamps.
- Beacon part:
    - *beacon clock*: marks every 5 seconds to send a beacon.
    - *beacon sender*: computes the fields of the beacon message and prepares the message.
    - MQTT OUT (*/10581197_10583298*): publishes the beacon message.
- Routing part:
    - MQTT IN (*/10581197_10583298*): receives every message published over the main topic
    - *routing*: filters the received messages to handle only association or data ones.
      If it filters an association message, it checks whether the network is already full or not; if it is full an error message is prepared, otherwise an ACK message is prepared with the new ID.
      If it filters a data message, a random actuator is uniformly extracted from the list of registered ones and the data is forwarded to it preparing a new data message.
    - MQTT OUT (*/10581197_10583298*) (same node of the beacon part): publishes error, ACK and data messages prepared by the previous node.
- Control & debug part:
    - *RESET*: button to stop and start the PANC.
    - *reset flow vars*: resets all global variables. For example, empties the array of registered clients or resets to 0 the counter for the IDs to be assigned
    - Various debug nodes: used to print some logs of the PANC's activity.

NB: by project specification, the PANC was not required to forward the data message because the actuators were already subscribed to the topic where those messages were published by the sensors. However, we decided that it would have been more correct to add the *dst* attribute and not allow the sensors (which are RFDs) to directly send their data to the actuators (which are RFDs too), but instead force them to pass through the PANC.

NB: we decided to let the PANC randomly choose the actuator to be the destination of each data message. This was done to prevent anti-patterns like sending information regarding the whole network (i.e. which registered RFDs are actuators) to the sensors by the PANC and to resolve all the logic concerned with the entire network in the PANC itself.