IoT Final Project 2023 - Project 1

Pasquale Castiglione
10657816

Lorenzo Campana
10605775

mqtt.h

The struct mqtt_msg is used to define the message format to be sent between nodes. It contains the following attributes:

- `type`
- `ID` (sender Node ID)
- `topic`
- `payload`
- `seq`, 32-bit number, used to identify a specific msg-ack sequence

mqttC.nc

Variables

- `locked`:
  This boolean variable is used to indicate whether a message sending operation is currently in progress. When a message is being sent, this variable is set to TRUE, preventing further message sending until the current operation completes.
- `time_delays[N_CLIENTS]`:
  This array holds time delay values associated with each client ID. It's used for setting specific time delays before certain actions occur for each client.
- `n_subscription_sim[N_CLIENTS]`:
  This array holds the number of remaining subscriptions each client needs to make. It's decremented as clients subscribe to topics.

- `subscribed_sim[N_CLIENTS][N_TOPICS]`:
  This 2D array indicates whether a client is subscribed to a particular topic. It's used to track the subscription status of each client for different topics.

- `panc_table`:
  This structure represents the table used to keep track of client connections and topic subscriptions.

- `conn_ack_wait`:
  This variable defines the initial waiting time for a CONNACK acknowledgment response from the broker.

- `sub_ack_wait`:
  This variable defines the initial waiting time for a SUBACK acknowledgment response from the broker.

- `waiting_CONNACK`:
  This structure is used to track the status and sequence number of the CONNECT acknowledgment response that the client is waiting for.

- `waiting_SUBACK`:
  Similarly to `waiting_CONNACK`, this structure tracks the status and sequence number of the SUBACK acknowledgment response that the client is waiting for.

- `queued_sub_topic`:
  This variable holds the topic that is currently queued for subscription. It's used to ensure that the correct topic is subscribed to after a timer event fires.

Functions

- `init_panc_table`:
  Initializes the Panc table structure to track client connections and subscriptions.

- `send_connect_to_PANC`:
  Sends a CONNECT message to the PAN Coordinator to establish a connection.

- `send_subscribe`:
  Sends a SUBSCRIBE message to the PAN Coordinator to subscribe to a specified topic.

- `send_publish`: Sends a PUBLISH message to the PAN Coordinator to publish on a specified topic.
- `create_connection`:
  Handles the creation of a connection and sends a CONNACK response to the client.
- `create_subscription`:
  Handles the creation of a subscription and sends a SUBACK response to the client.
- `forward_publish`:
  Forwards PUBLISH messages to subscribed clients.
- `isConnected`:
  Checks if a specific client is connected based on the Panc table.
- `isSubscribed`:
  Checks if a specific client is subscribed to a topic based on the Panc table.

Timers

- `Timer0`:
  Initiates the process of sending a CONNECT message to establis a connection.
- `Timer_wait_CONNACK`:
  Handles timeouts and retries for waiting for a CONNACK response.
- `Timer_wait_SUBACK`:
  Handles timeouts and retries for waiting for a SUBACK response.
- `Timer_SUB`:
  Initiates the process of sending a SUBSCRIBE message to subscribe to a topic.
- `Timer_PUB`:
  Initiates the process of sending a PUBLISH message.

Receive Event

Received messages are handled according to their type:

PANC

- `CONNECT Message:`
  The handler calls the `create_connection` function, which updates the texttttPanc_table to mark the client as connected and sends a CONNACK acknowledgment back to the client. The acknowledgment confirms the successful establishment of the connection between the client and the broker.

- `SUBSCRIBE Message:`
  The handler calls the `create_subscription` function, which updates the `Panc_table` to mark the client as subscribed to the specified topic. The handler sends a SUBACK acknowledgment back to the client, confirming the successful subscription to the requested topic.

- `PUBLISH Message:`
  The handler calls the `forward_publish` function, which checks if the sender client is connected and forwards the message to all subscribed clients for the specified topic. The handler iterates through the subscribed clients and sends the PUBLISH message to each subscribed client.

MOTE

- `CONNACK Message:`
  If the client was waiting for a CONNACK acknowledgment, the handler updates the acknowledgment status and performs actions such as initializing timers for subscription and publishing tasks.

- `SUBACK Message:`
  If the client was waiting for a SUBACK acknowledgment, the handler updates the acknowledgment status and marks the topic as subscribed. The handler also decrements the remaining subscription count for the client.

Simulation

We leveraged on both simulation envoironments.

TOSSIM

```
DEBUG (9): APP BOOTED.
DEBUG (8): APP BOOTED.
DEBUG (2): APP BOOTED.
DEBUG (6): APP BOOTED.
DEBUG (5): APP BOOTED.
DEBUG (1): APP BOOTED.
DEBUG (4): APP BOOTED.
DEBUG (3): APP BOOTED.
DEBUG (7): APP BOOTED.
DEBUG (2): Radio started.
DEBUG (9): Radio started.
DEBUG (8): Radio started.
DEBUG (7): Radio started.
DEBUG (3): Radio started.
DEBUG (6): Radio started.
DEBUG (4): Radio started.
DEBUG (1): Radio started.
DEBUG (5): Radio started.
DEBUG (1): Send CONNECT packet
DEBUG (9): CONNECT received from 1 and seq 33614.
DEBUG (9): Creating connection with client 1 and seq 33614.
DEBUG (9): Send CONNACK packet to 1.
DEBUG (1): CONNACK received with seq 33614.
DEBUG (1): Connected to PANC.
...
...
DEBUG (1): Send SUBSCRIBE packet
DEBUG (9): SUBSCRIBE received from 1 with topic 1 and seq 940422544.
DEBUG (9): Creating subscription with client 1 and topic 1.
DEBUG (9): Send SUBACK packet to 1.
DEBUG (1): SUBACK received with seq.
DEBUG (1): Subscribed to a topic 1.
DEBUG (1): Send SUBSCRIBE packet
DEBUG (9): SUBSCRIBE received from 1 with topic 0 and seq 1866991771.
DEBUG (9): Creating subscription with client 1 and topic 0.
DEBUG (9): Send SUBACK packet to 1.
DEBUG (1): SUBACK received with seq 1866991771.
DEBUG (1): Subscribed to a topic 0.
DEBUG (2): Send SUBSCRIBE packet
...
```

```
...
DEBUG (4): Send PUBLISH packet
DEBUG (9): PUBLISH received from 4 on topic:1 with payload:72.
DEBUG (9): Send PUBLISH packet to 1.
DEBUG (5): PUBLISH received on TOPIC:1 VALUE:72.
...
...
```

COOJA

We create a simulation in the Cooja environment with nine motes grouped in a star topology. Among these, we setup the ninth mote, PanC, to function as a serial socket server on port 60001.

Node-Red

The Node-Red flow is indeed straighforward, it contains a network interface that intercept local input, and pass it towards the split function. Then the function divides the `PUBLISH` message in the form `Topic,Payload` to craft a proper MQTT publish request understandable by our public ThingSpeak channel, with topic `channels/channel_id/publish` and payload `status=MQTTPUBLISH&fieldX=data`

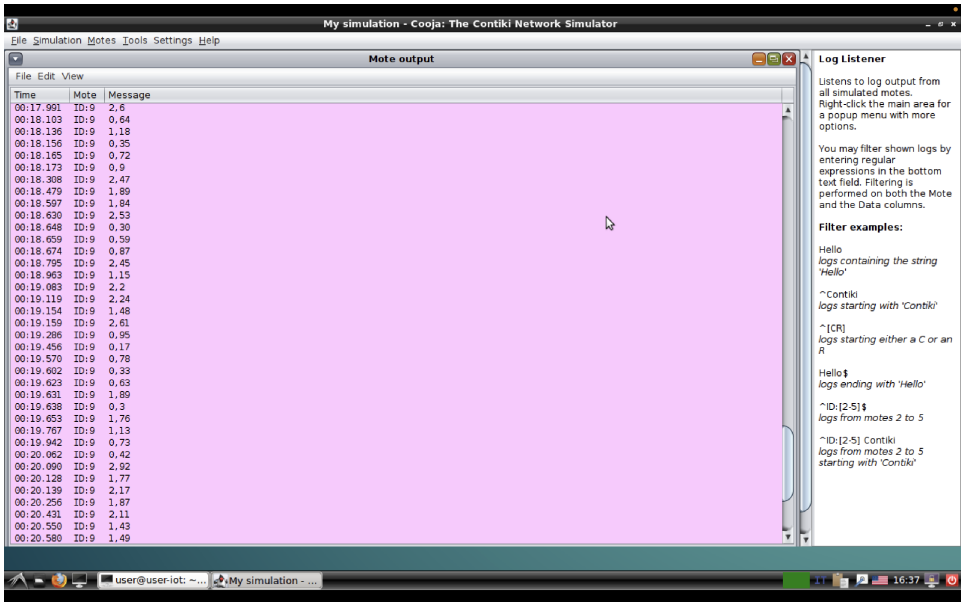

Figure 1: Cooja log of publish messages sent by the PANC to Node-Red

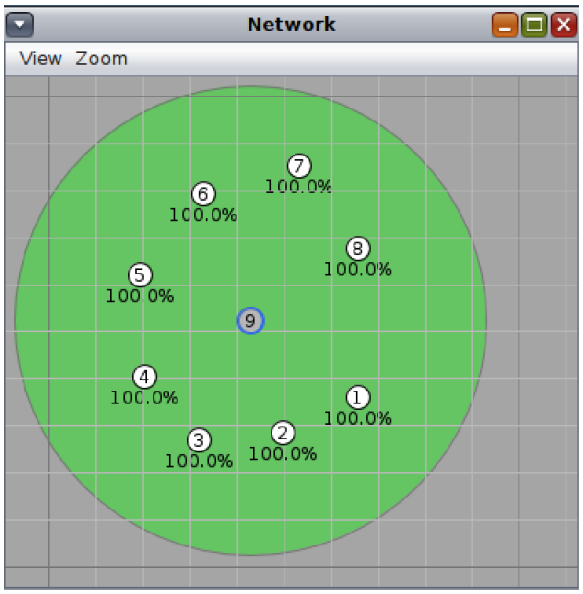

Figure 2: Network topology

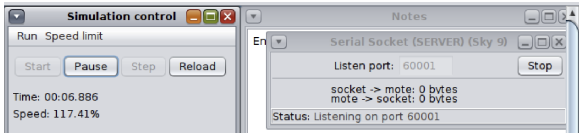

Figure 3: Cooja simulation control and Serial Socket setup

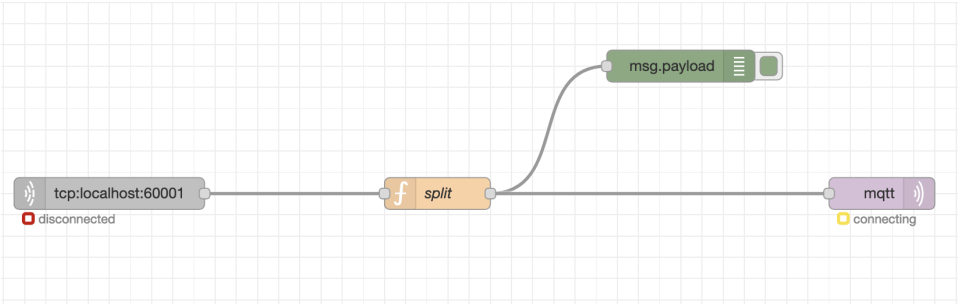

Figure 4: Node-Red flow

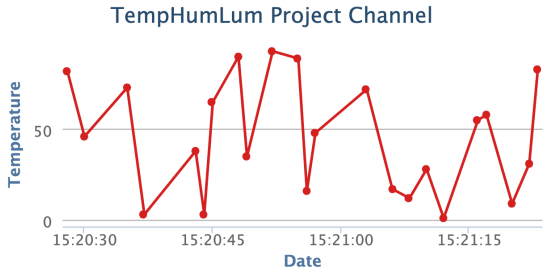

Figure 5: Temperature chart

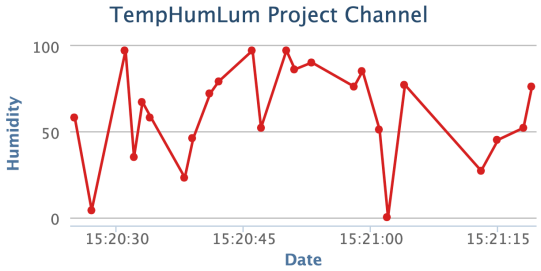

Figure 6: Humidity chart

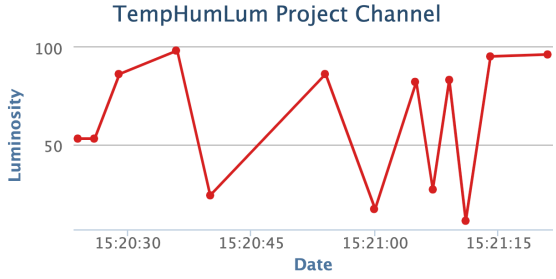

Figure 7: Luminosity chart