IOT Final Project / Project 1/ Lightweight publish-subscribe application protocol

Name: Nadia Sadeghi
Person Code: 10916407

The chosen project focuses on developing a lightweight publish-subscribe application protocol. It involves creating an MQTT Broker and eight clients, with three phases:

- connection
- subscription
- publishing.

The project simulation environment is TOSSIM. After code completed we can add printf command and open code in Cooja finally by making channel in Thingspeak URL, find three plot for our three topic : TEMPERATURE, HUMIDITY, LUMINOSITY

Unfortunately, due to tight exams and short time, I only had the opportunity to write the program and run it in TOSSIM. Therefore, the project has not been fully implemented.

Booting phase

During the first phase of implementation, all nodes undergo a booting process to set up their radios. The MQTT broker initializes variables for network management, while the clients start the connection phase.

```
module mqttC {

  uses {
    interface Boot;
        interface AMPacket;
    interface Packet;
    interface PacketAcknowledgements;
        interface AMSend;
        interface SplitControl;
        interface Receive;
    interface Read<uint16_t>;
    interface Timer<TMilli> as Timer0;
    interface Timer<TMilli> as Timer1;
    interface Timer<TMilli> as Timer2;
    interface Timer<TMilli> as Timer3;
    interface Timer<TMilli> as Timer4;
  }

} implementation {

  //Declaration of the different variables used
  uint8_t nbNodesConnected=0;
  uint8_t messageID = 1;
  uint8_t retransmission = 0;
  uint16_t valueRetrans = -1;
  uint16_t valueToForward = -1;
  uint8_t topicToForward = -1;
  uint8_t sourceToForward = -1;
  uint8_t index = 0;
  uint8_t forwardRetrans = 0;
  uint8_t forwarding = 0;
  message_t packet;
```

Connection Phase

After a successful booting process, the clients need to establish a connection with the MQTT broker. They send a CONNECT message containing the message type. The message types include CONNECT=1, SUBSCRIBE=2, PUBLISH=3, and FORWARD=4. Each CONNECT message must be acknowledged by the MQTT broker. If no acknowledgment is received, the node retries sending the message after a 0.5-second delay.

```
//***************** Task sendConnect ******************//
// This task is used by nodes to send a connect request to the MQTT
// broker which is supposed to answer by a CONNACK
task void sendConnect() {
  my_msg_connect_t* mess=(my_msg_connect_t*)(call Packet.getPayload(&packet,sizeof(my_msg_connect_t)));
  mess->type = CONNECT; //Fills the type field
  dbg("radio_send", "Try to send a connect request to node 1 (MQTT Broker) at time %s \n", sim_time_string());
  call PacketAcknowledgements.requestAck( &packet );//Asks for an ACK
  if(call AMSend.send(1,&packet,sizeof(my_msg_connect_t)) == SUCCESS){ //Tries to send the packet to the MQTT Broker
    dbg("radio_send", "Packet passed to lower layer successfully!\n");
    dbg("radio_pack",">>>Pack\n \t Payload length %hhu \n", call Packet.payloadLength( &packet ) );//Displays the payload length
    dbg_clear("radio_pack","\t Source: %hhu \n ", call AMPacket.source( &packet ) );//Displays the source
    dbg_clear("radio_pack","\t Destination: %hhu \n ", call AMPacket.destination( &packet ) );//Displays the destination
    dbg_clear("radio_pack","\t AM Type: %hhu \n ", call AMPacket.type( &packet ) );//Displays the AM Type
    dbg_clear("radio_pack", "\t\t Message type: %hhu \n", mess->type); //Displays the type of msg
      }
 }
```

Subscription Phase

In the MQTT model, nodes subscribe to three topics with specified QoS levels. The SUBSCRIBE message includes the message type, topic array, and QoS array. We assigned subscription rules based on node indices and implemented arbitrary rules for QoS levels. If an acknowledgment is not received, the client waits for 0.5 second before resending the SUBSCRIBE message to the MQTT broker.

To store subscriptions, the MQTT Broker utilizes a matrix where rows represent topics and columns represent client nodes. The matrix stores values denoting subscription status: "-1" indicates no subscription, and "0" indicates a subscription with Low QoS.

```
//***************** Task sendSubscribe ******************//
// This task is used by nodes to send a subscribe request to
// the MQTT broker which is supposed to answer by a SUBACK
task void sendSubscribe() {
  my_msg_subscribe_t* mess=(my_msg_subscribe_t*)(call Packet.getPayload(&packet,sizeof(my_msg_subscribe_t)));
  mess->type = SUBSCRIBE; //Fills the type field
  if(TOS_NODE_ID<9){ //All nodes, we define a low QOS on all topics
      mess->qos[TEMPERATURE-1] = QOS_LOW;
      mess->qos[HUMIDITY-1] = QOS_LOW;
      mess->qos[LUMINOSITY-1] = QOS_LOW;
  }
  if(TOS_NODE_ID%2 == 0){ //If the nodeID is even, we subscribe to all the topics, this choice is an arbitrary one
      mess->topic[TEMPERATURE-1] = 1;
      mess->topic[HUMIDITY-1] = 1;
      mess->topic[LUMINOSITY-1] = 1;
  } else { //If the nodeID is odd, we subscribe to TEMPERATURE and LUMINOSITY only, this choice is an arbitrary one
      mess->topic[TEMPERATURE-1] = 1;
      mess->topic[HUMIDITY-1] = 0;
      mess->topic[LUMINOSITY-1] = 1;
  }
  dbg("radio_send", "Try to send a subscribe request to node 1 (MQTT Broker) at time %s \n", sim_time_string());
  call PacketAcknowledgements.requestAck( &packet ); //Asks for an ACK
  if(call AMSend.send(1,&packet,sizeof(my_msg_subscribe_t)) == SUCCESS){ //Tries to send the packet to the MQTT Broker
    dbg("radio_send", "Packet passed to lower layer successfully!\n");
    dbg("radio_pack",">>>Pack\n \t Payload length %hhu \n", call Packet.payloadLength( &packet ) ); //Displays the payload length
    dbg_clear("radio_pack","\t Source: %hhu \n ", call AMPacket.source( &packet ) ); //Displays the source
    dbg_clear("radio_pack","\t Destination: %hhu \n ", call AMPacket.destination( &packet ) ); //Displays the destination
    dbg_clear("radio_pack","\t AM Type: %hhu \n ", call AMPacket.type( &packet ) ); //Displays the AM Type
    dbg_clear("radio_pack", "\t\t Message type: %hhu \n", mess->type); //Displays the type of msg
    dbg_clear("radio_pack", "\t\t Topic: [%hhu %hhu %hhu] \n", mess->topic[0], mess->topic[1], mess->topic[2]); //Displays the topics to
subscribe to*/
    dbg_clear("radio_pack", "\t\t QOS: [%d %d %d] \n", mess->qos[0], mess->qos[1], mess->qos[2]); /*Displays the different QOS for each
topic*/
      }
 }
```

Publishing & Forwarding Phase

The publish phase is divided into two subphases: publish and forward. In the publish subphase, the client sends the message to the MQTT Broker. The forward subphase involves distributing the message from the Broker to the subscribed nodes.

In the publish subphase, a client node sends a PUBLISH message to the MQTT Broker. The message includes the message type, value, message ID, topic, and QoS. If the QoS is set to High and the acknowledgement is not received, the node resends the same value until the ACK is received. Each node publishes on a specific topic, and the QoS between the node and the Broker is arbitrarily chosen. Once the MQTT Broker receives the PUBLISH message and fulfills the QoS requirements, the forward subphase can proceed.

In the FORWARD subphase, the MQTT Broker distributes the message to nodes subscribed to the topic. The Broker copies the relevant row from the subscription's matrix into a temporary array. It reads the array and sends a FORWARD message to each node with the corresponding QoS (0 for Low QoS). This process continues until all nodes have received the message.

During the FORWARD subphase, the MQTT Broker updates the temporary array after a successful transmission by replacing the previous value with "-1". If a transmission fails, the array remains unchanged, and the Broker moves on to the next value. The process continues until all values in the array become "-1". If a node tries to publish while the Broker is already forwarding, the publish message is discarded to indicate the Broker's busy state.

```
118 //***************** Task sendPublish ******************//
119  //This task is used by the different nodes to sending a
120  task void sendPublish(){
121    call Read.read(); //
122  }
123
124  //***************** Task sendForward ******************//
125  //This task is used by the MQTT broker after receiving a
126  //publish message from a node
127  task void sendForward() {
128    int l, found = 0;
129    my_msg_forward_t* mess=(my_msg_forward_t*)(call Packet.getPayload(&packet,sizeof(my_msg_forward_t)));
130    mess->type = FORWARD;
131    mess->value = valueToForward;
132    mess->source = sourceToForward;
133    mess->topic = topicToForward;
134    for(l=index; l<MAX_NB_NODES && found==0; l++){
135        if(sourceToForward != l+2){
136        if(nodesToForward[l] == QOS_LOW){
137                    mess->qos = subscriptions[l][topicToForward-1];
138                    call PacketAcknowledgements.noAck( &packet );
139                    found = 1;
140                    index = l;
141                    if(call AMSend.send(l+2,&packet,sizeof(my_msg_forward_t)) == SUCCESS){
142                        dbg("radio_send", "The MQTT broker tries to forward the publish request to node %hhu at time %s \n", l+2,
sim_time_string());
143                        dbg("radio_pack",">>>Pack \n \t Payload length %hhu \n", call Packet.payloadLength( &packet ) ); //Displays the
payload length
144                        dbg_clear("radio_pack","\t Source: %hhu \n", call AMPacket.source( &packet )); //Displays the source
145                        dbg_clear("radio_pack","\t Destination: %hhu \n", call AMPacket.destination( &packet ) ); //Displays the
destination
146                        dbg_clear("radio_pack","\t AM Type: %hhu \n", call AMPacket.type( &packet ) ); //Displays the AM Type
147                        dbg_clear("radio_pack", "\t\t Message type: %hhu \n", mess->type); //Displays the type of msg
148                        dbg_clear("radio_pack", "\t\t Topic: %hhu \n", mess->topic); //Displays the topic concerned by the publication
```

```
destination
146                        dbg_clear("radio_pack","\t AM Type: %hhu \n", call AMPacket.type( &packet ) ); //Displays the AM Type
147                        dbg_clear("radio_pack", "\t\t Message type: %hhu \n", mess->type); //Displays the type of msg
148                        dbg_clear("radio_pack", "\t\t Topic: %hhu \n", mess->topic); //Displays the topic concerned by the publication
149                        dbg_clear("radio_pack", "\t\t QOS: %hhu \n", mess->qos); //Displays the QOS
150                        dbg_clear("radio_pack", "\t\t Value: %hhu \n", mess->value); //Displays the value
151                        dbg_clear("radio_pack", "\t\t Initial source: %hhu \n", mess->source); //Displays the value
152                    }
153
154            }
155        }
156        if(found == 0 && l == MAX_NB_NODES-1 && forwardRetrans == 0){
157            forwarding=0;
158            dbg_clear("radio_pack", "\t\t forwarding: %hhu \n", forwarding);
159        }
160        if (forwardRetrans==1 && l==MAX_NB_NODES-1){
161            forwardRetrans=0;
162            index=0;
163            post sendForward();
164        }
165    }
166  }
```

Result

In this project, there are nine nodes (one MQTT Broker and eight clients) with staggered boot times. Running the application resulted in detailed output, showcasing the entire process from booting to publishing. According to the log results, different values have been obtained for all three topics based on QOS=0.