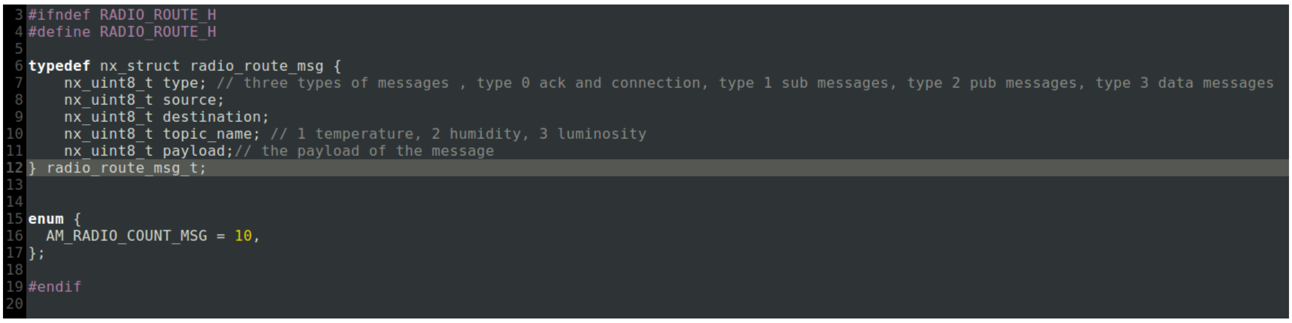
PROJECT 1

Team Leader: Pasquale Lobaccaro (person code) 10958708
Andrea Ricciardi (person code) 10931392

TINYOS CODE

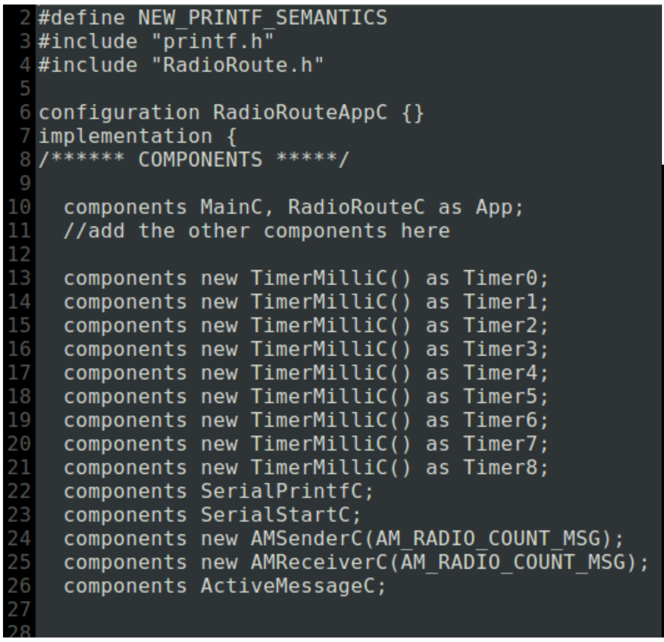
First of all, the header file (RadioRoute.h) has been implemented. A struct for our messages (radio_route_msg_t) has been created with five attributes:

- type: it indicates which type of message is being used. Type 0 for ACK and CONNECT messages, type 1 for SUBSCRIBE messages, type 2 for PUB messages and type 3 for DATA messages;
- source: it indicates the source of the message;
- destination: it indicates the destination of the message
- topic_name: it indicates the topic to which the message is related to. Topic 1 for TEMPERATURE, topic 2 for HUMIDITY and topic 3 for LUMINOSITY;
- payload: it indicates the payload of the message.

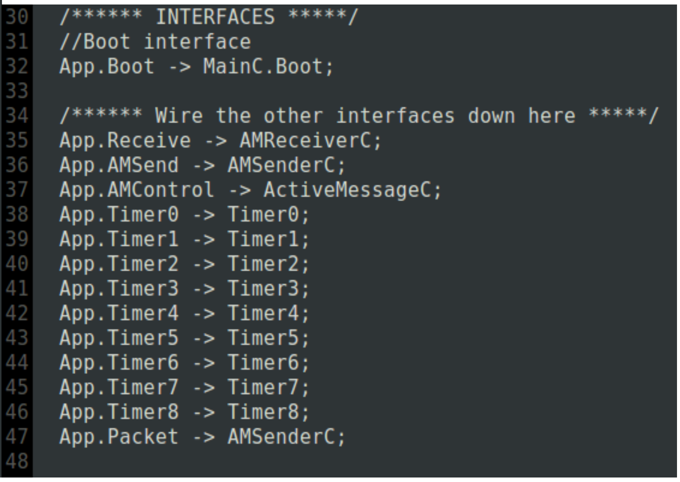


```
#ifndef RADIO_ROUTE_H
#define RADIO_ROUTE_H

typedef nx_struct radio_route_msg {
    nx_uint8_t type; // three types of messages , type 0 ack and connection, type 1 sub messages, type 2 pub messages, type 3 data messages
    nx_uint8_t source;
    nx_uint8_t destination;
    nx_uint8_t topic_name; // 1 temperature, 2 humidity, 3 luminosity
    nx_uint8_t payload;// the payload of the message
} radio_route_msg_t;


enum {
  AM_RADIO_COUNT_MSG = 10,
};

#endif
```

Subsequently the AppC file (RadioRouteAppC.nc) has been implemented. In the first part the components have been defined and in the second part the wiring with the interfaces has been implemented.

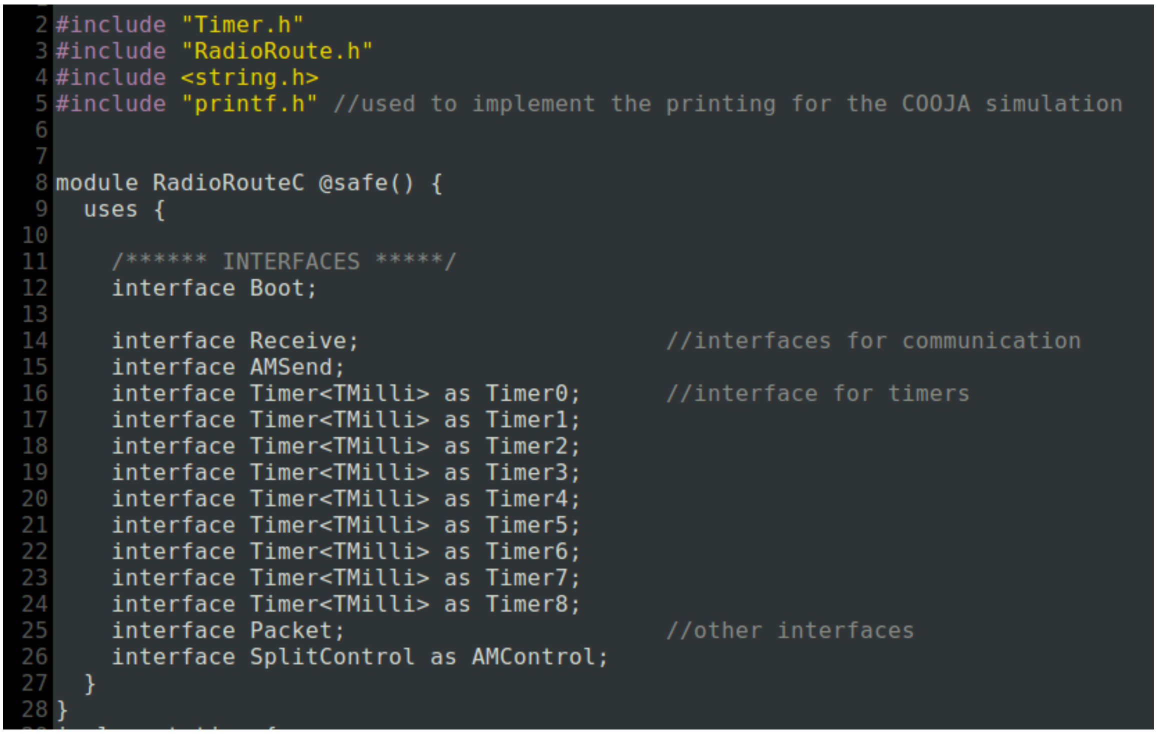


```
#define NEW_PRINTF_SEMANTICS
#include "printf.h"
#include "RadioRoute.h"

configuration RadioRouteAppC {}
implementation {
/****** COMPONENTS *****/

  components MainC, RadioRouteC as App;
  //add the other components here

  components new TimerMilliC() as Timer0;
  components new TimerMilliC() as Timer1;
  components new TimerMilliC() as Timer2;
  components new TimerMilliC() as Timer3;
  components new TimerMilliC() as Timer4;
  components new TimerMilliC() as Timer5;
  components new TimerMilliC() as Timer6;
  components new TimerMilliC() as Timer7;
  components new TimerMilliC() as Timer8;
  components SerialPrintfC;
  components SerialStartC;
  components new AMSenderC(AM_RADIO_COUNT_MSG);
  components new AMReceiverC(AM_RADIO_COUNT_MSG);
  components ActiveMessageC;
```



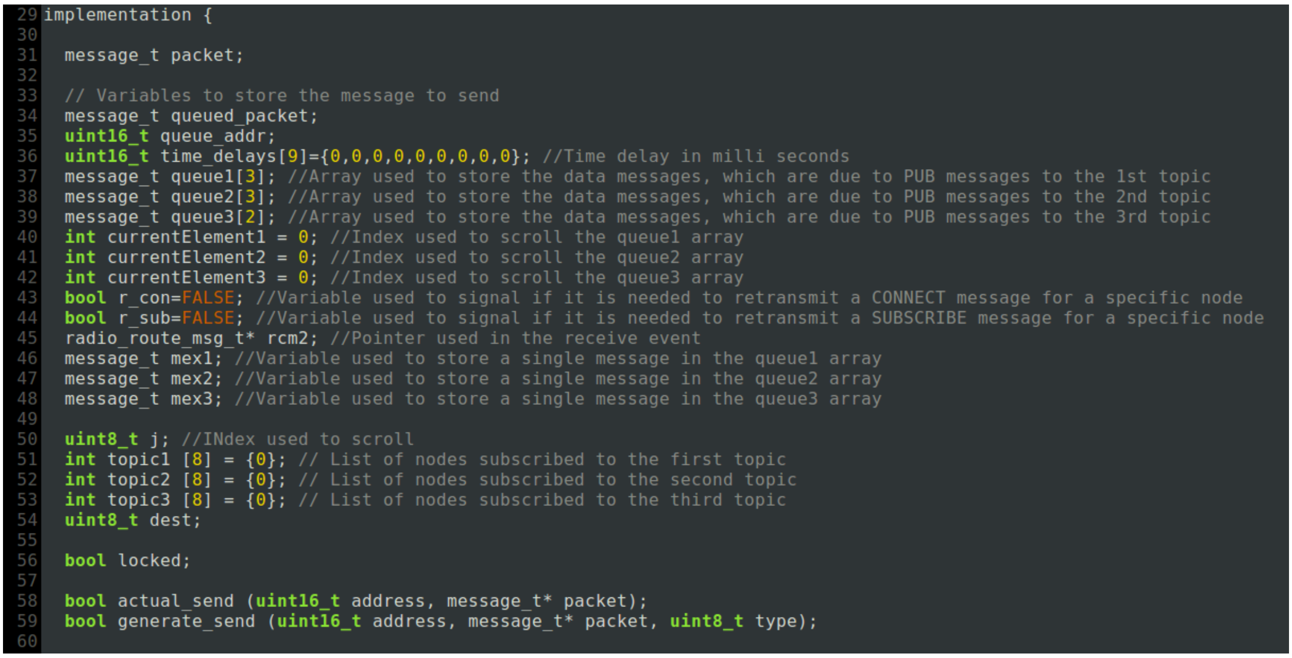
```
  /****** INTERFACES *****/
  //Boot interface
  App.Boot -> MainC.Boot;

  /****** Wire the other interfaces down here *****/
  App.Receive -> AMReceiverC;
  App.AMSend -> AMSenderC;
  App.AMControl -> ActiveMessageC;
  App.Timer0 -> Timer0;
  App.Timer1 -> Timer1;
  App.Timer2 -> Timer2;
  App.Timer3 -> Timer3;
  App.Timer4 -> Timer4;
  App.Timer5 -> Timer5;
  App.Timer6 -> Timer6;
  App.Timer7 -> Timer7;
  App.Timer8 -> Timer8;
  App.Packet -> AMSenderC;
```

Finally, the C.nc file (RadioRouteC.nc) has been implemented.
In the first part of the RadioRouteC.nc file the interfaces have been defined:



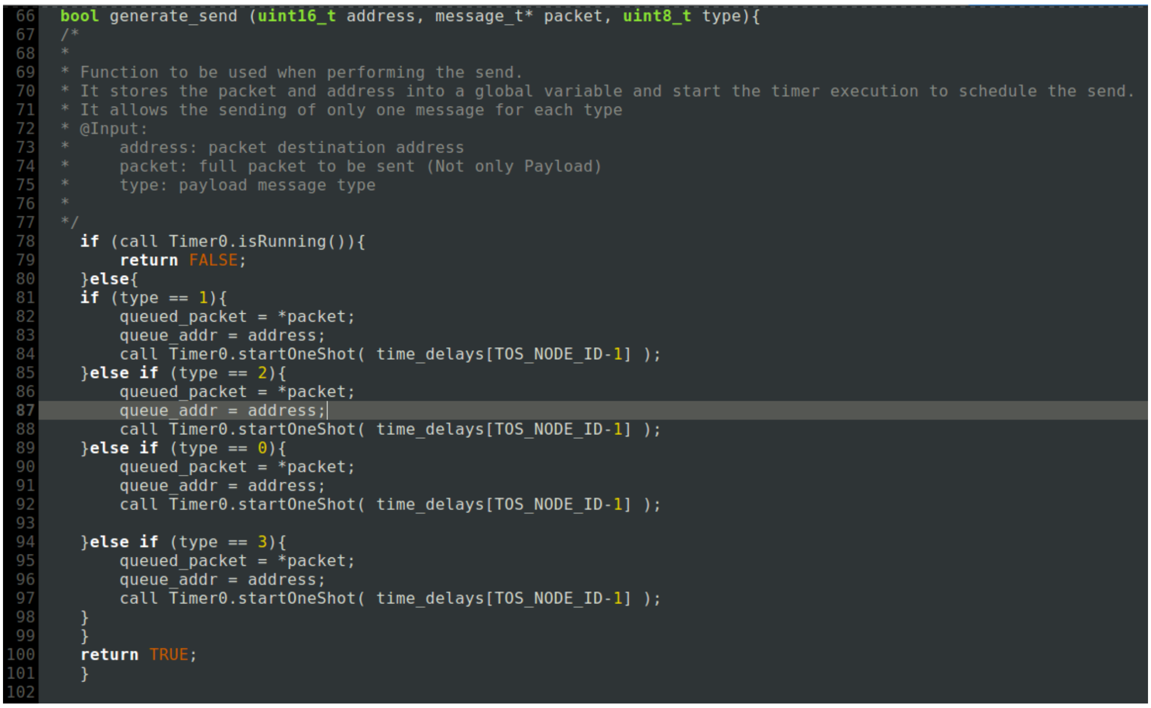
```
#include "Timer.h"
#include "RadioRoute.h"
#include <string.h>
#include "printf.h" //used to implement the printing for the COOJA simulation


module RadioRouteC @safe() {
  uses {

    /****** INTERFACES *****/
    interface Boot;

    interface Receive;                      //interfaces for communication
    interface AMSend;
    interface Timer<TMilli> as Timer0;      //interface for timers
    interface Timer<TMilli> as Timer1;
    interface Timer<TMilli> as Timer2;
    interface Timer<TMilli> as Timer3;
    interface Timer<TMilli> as Timer4;
    interface Timer<TMilli> as Timer5;
    interface Timer<TMilli> as Timer6;
    interface Timer<TMilli> as Timer7;
    interface Timer<TMilli> as Timer8;
    interface Packet;                       //other interfaces
    interface SplitControl as AMControl;
  }
}
```

Then the global variables have been defined:



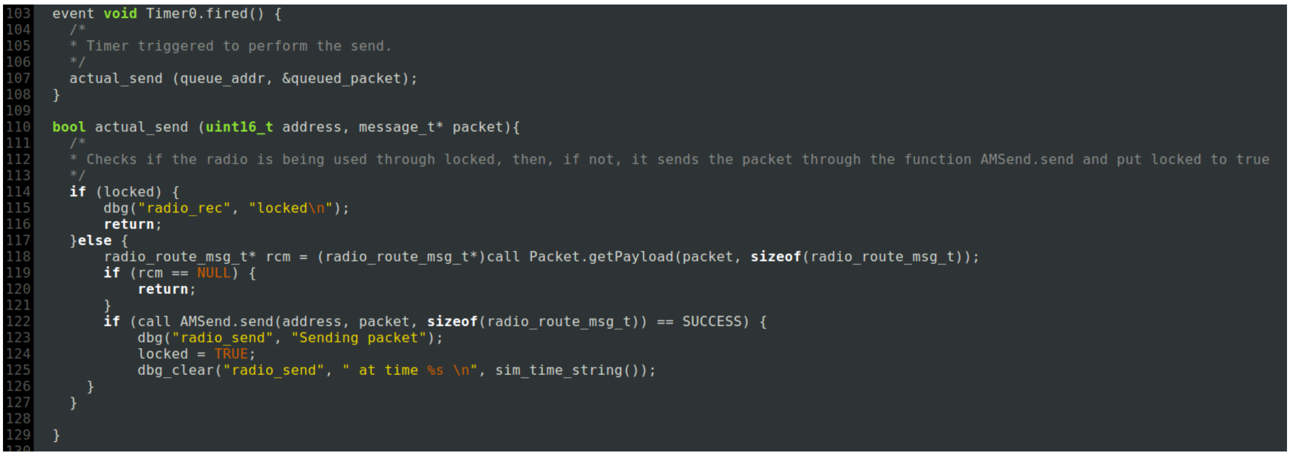
```
implementation {

  message_t packet;

  // Variables to store the message to send
  message_t queued_packet;
  uint16_t queue_addr;
  uint16_t time_delays[9]={0,0,0,0,0,0,0,0,0}; //Time delay in milli seconds
  message_t queue1[3]; //Array used to store the data messages, which are due to PUB messages to the 1st topic
  message_t queue2[3]; //Array used to store the data messages, which are due to PUB messages to the 2nd topic
  message_t queue3[2]; //Array used to store the data messages, which are due to PUB messages to the 3rd topic
  int currentElement1 = 0; //Index used to scroll the queue1 array
  int currentElement2 = 0; //Index used to scroll the queue2 array
  int currentElement3 = 0; //Index used to scroll the queue3 array
  bool r_con=FALSE; //Variable used to signal if it is needed to retransmit a CONNECT message for a specific node
  bool r_sub=FALSE; //Variable used to signal if it is needed to retransmit a SUBSCRIBE message for a specific node
  radio_route_msg_t* rcm2; //Pointer used in the receive event
  message_t mex1; //Variable used to store a single message in the queue1 array
  message_t mex2; //Variable used to store a single message in the queue2 array
  message_t mex3; //Variable used to store a single message in the queue3 array

  uint8_t j; //INdex used to scroll
  int topic1 [8] = {0}; // List of nodes subscribed to the first topic
  int topic2 [8] = {0}; // List of nodes subscribed to the second topic
  int topic3 [8] = {0}; // List of nodes subscribed to the third topic
  uint8_t dest;

  bool locked;

  bool actual_send (uint16_t address, message_t* packet);
  bool generate_send (uint16_t address, message_t* packet, uint8_t type);

```

At this point the first bool function that has been implemented is the **generate_send**, it takes in input the destination of the message, the pointer to the packet that has to be sent and the type of the message. The main goal of this function is to store the packet and address into a global variable and start the timer 0 execution to schedule the send.



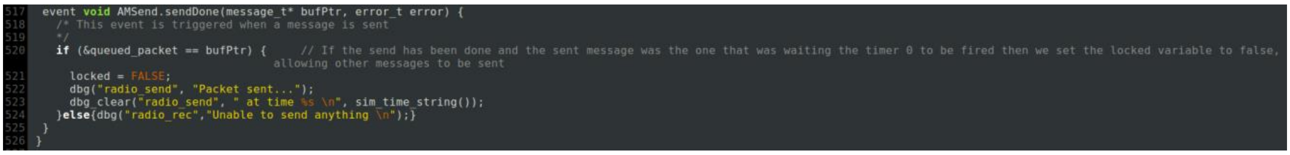
```
bool generate_send (uint16_t address, message_t* packet, uint8_t type){
/*
*
* Function to be used when performing the send.
* It stores the packet and address into a global variable and start the timer execution to schedule the send.
* It allows the sending of only one message for each type
* @Input:
*     address: packet destination address
*     packet: full packet to be sent (Not only Payload)
*     type: payload message type
*
*/
  if (call Timer0.isRunning()){
      return FALSE;
  }else{
  if (type == 1){
      queued_packet = *packet;
      queue_addr = address;
      call Timer0.startOneShot( time_delays[TOS_NODE_ID-1] );
  }else if (type == 2){
      queued_packet = *packet;
      queue_addr = address;
      call Timer0.startOneShot( time_delays[TOS_NODE_ID-1] );
  }else if (type == 0){
      queued_packet = *packet;
      queue_addr = address;
      call Timer0.startOneShot( time_delays[TOS_NODE_ID-1] );

  }else if (type == 3){
      queued_packet = *packet;
      queue_addr = address;
      call Timer0.startOneShot( time_delays[TOS_NODE_ID-1] );
  }
  }
  return TRUE;
  }
```

When the timer 0 is fired the function **actual_send** is called. The main purpose of this function is to check if the radio is being used through the variable locked, then, if not, it sends the packet through the function **AMSend.send** and put locked to true. This function takes as input the address of the destination and the pointer to the message that is being sent.



```
event void Timer0.fired() {
  /*
  * Timer triggered to perform the send.
  */
  actual_send (queue_addr, &queued_packet);
}

bool actual_send (uint16_t address, message_t* packet){
  /*
  * Checks if the radio is being used through locked, then, if not, it sends the packet through the function AMSend.send and put locked to true
  */
  if (locked) {
      dbg("radio_rec", "locked\n");
      return;
  }else {
      radio_route_msg_t* rcm = (radio_route_msg_t*)call Packet.getPayload(packet, sizeof(radio_route_msg_t));
      if (rcm == NULL) {
          return;
      }
      if (call AMSend.send(address, packet, sizeof(radio_route_msg_t)) == SUCCESS) {
          dbg("radio_send", "Sending packet");
          locked = TRUE;
          dbg_clear("radio_send", " at time %s \n", sim_time_string());
    }
  }

}
```

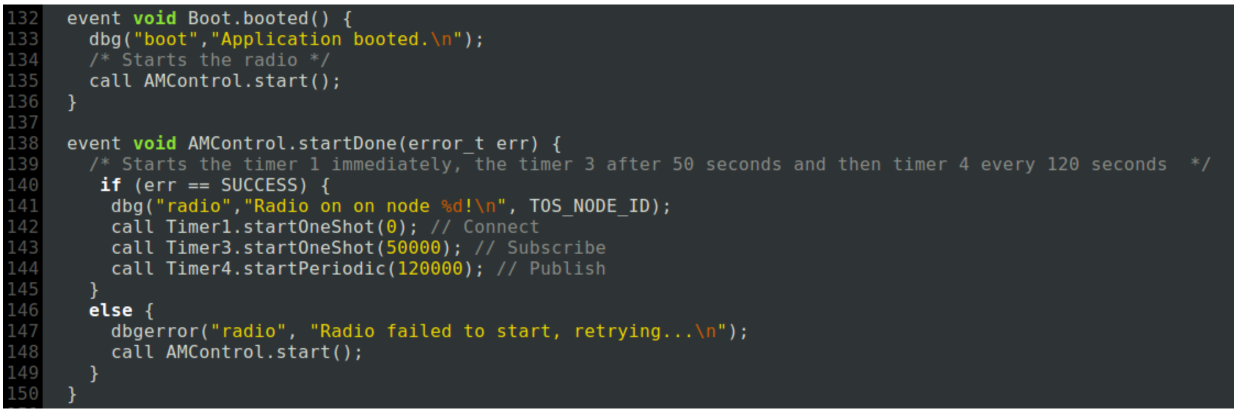
When the message is successfully sent, the event **AMSend.sendDone** is triggered.



```
event void AMSend.sendDone(message_t* bufPtr, error_t error) {
  /* This event is triggered when a message is sent
  */
  if (&queued_packet == bufPtr) {     // If the send has been done and the sent message was the one that was waiting the timer 0 to be fired then we set the locked variable to false,
                                      allowing other messages to be sent
    locked = FALSE;
    dbg("radio_send", "Packet sent...");
    dbg_clear("radio_send", " at time %s \n", sim_time_string());
  }else{dbg("radio_rec","Unable to send anything \n");}
 }
}
```

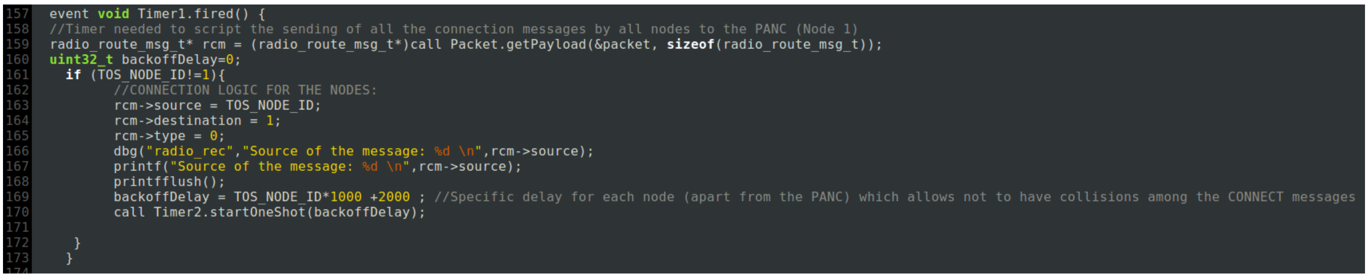
Successively the logic of the events **Boot.booted** and **AMControl.startDone** has been implemented.

For the first event the radio has been started and for the second one the timers 1, 3 and 4 have been started for each node (timer 1 is started immediately, timer 3 is started after 50 seconds and timer 4 is started periodically every 120 seconds).



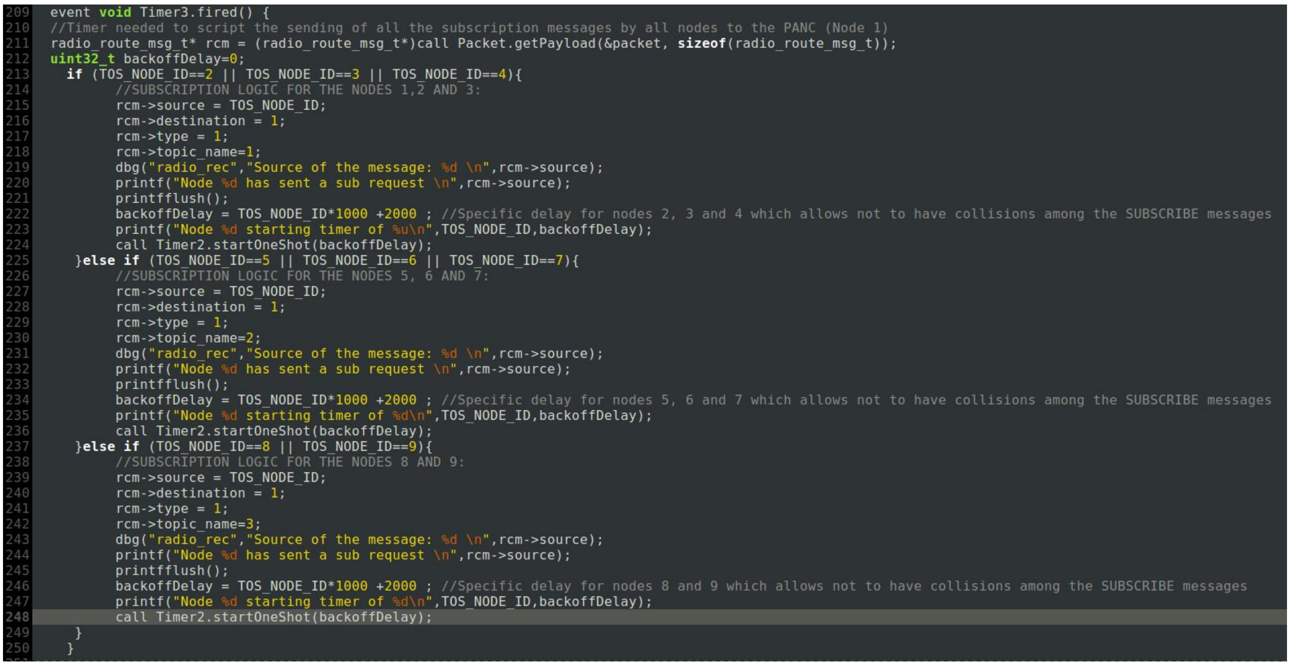
```
event void Boot.booted() {
  dbg("boot","Application booted.\n");
  /* Starts the radio */
  call AMControl.start();
}

event void AMControl.startDone(error_t err) {
  /* Starts the timer 1 immediately, the timer 3 after 50 seconds and then timer 4 every 120 seconds  */
   if (err == SUCCESS) {
    dbg("radio","Radio on on node %d!\n", TOS_NODE_ID);
    call Timer1.startOneShot(0); // Connect
    call Timer3.startOneShot(50000); // Subscribe
    call Timer4.startPeriodic(120000); // Publish
  }
  else {
    dbgerror("radio", "Radio failed to start, retrying...\n");
    call AMControl.start();
  }
}
```

As far as the logic of timer 1 is concerned, it is used to script the sending of all the CONNECT messages from all the nodes apart node 1 (that is the PANC) to the PANC. In order to avoid collisions, the call to timer 2 (the timer for effectively sending the messages) has been performed at different times for each node, using the local variable backoffDelay.



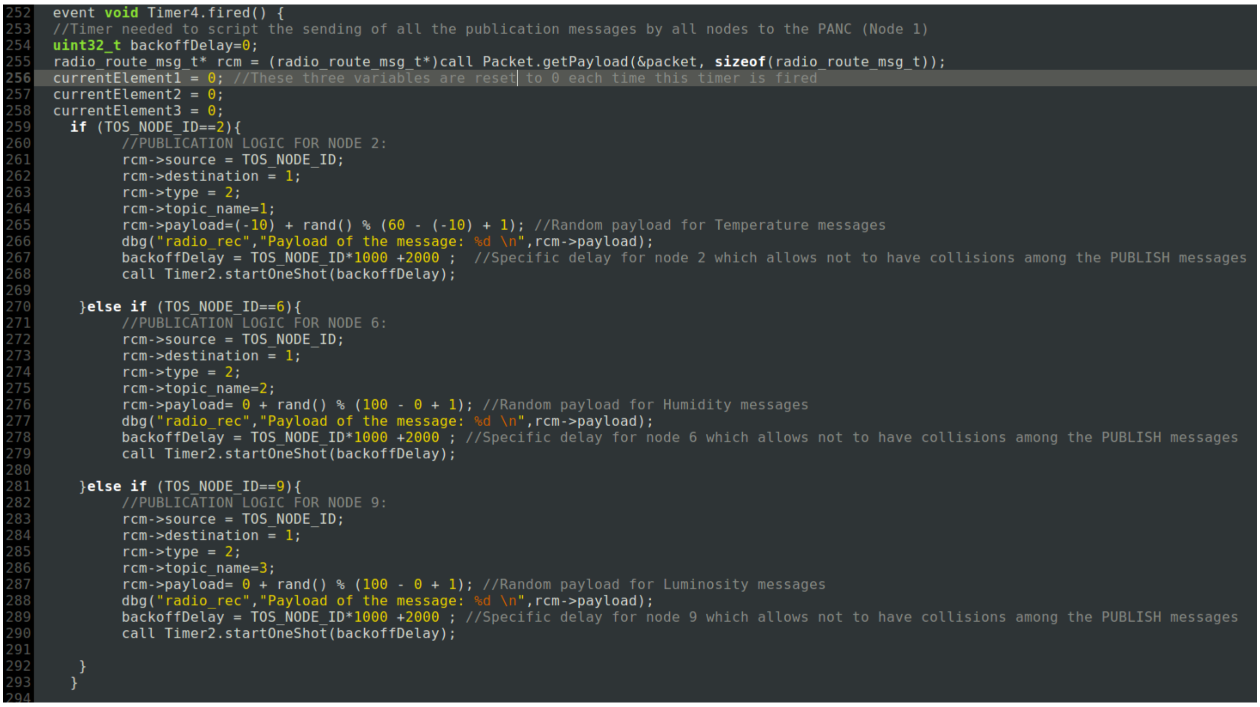
```
event void Timer1.fired() {
//Timer needed to script the sending of all the connection messages by all nodes to the PANC (Node 1)
radio_route_msg_t* rcm = (radio_route_msg_t*)call Packet.getPayload(&packet, sizeof(radio_route_msg_t));
uint32_t backoffDelay=0;
  if (TOS_NODE_ID!=1){
        //CONNECTION LOGIC FOR THE NODES:
        rcm->source = TOS_NODE_ID;
        rcm->destination = 1;
        rcm->type = 0;
        dbg("radio_rec","Source of the message: %d \n",rcm->source);
        printf("Source of the message: %d \n",rcm->source);
        printfflush();
        backoffDelay = TOS_NODE_ID*1000 +2000 ; //Specific delay for each node (apart from the PANC) which allows not to have collisions among the CONNECT messages
        call Timer2.startOneShot(backoffDelay);

   }
  }
```

For what regards the logic of timer 3, it has been used to script the sending of all the SUBSCRIBE messages from all the nodes apart node 1 to the PANC. Nodes 2, 3 and 4 are subscribing to topic 1 (TEMPERATURE); nodes 5, 6 and 7 are subscribing to topic 2 (HUMIDITY) and nodes 8, 9 are subscribing to topic 3 (LUMINOSITY). The same logic of timer 1 for avoiding collisions has been applied.



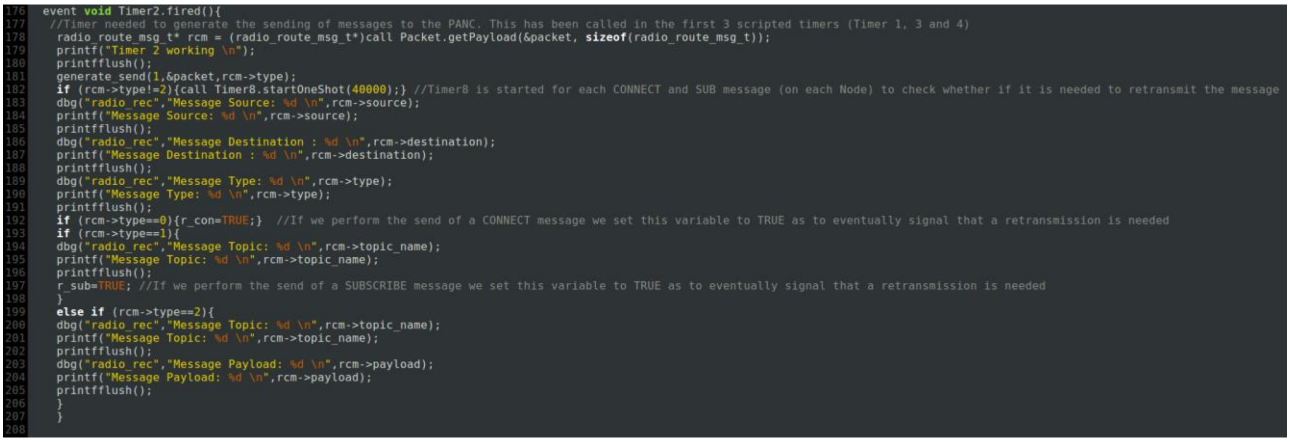
```
event void Timer3.fired() {
//Timer needed to script the sending of all the subscription messages by all nodes to the PANC (Node 1)
radio_route_msg_t* rcm = (radio_route_msg_t*)call Packet.getPayload(&packet, sizeof(radio_route_msg_t));
uint32_t backoffDelay=0;
  if (TOS_NODE_ID==2 || TOS_NODE_ID==3 || TOS_NODE_ID==4){
        //SUBSCRIPTION LOGIC FOR THE NODES 1,2 AND 3:
        rcm->source = TOS_NODE_ID;
        rcm->destination = 1;
        rcm->type = 1;
        rcm->topic_name=1;
        dbg("radio_rec","Source of the message: %d \n",rcm->source);
        printf("Node %d has sent a sub request \n",rcm->source);
        printfflush();
        backoffDelay = TOS_NODE_ID*1000 +2000 ; //Specific delay for nodes 2, 3 and 4 which allows not to have collisions among the SUBSCRIBE messages
        printf("Node %d starting timer of %u\n",TOS_NODE_ID,backoffDelay);
        call Timer2.startOneShot(backoffDelay);
   }else if (TOS_NODE_ID==5 || TOS_NODE_ID==6 || TOS_NODE_ID==7){
        //SUBSCRIPTION LOGIC FOR THE NODES 5, 6 AND 7:
        rcm->source = TOS_NODE_ID;
        rcm->destination = 1;
        rcm->type = 1;
        rcm->topic_name=2;
        dbg("radio_rec","Source of the message: %d \n",rcm->source);
        printf("Node %d has sent a sub request \n",rcm->source);
        printfflush();
        backoffDelay = TOS_NODE_ID*1000 +2000 ; //Specific delay for nodes 5, 6 and 7 which allows not to have collisions among the SUBSCRIBE messages
        printf("Node %d starting timer of %d\n",TOS_NODE_ID,backoffDelay);
        call Timer2.startOneShot(backoffDelay);
   }else if (TOS_NODE_ID==8 || TOS_NODE_ID==9){
        //SUBSCRIPTION LOGIC FOR THE NODES 8 AND 9:
        rcm->source = TOS_NODE_ID;
        rcm->destination = 1;
        rcm->type = 1;
        rcm->topic_name=3;
        dbg("radio_rec","Source of the message: %d \n",rcm->source);
        printf("Node %d has sent a sub request \n",rcm->source);
        printfflush();
        backoffDelay = TOS_NODE_ID*1000 +2000 ; //Specific delay for nodes 8 and 9 which allows not to have collisions among the SUBSCRIBE messages
        printf("Node %d starting timer of %d\n",TOS_NODE_ID,backoffDelay);
        call Timer2.startOneShot(backoffDelay);
   }
  }
```

For what concerns the logic of timer 4, it has been used to script the sending of all the PUBLISH messages from nodes 2, 6 and 9 to the PANC. Node 2 publishes messages with a random payload to topic 1, node 6 publishes messages with a random payload to topic 2 and node 9 publishes messages with a random payload to topic 3. The variables currentElement1, currentElement2 and currentElement3 are reset to 0 as to guarantee the correct functioning of timers 5, 6 and 7.



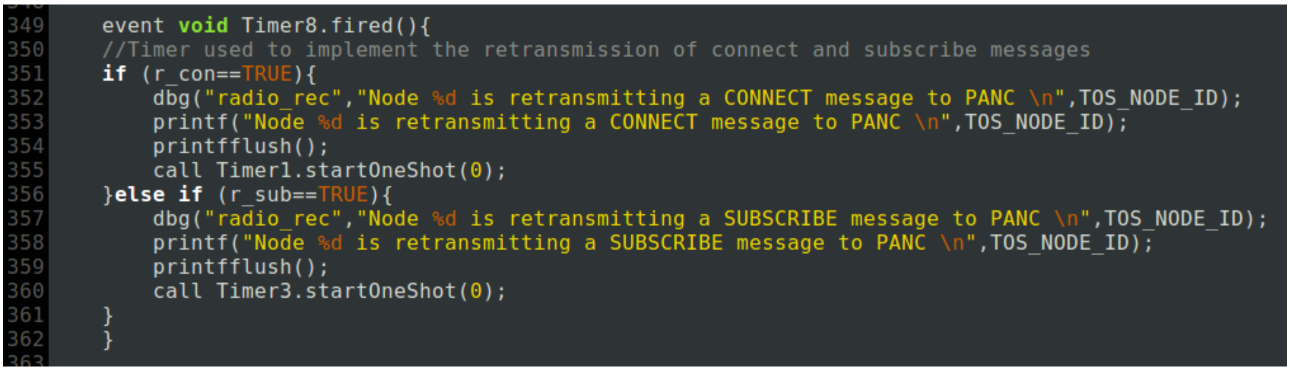
```
 event void Timer4.fired() {
 //Timer needed to script the sending of all the publication messages by all nodes to the PANC (Node 1)
 uint32_t backoffDelay=0;
 radio_route_msg_t* rcm = (radio_route_msg_t*)call Packet.getPayload(&packet, sizeof(radio_route_msg_t));
 currentElement1 = 0; //These three variables are reset to 0 each time this timer is fired
 currentElement2 = 0;
 currentElement3 = 0;
   if (TOS_NODE_ID==2){
         //PUBLICATION LOGIC FOR NODE 2:
         rcm->source = TOS_NODE_ID;
         rcm->destination = 1;
         rcm->type = 2;
         rcm->topic_name=1;
         rcm->payload=(-10) + rand() % (60 - (-10) + 1); //Random payload for Temperature messages
         dbg("radio_rec","Payload of the message: %d \n",rcm->payload);
         backoffDelay = TOS_NODE_ID*1000 +2000 ;  //Specific delay for node 2 which allows not to have collisions among the PUBLISH messages
         call Timer2.startOneShot(backoffDelay);

    }else if (TOS_NODE_ID==6){
         //PUBLICATION LOGIC FOR NODE 6:
         rcm->source = TOS_NODE_ID;
         rcm->destination = 1;
         rcm->type = 2;
         rcm->topic_name=2;
         rcm->payload= 0 + rand() % (100 - 0 + 1); //Random payload for Humidity messages
         dbg("radio_rec","Payload of the message: %d \n",rcm->payload);
         backoffDelay = TOS_NODE_ID*1000 +2000 ; //Specific delay for node 6 which allows not to have collisions among the PUBLISH messages
         call Timer2.startOneShot(backoffDelay);

    }else if (TOS_NODE_ID==9){
         //PUBLICATION LOGIC FOR NODE 9:
         rcm->source = TOS_NODE_ID;
         rcm->destination = 1;
         rcm->type = 2;
         rcm->topic_name=3;
         rcm->payload= 0 + rand() % (100 - 0 + 1); //Random payload for Luminosity messages
         dbg("radio_rec","Payload of the message: %d \n",rcm->payload);
         backoffDelay = TOS_NODE_ID*1000 +2000 ; //Specific delay for node 9 which allows not to have collisions among the PUBLISH messages
         call Timer2.startOneShot(backoffDelay);

    }
   }
```

As previously said timer 2 has been used to send messages to the PANC. In addition to this the timer has also been used to manage the logic of retransmissions, in fact when the sending has been performed if the sent message was of type 0 (CONNECT message) the variable r_con was set to true, underlying that could be necessary to retransmit the message; the same logic is used for the transmission of SUBSCRIBE messages, but in this case the variable used to indicate that could be necessary to retransmit the message is r_sub. Moreover, the timer 8 has also been started (only for CONNECT and SUB messages) to effectively check if the retransmission is needed.



```
 event void Timer2.fired(){
  //Timer needed to generate the sending of messages to the PANC. This has been called in the first 3 scripted timers (Timer 1, 3 and 4)
   radio_route_msg_t* rcm = (radio_route_msg_t*)call Packet.getPayload(&packet, sizeof(radio_route_msg_t));
   printf("Timer 2 working \n");
   printfflush();
   generate_send(1,&packet,rcm->type);
   if (rcm->type!=2){call Timer8.startOneShot(40000);} //Timer8 is started for each CONNECT and SUB message (on each Node) to check whether if it is needed to retransmit the message
   dbg("radio_rec","Message Source: %d \n",rcm->source);
   printf("Message Source: %d \n",rcm->source);
   printfflush();
   dbg("radio_rec","Message Destination : %d \n",rcm->destination);
   printf("Message Destination : %d \n",rcm->destination);
   printfflush();
   dbg("radio_rec","Message Type: %d \n",rcm->type);
   printf("Message Type: %d \n",rcm->type);
   printfflush();
   if (rcm->type==0){r_con=TRUE;}  //If we perform the send of a CONNECT message we set this variable to TRUE as to eventually signal that a retransmission is needed
   if (rcm->type==1){
   dbg("radio_rec","Message Topic: %d \n",rcm->topic_name);
   printf("Message Topic: %d \n",rcm->topic_name);
   printfflush();
   r_sub=TRUE; //If we perform the send of a SUBSCRIBE message we set this variable to TRUE as to eventually signal that a retransmission is needed
   }
   else if (rcm->type==2){
   dbg("radio_rec","Message Topic: %d \n",rcm->topic_name);
   printf("Message Topic: %d \n",rcm->topic_name);
   printfflush();
   dbg("radio_rec","Message Payload: %d \n",rcm->payload);
   printf("Message Payload: %d \n",rcm->payload);
   printfflush();
   }
   }
```

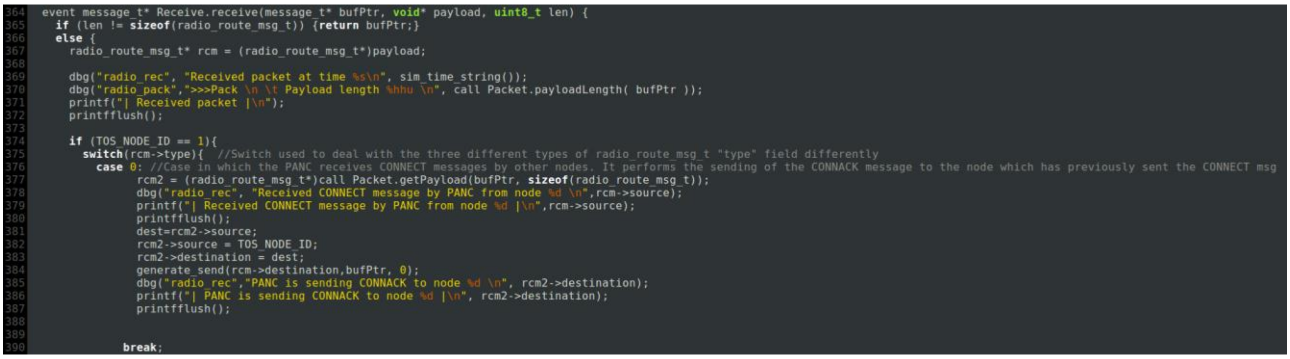
Timer 8, which has been called from each node which has performed a send of a CONNECT or SUB message, checks whether if the variable r_con is true or false (if the timer has been started during the sending of a CONNECT message) or if the variable r_sub is true or false (if the timer has been started during the sending of a SUB message) for the considered node. If one of those two variables is true, the retransmission is performed by calling the timer that scripts the sending of the desired message only for the specified node.



```
    event void Timer8.fired(){
    //Timer used to implement the retransmission of connect and subscribe messages
    if (r_con==TRUE){
        dbg("radio_rec","Node %d is retransmitting a CONNECT message to PANC \n",TOS_NODE_ID);
        printf("Node %d is retransmitting a CONNECT message to PANC \n",TOS_NODE_ID);
        printfflush();
        call Timer1.startOneShot(0);
    }else if (r_sub==TRUE){
        dbg("radio_rec","Node %d is retransmitting a SUBSCRIBE message to PANC \n",TOS_NODE_ID);
        printf("Node %d is retransmitting a SUBSCRIBE message to PANC \n",TOS_NODE_ID);
        printfflush();
        call Timer3.startOneShot(0);
    }
    }
```

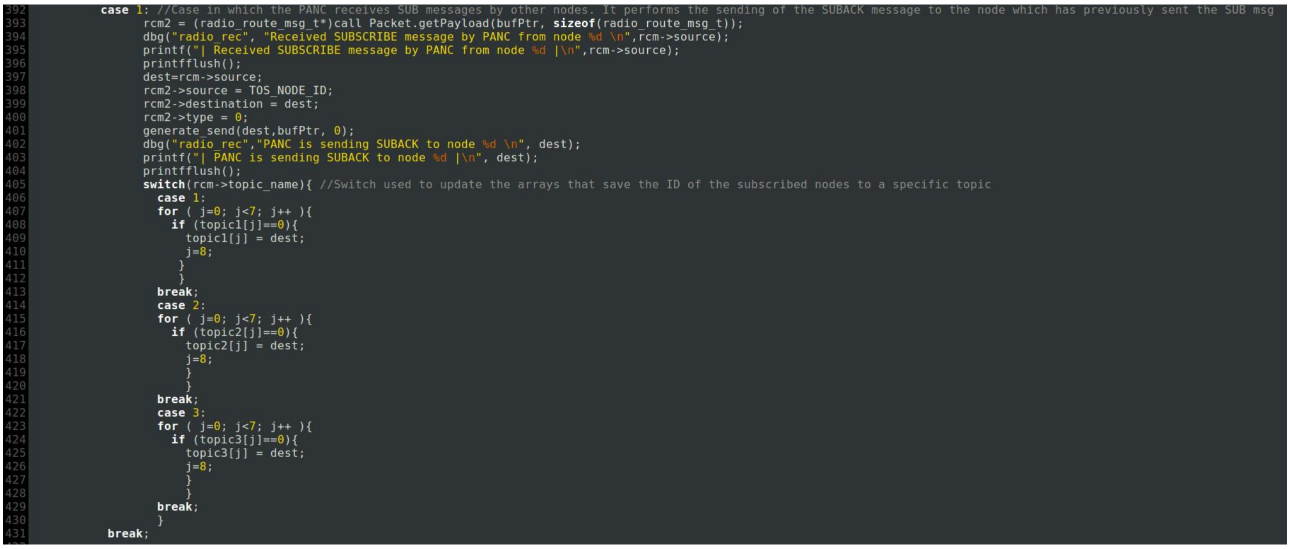
As far as the reception of messages is concerned, the logic has been mainly divided in two parts: the reception of messages from the PANC and the reception of messages from all the other nodes. In the first part an additional division has been done:

- the reception of CONNECT messages, where, when the PANC receives a CONNECT message, it sends back to the source of the message a CONNACK.



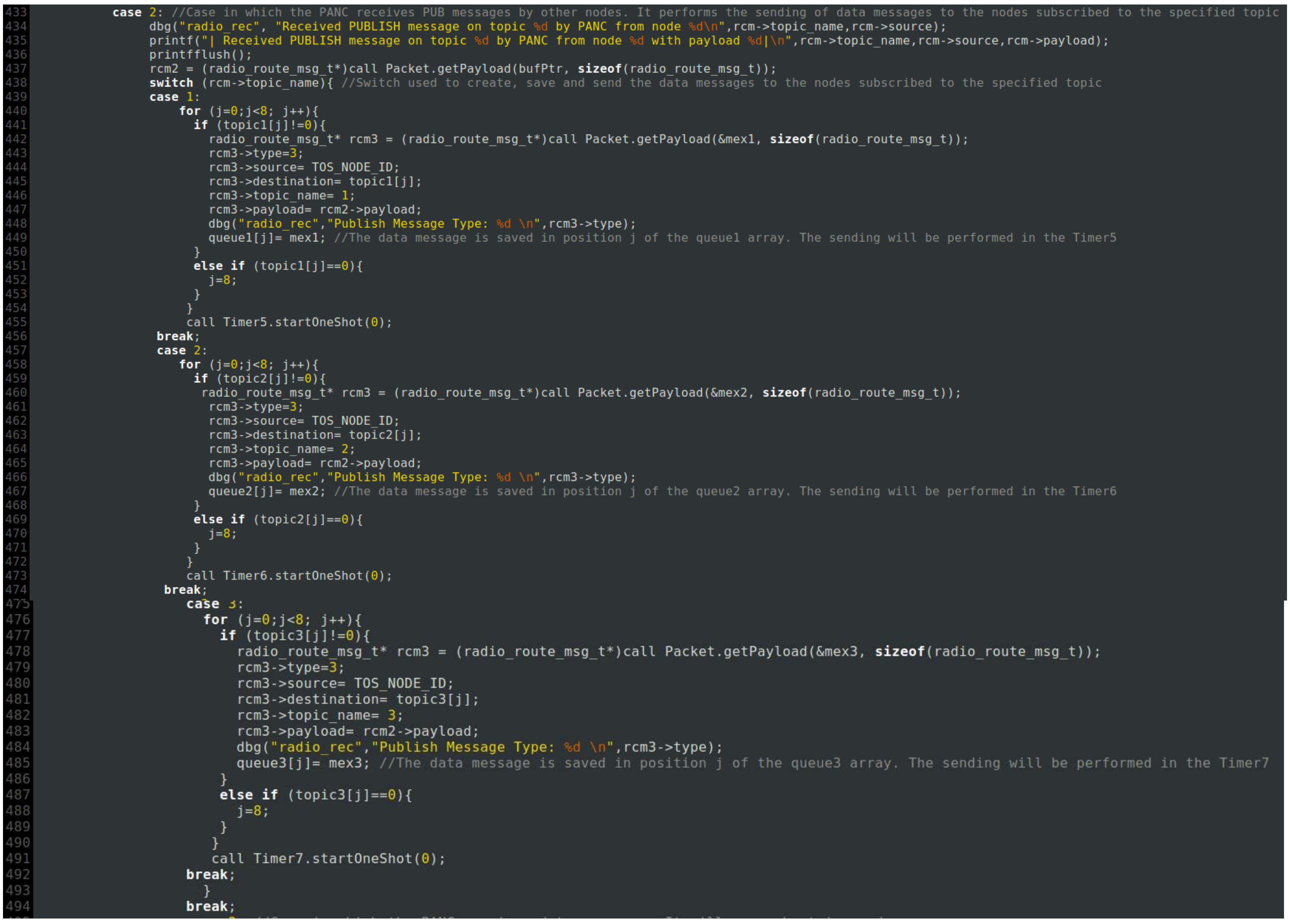
```
event message_t* Receive.receive(message_t* bufPtr, void* payload, uint8_t len) {
  if (len != sizeof(radio_route_msg_t)) {return bufPtr;}
  else {
    radio_route_msg_t* rcm = (radio_route_msg_t*)payload;

    dbg("radio_rec", "Received packet at time %s\n", sim_time_string());
    dbg("radio_pack",">>>Pack \n \t Payload length %hhu \n", call Packet.payloadLength( bufPtr ));
    printf("| Received packet |\n");
    printfflush();

    if (TOS_NODE_ID == 1){
      switch(rcm->type){  //Switch used to deal with the three different types of radio_route_msg_t "type" field differently
        case 0: //Case in which the PANC receives CONNECT messages by other nodes. It performs the sending of the CONNACK message to the node which has previously sent the CONNECT msg
              rcm2 = (radio_route_msg_t*)call Packet.getPayload(bufPtr, sizeof(radio_route_msg_t));
              dbg("radio_rec", "Received CONNECT message by PANC from node %d \n",rcm->source);
              printf("| Received CONNECT message by PANC from node %d |\n",rcm->source);
              printfflush();
              dest=rcm2->source;
              rcm2->source = TOS_NODE_ID;
              rcm2->destination = dest;
              generate_send(rcm->destination,bufPtr, 0);
              dbg("radio_rec","PANC is sending CONNACK to node %d \n", rcm2->destination);
              printf("| PANC is sending CONNACK to node %d |\n", rcm2->destination);
              printfflush();


            break;
```

- the reception of SUBSCRIBE messages where, when the PANC receives a SUB message, it sends back to the source of the message a SUBACK. In addition to this a switch has been used to update the arrays that save the ID of the subscribed nodes to a specific topic.



```
        case 1: //Case in which the PANC receives SUB messages by other nodes. It performs the sending of the SUBACK message to the node which has previously sent the SUB msg
              rcm2 = (radio_route_msg_t*)call Packet.getPayload(bufPtr, sizeof(radio_route_msg_t));
              dbg("radio_rec", "Received SUBSCRIBE message by PANC from node %d \n",rcm->source);
              printf("| Received SUBSCRIBE message by PANC from node %d |\n",rcm->source);
              printfflush();
              dest=rcm->source;
              rcm2->source = TOS_NODE_ID;
              rcm2->destination = dest;
              rcm2->type = 0;
              generate_send(dest,bufPtr, 0);
              dbg("radio_rec","PANC is sending SUBACK to node %d \n", dest);
              printf("| PANC is sending SUBACK to node %d |\n", dest);
              printfflush();
              switch(rcm->topic_name){ //Switch used to update the arrays that save the ID of the subscribed nodes to a specific topic
                case 1:
                for ( j=0; j<7; j++ ){
                  if (topic1[j]==0){
                    topic1[j] = dest;
                    j=8;
                   }
                   }
                break;
                case 2:
                for ( j=0; j<7; j++ ){
                  if (topic2[j]==0){
                    topic2[j] = dest;
                    j=8;
                    }
                    }
                break;
                case 3:
                for ( j=0; j<7; j++ ){
                  if (topic3[j]==0){
                    topic3[j] = dest;
                    j=8;
                    }
                    }
                break;
                }
         break;
```

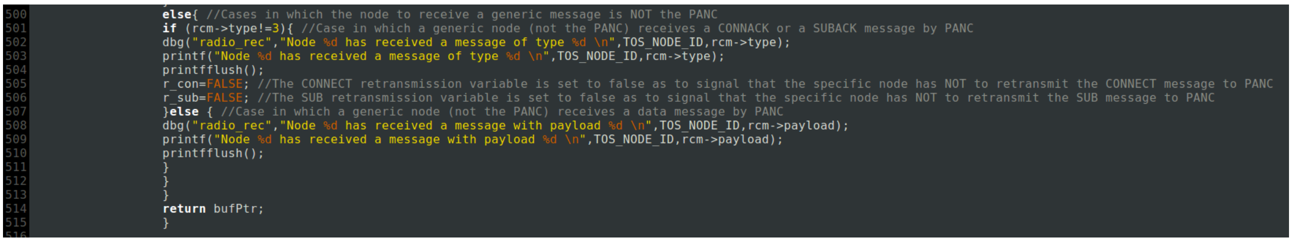
- The reception of PUBLISH messages, where, when the PANC receives a PUB message to a specific topic, it sends data messages containing the received payload to all the nodes subscribed to that specific topic. This has been achieved by saving the data messages that have to be sent in an array. At the end of this operation the specific timer (timer 5 for topic 1, timer 6 for topic 2 and timer 7 for topic 3) for sending those messages is started.



```
          case 2: //Case in which the PANC receives PUB messages by other nodes. It performs the sending of data messages to the nodes subscribed to the specified topic
               dbg("radio_rec", "Received PUBLISH message on topic %d by PANC from node %d\n",rcm->topic_name,rcm->source);
               printf("| Received PUBLISH message on topic %d by PANC from node %d with payload %d|\n",rcm->topic_name,rcm->source,rcm->payload);
               printfflush();
               rcm2 = (radio_route_msg_t*)call Packet.getPayload(bufPtr, sizeof(radio_route_msg_t));
               switch (rcm->topic_name){ //Switch used to create, save and send the data messages to the nodes subscribed to the specified topic
               case 1:
                   for (j=0;j<8; j++){
                     if (topic1[j]!=0){
                       radio_route_msg_t* rcm3 = (radio_route_msg_t*)call Packet.getPayload(&mex1, sizeof(radio_route_msg_t));
                       rcm3->type=3;
                       rcm3->source= TOS_NODE_ID;
                       rcm3->destination= topic1[j];
                       rcm3->topic_name= 1;
                       rcm3->payload= rcm2->payload;
                       dbg("radio_rec","Publish Message Type: %d \n",rcm3->type);
                       queue1[j]= mex1; //The data message is saved in position j of the queue1 array. The sending will be performed in the Timer5
                     }
                     else if (topic1[j]==0){
                       j=8;
                     }
                    }
                    call Timer5.startOneShot(0);
                break;
                case 2:
                   for (j=0;j<8; j++){
                     if (topic2[j]!=0){
                      radio_route_msg_t* rcm3 = (radio_route_msg_t*)call Packet.getPayload(&mex2, sizeof(radio_route_msg_t));
                       rcm3->type=3;
                       rcm3->source= TOS_NODE_ID;
                       rcm3->destination= topic2[j];
                       rcm3->topic_name= 2;
                       rcm3->payload= rcm2->payload;
                       dbg("radio_rec","Publish Message Type: %d \n",rcm3->type);
                       queue2[j]= mex2; //The data message is saved in position j of the queue2 array. The sending will be performed in the Timer6
                     }
                     else if (topic2[j]==0){
                       j=8;
                     }
                    }
                    call Timer6.startOneShot(0);
                 break;
                    case 3:
                      for (j=0;j<8; j++){
                        if (topic3[j]!=0){
                          radio_route_msg_t* rcm3 = (radio_route_msg_t*)call Packet.getPayload(&mex3, sizeof(radio_route_msg_t));
                          rcm3->type=3;
                          rcm3->source= TOS_NODE_ID;
                          rcm3->destination= topic3[j];
                          rcm3->topic_name= 3;
                          rcm3->payload= rcm2->payload;
                          dbg("radio_rec","Publish Message Type: %d \n",rcm3->type);
                          queue3[j]= mex3; //The data message is saved in position j of the queue3 array. The sending will be performed in the Timer7
                        }
                        else if (topic3[j]==0){
                          j=8;
                        }
                       }
                       call Timer7.startOneShot(0);
                    break;
                      }
                    break;
```

The second part of the receive event, where all the nodes apart from the PANC receive messages, is also divided in two parts:

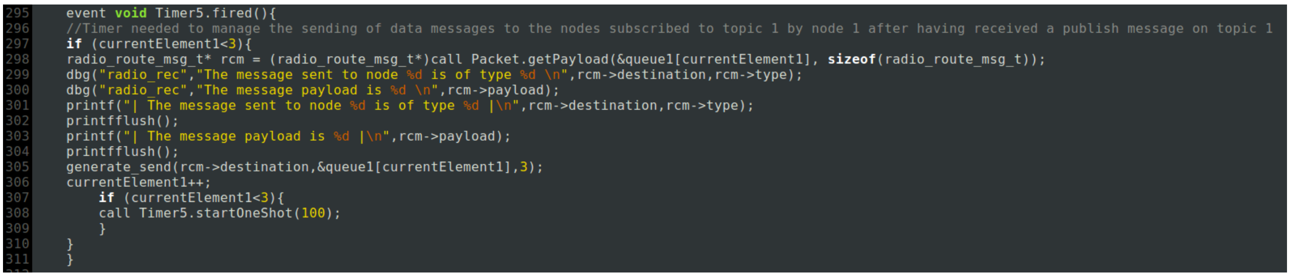
- the first one where a node receives an ACK. In this case the variables r_con and r_sub have been set to FALSE as to signal that there is no need to retransmit for that specified node.
- the second one where a node receives a data message.



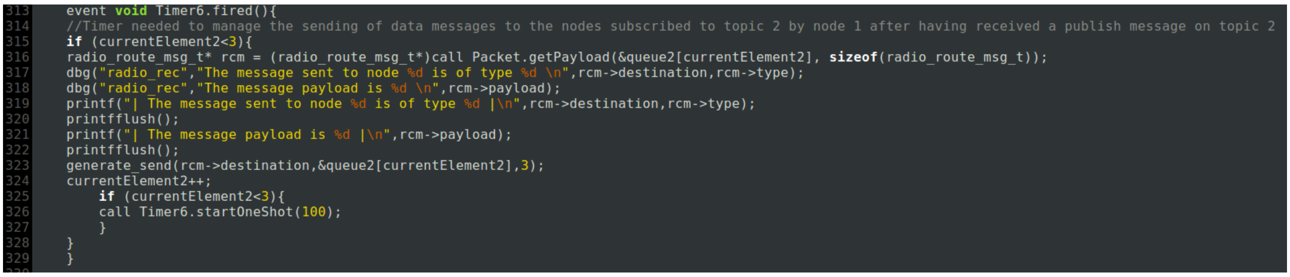
```
                    else{ //Cases in which the node to receive a generic message is NOT the PANC
                    if (rcm->type!=3){ //Case in which a generic node (not the PANC) receives a CONNACK or a SUBACK message by PANC
                    dbg("radio_rec","Node %d has received a message of type %d \n",TOS_NODE_ID,rcm->type);
                    printf("Node %d has received a message of type %d \n",TOS_NODE_ID,rcm->type);
                    printfflush();
                    r_con=FALSE; //The CONNECT retransmission variable is set to false as to signal that the specific node has NOT to retransmit the CONNECT message to PANC
                    r_sub=FALSE; //The SUB retransmission variable is set to false as to signal that the specific node has NOT to retransmit the SUB message to PANC
                    }else { //Case in which a generic node (not the PANC) receives a data message by PANC
                    dbg("radio_rec","Node %d has received a message with payload %d \n",TOS_NODE_ID,rcm->payload);
                    printf("Node %d has received a message with payload %d \n",TOS_NODE_ID,rcm->payload);
                    printfflush();
                    }
                    }
                    }
                    return bufPtr;
                    }
```

As previously said timer 5, timer 6 and timer 7 have been used to send data messages to nodes subscribed to the three topics.

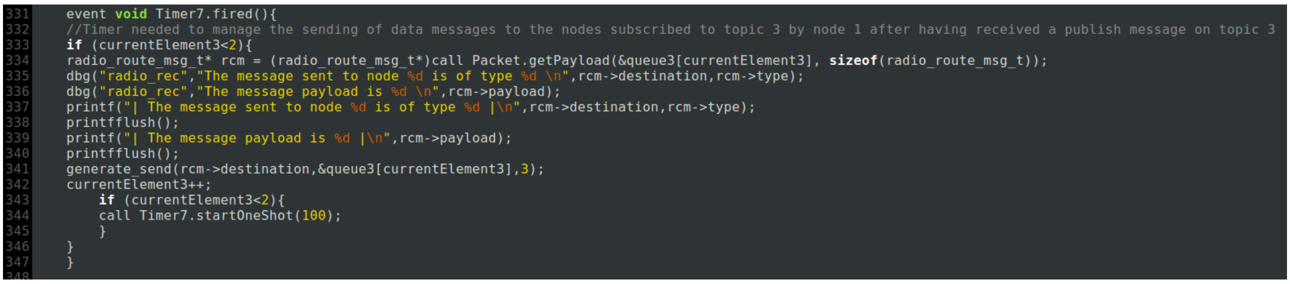
All these timers have been implemented as follows: the array of the stored data messages is scrolled through the currentElement variables, and each message is sent to the right node.



```
    event void Timer5.fired(){
    //Timer needed to manage the sending of data messages to the nodes subscribed to topic 1 by node 1 after having received a publish message on topic 1
    if (currentElement1<3){
    radio_route_msg_t* rcm = (radio_route_msg_t*)call Packet.getPayload(&queue1[currentElement1], sizeof(radio_route_msg_t));
    dbg("radio_rec","The message sent to node %d is of type %d \n",rcm->destination,rcm->type);
    dbg("radio_rec","The message payload is %d \n",rcm->payload);
    printf("| The message sent to node %d is of type %d |\n",rcm->destination,rcm->type);
    printfflush();
    printf("| The message payload is %d |\n",rcm->payload);
    printfflush();
    generate_send(rcm->destination,&queue1[currentElement1],3);
    currentElement1++;
        if (currentElement1<3){
        call Timer5.startOneShot(100);
        }
    }
    }
```

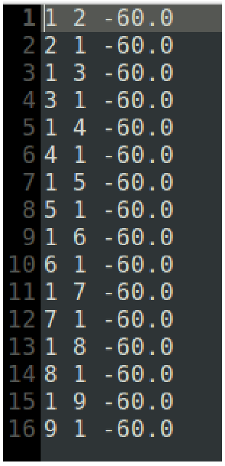


```
	event void Timer6.fired(){
	//Timer needed to manage the sending of data messages to the nodes subscribed to topic 2 by node 1 after having received a publish message on topic 2
	if (currentElement2<3){
	radio_route_msg_t* rcm = (radio_route_msg_t*)call Packet.getPayload(&queue2[currentElement2], sizeof(radio_route_msg_t));
	dbg("radio_rec","The message sent to node %d is of type %d \n",rcm->destination,rcm->type);
	dbg("radio_rec","The message payload is %d \n",rcm->payload);
	printf("| The message sent to node %d is of type %d |\n",rcm->destination,rcm->type);
	printfflush();
	printf("| The message payload is %d |\n",rcm->payload);
	printfflush();
	generate_send(rcm->destination,&queue2[currentElement2],3);
	currentElement2++;
		if (currentElement2<3){
		call Timer6.startOneShot(100);
		}
	}
	}
```



```
	event void Timer7.fired(){
	//Timer needed to manage the sending of data messages to the nodes subscribed to topic 3 by node 1 after having received a publish message on topic 3
	if (currentElement3<2){
	radio_route_msg_t* rcm = (radio_route_msg_t*)call Packet.getPayload(&queue3[currentElement3], sizeof(radio_route_msg_t));
	dbg("radio_rec","The message sent to node %d is of type %d \n",rcm->destination,rcm->type);
	dbg("radio_rec","The message payload is %d \n",rcm->payload);
	printf("| The message sent to node %d is of type %d |\n",rcm->destination,rcm->type);
	printfflush();
	printf("| The message payload is %d |\n",rcm->payload);
	printfflush();
	generate_send(rcm->destination,&queue3[currentElement3],3);
	currentElement3++;
		if (currentElement3<2){
		call Timer7.startOneShot(100);
		}
	}
	}
```

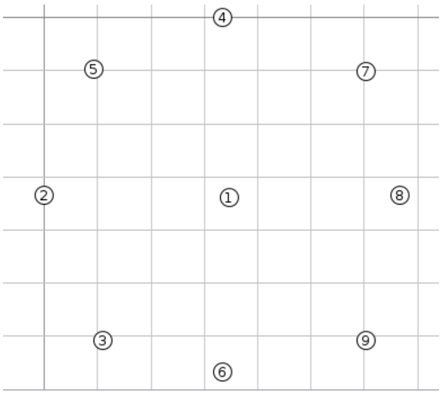
The topology considered for the simulation is a star shape one which consists in 9 nodes, where the first one is the PANC, the centre of the star.



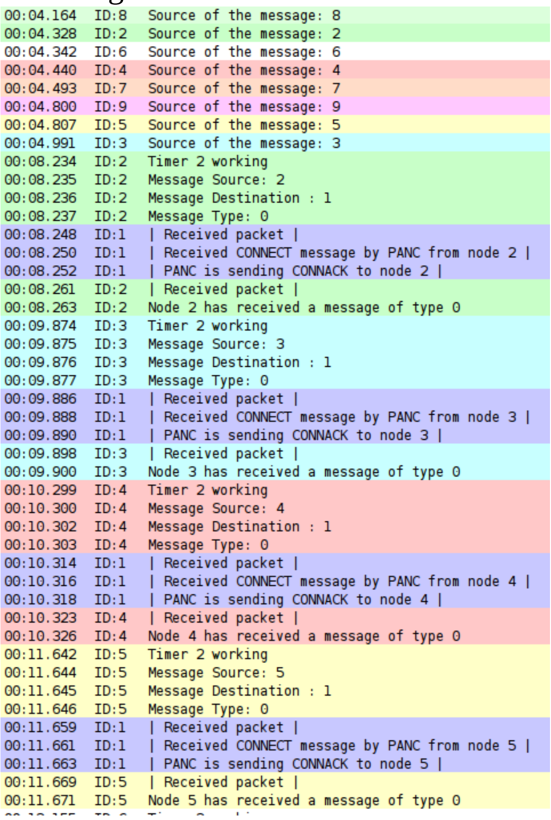
```
1 2 -60.0
2 1 -60.0
1 3 -60.0
3 1 -60.0
1 4 -60.0
4 1 -60.0
1 5 -60.0
5 1 -60.0
1 6 -60.0
6 1 -60.0
1 7 -60.0
7 1 -60.0
1 8 -60.0
8 1 -60.0
1 9 -60.0
9 1 -60.0
```

COOJA SIMULATION

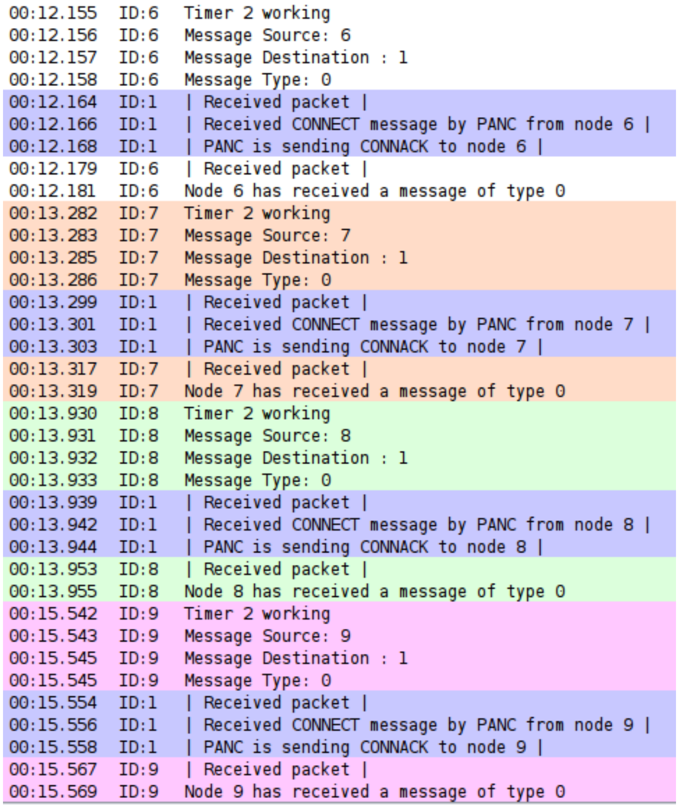
For simulating the logic of the TinyOs code the Cooja simulator has been used.
First of all, it has been needed to create a new simulation and then the file needed for simulating (main.exe) has been created by using the command *make telosb* in the folder where all the other files are. In order to create the simulation, the creation of the nodes has been needed, this has been done by adding sky motes that use as firmware the main.exe file generated by the previous command. The number of motes that have been used is 9 and they have been organized in a star shape topology.



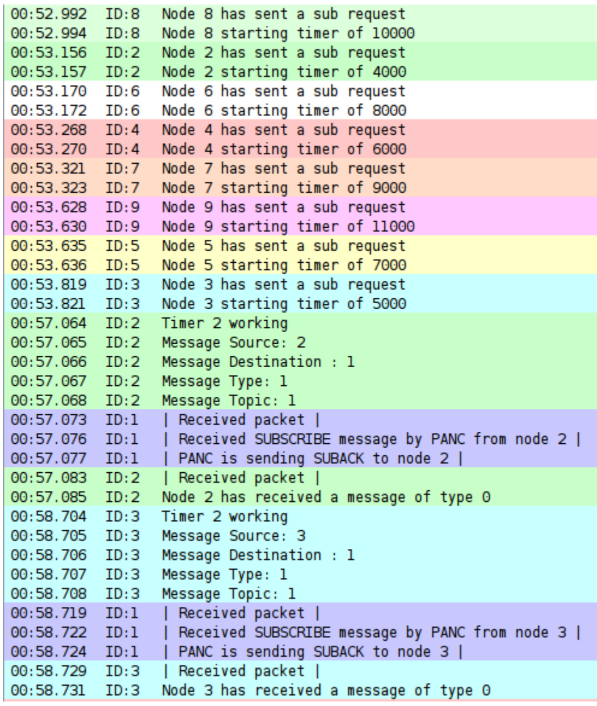
Successively as to connect node 1 to NodeRed has been used the Cooja tool to allow node 1 to work as a server which has an entry socket 60001. Then the simulation has been started giving the following result:



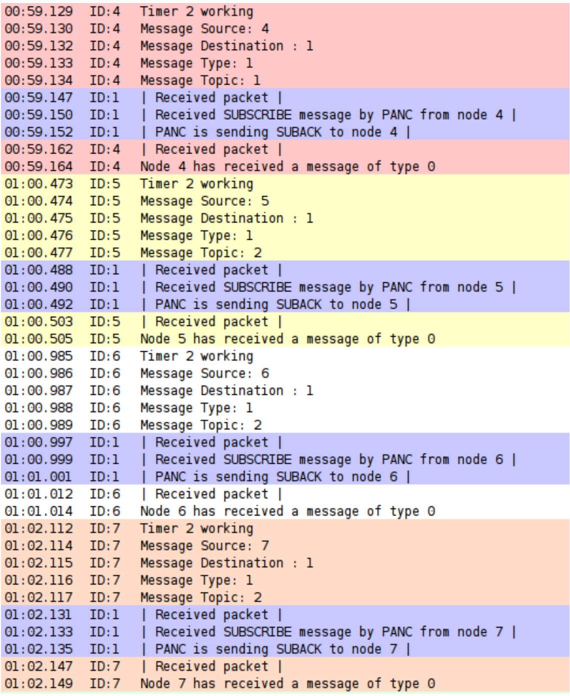
```
00:04.164  ID:8  Source of the message: 8
00:04.328  ID:2  Source of the message: 2
00:04.342  ID:6  Source of the message: 6
00:04.440  ID:4  Source of the message: 4
00:04.493  ID:7  Source of the message: 7
00:04.800  ID:9  Source of the message: 9
00:04.807  ID:5  Source of the message: 5
00:04.991  ID:3  Source of the message: 3
00:08.234  ID:2  Timer 2 working
00:08.235  ID:2  Message Source: 2
00:08.236  ID:2  Message Destination : 1
00:08.237  ID:2  Message Type: 0
00:08.248  ID:1  | Received packet |
00:08.250  ID:1  | Received CONNECT message by PANC from node 2 |
00:08.252  ID:1  | PANC is sending CONNACK to node 2 |
00:08.261  ID:2  | Received packet |
00:08.263  ID:2  Node 2 has received a message of type 0
00:09.874  ID:3  Timer 2 working
00:09.875  ID:3  Message Source: 3
00:09.876  ID:3  Message Destination : 1
00:09.877  ID:3  Message Type: 0
00:09.886  ID:1  | Received packet |
00:09.888  ID:1  | Received CONNECT message by PANC from node 3 |
00:09.890  ID:1  | PANC is sending CONNACK to node 3 |
00:09.898  ID:3  | Received packet |
00:09.900  ID:3  Node 3 has received a message of type 0
00:10.299  ID:4  Timer 2 working
00:10.300  ID:4  Message Source: 4
00:10.302  ID:4  Message Destination : 1
00:10.303  ID:4  Message Type: 0
00:10.314  ID:1  | Received packet |
00:10.316  ID:1  | Received CONNECT message by PANC from node 4 |
00:10.318  ID:1  | PANC is sending CONNACK to node 4 |
00:10.323  ID:4  | Received packet |
00:10.326  ID:4  Node 4 has received a message of type 0
00:11.642  ID:5  Timer 2 working
00:11.644  ID:5  Message Source: 5
00:11.645  ID:5  Message Destination : 1
00:11.646  ID:5  Message Type: 0
00:11.659  ID:1  | Received packet |
00:11.661  ID:1  | Received CONNECT message by PANC from node 5 |
00:11.663  ID:1  | PANC is sending CONNACK to node 5 |
00:11.669  ID:5  | Received packet |
00:11.671  ID:5  Node 5 has received a message of type 0
```



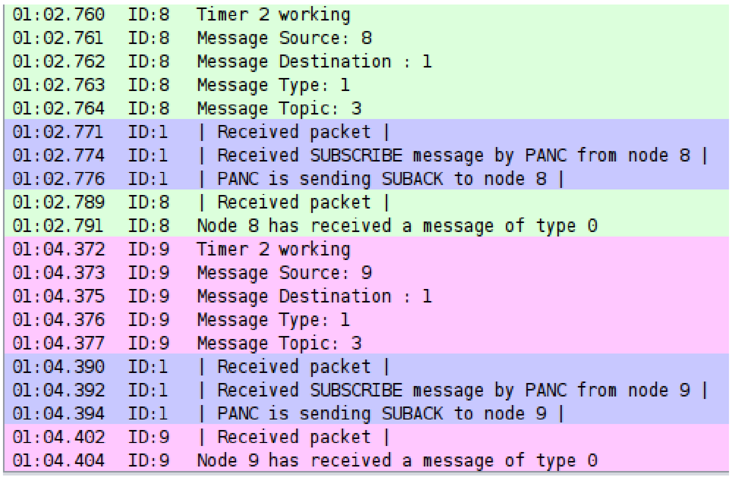
```
00:12.155  ID:6  Timer 2 working
00:12.156  ID:6  Message Source: 6
00:12.157  ID:6  Message Destination : 1
00:12.158  ID:6  Message Type: 0
00:12.164  ID:1  | Received packet |
00:12.166  ID:1  | Received CONNECT message by PANC from node 6 |
00:12.168  ID:1  | PANC is sending CONNACK to node 6 |
00:12.179  ID:6  | Received packet |
00:12.181  ID:6  Node 6 has received a message of type 0
00:13.282  ID:7  Timer 2 working
00:13.283  ID:7  Message Source: 7
00:13.285  ID:7  Message Destination : 1
00:13.286  ID:7  Message Type: 0
00:13.299  ID:1  | Received packet |
00:13.301  ID:1  | Received CONNECT message by PANC from node 7 |
00:13.303  ID:1  | PANC is sending CONNACK to node 7 |
00:13.317  ID:7  | Received packet |
00:13.319  ID:7  Node 7 has received a message of type 0
00:13.930  ID:8  Timer 2 working
00:13.931  ID:8  Message Source: 8
00:13.932  ID:8  Message Destination : 1
00:13.933  ID:8  Message Type: 0
00:13.939  ID:1  | Received packet |
00:13.942  ID:1  | Received CONNECT message by PANC from node 8 |
00:13.944  ID:1  | PANC is sending CONNACK to node 8 |
00:13.953  ID:8  | Received packet |
00:13.955  ID:8  Node 8 has received a message of type 0
00:15.542  ID:9  Timer 2 working
00:15.543  ID:9  Message Source: 9
00:15.545  ID:9  Message Destination : 1
00:15.545  ID:9  Message Type: 0
00:15.554  ID:1  | Received packet |
00:15.556  ID:1  | Received CONNECT message by PANC from node 9 |
00:15.558  ID:1  | PANC is sending CONNACK to node 9 |
00:15.567  ID:9  | Received packet |
00:15.569  ID:9  Node 9 has received a message of type 0
```



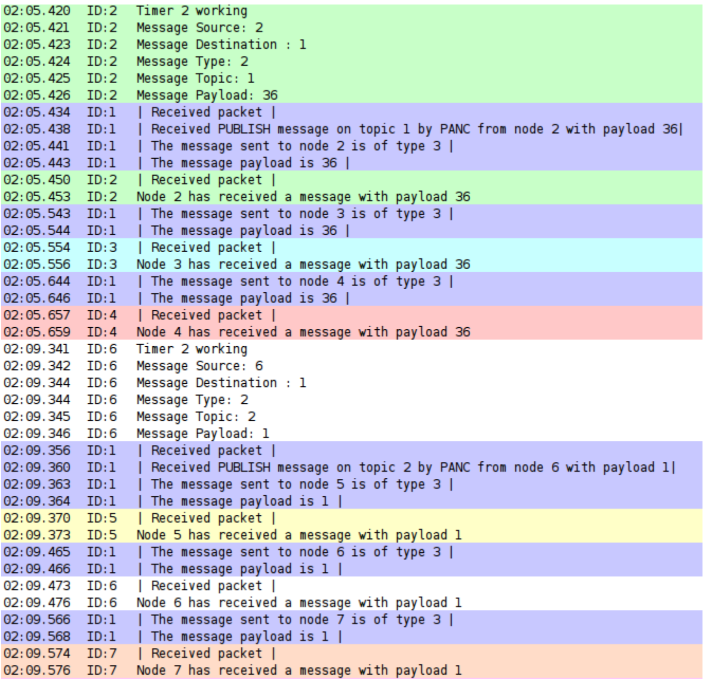
```
00:52.992   ID:8   Node 8 has sent a sub request
00:52.994   ID:8   Node 8 starting timer of 10000
00:53.156   ID:2   Node 2 has sent a sub request
00:53.157   ID:2   Node 2 starting timer of 4000
00:53.170   ID:6   Node 6 has sent a sub request
00:53.172   ID:6   Node 6 starting timer of 8000
00:53.268   ID:4   Node 4 has sent a sub request
00:53.270   ID:4   Node 4 starting timer of 6000
00:53.321   ID:7   Node 7 has sent a sub request
00:53.323   ID:7   Node 7 starting timer of 9000
00:53.628   ID:9   Node 9 has sent a sub request
00:53.630   ID:9   Node 9 starting timer of 11000
00:53.635   ID:5   Node 5 has sent a sub request
00:53.636   ID:5   Node 5 starting timer of 7000
00:53.819   ID:3   Node 3 has sent a sub request
00:53.821   ID:3   Node 3 starting timer of 5000
00:57.064   ID:2   Timer 2 working
00:57.065   ID:2   Message Source: 2
00:57.066   ID:2   Message Destination : 1
00:57.067   ID:2   Message Type: 1
00:57.068   ID:2   Message Topic: 1
00:57.073   ID:1   | Received packet |
00:57.076   ID:1   | Received SUBSCRIBE message by PANC from node 2 |
00:57.077   ID:1   | PANC is sending SUBACK to node 2 |
00:57.083   ID:2   | Received packet |
00:57.085   ID:2   Node 2 has received a message of type 0
00:58.704   ID:3   Timer 2 working
00:58.705   ID:3   Message Source: 3
00:58.706   ID:3   Message Destination : 1
00:58.707   ID:3   Message Type: 1
00:58.708   ID:3   Message Topic: 1
00:58.719   ID:1   | Received packet |
00:58.722   ID:1   | Received SUBSCRIBE message by PANC from node 3 |
00:58.724   ID:1   | PANC is sending SUBACK to node 3 |
00:58.729   ID:3   | Received packet |
00:58.731   ID:3   Node 3 has received a message of type 0
```



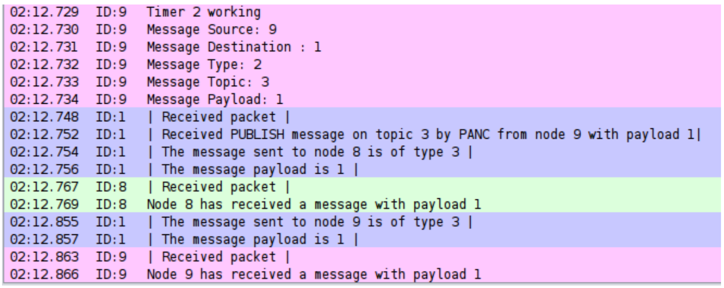
```
00:59.129   ID:4   Timer 2 working
00:59.130   ID:4   Message Source: 4
00:59.132   ID:4   Message Destination : 1
00:59.133   ID:4   Message Type: 1
00:59.134   ID:4   Message Topic: 1
00:59.147   ID:1   | Received packet |
00:59.150   ID:1   | Received SUBSCRIBE message by PANC from node 4 |
00:59.152   ID:1   | PANC is sending SUBACK to node 4 |
00:59.162   ID:4   | Received packet |
00:59.164   ID:4   Node 4 has received a message of type 0
01:00.473   ID:5   Timer 2 working
01:00.474   ID:5   Message Source: 5
01:00.475   ID:5   Message Destination : 1
01:00.476   ID:5   Message Type: 1
01:00.477   ID:5   Message Topic: 2
01:00.488   ID:1   | Received packet |
01:00.490   ID:1   | Received SUBSCRIBE message by PANC from node 5 |
01:00.492   ID:1   | PANC is sending SUBACK to node 5 |
01:00.503   ID:5   | Received packet |
01:00.505   ID:5   Node 5 has received a message of type 0
01:00.985   ID:6   Timer 2 working
01:00.986   ID:6   Message Source: 6
01:00.987   ID:6   Message Destination : 1
01:00.988   ID:6   Message Type: 1
01:00.989   ID:6   Message Topic: 2
01:00.997   ID:1   | Received packet |
01:00.999   ID:1   | Received SUBSCRIBE message by PANC from node 6 |
01:01.001   ID:1   | PANC is sending SUBACK to node 6 |
01:01.012   ID:6   | Received packet |
01:01.014   ID:6   Node 6 has received a message of type 0
01:02.112   ID:7   Timer 2 working
01:02.114   ID:7   Message Source: 7
01:02.115   ID:7   Message Destination : 1
01:02.116   ID:7   Message Type: 1
01:02.117   ID:7   Message Topic: 2
01:02.131   ID:1   | Received packet |
01:02.133   ID:1   | Received SUBSCRIBE message by PANC from node 7 |
01:02.135   ID:1   | PANC is sending SUBACK to node 7 |
01:02.147   ID:7   | Received packet |
01:02.149   ID:7   Node 7 has received a message of type 0
```



```
01:02.760   ID:8   Timer 2 working
01:02.761   ID:8   Message Source: 8
01:02.762   ID:8   Message Destination : 1
01:02.763   ID:8   Message Type: 1
01:02.764   ID:8   Message Topic: 3
01:02.771   ID:1   | Received packet |
01:02.774   ID:1   | Received SUBSCRIBE message by PANC from node 8 |
01:02.776   ID:1   | PANC is sending SUBACK to node 8 |
01:02.789   ID:8   | Received packet |
01:02.791   ID:8   Node 8 has received a message of type 0
01:04.372   ID:9   Timer 2 working
01:04.373   ID:9   Message Source: 9
01:04.375   ID:9   Message Destination : 1
01:04.376   ID:9   Message Type: 1
01:04.377   ID:9   Message Topic: 3
01:04.390   ID:1   | Received packet |
01:04.392   ID:1   | Received SUBSCRIBE message by PANC from node 9 |
01:04.394   ID:1   | PANC is sending SUBACK to node 9 |
01:04.402   ID:9   | Received packet |
01:04.404   ID:9   Node 9 has received a message of type 0
```



```
02:05.420   ID:2   Timer 2 working
02:05.421   ID:2   Message Source: 2
02:05.423   ID:2   Message Destination : 1
02:05.424   ID:2   Message Type: 2
02:05.425   ID:2   Message Topic: 1
02:05.426   ID:2   Message Payload: 36
02:05.434   ID:1   | Received packet |
02:05.438   ID:1   | Received PUBLISH message on topic 1 by PANC from node 2 with payload 36|
02:05.441   ID:1   | The message sent to node 2 is of type 3 |
02:05.443   ID:1   | The message payload is 36 |
02:05.450   ID:2   | Received packet |
02:05.453   ID:2   Node 2 has received a message with payload 36
02:05.543   ID:1   | The message sent to node 3 is of type 3 |
02:05.544   ID:1   | The message payload is 36 |
02:05.554   ID:3   | Received packet |
02:05.556   ID:3   Node 3 has received a message with payload 36
02:05.644   ID:1   | The message sent to node 4 is of type 3 |
02:05.646   ID:1   | The message payload is 36 |
02:05.657   ID:4   | Received packet |
02:05.659   ID:4   Node 4 has received a message with payload 36
02:09.341   ID:6   Timer 2 working
02:09.342   ID:6   Message Source: 6
02:09.344   ID:6   Message Destination : 1
02:09.344   ID:6   Message Type: 2
02:09.345   ID:6   Message Topic: 2
02:09.346   ID:6   Message Payload: 1
02:09.356   ID:1   | Received packet |
02:09.360   ID:1   | Received PUBLISH message on topic 2 by PANC from node 6 with payload 1|
02:09.363   ID:1   | The message sent to node 5 is of type 3 |
02:09.364   ID:1   | The message payload is 1 |
02:09.370   ID:5   | Received packet |
02:09.373   ID:5   Node 5 has received a message with payload 1
02:09.465   ID:1   | The message sent to node 6 is of type 3 |
02:09.466   ID:1   | The message payload is 1 |
02:09.473   ID:6   | Received packet |
02:09.476   ID:6   Node 6 has received a message with payload 1
02:09.566   ID:1   | The message sent to node 7 is of type 3 |
02:09.568   ID:1   | The message payload is 1 |
02:09.574   ID:7   | Received packet |
02:09.576   ID:7   Node 7 has received a message with payload 1
```

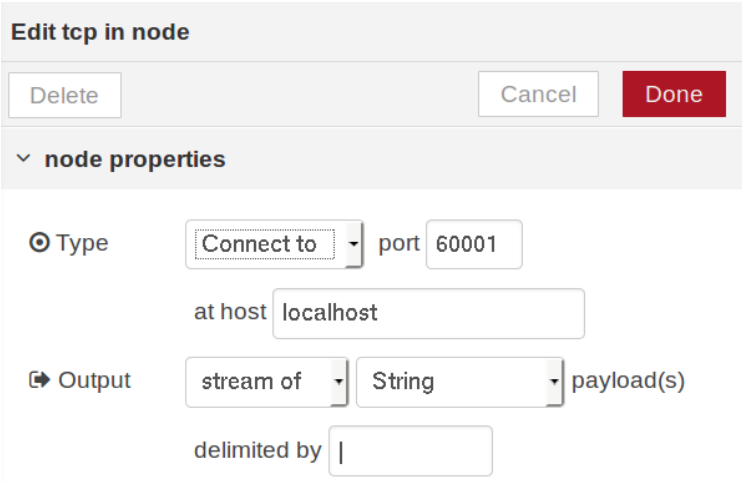


```
02:12.729   ID:9   Timer 2 working
02:12.730   ID:9   Message Source: 9
02:12.731   ID:9   Message Destination : 1
02:12.732   ID:9   Message Type: 2
02:12.733   ID:9   Message Topic: 3
02:12.734   ID:9   Message Payload: 1
02:12.748   ID:1   | Received packet |
02:12.752   ID:1   | Received PUBLISH message on topic 3 by PANC from node 9 with payload 1|
02:12.754   ID:1   | The message sent to node 8 is of type 3 |
02:12.756   ID:1   | The message payload is 1 |
02:12.767   ID:8   | Received packet |
02:12.769   ID:8   Node 8 has received a message with payload 1
02:12.855   ID:1   | The message sent to node 9 is of type 3 |
02:12.857   ID:1   | The message payload is 1 |
02:12.863   ID:9   | Received packet |
02:12.866   ID:9   Node 9 has received a message with payload 1
```

Since timer 4, used for scripting the PUBLISH messages, is periodic, if the simulation goes on it will be possible to see many other PUB messages.

NODE RED

First of all, a tcp block has been used in order to connect the PANC. The node properties are:



Edit tcp in node

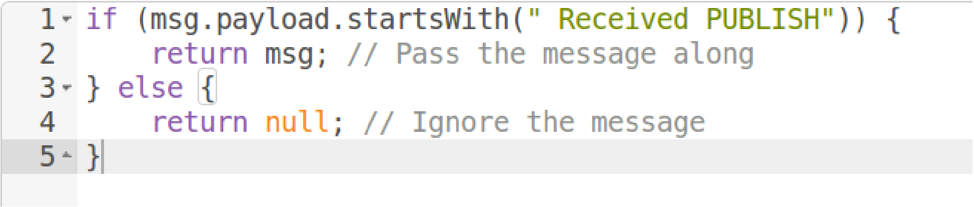
Type: Connect to port 60001

at host localhost

Output: stream of String payload(s)

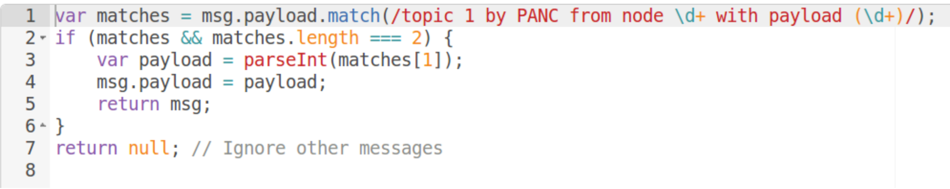
delimited by |

The delimiter “|” has been used because in the printf functions in TinyOS each message related to node 1 was delimited by that character. Then a filtering function has been used as to filter all the publish messages. The function logic is:

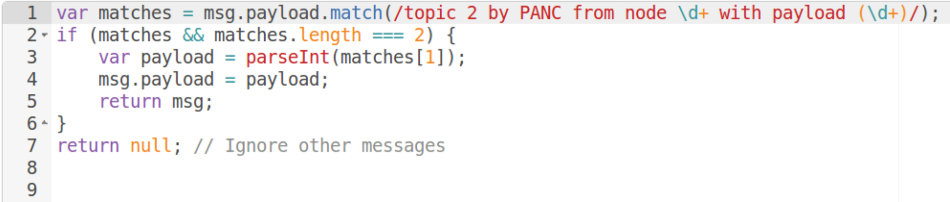


```
if (msg.payload.startsWith(" Received PUBLISH")) {
    return msg; // Pass the message along
} else {
    return null; // Ignore the message
}
```

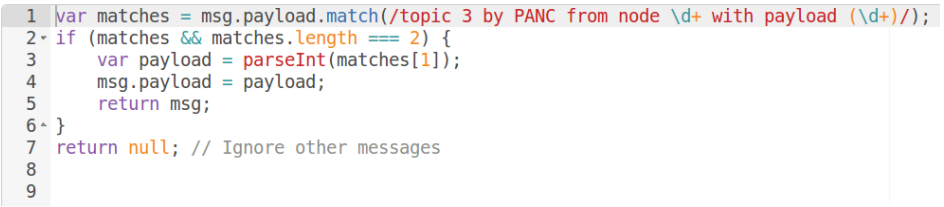
The output of the filtering function has been used as input of three other functions, which are used to extract the payload of the messages. Each function is related to one specific topic.



```
var matches = msg.payload.match(/topic 1 by PANC from node \d+ with payload (\d+)/);
if (matches && matches.length === 2) {
    var payload = parseInt(matches[1]);
    msg.payload = payload;
    return msg;
}
return null; // Ignore other messages

```

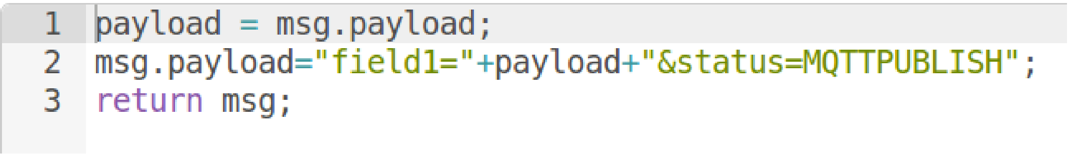


```
var matches = msg.payload.match(/topic 2 by PANC from node \d+ with payload (\d+)/);
if (matches && matches.length === 2) {
    var payload = parseInt(matches[1]);
    msg.payload = payload;
    return msg;
}
return null; // Ignore other messages


```

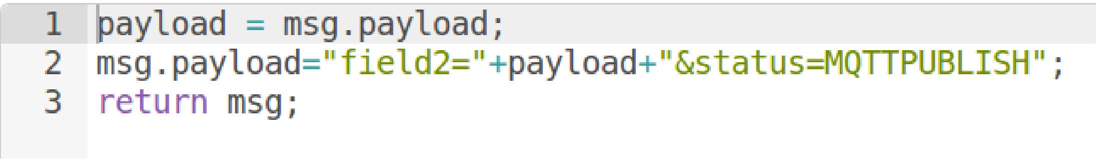


```
var matches = msg.payload.match(/topic 3 by PANC from node \d+ with payload (\d+)/);
if (matches && matches.length === 2) {
    var payload = parseInt(matches[1]);
    msg.payload = payload;
    return msg;
}
return null; // Ignore other messages


```

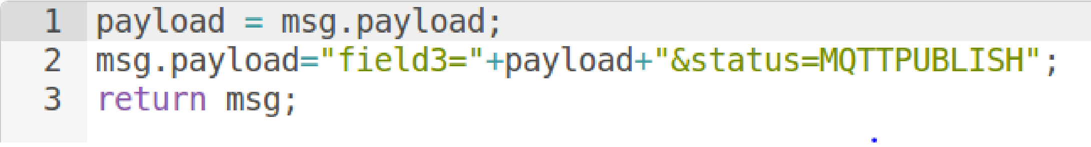
Then the output of each function related to a specific topic enter into another function used to format the message for ThingSpeak.



```
payload = msg.payload;
msg.payload="field1="+payload+"&status=MQTTPUBLISH";
return msg;
```

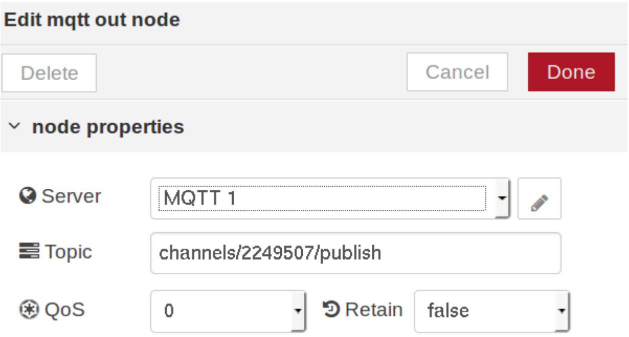


```
payload = msg.payload;
msg.payload="field2="+payload+"&status=MQTTPUBLISH";
return msg;
```



```
payload = msg.payload;
msg.payload="field3="+payload+"&status=MQTTPUBLISH";
return msg;
```

At the end has been used an MQTT block in order to send the messages to ThingSpeak.



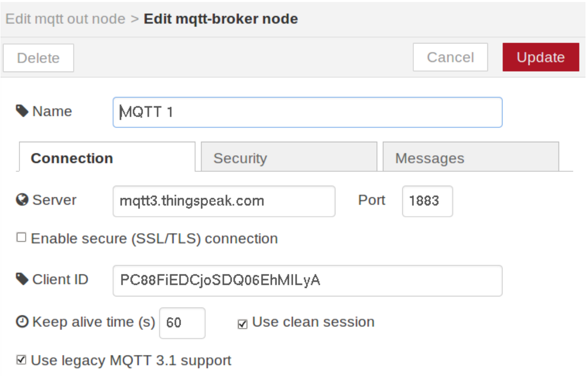
Edit mqtt out node

Server: MQTT 1

Topic: channels/2249507/publish

QoS: 0

Retain: false



Edit mqtt out node > Edit mqtt-broker node

Name: MQTT 1

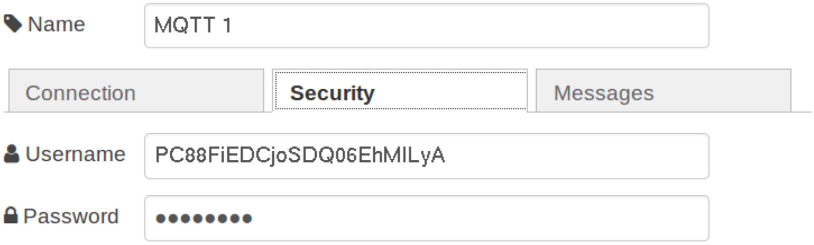
Server: mqtt3.thingspeak.com Port: 1883

□ Enable secure (SSL/TLS) connection

Client ID: PC88FiEDCjoSDQ06EhMILyA

Keep alive time (s): 60 ☑ Use clean session

☑ Use legacy MQTT 3.1 support



Name: MQTT 1

Username: PC88FiEDCjoSDQ06EhMILyA

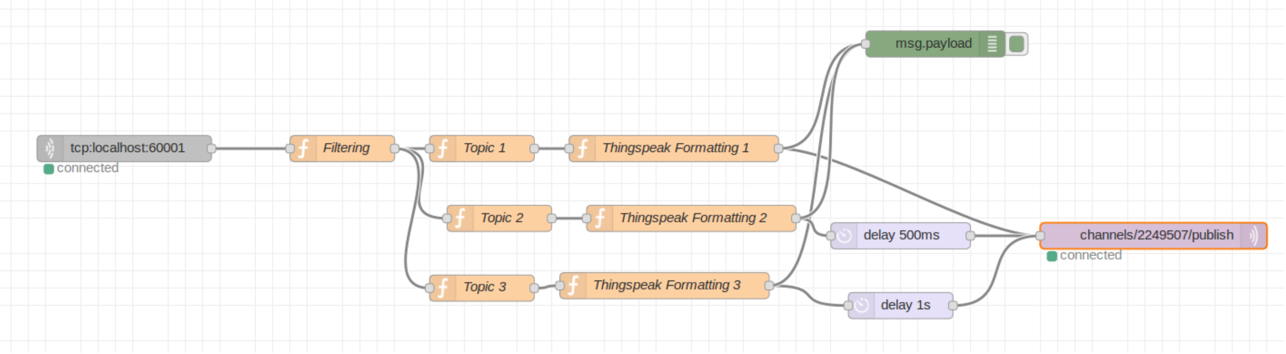
Password: ••••••••

The credentials for MQTT ThingSpeak mote are:

username = PC88FiEDCjoSDQ06EhMILyA
clientId = PC88FiEDCjoSDQ06EhMILyA
password = lyS+eTmzYJbzUaNiBzDZva82

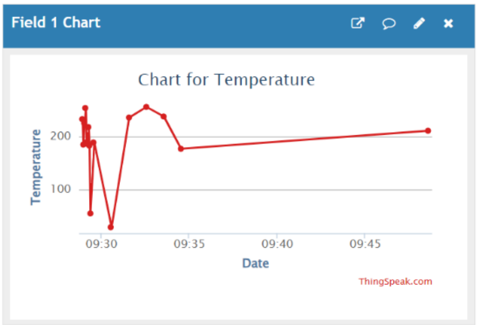
There have been also inserted two delay blocks on the outputs of the last two formatting functions in order to avoid collisions.

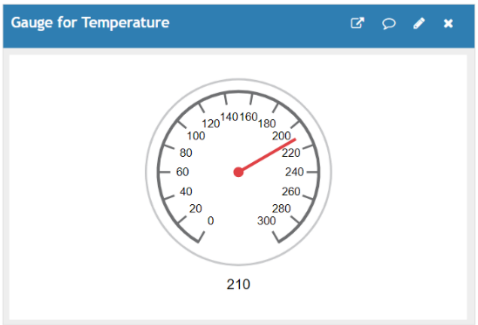
The complete flow is:

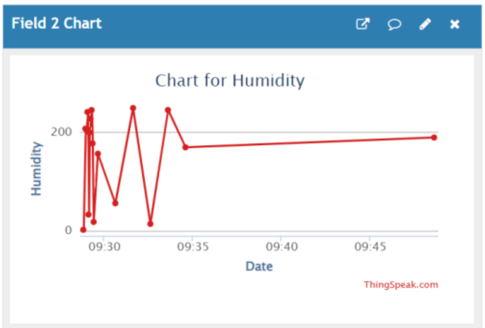


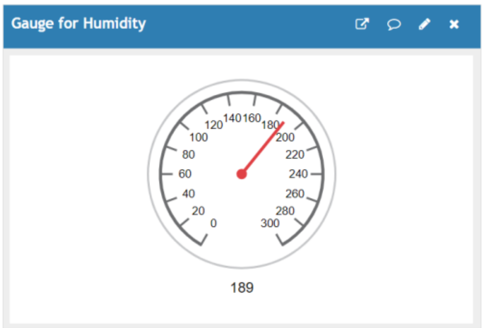
THINGSPEAK

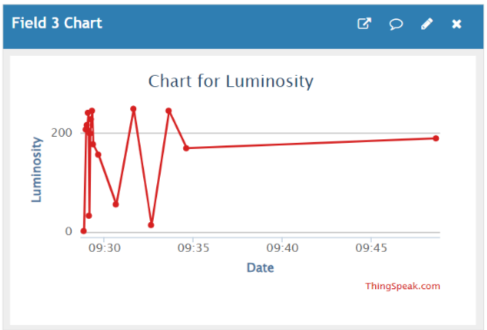
First of all, a new channel with three fields has been created, those fields are the TEMPERATURE, HUMIDITY and LUMINOSITY. Then a new MQTT device has been created and the credentials of this device have been used to connect NodeRed to ThingSpeak. The PUB messages received by the PANC, formatted through NodeRed, have been represented in ThingSpeak through these following graphs.

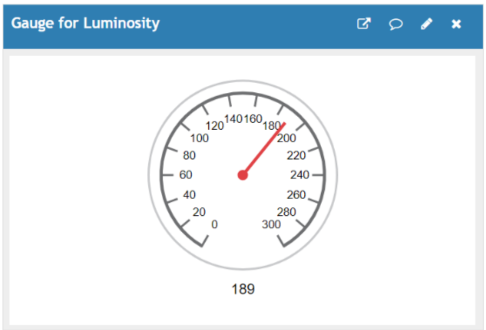

Field 1 Chart
Chart for Temperature
Temperature
09:30
09:35
09:40
09:45
Date
ThingSpeak.com


Gauge for Temperature


Field 2 Chart
Chart for Humidity
Humidity
09:30
09:35
09:40
09:45
Date
ThingSpeak.com


Gauge for Humidity


Field 3 Chart
Chart for Luminosity
Luminosity
09:30
09:35
09:40
09:45
Date
ThingSpeak.com


Gauge for Luminosity