IoT 2023 CHALLENGE 3

1) Configuration part:

Inside the ***RadioRoute.h*** file we are defining the message format as below code.

```
typedef nx_struct radio_route_msg {
  nx_uint8_t type;
  nx_uint16_t sender;
  nx_uint16_t destination;
  nx_uint16_t value; // is equal to the cost for type 2 message
} radio_route_msg_t;
```

Inside the ***RunSimulationScript.py*** file, we should initialize all nodes at time zero. With the below code, we did this.

```
for i in range(1,8):
    print ("Creating node",i,"...");
    node1 =t.getNode(i);
    node1.bootAtTime(time);
    print ">>>Will boot at time", time/t.ticksPerSecond(), "[sec]";
```

Also, we defined a new channel for printing the node 6 status as below code.

```
# adding a new channel for showing the LEDS output for the node 6.
print "Activate debug message on channel node6_leds"
t.addChannel("node6_leds",out);
```

Inside the ***RadioRouteAppC.nc*** file, we need to define the components and interfaces.

```
  components MainC, RadioRouteC as App, LedsC;
  components new AMSenderC(AM_RADIO_COUNT_MSG);
  components new AMReceiverC(AM_RADIO_COUNT_MSG);
  components ActiveMessageC;
  components new TimerMilliC() as Timer0;
  components new TimerMilliC() as Timer1;
  App.Boot -> MainC.Boot;
  App.Receive -> AMReceiverC;
  App.AMSend -> AMSenderC;
  App.AMControl -> ActiveMessageC;
  App.Leds -> LedsC;
  App.Timer0 -> Timer0;
  App.Timer1 -> Timer1;
  App.Packet -> AMSenderC;
```

2) Implementation part

Now configuration part is done and we need to modify ***RadioRouteC.nc*** file to do the challenge requirements.

A) First of all, we need to define the routing table. This way we defined a structure as an entry to store variables inside it and then we created an array of this structure.

```
typedef nx_struct rt_struct{
  nx_uint16_t destination;
  nx_uint16_t Next_Hop;
  nx_uint16_t cost; // cost of the path
} rt_struct;
rt_struct routing_table[6];
```

Then we are creating an array of char for storing our student number and an array of type in LED status of node 6.

```
const char *student_id = "10832693";
uint16_t Node6_LED_status[30];
```

Then, we have defined several custom functions for handling messages and routing table.

```
void Route_Req_Msg (uint16_t node_requested);
void Route_Rep_Msg(uint16_t node_requested, uint16_t cost);
void Data_Message_Send(uint16_t dest, uint16_t value);
uint8_t Dest_searching(uint16_t dest);
// function for routing table initializing
void routing_table_initialize ();
```

For initializing the routing table, we are using ***routing_table_initialize*** function and simply we initialized all variables with zero. We are calling this function inside Boot.Booted().

The next function is ***Dest_searching()***. This function gets the destination node as an input and searches on the routing table for the node that we can reach to the destination by it. If the routing table is empty, it returns 0, otherwise, it returns the index of the node that we can reach to the destination by it. Below is the code block of this function.

```
uint8_t Dest_searching(uint16_t dest){
    uint16_t k;
    /* simply here we are checking that if the destination is reachable from
current node or not. if this is not reachable then return 0 and in this case
we need to broadcast route reply or route request. But if it's reachable
return the index of the node.*/
    if (rt_elements != 0){
    for (k=0; k<rt_elements; k++){
      if(routing_table[k].destination == dest){
        return k+1;
    }
    else{
        return 0;
      }}}}
```

The next function is Data_Message_Send. Here simply we are checking the message is sent do not send it again. Here we are finding the next node to reach the destination. Then we fill the message rcm with the proper type, sender, destination, and value. Type is always 0 in this case and the value is 5, and the destination is 7, but the sender can vary in each different case.

```
void Data_Message_Send(uint16_t dest, uint16_t value){
    if (!msg_sent_flag){
```

```
      uint8_t rt_currentIndex = Dest_searching(dest);
      // define type as zero for Data messages
      uint8_t type = 0;
      // just define rcm as a message.then fill it
      radio_route_msg_t* rcm = (radio_route_msg_t*)call
Packet.getPayload(&packet, sizeof(radio_route_msg_t));
      //dbg("timer", "msg dest is %u \n", routing_table[rt_currentIndex-
1].Next_Hop);
      if (rcm == NULL) {
        return;
      }
      // filling type
      rcm->type = type;
      // sender node
      rcm->sender = TOS_NODE_ID;
      // final destination::::::: in this case it is 7
      rcm->destination = dest;
      // explicit value of message
      rcm->value = value;
      generate_send(routing_table[rt_currentIndex-1].Next_Hop, &packet, type);
      msg_sent_flag = TRUE;
      //dbg("radio_send", "Data message generated with type: %u\tand
destination: %u\tand sender: %u\tand value: %u \n", rcm->type, rcm-
>destination, rcm->sender, rcm->value);
    }}
```

The next function is ***Route_req_Msg.*** First, we are checking the status of route_req_sent variable and if it's false then we can send a route request message, otherwise do not send it. This variable checking is because of sending just one route request message by each node. Then like the message type 0 we fill this message but with this difference that here we have not sender anymore.

```
void Route_Req_Msg(uint16_t node_requested){
  // in the beggining route_req_sent is false
    if (!route_req_sent){
    // this is route req message then give it type=1
    uint8_t type = 1;
    // just define rcm as a message.then fill it
    radio_route_msg_t* rcm = (radio_route_msg_t*)call
Packet.getPayload(&packet, sizeof(radio_route_msg_t));
    if (rcm == NULL) {
      return;
    }
    rcm->type = type;
    // in this case there is no any sender then put it zero
    rcm->sender = 0;
    // just node_requested is exist.
    rcm->destination = node_requested;
    // there is no value in this type of message.
    rcm->value = 0;
```

```
    // broadcast this message
    generate_send(AM_BROADCAST_ADDR, &packet, type);
    //dbg("radio_send", "route request message generated with type: %u\tand
destination: %u \n", rcm->type, rcm->destination);
    }}
```

The next function is ***Route_Rep_Msg***. This function is so similar to the route_req_msg and the only difference is the way we should fill the payload. Here we have cost instead of value and our destination is the requested node and we also have the sender node here.

```
void Route_Rep_Msg(uint16_t node_requested, uint16_t cost){
  // route_rep_sent is FALSE from the beggining of the code
    if (!route_rep_sent){
    // define the type = 2 for route rep messages
    uint8_t type = 2;
    // just define rcm as a message.then fill it
    radio_route_msg_t* rcm = (radio_route_msg_t*)call
Packet.getPayload(&packet, sizeof(radio_route_msg_t));
    if (rcm == NULL) {
      return;
    }
    // filling the type of message
    rcm->type = type;
    // define the sender node
    rcm->sender = (uint16_t) TOS_NODE_ID;
    // define the node requested
    rcm->destination = node_requested;
    // consider value as cost for route reply messages
    rcm->value = cost;
    // broadcast this message
    generate_send(AM_BROADCAST_ADDR, &packet, type);
    //dbg("radio_send", "route reply message generated with type: %u\tand
destination: %u\tand sender: %u\tand value: %u \n", rcm->type, rcm-
>destination, rcm->sender, rcm->value);
    }}
```

Then inside the ***actual_send*** function we check the looked and if it is False then we are in the right place and we are sending the packet.

```
bool actual_send (uint16_t address, message_t* packet){
    /*
    * Implement here the logic to perform the actual send of the packet using
the tinyOS interfaces
    */
    if (!locked) {
if (call AMSend.send(address, packet, sizeof(radio_route_msg_t)) == SUCCESS) {
        locked = TRUE;
        dbg_clear("radio_send", " sending packet at time %s \n",
sim_time_string());}}
    else {
```

```
    return FALSE;}}
```

After this, inside the Boot.booted() function, application booted, and also we initial the routing table and then if we are inside node 1 then we call timer1 with the delay of 5s.

```
event void Boot.booted() {
    dbg("boot","Application booted.\n");
    routing_table_initialize();
    call AMControl.start();
    if (TOS_NODE_ID == 1){
      call Timer1.startOneShot(5000);
      //dbg("timer","application have booted and timer1 called.\n");
    }}
```

When the timer1 fired, inside the we are checking destination to the node 7, if it exists send data message, otherwise send a route request message.

```
  event void Timer1.fired() {
    uint8_t rt_currentIndex=0;
    rt_currentIndex = Dest_searching(7);
    // in first run the returned value by Dest_searching is zero then we need
to send a route req message.
    if (rt_currentIndex == 0){
    // send route req message
        Route_Req_Msg(7);
    }
    else {
    // sending Data message
     Data_Message_Send(7, 5);
    }
  }
```

B) Receive procedure:

This part is the main part that we are implementing the logic of sending messages and updating the routing tables.
First of all, we need to define the LEDs and modify their status. We need to pick one of the numbers of our student person ID and modulo to 3. This number is the index of LED that we have to toggle it. Variable ***msgs_number*** is the total number of messages that a node received.

```
uint8_t rt_currentIndex=0;
    uint16_t k=0;
    if (len != sizeof(radio_route_msg_t)) {return bufPtr;}
    else {
      uint8_t LED_index = student_id[msgs_number%8]%3;
      radio_route_msg_t* rcm = (radio_route_msg_t*)payload;
      switch(LED_index){
        case 0:
            call Leds.led0Toggle();
            break;
```

```
        case 1:
            call Leds.led1Toggle();
            break;
        case 2:
            call Leds.led2Toggle();
            break;
      }
      msgs_number ++;
```

Then we show the status of LED for this node by the below code. This code (`(call Leds.get() & LEDS_LED0) >0`) implies receiving 0 or 1 as the status of the LED.

```
      dbg("radio_rec", "node  %u  received a message of type %u at time %s and
LEDs status is like this:\n", TOS_NODE_ID, rcm->type , sim_time_string());
      dbg("led_0", "Led 0 status %u\n",  (call Leds.get() & LEDS_LED0) >0);
      dbg("led_1", "Led 1 status %u\n",  (call Leds.get() & LEDS_LED1) >0);
      dbg("led_2", "Led 2 status %u\n",  (call Leds.get() & LEDS_LED2) >0);
```

In our application, we need to store the status of LEDs on node 6. This way we are saving these inside the variable ***Node6_LED_status***. As far as our LEDs are 3 for each node our indexes grow in the factor of 3.

```
if(TOS_NODE_ID == 6){
        dbg("node6_leds", " message received by node 6. save the LEDs
status\n");
        Node6_LED_status[(msgs_number)*3+0] = (call Leds.get() & LEDS_LED0)
>0;
        Node6_LED_status[(msgs_number)*3+1] = (call Leds.get() & LEDS_LED1)
>0;
        Node6_LED_status[(msgs_number)*3+2] = (call Leds.get() & LEDS_LED2)
>0;
      }
```

After saving LEDs status we print them.

```
while ( TOS_NODE_ID == 6 && k<=((msgs_number)*3)+2){
      if(k == 0)
      dbg_clear("node6_leds", "Node 6 LEDs status:\n");

        dbg_clear("node6_leds", "%u",  Node6_LED_status[k]);
        k++;
      }
      dbg_clear("node6_leds", "\n\n");
```

Now we need to handle the received message; we performed a Switch Case block for this purpose with respect to each different message type.
If the type is 0, then if we already reached to the destination print a message to show we are at the destination. If the current node is not the destination send again a Data message to the destination.

```
case 0:
```

```
        // if we are in destination then do nothing just print that Data
message received by the destination.
        if (rcm->destination == TOS_NODE_ID){
          dbg("radio_pack", "Data message received by the destination node: %u
with the value of: %u\n", rcm->destination, rcm->value);

            }
        // otherwise call data message send function till reaching to the
desired destination
        else {
          Data_Message_Send(rcm->destination, rcm->value);
            }
        break;
```

If the message type is 1, check if we are not in the destination node and the routing table is empty perform a route request message. Else if we are in the destination node perform a route reply message with a cost of 1. And finally, else if check if the requested destination node is in the routing table send a route reply message with cost+1.

```
case 1:
      // if we are in required destination then send a route reply message

        rt_currentIndex = Dest_searching(rcm->destination);
      // check if requested node is not in my routing table and not me
      if (rcm->destination != TOS_NODE_ID && rt_currentIndex ==0){
        Route_Req_Msg(rcm->destination);
      }
      // if I'm the requested node
      else if (rcm->destination == TOS_NODE_ID) {
      // second argument of the Route_Rep_Msg is cost
          Route_Rep_Msg(rcm->destination, 1);
        }
      // if requested node is in my table.
        else if (rt_currentIndex !=0){
            // update the cost with cost of routing table +1 and send the
reply message.
            Route_Rep_Msg(rcm->destination, routing_table[rt_currentIndex-
1].cost+1);
        }
        break;
```

if the message type is 2, if the routing table doesn't have any entry then update the routing table with current information and if the destination node is not node 1 then perform a route reply message with the cost+1. Else if the newly received reply message has less cost update the previous values, and also again if we are not in node 1 forward the route reply message with cost+1. And finally, if we are in node 1 then send a Data message to node 7 with the value of 5.

```
case 2:
            rt_currentIndex = Dest_searching(rcm->destination);
```

```
        if (rt_currentIndex == 0) {

        routing_table[rt_elements].destination = rcm->destination;
        routing_table[rt_elements].Next_Hop = rcm->sender;
        routing_table[rt_elements].cost = rcm->value;
        rt_elements ++;
        if (TOS_NODE_ID != 1)
            Route_Rep_Msg(rcm->destination, rcm->value + 1);
        }

        else if (routing_table[rt_currentIndex-1].cost > rcm->value){
        routing_table[rt_currentIndex-1].destination = rcm->destination;
        routing_table[rt_currentIndex-1].cost = rcm->value + 1;
        routing_table[rt_currentIndex-1].Next_Hop = rcm->sender;
        if (TOS_NODE_ID != 1)
            Route_Rep_Msg(rcm->destination, rcm->value + 1);
            }

        // if i'm the requested node in reply
        if (TOS_NODE_ID == 1) {
          Data_Message_Send(7, 5);
            }
    break;
```

Our final result of LEDs for node 6 with initial status is 000,010,110,111,011,010.