IOT 2023 / Challenge 3

(1) Name: Nadia Sadeghi Person Code: 10916407

(2) Name: Alireza Tahmasebi Person Code: 10837878

There are two phases for implementation this challenge:

1. Configuration
2. Implementation

✓ Configuration:

- The "RadioRoute.h" file specifies the packet fields used in the network. There are three types of packets: type 0, type 1, and type 2. For type 0 packets, the fields are as expected. Type 1 packets only require the type and node requested fields, with the destination field used to store the node requested information, and sender and value fields set to 0. Type 2 packets need the type, sender, node requested, and cost fields, with the destination field storing the node requested information and the value field containing the cost.

```


#ifndef RADIO_ROUTE_H
#define RADIO_ROUTE_H

typedef nx_struct radio_route_msg {
  nx_uint8_t type;
  nx_uint16_t sender;
  nx_uint16_t destination;
  nx_uint16_t value;
} radio_route_msg_t;

enum {
  AM_RADIO_COUNT_MSG = 10,
};

#endif
```

✓ Implementation:

- In "RadioRouteC.nc" we implement our simulation.

we define the routing table which consists of fields such as dest, next_hop, and cost. To allow each node to add information about other nodes, we create a global array of this structure. Each row in the routing table corresponds to a node and its associated information. Additionally, we have a global variable called "index" that indicates the current index in the routing table where new information can be inserted.

```
// definign routing table.
typedef nx_struct rounting_table_struct{
  nx_uint16_t destination;
  nx_uint16_t nextHop;
  nx_uint16_t cost;
} rounting_table_struct;
```

- In below you can find all assumption of question, such as Time delay, Student Id, Timers, Data message, Route Request & Reply message …. As below:

```
rounting_table_struct routing_table[6];
// Variables to store the message to send
message_t queued_packet;
uint16_t queue_addr;
uint16_t time_delays[7]={61,173,267,371,479,583,689}; //Time delay in milli seconds



bool route_req_sent=FALSE;
bool route_rep_sent=FALSE;
// flag for checking if the message in sent or not
bool message_sent = FALSE;
// define variable to count the messages
uint16_t Nof_msgs = 0;
// student ID as a string
const char *student_id = "10916407";
uint16_t Node6_history[30];


bool locked;
// actual send function declaration
bool actual_send (uint16_t address, message_t* packet);
// generate send function declaration. this function shouldn't be modified
bool generate_send (uint16_t address, message_t* packet, uint8_t type);
// our custom functions.
uint8_t Next_Node_Search(uint16_t dest, uint16_t* cost);
void Route_Req_Msg (uint16_t node_requested);
void Route_Rep_Msg(uint16_t node_requested, uint16_t cost);
void D_M_Send(uint16_t dest, uint16_t value);
void rt_has_initialize ();
```

```
uint8_t rt_TopIndex = 0;

void rt_has_initialize(){
  uint8_t i;
  for(i=0; i<6; i++){
    routing_table[i].destination = 0;
    routing_table[i].nextHop = 0;
    routing_table[i].cost = 0;
  }
}
```

```
bool generate_send (uint16_t address, message_t* packet, uint8_t type){
/*
*
* Function to be used when performing the send after the receive message event.
* It store the packet and address into a global variable and start the timer execution to schedule the send.
* It allow the sending of only one message for each REQ and REP type
* @Input:
*     address: packet destination address
*     packet: full packet to be sent (Not only Payload)
*     type: payload message type
*
* MANDATORY: DO NOT MODIFY THIS FUNCTION
*/
  if (call Timer0.isRunning()){
      return FALSE;
  }else{
  if (type == 1 && !route_req_sent ){
      route_req_sent = TRUE;
      call Timer0.startOneShot( time_delays[TOS_NODE_ID-1] );
      queued_packet = *packet;
      queue_addr = address;
  }else if (type == 2 && !route_rep_sent){
      route_rep_sent = TRUE;
      call Timer0.startOneShot( time_delays[TOS_NODE_ID-1] );
      queued_packet = *packet;
      queue_addr = address;
  }else if (type == 0){
      call Timer0.startOneShot( time_delays[TOS_NODE_ID-1] );
      queued_packet = *packet;
      queue_addr = address;
  }
  }
  return TRUE;
```

```
event void Timer0.fired() {
  /*
  * Timer triggered to perform the send.
  * MANDATORY: DO NOT MODIFY THIS FUNCTION
  */
  actual_send (queue_addr, &queued_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) {
    return FALSE;
  }
  else {
    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());
      //return TRUE;
    }
    //return FALSE;
  }
}
```

- The implementation starts with the "Boot.booted" function, where the routing table is initialized with all fields set to 0. Then, the "AMControl.start" function is called to activate the radios. Finally, the timer1 of node 1 starts counting for a duration of 5 seconds.

```
event void Boot.booted() {
  uint8_t i;
  dbg("boot","Application booted.\n");
  /*for(i=0; i<6; i++){
    routing_table[i].destination = 0;
    routing_table[i].nextHop = 0;
    routing_table[i].cost = 0;
  } */
  t_has_initialize();

  call AMControl.start();

  if (TOS_NODE_ID == 1){
    call Timer1.startOneShot(5000);
  }
}
```

- In this step, searching in routing table for finding destination. After that show the value of Cost & Next_hop.

```
event void AMControl.startDone(error_t err) {
  if (err == SUCCESS) {
    dbg("radio","Radio on on node %d!\n", TOS_NODE_ID);
  }
  else {
    dbgerror("radio", "Radio failed to start, retrying...\n");
    call AMControl.start();
  }
}

event void AMControl.stopDone(error_t err) {
  dbg("boot", "Radio stopped!\n");
}

uint8_t Next_Node_Search(uint16_t dest, uint16_t* cost){
  uint8_t i;
  /* simply here we are checking that if the destination is reachable from current noded 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 1.*/
  if (rt_TopIndex == 0){
      return 0;
  }

  for (i=0; i<rt_TopIndex; i++){
    if(routing_table[i].destination == dest){
      *cost = routing_table[i].cost;
      return routing_table[i].next_hop;
    }
  }
  return 0;
}
```

- in the "SendMessage" function, the routing table is checked for the destination node. Since no routing request/response has been received yet, node 1 does not have node 7 in its routing table. To handle this, the function stores the destination and message in global variables for future sending. It then calls the "SendRoutingReq" function to send routing packets to other nodes in order to find the destination.

```
void D_M_Send(uint16_t dest, uint16_t value){

  if (!message_sent){
    uint16_t cost;
    uint16_t next_hop = Next_Node_Search(dest, &cost);
    // 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 type
    // define type as zero for Data messages
    rcm->type = 0;

    // sender node
    rcm->sender = (uint16_t) TOS_NODE_ID;

    // final destination::::::: in this case it is 7
    rcm->destination = dest;

    // explicit value of message
    rcm->value = value;

    message_sent = TRUE;
    generate_send(next_hop, &packet, 0);
  }
}
```

- In the "SendRoutingReq" function, we create a packet with the necessary values to search for the destination (e.g., node 7). The packet is then sent using the "generate_send" function, with the address set to "AM_BROADCAST_ADDR." This function handles delays, and after the delay, the "actual_send" function is called to transmit the packet. This packet-sending procedure remains consistent throughout.

```
// sending route request message
void Route_Req_Msg(uint16_t node_requested){
// in the beggining route_req_sent is false
  if (!route_req_sent){

  // 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;
  }

  // this is route req message then give it type=1
  rcm->type = 1;

  // 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, 1);
  }
}
```

- when each node receives a type 0 packet, it checks if the destination is itself. If it is, the transmission is complete and a success message is removed. If not, the message and destination are passed to the "SendMessage" function, which searches for the destination in the routing table and forwards the message to the next hop indicated in the table. In summary, when the routing request packet is sent, neighboring nodes (e.g., nodes 2 and 3) receive it and trigger the "Receive.receive" function. The nodes first change their LEDs as per the challenge requirements. They then check the type of the received packet, identifying it as type 1. If the requested node is not the current node, and the routing table is empty, the nodes broadcast a routing request for the requested node.

```
dbg("radio_rec", "Received packet at time %s\n", sim_time_string());

if (rcm->type == 0){
  // if we are in destination then do nothing just print that Data message received by the destination.
  if (rcm->destination == (uint16_t) TOS_NODE_ID){
    dbg("radio_pack", "packet 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 {
    D_M_Send(rcm->destination, rcm->value);
  }
}
else if (rcm->type == 1) {
  // if we are in required destination then send a route reply message
  rt_currentIndex = Next_Node_Search(rcm->destination, &cost);
  // check if requested node is not in my routing table and not me
  if (rcm->destination != (uint16_t) TOS_NODE_ID && next_hop ==0){
    Route_Req_Msg(rcm->destination);
  }
  // if I'm the requested node
  else if (rcm->destination == (uint16_t) 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 (next_hop !=0){
      // update the cost with cost of routing table +1 and send the reply message.

      Route_Rep_Msg(rcm->destination, cost+1);
  }
}
```

- when node 7 broadcasts a routing response, its neighboring nodes (6 and 5) receive it. Upon receiving a type 2 packet, a node checks its routing table. If the destination specified in the packet is not found, a new entry is added with the destination, next_hop, and cost values from the packet. The packet is then retransmitted. If the destination already exists, the node compares costs to decide whether to update the routing table. If the desired destination matches the requested node in the packet, the node calls the "SendMessage" function. The routing responses propagate back to node 1, which sends the stored message to the desired destination, node 7. In the "SendMessage" function, the next_hop is determined (node 3) and a packet is generated and sent accordingly.

```
      else if (rcm->type == 2) {
        next_hop = Next_Node_Search(rcm->destination, &cost);

        if (next_hop == 0) {
          routing_table[rt_TopIndex].destination = rcm->destination;
          routing_table[rt_TopIndex].nextHop = rcm->sender;
          routing_table[rt_TopIndex].cost = rcm->value;

          rt_TopIndex ++;
          if (TOS_NODE_ID != rcm->destination)
            Route_Rep_Msg(rcm->destination, rcm->value + 1);
        }

        else if (routing_table[rt_currentIndex-1].cost > rcm->value+1){
          uint8_t i;
          for(i=0; i<rt_TopIndex; i++){
            if(routing_table[i].destination == rcm->destination)
              break;
          }
          routing_table[i].cost = rcm->value + 1;
          routing_table[i].nextHop = rcm->sender;
          if (TOS_NODE_ID != rcm->destination)
            Route_Rep_Msg(rcm->destination, rcm->value + 1);
          }

          if (TOS_NODE_ID == 1) {  //rcm->destination
            D_M_Send(7, 5);
          }
        }
        //end of if

      return bufPtr;
    }
  }
```

Final result is:

The LED of 6 nodes shown as below.

010,110,111,011,010