18COP531 - Wireless Networks

By Group

- Helin Liu - B516464
- Sayali Chavan - B812081
- Hitesh Rathi - B813146
- Tom Coombs - B831314

Index

1. Introduction

In this coursework we are implementing the Ad-hoc Wireless network with AODV routing protocol using sensinode hardware. Ad-hoc wireless network is Infrastructure-less, peer to peer set up temporarily to meet immediate need.This Wireless Network consists of six sensinodes one acts as a source, one destination and other 4 nodes act as a router. Router receives the data and forward it to the destination hence, they acts as a transceiver.

AODV protocol broadcast Route Request message from source to neighboring node and it keeps rebroadcasting till it reaches the destination. Once reached destination that node reverts back the Route Reply message through unicast path as it has its path stored in reverse table with that it creates forward table to send the RR message and forwards the Data through the same route.The routing algorithm forwards the sensor data from source device to next hop by analyzing the route response received from the neighbor nodes. The route response information includes the number of hops from the destination node, battery level (voltage sensor) and Radio Signal Strength Indication (RSSI) of the neighboring nodes.

2. Requirement and Specification

The designed sensor network have the following specifications:

1. Build an Ad Hoc wireless sensor network by developing AODV routing protocol with six sensor nodes from which four router, one destination and one source

2. Send the measurement of temperature and voltage from source to destination device

3. AODV is designed and implemented depending upon the number of hops from the destination node, battery level (voltage sensor) and Radio Signal Strength Indication (RSSI) of the neighbour nodes.

4. The temperature reading and the battery level of the sender is read, reported, and displayed regularly in all the nodes over which the readings are transmitted.

5. Also , every time button is pressed the temperature reading and the battery level of the source device should be read, reported, and displayed.

6. If the conditions of neighboring node is changed, the source node will be notified and change the next hop selection correspondingly.
   The change can be achieved by:
   - Changing the position of the intermediate node
   - Changing the battery level by switching off the power. The output message from the destination device should indicate the change of the intermediate node selected.

The designed AODV protocol have the following specifications:

1. Each node must be able to generate random data.

2. Each node must be able to save the next hop of a route towards a given destination.

3. Each node must be able to forward a DATA package towards its required destination if the required route is available.

4. Each node must be able to broadcast a ROUTE_REQUEST and initialize the route discovery process if there is no route in the reverse table for the required destination.

5. Each node must send a DATA package once the broadcast message reaches the destination

6. Each node must be able to discard ROUTE_REQUEST it has already received.

8. Each node must be able to improve the route towards a destination depending on upon the number of hops from the destination node, battery level (voltage sensor) and Radio Signal Strength Indication (RSSI) of the intermediate nodes.

10. Each node must be able to forward a ROUTE_REPLY to all neighbours which sent the corresponding ROUTE_REQUEST

3. Design

Sender is responsible for Transmission, sending route discovery and forwarding the packet.

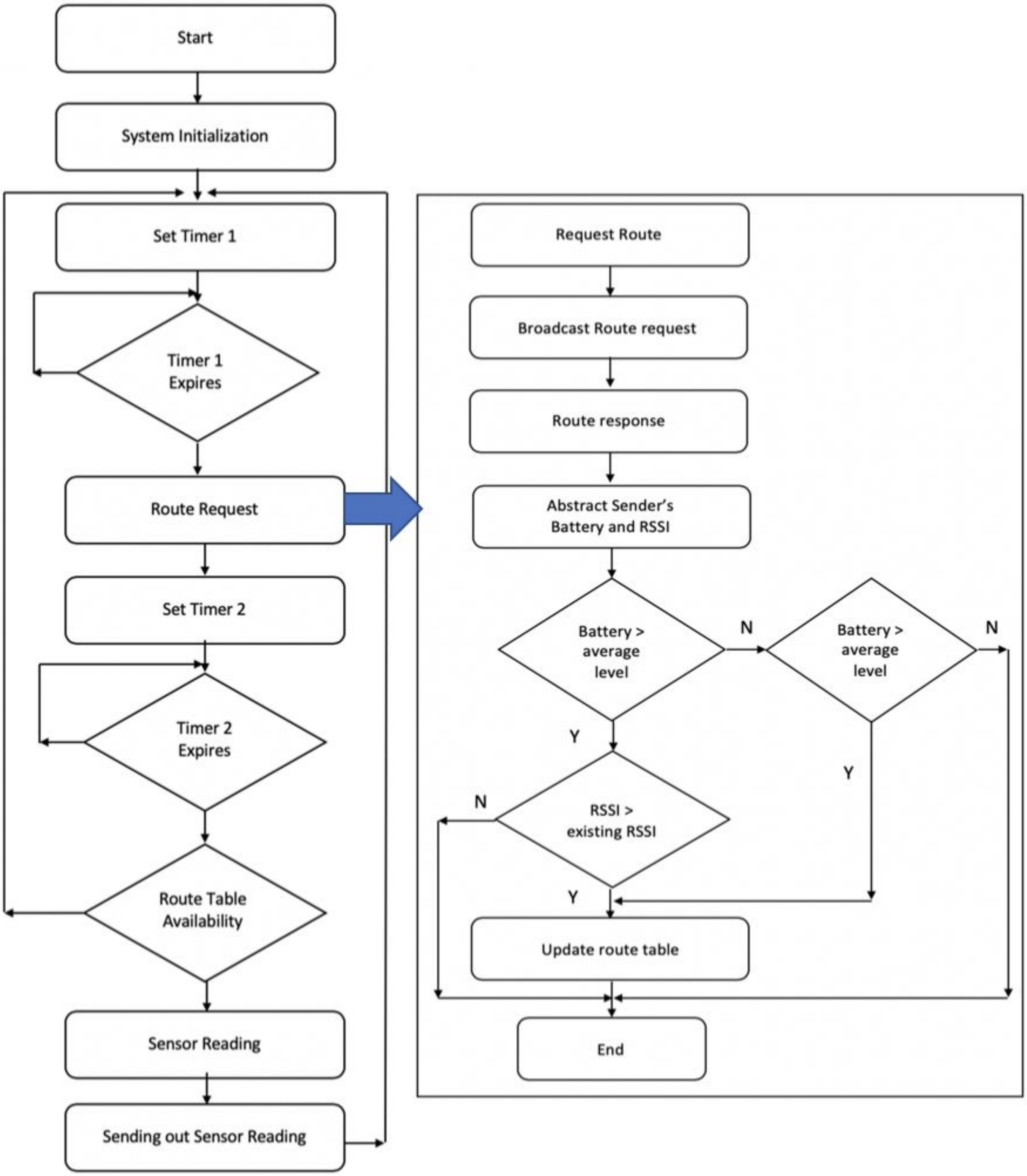


Fig. Sender Flowchart

I are using two timers:
Timer 1 : Sensor reading time interval
Timer 2 : For Route Response

To send a data to destination “Route Discovery” method is required. Which has two important signal RREQ and RREP. RREQ is the route request message broadcasted to the neighbours and wait for RREP route Response message.Then sender will update the routing table with the nest path to the destination.Best hop is decided by node with the higher level, less number of hops and then higher RSSI level. If the current node is the destination node then it will revert back the RREP.

3.2 Routing principle

We are using AODV protocol for designing this Ad-hoc network.Ad Hoc On Demand Distance Vector is one of the most advanced routing protocol. AODV is a dynamic , multi hop routing protocol and on-demand routing. It can execute both unicast as well as multicast routing. AODV is idle when there is no connection establishment request. Once the demand is made an valid route is established and information will always be saved when data is sent.

Features of AODV:

1. Reactive or on Demand
2. Descendant of DSDV
3. Uses bi-directional links
4. Route discovery cycle used for route finding
5. Maintenance of active routes
6. sequence numbers used for loop prevention and as route freshness criteria
7. Provides unicast and multicast communication

There are three types of messages :
1. Route Request (RREQ)
2. Route Replies (RREP)
3. Route Errors(RERR)

Initially,S - Sender broadcasts the RREQ to find the route to required destination node. All neighboring node receives the request and caches the route back to the initial node in backward table. Sender broadcasts the RREQ packet to its neighbors, If it's not the destination node then re-broadcasts a RREQ, also adds up reverse path pointing towards the source.

When the intended destination node receives a RREQ, it replies by sending a Route Reply (RR). RR travels along the reverse path set-up when RREQ was forwarded So using that the RREP unicast message can forwarded from destination to the initial node maintaining routing tables. Routes are maintained only between nodes which need to communicate and who doesn't have have a valid route to that destination, it initiates a Path Discovery Process to detect D-Destination.

3.3 Buffer Management

The buffer management in this architecture and the Rime Stack is simple. All packets, incoming and outgoing are stored in a single buffer, which is called Rime Buffer. Buffer contains the application data as well as packet attribute data i.e. temperature, battery level and rssi. There is a single priority level for accessing the Rime Buffer, so there is no locking mechanism needed to be used.

4. Implementation

4.1 Description of the source code

The sender consists of 3 functions; receive unicast, receive broadcast and the main function. Receive broadcast is never implemented. Receive unicast receives the packet and then performs the appropriate functionality depending the descriptor of the packet. If it is a route reply it calculates the required variables and the triggers the unicast transmission to the next hop.

4.2 Description of the Receiver code

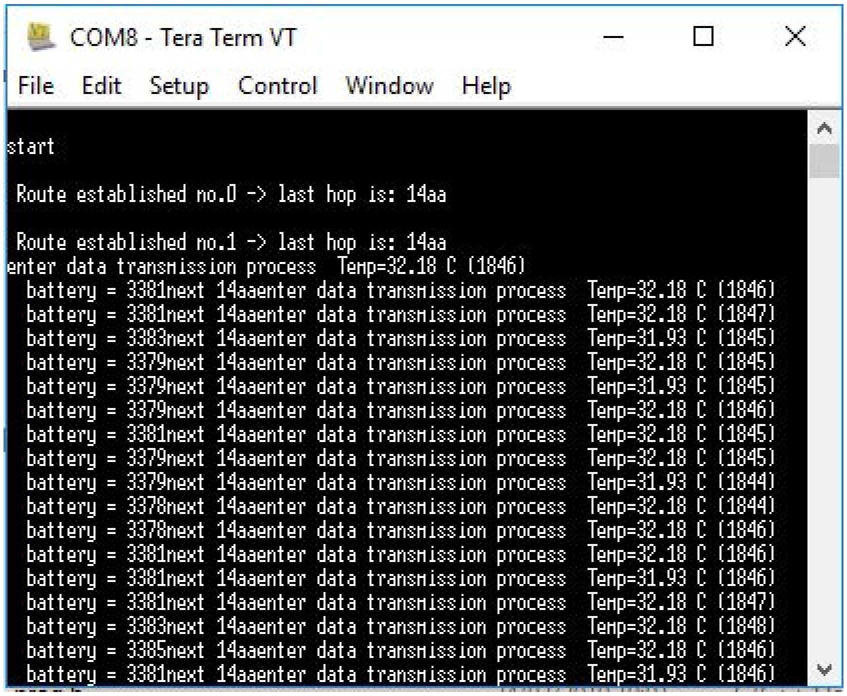
The receiver code has 2 main function, one is unicast callbacks function and another is the broadcast callbacks function, the unicast function receives a packet and depending on the type of packet it carries out a particular function. If it receives a route response it will forward the pack to the next hop (backwards). If it receives a data packet it prints the sensor output and then forwards the pack to the next hop in the forward table.

5. Testing Functions

We established the multi-hop wireless sensor network using AODV and observed the below results:

The temperature reading and the battery level from the source device and The destination node connects to a computer to display the received message, which includes the temperature reading and the battery level of the source device, the result on the:

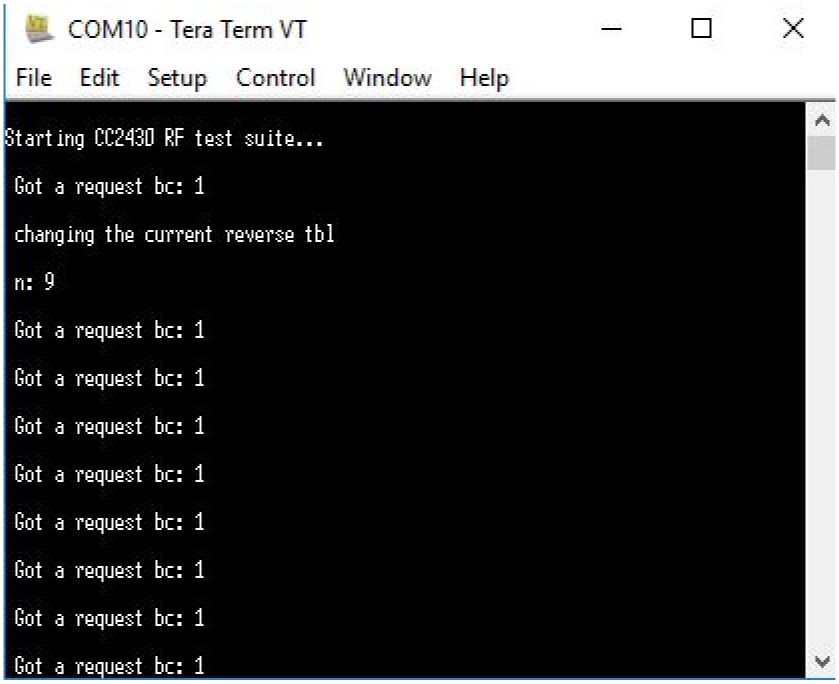
Sender A node



```
start

 Route established no.0 -> last hop is: 14aa

 Route established no.1 -> last hop is: 14aa
enter data transmission process  Temp=32.18 C (1846)
  battery = 3381next 14aaenter data transmission process  Temp=32.18 C (1846)
  battery = 3381next 14aaenter data transmission process  Temp=32.18 C (1847)
  battery = 3383next 14aaenter data transmission process  Temp=31.93 C (1845)
  battery = 3379next 14aaenter data transmission process  Temp=32.18 C (1845)
  battery = 3379next 14aaenter data transmission process  Temp=31.93 C (1845)
  battery = 3379next 14aaenter data transmission process  Temp=32.18 C (1846)
  battery = 3381next 14aaenter data transmission process  Temp=32.18 C (1845)
  battery = 3379next 14aaenter data transmission process  Temp=32.18 C (1845)
  battery = 3379next 14aaenter data transmission process  Temp=31.93 C (1844)
  battery = 3378next 14aaenter data transmission process  Temp=32.18 C (1844)
  battery = 3378next 14aaenter data transmission process  Temp=32.18 C (1846)
  battery = 3381next 14aaenter data transmission process  Temp=32.18 C (1846)
  battery = 3381next 14aaenter data transmission process  Temp=31.93 C (1846)
  battery = 3381next 14aaenter data transmission process  Temp=32.18 C (1847)
  battery = 3383next 14aaenter data transmission process  Temp=32.18 C (1848)
  battery = 3385next 14aaenter data transmission process  Temp=32.18 C (1846)
  battery = 3381next 14aaenter data transmission process  Temp=31.93 C (1846)
```

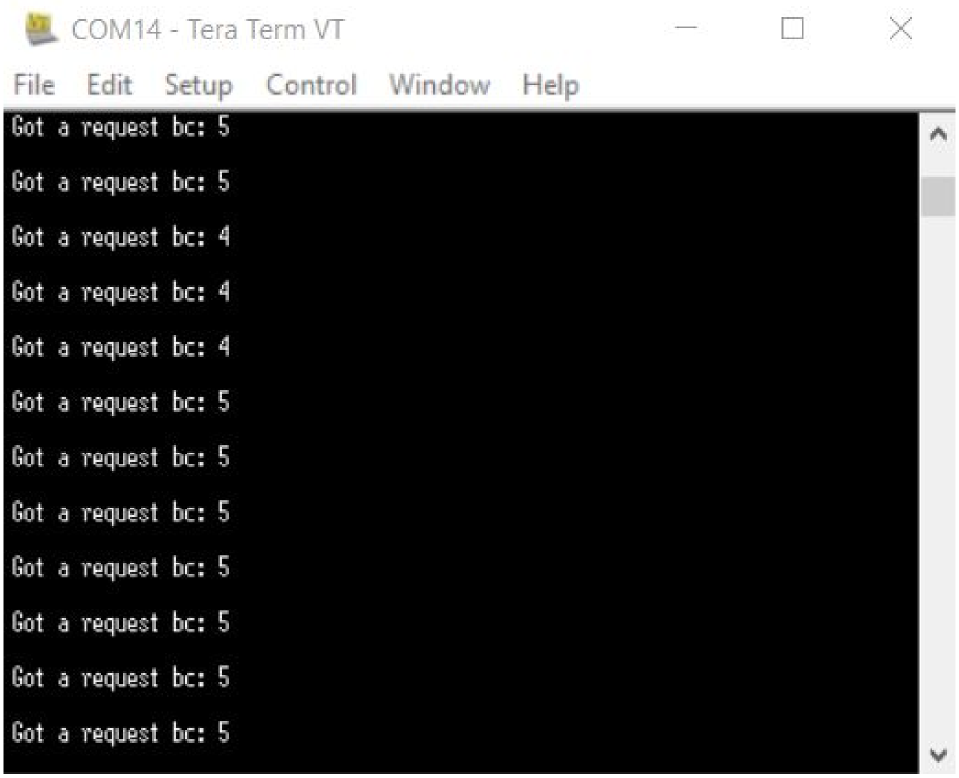
Router B node

When it broadcasts:

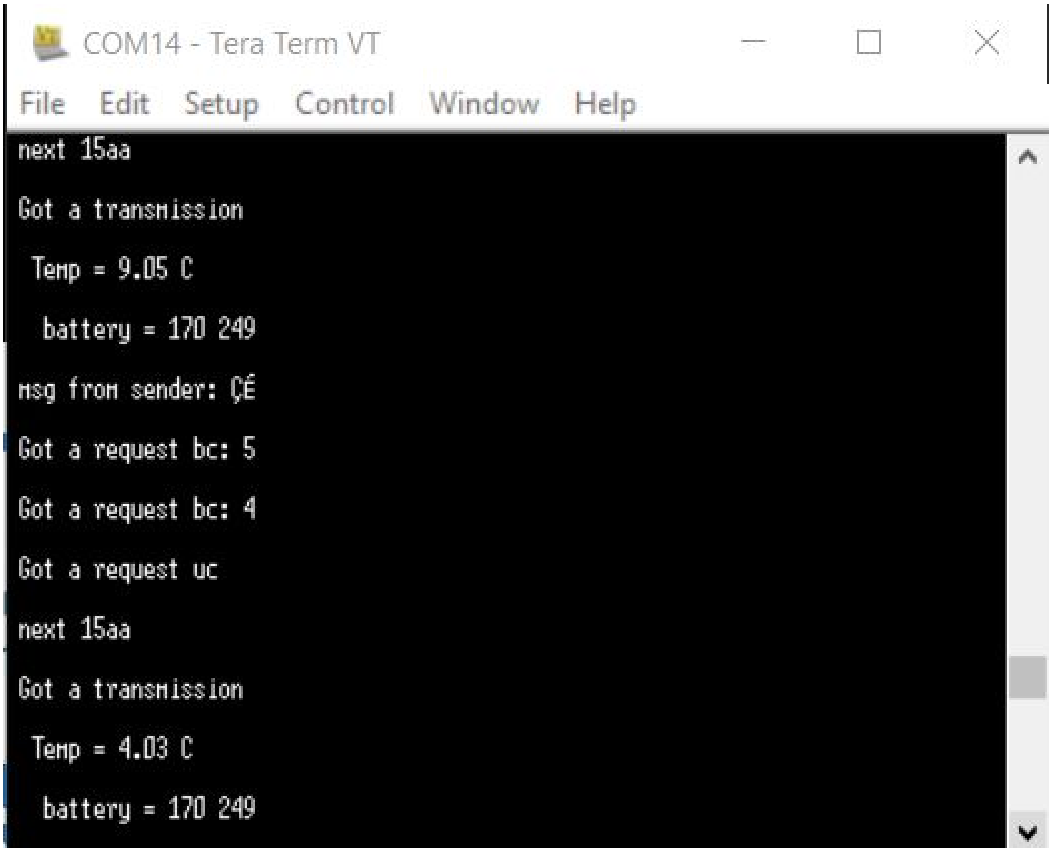


```
Starting CC2430 RF test suite...

 Got a request bc: 1

 changing the current reverse tbl

 n: 9

 Got a request bc: 1

 Got a request bc: 1

 Got a request bc: 1

 Got a request bc: 1

 Got a request bc: 1

 Got a request bc: 1

 Got a request bc: 1

 Got a request bc: 1
```

Router C node

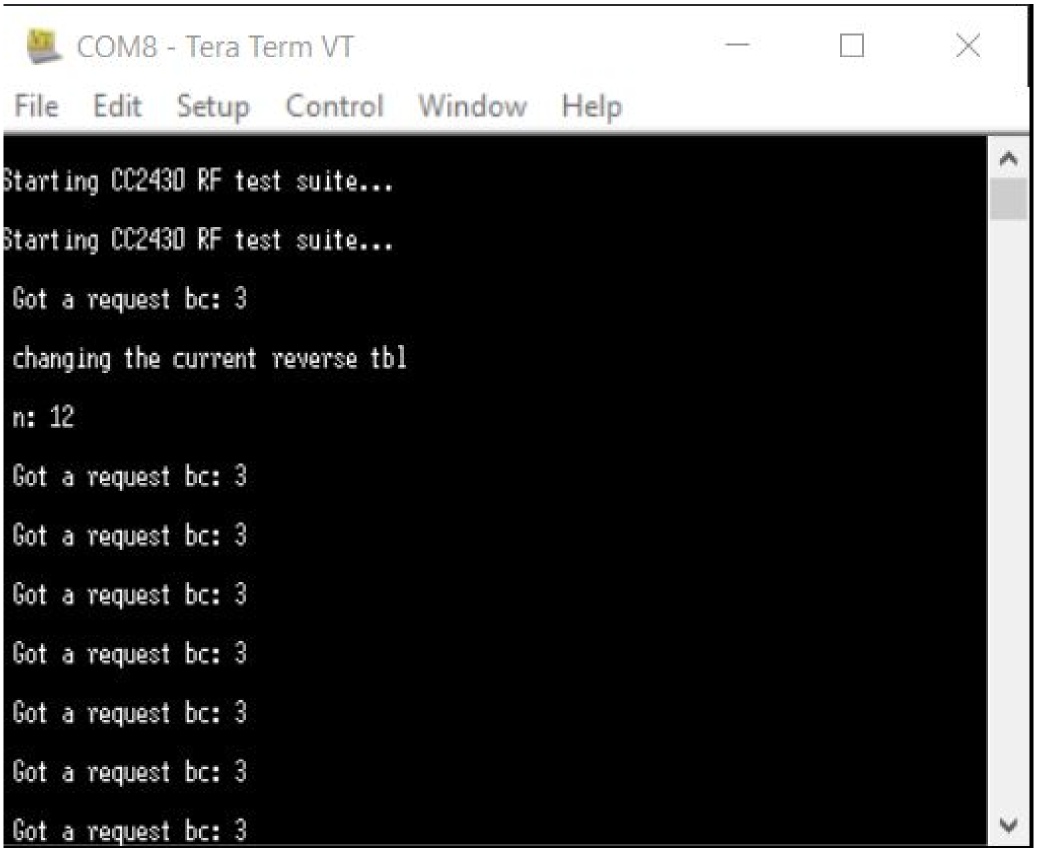


```
Got a request bc: 5

Got a request bc: 5

Got a request bc: 4

Got a request bc: 4

Got a request bc: 4

Got a request bc: 5

Got a request bc: 5

Got a request bc: 5

Got a request bc: 5

Got a request bc: 5

Got a request bc: 5

Got a request bc: 5
```

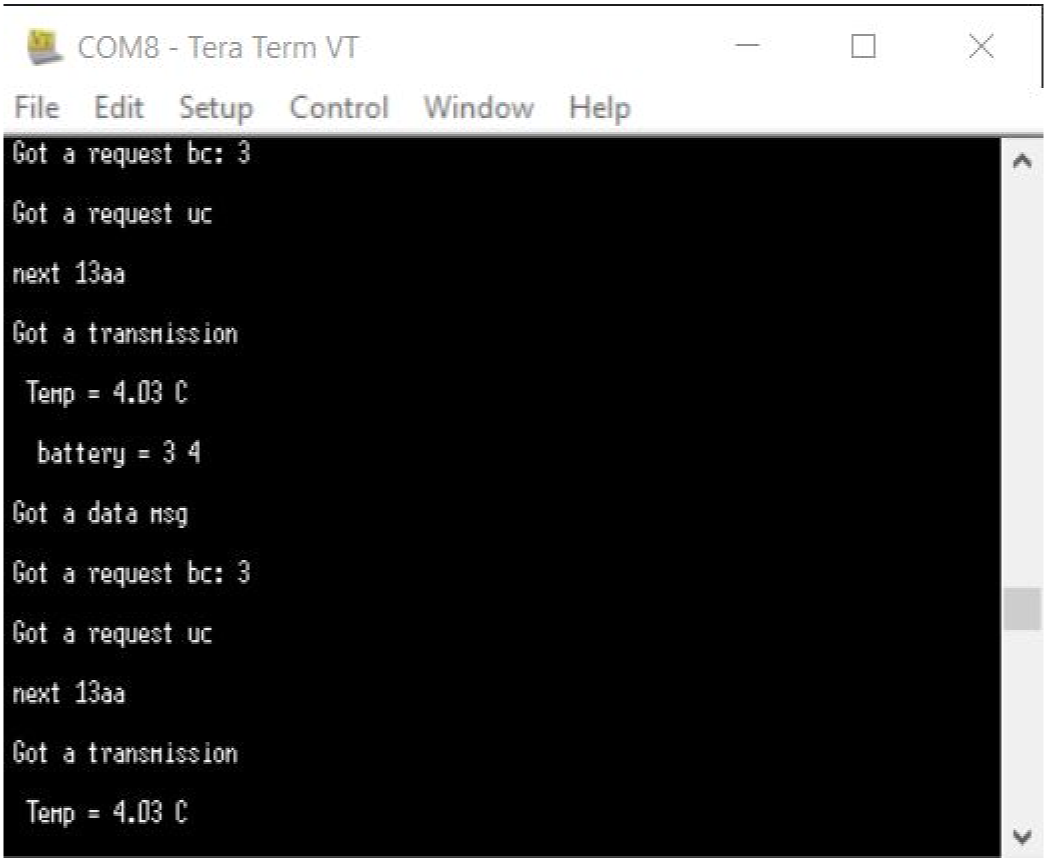


```
next 15aa

Got a transmission

 Temp = 9.05 C

  battery = 170 249

msg from sender: ÇÉ

Got a request bc: 5

Got a request bc: 4

Got a request uc

next 15aa

Got a transmission

 Temp = 4.03 C

  battery = 170 249
```

Router D node

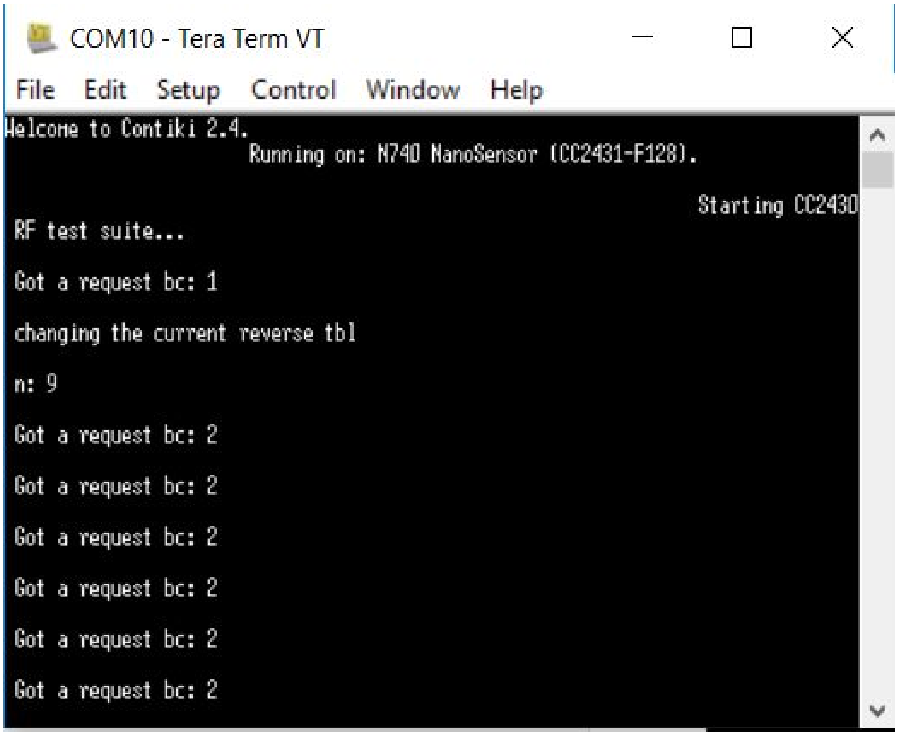


```
Starting CC2430 RF test suite...
Starting CC2430 RF test suite...
 Got a request bc: 3
 changing the current reverse tbl
 n: 12
 Got a request bc: 3
 Got a request bc: 3
 Got a request bc: 3
 Got a request bc: 3
 Got a request bc: 3
 Got a request bc: 3
 Got a request bc: 3
```

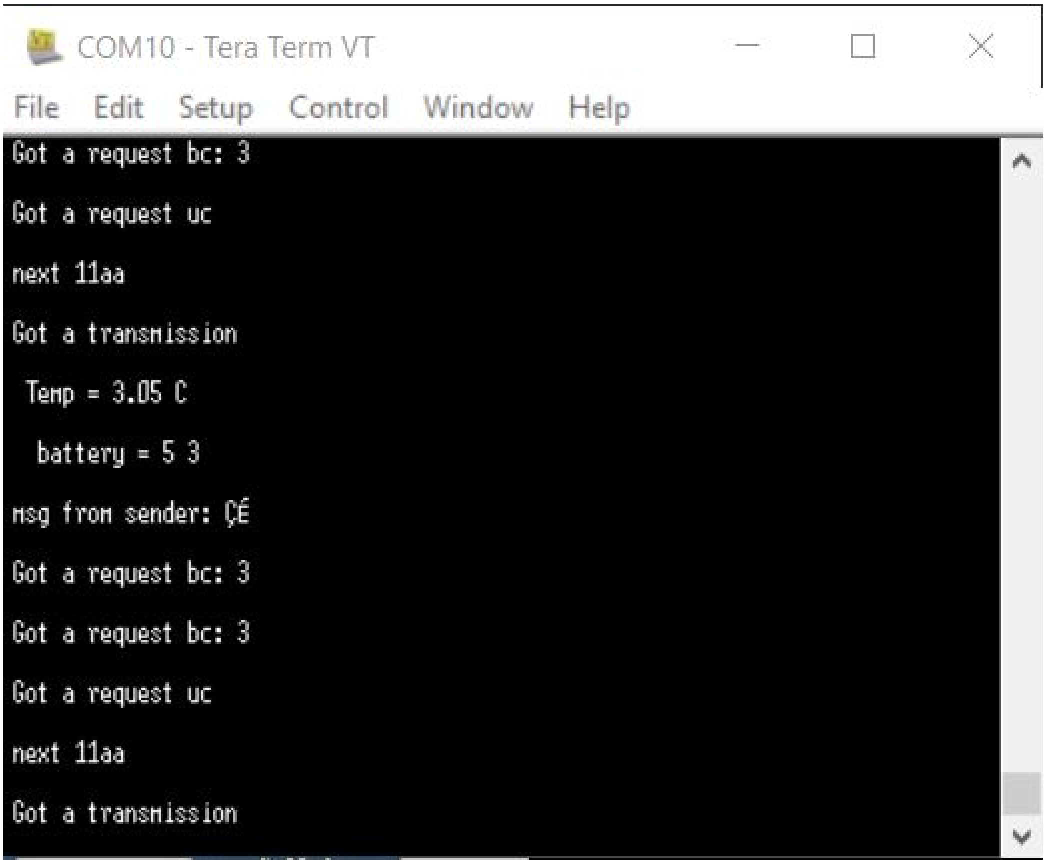


```
 Got a request bc: 3
 Got a request uc
 next 13aa
 Got a transmission
  Temp = 4.03 C
   battery = 3 4
 Got a data msg
 Got a request bc: 3
 Got a request uc
 next 13aa
 Got a transmission
  Temp = 4.03 C
```

Router E node

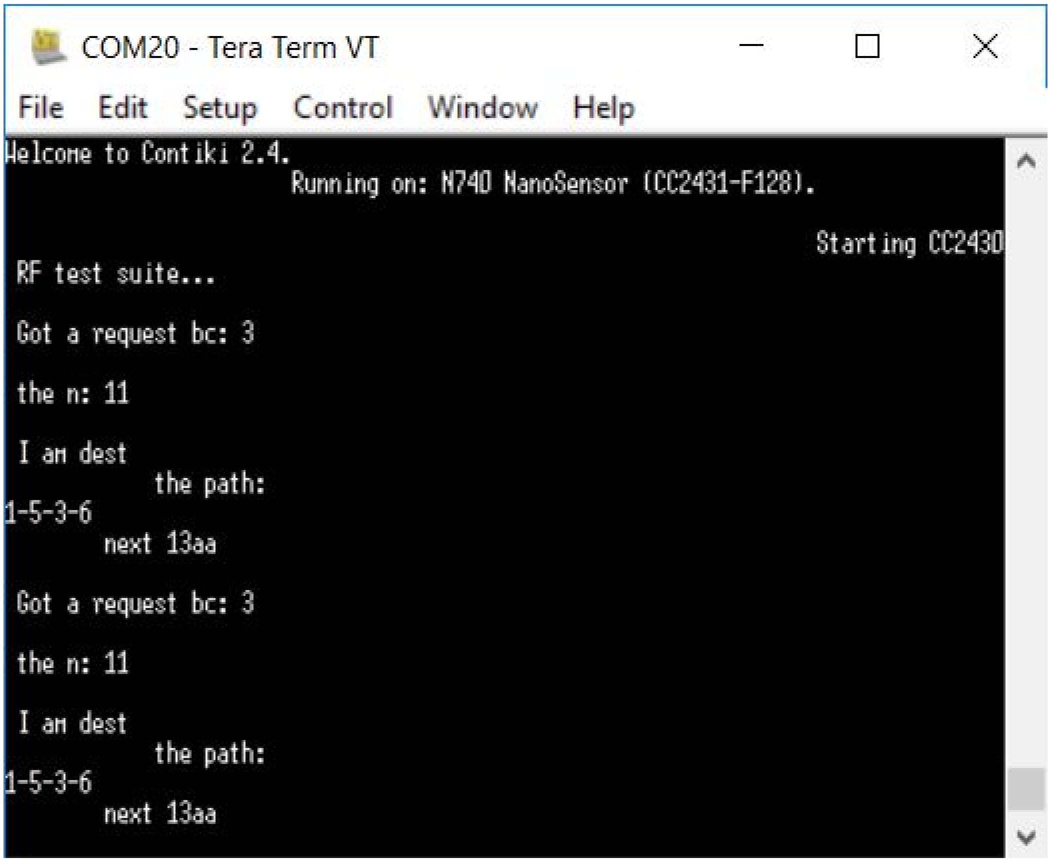


```
Welcome to Contiki 2.4.
                                 Running on: N740 NanoSensor (CC2431-F128).

                                                             Starting CC2430
 RF test suite...

 Got a request bc: 1

 changing the current reverse tbl

 n: 9

 Got a request bc: 2

 Got a request bc: 2

 Got a request bc: 2

 Got a request bc: 2

 Got a request bc: 2

 Got a request bc: 2
```



```
Got a request bc: 3

Got a request uc

next 11aa

Got a transmission

 Temp = 3.05 C

  battery = 5 3

msg from sender: ÇÉ

Got a request bc: 3

Got a request bc: 3

Got a request uc

next 11aa

Got a transmission
```

Destination F node



```
Welcome to Contiki 2.4.
                         Running on: N740 NanoSensor (CC2431-F128).

                                                                              Starting CC2430
 RF test suite...

 Got a request bc: 3

 the n: 11

 I am dest
             the path:
1-5-3-6
        next 13aa

 Got a request bc: 3

 the n: 11

 I am dest
             the path:
1-5-3-6
        next 13aa
```

6. Conclusion and Further work

From above experiment, we can conclude that we have created the Ad-hoc wireless network with six sensinodes. The best path is decided by AODV protocol depending upon the number of hops from the destination node. We tried incorporating RSSI and Battery level change functionality but due to time constraint we were not able to finish it.However, in future we are planning to complete these functionality along with security policies.

7. Appendix

Code: We have used the C programming language.

1. **Sender**

```
#include "contiki.h"
#include "net/rime.h"
#include <string.h>
#include "dev/button-sensor.h"
#include "dev/sensinode-sensors.h"
#include "dev/leds.h"

#include <stdio.h>
#define TABLELENGTH       10

#define COMMAND_ROUTREQUEST   0x20   //Command for requesting route
#define COMMAND_ROUTERESPONSE 0x21//Command for route response
#define COMMAND_DTATTX        0x22 //Command for sending data through unicast

#define BATTERY_AVERGE_LVL    3000

typedef struct          //group of data elements grouped together under one name
{
        uint16_t u16Dest;    //The destation address
        uint16_t u16NextHop;        //The next hop which point to the destination address
        uint16_t u16Battery;
        uint16_t u16Rssi;
} tsRouteTable;

typedef struct          //group of data elements grouped together under one name
{
   uint16_t fDest;
   uint16_t nextHop;
   uint16_t origin;
   uint16_t count;
} rFwdTable;

static rFwdTable fwdTable;
static tsRouteTable sRouteTable[TABLELENGTH];     //creates arrary of type
tsRouteTable and size of TABLELENGTH

static rimeaddr_t addr;
static uint8_t destination;
static struct unicast_conn uc;
static struct broadcast_conn bc;
static uint8_t u8DataBuffer[50];
static char u8DataBufferText[50];

static uint8_t name = 1;

static uint8_t path;

static const struct broadcast_callbacks broadcast_callbacks = {recv_bc};    //set the
broadcast callback function to recv_bc
static const struct unicast_callbacks unicast_callbacks = {recv_uc};  //set the unicast
callback function to recv_uc


static int rv;
static struct sensors_sensor * sensor;
static float sane = 0;
static uint16_t battery;
static uint8_t temperature1=0;
static uint8_t temperature2=0;

static uint8_t brdcstCounter=0;
static uint8_t brdcstLimit=4;
static uint8_t brdcstID=1;

uint16_t dest=0;
uint16_t origin=0;
uint16_t source=0;


static uint8_t jj = 0;

static uint8_t ch = 1;
```

```
/*---------------------------------------------------------------------------*/
PROCESS(routenode_process, "Example unicast");
AUTOSTART_PROCESSES(&routenode_process);
/*---------------------------------------------------------------------------*/

static void
recv_uc(struct unicast_conn *c, const rimeaddr_t *from)
{

  uint8_t * data;
  uint16_t dest=0;
  uint16_t nexthop=0;
  uint16_t source=0;
  uint16_t src=0;
  uint16_t battery=0;
  uint16_t rssi=0;
  static int i=0;
  static int m=0;

  unsigned int bSuccess=0;
  unsigned int bFound=0;

  data = packetbuf_dataptr();

 switch(data[0])
 {
          case COMMAND_ROUTERESPONSE:
                    printf("\n Route established no.%d -> ",jj);
                    jj++;

                    //get the destination
       dest = data[1];
       dest = dest << 8;
       dest = dest | data[2];

       //get the origin of the packet
       origin = data[6];
       origin = origin << 8;
       origin = origin | data[7];
```

```
          source = from->u8[0];
source = source <<8;
source = source | from->u8[1];
          addr.u8[1] = source;
addr.u8[0] = source>>8;
          printf("last hop is: %02x%02x \n\r",addr.u8[1],addr.u8[0]);

          ch = 2;



          fwdTable.fDest = dest;
          fwdTable.nextHop = source;


          /* bSuccess=0;
               dest = data[1];
                    dest = dest << 8;
                    dest = dest | data[2];

            src = from->u8[1];
  src = src <<8;
  src = src | from->u8[0];

  battery = data[4];
  battery = battery << 8;
  battery = battery | data[3];
  //printf("has respo %d%d\r\n",from->u8[0],from->u8[1],battery);
  rssi = packetbuf_attr(PACKETBUF_ATTR_RSSI);

  for(i=0; i<TABLELENGTH; i++)
  {
          if(sRouteTable[i].u16Dest==dest)
                 {
                         bSuccess=1;

                         if(sRouteTable[i].u16NextHop == src)
                                {
                                           sRouteTable[i].u16Rssi = rssi;
                         sRouteTable[i].u16Battery = battery;
                    }else{
                if(battery > BATTERY_AVERGE_LVL)
                    {

                        //printf("Received rssi=%d, from
%d\r\n",rssi,src);
                        if(rssi > sRouteTable[i].u16Rssi)
                            {
                                //printf("stored rssi=%d, from
%d\r\n",sRouteTable[i].u16Rssi,sRouteTable[i].u16NextHop);
                                sRouteTable[i].u16NextHop =
src;
                         sRouteTable[i].u16Rssi = rssi;
                         sRouteTable[i].u16Battery = battery;
                            }
                    }else{
                         if(battery > sRouteTable[i].u16Battery)
                            {
                                sRouteTable[i].u16NextHop =
src;
                         sRouteTable[i].u16Rssi = rssi;
                         sRouteTable[i].u16Battery = battery;
                            }
                    }
                    break;
            }
        }
    }

    if(!bSuccess)
     {
         for(i=0; i<TABLELENGTH; i++)
     {
         if(sRouteTable[i].u16Dest==0x0000)
             {
                sRouteTable[i].u16Dest=dest;
                        sRouteTable[i].u16NextHop = src;
                         sRouteTable[i].u16Rssi = rssi;
                         sRouteTable[i].u16Battery = battery;
            }
        }
    }
        break; */

        default:
            break;
}

  packetbuf_clear();
}

static void
recv_bc(struct broadcast_conn *c, rimeaddr_t *from)
{


          /*from->u8[0],
          from->u8[1],
          packetbuf_datalen(),
          (char *)packetbuf_dataptr());*/

  packetbuf_clear();
}


/*---------------------------------------------------------------------------*/
PROCESS_THREAD(routenode_process, ev, data)
{
  static struct etimer et;
  static uint8_t i=0;
  static uint8_t m=0;
  static int dec;
  static float frac;
  static uint16_t u16Dest=0xAA16;
  static uint16_t u16Origin=0xAA11;
  static uint8_t bFound=0;
  static uint8_t fstbuffer = name;
  static uint8_t sndbuffer = '\0';
  static uint8_t trdbuffer = '\0';
```

```
 static uint8_t couter = 1;


 //PROCESS_EXITHANDLER(unicast_close(&uc);)
 PROCESS_BEGIN();

 for(i=0; i<TABLELENGTH; i++)   //for each entry in the routing set the default values
 {
        sRouteTable[i].u16Dest=0x0000;
        sRouteTable[i].u16NextHop=0xffff;
        sRouteTable[i].u16Battery=0;
        sRouteTable[i].u16Rssi=0;
 }


 printf("\nstart\n\r");
 broadcast_open(&bc, 128, &broadcast_callbacks);    //set up broadcast
 unicast_open(&uc, 129, &unicast_callbacks);  //set up unicasting


etimer_set(&et, CLOCK_SECOND * 2); //set a 2 second timer
  while(ch == 1)      //start infinite loop
  {

    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et)); //if 2seconds has passed
    if(i==0)    //do route request
        {
                //set some variables to some sensor readings
      /* sensor = sensors_find(ADC_SENSOR);

      rv = sensor->value(ADC_SENSOR_TYPE_TEMP);
      if(rv != -1) {
      sane = ((rv * 0.61065 - 773) / 2.45);
      dec = sane;
      temperature1 = dec;
      frac = sane - dec;
      temperature2 = (unsigned int)(frac*100);
      //printf("  Temp=%d.%02u C (%d)\n\r", dec, (unsigned int)(frac*100), rv); */
```

```
          //set some more variables from sensor values
        rv = sensor->value(ADC_SENSOR_TYPE_VDD);


           //put some stuff in a buffer
      //printf("  Supply=%d\n\r", battery);
      u8DataBuffer[0] = COMMAND_ROUTREQUEST;
      u8DataBuffer[1] = u16Dest>>8;
      u8DataBuffer[2] = u16Dest;
      u8DataBuffer[3] = brdcstCounter;
      u8DataBuffer[4] = brdcstLimit;
      u8DataBuffer[5] = brdcstID;
                 u8DataBuffer[6] = u16Origin>>8;
      u8DataBuffer[7] = u16Origin;
                 u8DataBuffer[8] = name;    //0
                 /* u8DataBuffer[9] = sndbuffer; //1
                 u8DataBuffer[10] = sndbuffer; //2
                 u8DataBuffer[11] = trdbuffer; //3
                 u8DataBuffer[12] = trdbuffer; //4
                 u8DataBuffer[13] = trdbuffer; //5
                 u8DataBuffer[14] = couter;// */



       brdcstID++;
       packetbuf_copyfrom(u8DataBuffer, 9);     //copy some stuff from buffer and
broadcast it
       broadcast_send(&bc);
                 packetbuf_clear();
                 couter = 0;
       //printf("brdcst");

    }

         else{  //do if route table is available then send packet

        for(m=0; m<TABLELENGTH; m++)        //find the destination node that the
sender wishes to send to
        {
                 if(u16Dest == sRouteTable[m].u16Dest)
```

```
                    {
                            bFound=1;

                            break;
                    }
        }

        if(bFound){    //for the destination node
                u8DataBuffer[0] = COMMAND_DTATTX;
                u8DataBuffer[1] = u16Dest>>8;
                u8DataBuffer[2] = u16Dest;
                u8DataBuffer[3] = rimeaddr_node_addr.u8[0];   //first 2 digits
                u8DataBuffer[4] = rimeaddr_node_addr.u8[1];   //next 2 digits
                u8DataBuffer[5] = temperature1;
                u8DataBuffer[6] = temperature2;
                u8DataBuffer[7] = battery>>8;
                u8DataBuffer[8] = battery;

                        packetbuf_copyfrom(u8DataBuffer, 9);
                addr.u8[0] = sRouteTable[m].u16NextHop;
                addr.u8[1] = sRouteTable[m].u16NextHop>>8;
                //printf("next %d%d",addr.u8[0],addr.u8[1]);
                unicast_send(&uc, &addr);
        }
}

  if(i==0)
        {
                etimer_set(&et, CLOCK_SECOND * 2);
                i=1;
        }else{
                etimer_set(&et, CLOCK_SECOND * 1);
                i=0;
        }

}

        while(ch == 2){

                //take readings
```

```
             sensor = sensors_find(ADC_SENSOR);

     rv = sensor->value(ADC_SENSOR_TYPE_TEMP);
     if(rv != -1) {
     sane = ((rv * 0.61065 - 773) / 2.45);
     dec = sane;
     temperature1 = dec;
     frac = sane - dec;
     temperature2 = (unsigned int)(frac*100);
     //printf("  Temp=%d.%02u C (%d)\n\r", dec, (unsigned int)(frac*100), rv);

             rv = sensor->value(ADC_SENSOR_TYPE_VDD);

             if(rv != -1) {
      sane = rv * 3.75 / 2047;
      battery = sane*1000;
              }


             printf("enter data transmission process");
             PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));       //if 2seconds
has passed

             u8DataBuffer[0] = COMMAND_DTATTX;
             u8DataBuffer[1] = 255;
       u8DataBuffer[2] = 6;
       u8DataBuffer[3] = temperature1;
       u8DataBuffer[4] = temperature2;
       u8DataBuffer[5] = battery>>8;
       u8DataBuffer[6] = battery;
             packetbuf_copyfrom(u8DataBuffer, 7);
       addr.u8[1] = fwdTable.nextHop;
       addr.u8[0] = fwdTable.nextHop>>8;
             printf("  Temp=%d.%02u C (%d)\n\r", dec, (unsigned int)(frac*100), rv);
             printf("  battery = %d", battery);
       printf("next %02x%02x",addr.u8[1],addr.u8[0]);
       unicast_send(&uc, &addr);
```

```
                if(i==0)
        {
                etimer_set(&et, CLOCK_SECOND * 2);
                i=1;
        }else{
                etimer_set(&et, CLOCK_SECOND * 1);
                i=0;
        }
        }


}
PROCESS_END();
        }
```

2. **Destination/Receiver**

```
#include "contiki.h"
#include "net/rime.h"
#include <stdio.h> /* For printf() */
#include "cc2430_sfr.h"

#define TABLELENGTH        10

#define COMMAND_ROUTREQUEST    0x20    //Command for requesting route
#define COMMAND_ROUTERESPONSE 0x21 //Command for route response
#define COMMAND_DTATTX         0x22 //Command for sending data through
unicast

#define BATTERY_AVERGE_LVL    3000


typedef struct            //group of data elements grouped together under one name
{
   uint16_t fDest;
   uint16_t nextHop;
   uint16_t origin;
   uint16_t count;
} rFwdTable;
```

```
typedef struct          //group of data elements grouped together under one name
{
   uint16_t bDest;
   uint16_t nextHop;
   uint16_t origin;
   uint16_t count;
} rBwdTable;

static rBwdTable bwdTable;
static rFwdTable fwdTable;

static rimeaddr_t addr;
static uint8_t destination;
static struct etimer et;
static struct unicast_conn uc;
static struct broadcast_conn bc;    //broadcast_conn struct
static const struct broadcast_callbacks broadcast_callbacks = {recv_bc};
//Register the callback routine
static const struct unicast_callbacks unicast_callbacks = {recv_uc};

static uint8_t u8DataBuffer[50];    //create my buffer
static int i = 0;
static uint16_t myAddress=0xAA16;
static uint16_t addressbook[6] =
{0xAA11,0xAA12,0xAA13,0xAA14,0xAA15,0xAA16};

static uint8_t name = 6;

//static char letter_name = "F";

uint16_t dest=0;
uint16_t origin=0;
uint16_t source=0;

static long path;
static uint8_t nonc;

uint8_t path_arr[6];
uint8_t num = 0;
```

```
uint8_t count = 0;
uint8_t counter = 0;
uint8_t y = 0;
uint8_t u;
uint8_t checker = 0;
uint16_t battery =0;




/*---------------------------------------------------------------------------*/
PROCESS(rf_test_process, "RF test RX process"); //declare a process  called
"rf_test_process" and a string to identify it
AUTOSTART_PROCESSES(&rf_test_process);  //defines the process which will
be loaded when the system starts up i.e rf_test_process
/*---------------------------------------------------------------------------*/

static void
recv_uc(struct unicast_conn *c, const rimeaddr_t *from)
{
    uint8_t * data;

          data = packetbuf_dataptr();
    //unsigned int atDest=0;
    //uint16_t source=0;

    switch(data[0])
    {
        case COMMAND_ROUTERESPONSE:
                                printf("\n Got a request uc \n\r");
            //update forward TABLE
            //look up nexthop in in backwards table and send there

            //get the destination
            dest = data[1];
            dest = dest << 8;
            dest = dest | data[2];

            //get the origin of the packet
            origin = data[6];
            origin = origin << 8;
            origin = origin | data[7];

            source = from->u8[0];
            source = source <<8;
            source = source | from->u8[1];

            fwdTable.fDest = dest;
            fwdTable.nextHop = source;
            //fwdTable[i].origin = origin;
            fwdTable.count = data[3];
            //maybe increment


            u8DataBuffer[0] = COMMAND_ROUTERESPONSE;
            u8DataBuffer[1] = data[1];
            u8DataBuffer[2] = data[2];
            u8DataBuffer[3] = data[3];
            u8DataBuffer[4] = data[4];
            u8DataBuffer[5] = data[5];
            u8DataBuffer[6] = data[6];
            u8DataBuffer[7] = data[7];
                    u8DataBuffer[8] = data[8];
                    u8DataBuffer[9] = data[9];
                    u8DataBuffer[10] = data[10];
                    u8DataBuffer[11] = data[11];
                    u8DataBuffer[12] = data[12];
            //strcpy(u8DataBuffer[8] ,route);
            //add battery level

            packetbuf_copyfrom(u8DataBuffer,13);
            addr.u8[1] = bwdTable.nextHop;
            addr.u8[0] = bwdTable.nextHop>>8;
            printf("\n next %02x%02x \n\r",addr.u8[1],addr.u8[0]);
            unicast_send(&uc, &addr);

                case COMMAND_DTATTX:
                printf("\n Got a transmission \n\r");
                printf(" \n Temp = %d.%02u C \n\r", data[3], data[4]);
                printf(" \n  battery = %d %d\n\r",data[6], data[5]);
```

```
                        if(data[2] != name){
                        printf("\n Got a data msg\n\r");
                        packetbuf_copyfrom(data,7);
                        addr.u8[1] = fwdTable.nextHop;
         addr.u8[0] = fwdTable.nextHop>>8;
                        unicast_send(&uc, &addr);
                        }
                        if(data[2] == name){

                                printf("\n msg from sender: %s\n\r",data[1]);

                        }

      default:
         break;
   }
}

//define callback function for broadcast reception
static void
recv_bc(struct broadcast_conn *c, rimeaddr_t *from)
{
        uint8_t * data;
        uint8_t n;

        data = packetbuf_dataptr();
        n =  packetbuf_datalen();



        //uint16_t dest=0;
        //uint16_t origin=0;
        //uint16_t source=0;

        switch(data[0])
   {

      case COMMAND_ROUTREQUEST:
                        printf("\n Got a request bc: %d \n\r",data[n-1]);
```

```
               checker = 0;
               // get the path
               //path = data[8];


     //get the destination
     dest = data[1];
     dest = dest << 8;
     dest = dest | data[2];

     //get the origin of the packet
     origin = data[6];
     origin = origin << 8;
     origin = origin | data[7];

     //get the next hop (backwards)
     source = from->u8[0];
     source = source <<8;
     source = source | from->u8[1];

               for ( y = 0;y < 6;y++){
                    if(source == addressbook[y]){
                         checker = 1;
                    }
               }
               /* printf("the checker: %d\n",checker);
               printf("the source: %02x\n",source);
               printf("the destination: %02x\n",dest);
               printf("the my addr: %02x\n",myAddress); */
     if((myAddress != dest)&&(checker == 1))   //if not at the destination
     {
                    printf('go in 1');
       if((bwdTable.bDest != origin) && (bwdTable.nextHop != source)) //if not
already recieved a packet
       {

                         printf("\n changing the current reverse tbl \n\r");

                         printf("\n n: %d \n\r",n);
                         data[n] = name;
```

```
                                        bwdTable.bDest = origin;
                        bwdTable.nextHop = source;
                        //bwdTable.origin = origin;
                        bwdTable.count = data[3];
                        //rebroadcast
                        packetbuf_copyfrom(data, n+1);     //copy some stuff from buffer and
broadcast it

                                        broadcast_send(&bc);
                      }
                                   data[n] = name;
                                   packetbuf_copyfrom(data, n+1);
                                   broadcast_send(&bc);
                }

                else{   //if at destation




                                   printf("\n the n: %d \n\r",n);
                                   printf("\n I am dest \n the path: \n\r");

                                   for(u = 8; u < n ; u++){

                                        printf("%d-",data[u]);

                                   }
                                   printf("%d",name);
                                   packetbuf_clear();


                      //update back table
                      bwdTable.bDest = origin;
                      bwdTable.nextHop = source;
                      //bwdTable[i].origin = origin;
                      bwdTable.count = data[3];
```

```
            //update forward table
            fwdTable.fDest = dest;
            fwdTable.nextHop = dest;
            fwdTable.origin = origin;
            //fwdTable.count = data[3];

            //send a unicast message to the next hop in reverse TABLE
            //unicast message must be RREP
            u8DataBuffer[0] = COMMAND_ROUTERESPONSE;
            u8DataBuffer[1] = data[1];
                            u8DataBuffer[2] = data[2];
                            u8DataBuffer[3] = data[3];
                            u8DataBuffer[4] = data[4];
                            u8DataBuffer[5] = data[5];
                            u8DataBuffer[6] = data[6];
                            u8DataBuffer[7] = data[7];
                            //strcpy(u8DataBuffer[8] ,route);
                            //u8DataBuffer[8] = nonc;
           // u8DataBuffer[8] = route;
                            //u8DataBuffer[9] = data[9];
                                    //u8DataBuffer[10] = data[10];
                                    //u8DataBuffer[11] = data[11];
                                    //u8DataBuffer[12] = data[12];
            //battery lvl goes here in buffer

            packetbuf_copyfrom(u8DataBuffer,8);
            addr.u8[1] = bwdTable.nextHop;
            addr.u8[0] = bwdTable.nextHop>>8;
            printf("\n next %02x%02x \n\r",addr.u8[1],addr.u8[0]);
            unicast_send(&uc, &addr);

        }

        default:
                    break;
    }
}


/*------------------------------------------------------------------------*/
```

```
PROCESS_THREAD(rf_test_process, ev, data) //define the body of the thread
{
    PROCESS_BEGIN(); //begin the process
    printf("\nStarting CC2430 RF test suite...\n\r");

    //initialise tables

           for (i=5;i>=0;i--){
                      path_arr[i] = 0;

           }

        bwdTable.bDest = 0x0000;
        bwdTable.nextHop = 0xffff;
        bwdTable.origin = 0xffff;
        bwdTable.count = 0;

        fwdTable.fDest = 0x0000;
        fwdTable.nextHop = 0xffff;
        fwdTable.origin = 0xffff;
        fwdTable.count = 0;


    broadcast_open(&bc, 128, &broadcast_callbacks); //Open the channel for
broadcast (bc = struct for broadcast, 128 = channel, )
                                                     //A struct broadcast_callbacks with function
pointers to functions that will be called when a packet has been received
    unicast_open(&uc, 129, &unicast_callbacks);           //set up unicasting

    etimer_set(&et, CLOCK_SECOND);  //Setup an event timer to let a process
execute once per second.
    while(1) {
        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));  //wait until timer
expires
        etimer_reset(&et);  //reset the timer
    }
             PROCESS_END();
}
/*---------------------------------------------------------------------------*/
```

3. **Router:**
   In each router we just change “myAddress” variable and “name” of its node all code is same as that of Receiver/Destination.

4. **Make file : the target changes for Router**

```
ifndef TARGET
TARGET=sensinode
endif

# Make absolutely certain that you specify your device here
DEFINES=MODEL_N740

# These examples don't need code banking so we turn it off
#HAVE_BANKING=1

CONTIKI_PROJECT = Receiver


all: $(CONTIKI_PROJECT)

CONTIKI = ../../..
include $(CONTIKI)/Makefile.include
```

5. **binary file :** sender.ihx and receiver.ihx are included in zip folder