**IoT 2023 CHALLENGE 3**

## INTERNET OF THINGS - 054323

## *Prof Cesana Matteo*

*P.hD Fabio Palmese*

(1) Name: Gustavo Santos Torrico Person Code: 10911816

(2) Name: Claudia Baz Alvarez Person Code: 10916443

## Index

[**Methodology 1**](#_ckz6zoyuopu2)

[**System Architecture 1**](#_i7kdhi54phfy)

[RadioRoute.h 1](#_t97orwqm62r)

[RadioRouteApp.nc 2](#_bdqkpw3gci83)

[RadioRoute.nc 2](#_6uux8fhyz1ca)

[Interface declarations 2](#_s9tm6a33us7l)

[Functionalities definition and Flow Diagram 3](#_qzk208v4q1u9)

[**Bibliography 5**](#_6560jssvdxsc)

## Methodology

This laboratory challenger was based on the documents presented in class[1] and the base code provided by previous laboratories.

Based on the documentation provided the flux diagram of the project was drawn to gain a more intuitive understanding of the system and to help develop the code base.

Sequentially, the base code of the TinyOS laboratory was imported as a base model. Even Though the functionalities were different, the main function calls of the *<component>.nc* were practically the same.

Finally, as a method to evaluate the optimality of the code and to help its debug, the *dbg()* function was used to output the program.

## System Architecture

The based code is composed of mainly three components[2]:

### RadioRoute.h

Where the message payload type is defined. All the packets have the same interface, using the payload attributes as needed. The interface ahead is the one used by the nodes to exchange packets:

#ifndef RADIO\_ROUTE\_H

#define RADIO\_ROUTE\_H

typedef nx\_struct radio\_route\_msg {

    /\*

    type: defines the message format

        0 - DATA MSG

        1 - ROUTE\_REQ MSG

        2 - ROUTE\_REPLY

    \*/

    nx\_uint16\_t type;

    nx\_uint16\_t src;  // Sender

    nx\_uint16\_t dest; // Node Requested

    /\*

    value: defines msg payload

    The meaning of the value attr changes

    with the type of the msg:

        if DATA MSG  - led value

        if ROUTE\_REQ - none

        if REPLY\_REQ - hop\_count

    \*/

*Figure 1: RadioRoute.h component, definition of payload interface*

### RadioRouteApp.nc

Wrapper component used to link all the components interfaces. It declares the RadioRoute component and connects all its attributes with the corresponding TinyOs components:

#include "RadioRoute.h"

configuration RadioRouteAppC {}

implementation {

/\*\*\*\*\*\* COMPONENTS \*\*\*\*\*/

  components MainC, RadioRouteC as App;

  components LedsC;

  components new AMSenderC(AM\_RADIO\_COUNT\_MSG);

  components new AMReceiverC(AM\_RADIO\_COUNT\_MSG);

  components new TimerMilliC() as Timer0;

  components new TimerMilliC() as Timer1;

  components ActiveMessageC;

  /\*\*\*\*\*\* INTERFACES \*\*\*\*\*/

  App.Boot -> MainC.Boot; //Boot interface

  /\*\*\*\*\*\* Wire the other interfaces down here \*\*\*\*\*/

  App.Receive -> AMReceiverC;

  App.AMSend -> AMSenderC;

  App.AMControl -> ActiveMessageC;

  App.Leds -> LedsC;

  App.Timer0 -> Timer0;

  App.Timer1 -> Timer1;

  App.Packet -> AMSenderC;

}

*Figure 2: RadioTouteApp.nc , component wrapper*

### RadioRoute.nc

This component holds all the functionalities of the RadioRoute component. Its interfaces are connected on the *RadioRoutApp.nc* and its payload type is defined at the *RadioRoute.h.* Its structure is defined by the function calls of its interfaces: defined by the TinyOS documentation[3]:

#### Interface declarations

#include "Timer.h"

#include "RadioRoute.h"

module RadioRouteC @safe() {

  uses {

interface Boot;

//interface for LED

interface Leds;

//interface for timers

interface Timer<TMilli> as Timer0;// used to impose a delay on sent packets

interface Timer<TMilli> as Timer1;// Used as the total time of the simulation

//interfaces for communication

interface SplitControl as AMControl;

interface AMSend;

interface Receive;

interface Packet;

}

}

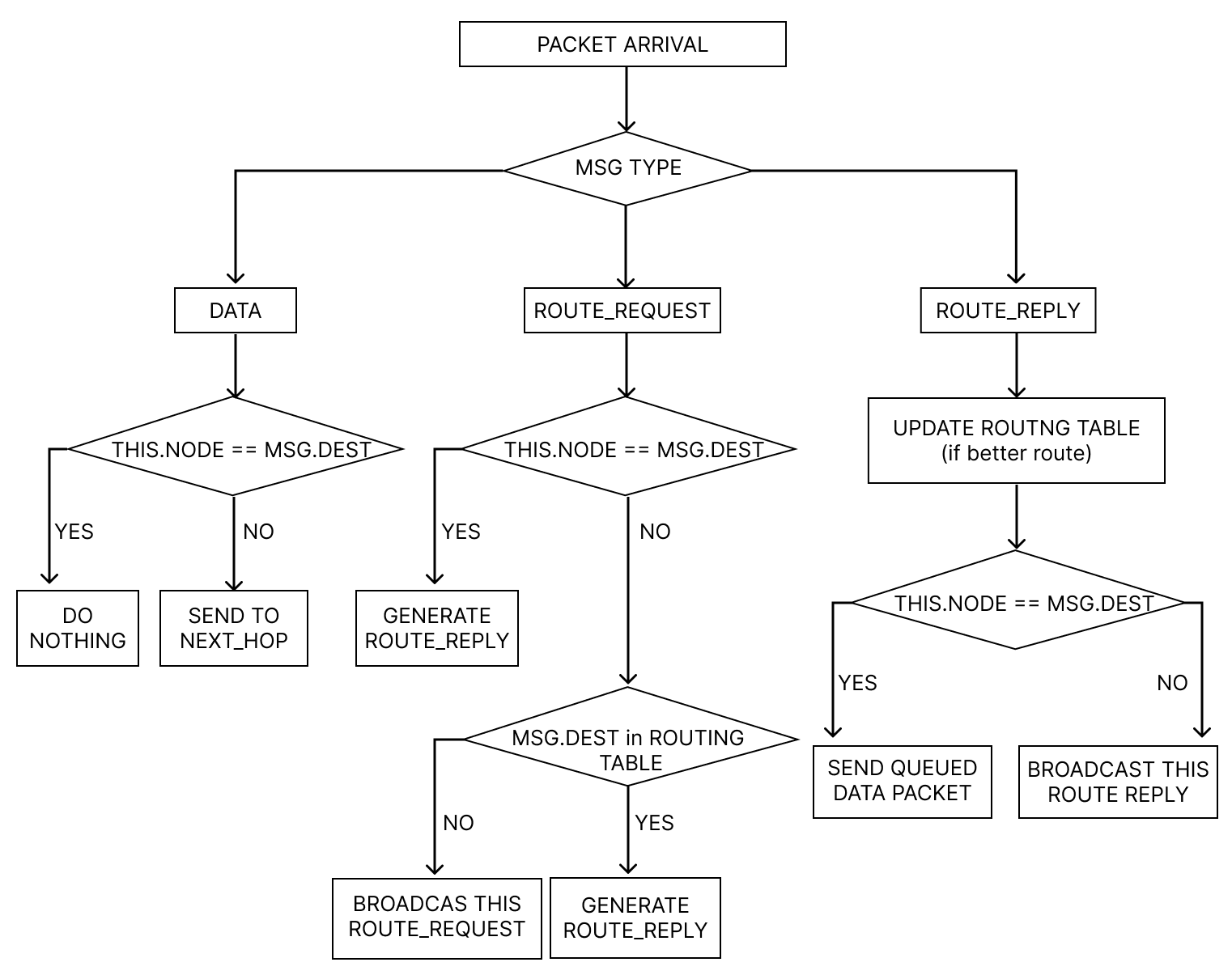
*Figure 3: Interface declaration of RadiouRoute.nc*

#### Functionalities definition and Flow Diagram

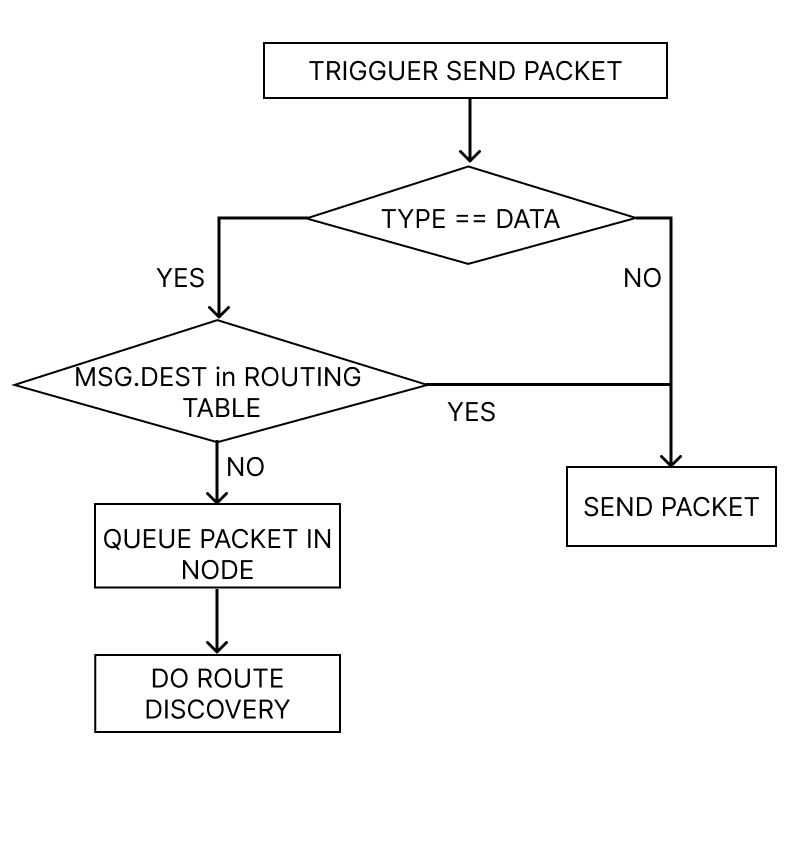
The main functionalities of the component are:

* **Receive packets:** decision making based on the receive message type, destination and source
* **Send packets:** decision making based on the received message type, destination and source and if the destination is or not in the routing table.

The decision making in of each main function is describe ahead:



*Figure 4: Receive packets flow diagram RadioRoute.nc*



*Figure 5: Send packets flow diagram RadioRoute.nc*

The base code for these functionalities can be found at **[2]**, commented and explained step by step. The joined with the triggered events, the functions and variables used in the component list ahead are the main pieces for the features described at Figures 4 and 5:

implementation {

  message\_t packet;

  // 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

  message\_t waiting\_packet; // Packet stored while route discovery is executed

  /\*\*\*\*\*  CONSTANTS  \*\*\*\*\*/

  uint16\_t NODES\_COUNT = 7;

  uint16\_t DATA = 0;

  uint16\_t ROUTE\_REQ = 1;

  uint16\_t ROUTE\_REP = 2;

  /\*\*\*\*\*  ROUTING TABLE  \*\*\*\*\*/

  // the index of the arrays is the destination address

  uint16\_t rt\_next\_hop[7]={NULL,NULL,NULL,NULL,NULL,NULL,NULL};

  uint16\_t rt\_hot\_count[7]={NULL,NULL,NULL,NULL,NULL,NULL,NULL};

  uint16\_t route\_req\_dest\_node = 0;

  /\*\*\*\*\*  LEDs  \*\*\*\*\*/

  uint16\_t leader\_code[8] = {1,0,9,1,1,8,1,6};

  uint16\_t led\_counter;

  uint16\_t led\_0, led\_1, led\_2;

  /\*\*\*\*\*  ROUTER VARIABLES  \*\*\*\*\*/

  bool route\_req\_sent=FALSE;

  bool route\_rep\_sent=FALSE;

  bool data\_sent=FALSE;

  bool locked;

  void change\_leds();

  bool actual\_send (uint16\_t address, message\_t\* packet);

  bool generate\_send (uint16\_t address, message\_t\* packet, uint8\_t type);

  bool clear\_queue(int8\_t type);

## System Outputs

seria legal um print do output do codigo e comentar minimamente

## Bibliography

[1] Matteo, Cesana. (2023). IoT Challenge #3: TinyOS and TOSSIM [Published course material]. Politecnico di Milano.

[2] System Implementation at: <https://github.com/claualc/iot>

[3] TinyOS Documentation Wiki. (n.d.). Retrieved from<http://tinyos.stanford.edu/tinyos-wiki/index.php/TinyOS_Documentation_Wiki>