***IOT Final Project***

*Tace 2*

1. Introduction
2. TinyOs
   1. Topology.txt
   2. RadioRoute.h
   3. RadioRouteAppC.nc
   4. RadioRouteC.nc
   5. RunSimulationScript.py
   6. Cooja simulation
   7. Implementation choices
3. Node-Red
   1. Workflow
      1. \*functions description\*
   2. ThingSpeak attachment
4. ThingSpeak
   1. Channel creation and settings
   2. Working flow
5. Observations
   1. Run instructions

(secondo me gli possiamo mettere anche il link a thingspeak e fare un public channel, così giusto per completezza)

1. ***Introduction***

The aim of the project was to create a LoraWan like network in TinyOs, network nodes should periodically generate messages with random data and send them (through the network server) to Node-Red, whom have to extract the relevant data from them and send them via MQTT to a ThingSpeak channel. The ThingSpeak channel must finally plot the random values on 5 charts (one for each node of the network).

TinyOs network has also an ack confirmation message functionality: when the server node receives a message, it must send an ack to the sender node, if the sender node doesn’t receives the ack within a 1 second window, it sends again the message (LoraWan-like). In addiction, server node must discard duplicates.

To connect TinyOs and Node-Red it is necessary to build and run the project using cooja, and send messages via TCP to Node-Red.

1. ***TinyOs***

TinyOs is able to simulate a network and its functions, it works thanks to interconnections between some files. Those files are:

* 1. ***Topology.txt***

This file contains the network topology (shown in figure).

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamenteAs we can see the network is composed by 5 sensor nodes (nodes 1,2,3,4,5 in our topology file), whom have to send messages to the server node (node 8 in our topology file); passing through the two gateways (nodes 6 and 7 in our topology).

N.B. all connections are bidirectional.

* 1. ***RadioRoute.h***

This file specifies which are the fields of the messages that go through the network.

Fields are:

* Type: just specifies the type of the message, if it is a data message (1) or an ack message (2).
* Sender: this field specifies which is the original sender node (so between 1 and 5).
* Gateway: this field specifies which is the gateway from which the message arrives to the server node, useful for knowing where to send the ack back.
* Destination: specifies the final destination of the ack message (it’s between 1 and 5).
* Value: contains the data, it must be a random generated data.
* ID: contains the message id, the message id is simply a generated number that is incremented by 1 every time a new message is sent.
  1. ***RadioRouteAppC.nc***

This module creates the interfaces and wires them to the RadioRouteC.nc file.

We can notice 3 timers, 2 used to send messages and one (ACK\_timer) used to count how much time is passed from the last sending.

We also have a RandomC used to create the random value.

We got also SerialPrintfC and SerialStartC necessary to use printfs in cooja.

* 1. ***RadioRouteC.nc***
  2. ***RunSimulationScript.py***
  3. ***Cooja simulation***

To run the project we have to compile it, then we have to open it in the cooja runner simulator (Contiki). Before running it we should ensure that the output node (8) is able to send his data through TCP, by opening a port on that server node.

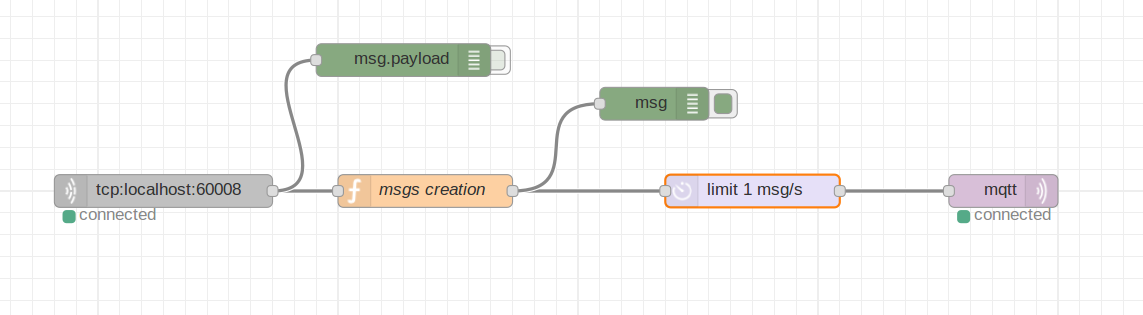
Then we can start the simulation and see all the packets that flows through the network.

* 1. ***Implementation choices***

Qui ci va la roba dei nodi 2 e 4

Diciamo che è sempre possibile buildare con tossim e decommentare tutti i dbg

1. ***Node-Red***

The main functionalities of Node-Red (in our project) were to “parse” the received message obtained via TCP, create a new MQTT message and send it via MQTT.

It also limits the number of messages/s that ThingSpeak can receive.

* 1. ***Function block***

“msgs creation” take as input the TCP message, and generate a new message to be sent via MQTT. In order to do so, it read the TCP message and in particular it extracts data about the sender node and the value of the message, and using them (and thingspeak credentials in order to transmit them successfully) it generates the new MQTT message.

* 1. ***Delay block***

Used in order to limit the number o messages per second that ThingSpeak can receive.

* 1. ***MQTT block***

Set in order to send data to out ThingSpeak channel.

1. ***ThingSpeak***

ThingSpeak is an online portal used to connect some kind of peripherals and read data. In our case we connected via MQTT and read data from Node-Red.

* 1. ***Channel creation and settings***

We created a channel with 5 fields (charts), in which each one contains the temporized value sent by each sensor node, so in the end we can see a diagram.

* 1. ***Working flow***

Immagine che contiene testo, software, Software multimediale, Icona del computer

Descrizione generata automaticamenteThingSpeak listens on an MQTT channel (non son sicuro sia giusto), the incoming messages can have up to 5 fields (one for each sensor node), ThingSpeak reads the messages and assigns the new values of each field to a corresponding new value in the corresponding chart.

1. ***Observations***