UNIVERSITÀ DI PISA

Computer Engineering

Advanced Network Architectures and Wireless Systems

IOT PROJECT

STUDENTS:
Alexander De Roberto
Antonio Di Tecco
Simone Pampaloni

Academic Year 2019/2020

Contents

1 | Overview

The aim of this project is the creation of a Client-Proxy-Server architecture that uses the CoAP application level protocol, with the function of server delegated to a multi-hop 6loWPAN Wireless Sensor Network (WSN).

2 | Architecture

The general architecture is the following.

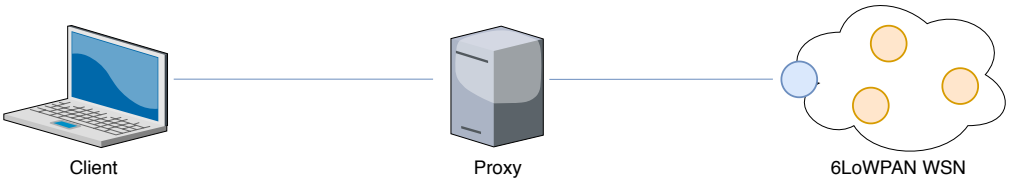

Figure 2.1: General Architecture Schema

The client is connected to a proxy, having the cache function, which requests and stores data from the 6LoWPAN sensor network. The 6LoWPAN will be described in more detail below, declaring and motivating the implementation choices.

2.1 WSN 6LowPAN RPL

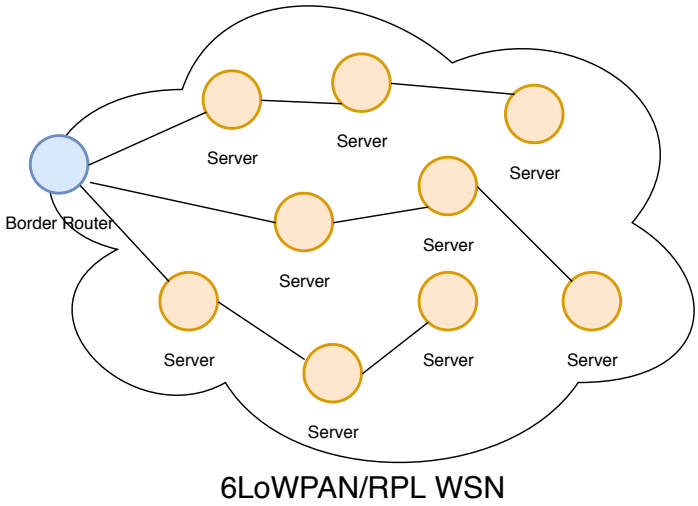

Figure 2.2: General Schema 6LoWPAN

WSN is a multi-hop 6loWPAN-based network that uses the RPL routing protocol. To run RPL on the WSN you need a border router that takes care

of building the DODAG and that allows the addressing of the network nodes outside the WSN. As per specifications, the network supports a number of nodes around 30 reachable by the border router with a maximum of 3-4 hops.
The nodes used are the Zolertia Z1. These types of devices impose constraints due to limited hardware capabilities, such as that relating to usable memory. To allow the correct functioning of the network, avoiding the overflow of the memory of the nodes, and at the same time a rapid formation of the DODAG of RPL, have been configured in the file *project-conf.h* the following parameters:

```
#ifndef PROJECT_CONF_H_
#define PROJECT_CONF_H_

#undef NETSTACK_CONF_RDC
#define NETSTACK_CONF_RDC   nullrdc_driver

// Configuro Trickle /*******************/

#undef RPL_CONF_DIO_REDUNDANCY
#define RPL_CONF_DIO_REDUNDANCY   2

#undef RPL_CONF_DIO_INTERVAL_MIN
#define RPL_CONF_DIO_INTERVAL_MIN   4

#undef RPL_CONF_DIO_INTERVAL_DOUBLINGS
#define RPL_CONF_DIO_INTERVAL_DOUBLINGS 14
/*************************************/

// TABLES /*****************************/

#undef NBR_TABLE_CONF_MAX_NEIGHBORS
#define NBR_TABLE_CONF_MAX_NEIGHBORS    10

#undef UIP_CONF_MAX_ROUTES
#define UIP_CONF_MAX_ROUTES       20
/*************************************/

#undef UIP_CONF_TCP
#define UIP_CONF_TCP  0 // Disable TCP

#endif
```

Listing 2.1: `project-conf.h server coap`

```
#ifndef PROJECT_ROUTER_CONF_H_
#define PROJECT_ROUTER_CONF_H_

#undef NETSTACK_CONF_RDC
#define NETSTACK_CONF_RDC     nullrdc_driver

// F0 /*********************************/
#undef RPL_OF_OCP
#define RPL_OF_OCP  RPL_OCP_OF0
```

```
/****************************************/

// Configuro Trickle /*******************/

#undef RPL_CONF_DIO_REDUNDANCY
#define RPL_CONF_DIO_REDUNDANCY   2

#undef RPL_CONF_DIO_INTERVAL_MIN
#define RPL_CONF_DIO_INTERVAL_MIN   4

#undef RPL_CONF_DIO_INTERVAL_DOUBLINGS
#define RPL_CONF_DIO_INTERVAL_DOUBLINGS 14
/****************************************/

// TABLES /*****************************/

#undef NBR_TABLE_CONF_MAX_NEIGHBORS
#define NBR_TABLE_CONF_MAX_NEIGHBORS   15

#undef UIP_CONF_MAX_ROUTES
#define UIP_CONF_MAX_ROUTES            30
/****************************************/

#ifndef UIP_FALLBACK_INTERFACE
#define UIP_FALLBACK_INTERFACE rpl_interface
#endif

#ifndef UIP_CONF_RECEIVE_WINDOW
#define UIP_CONF_RECEIVE_WINDOW  60
#endif

#ifndef WEBSERVER_CONF_CFS_CONNS
#define WEBSERVER_CONF_CFS_CONNS 2
#endif

#endif /* PROJECT_ROUTER_CONF_H_ */
```

Listing 2.2: `project-conf.h border router`

UIP_CONF_MAX_ROUTES and NBR_TABLE_CONF_MAX_NEIGHBORS have been set up to ensure good performance of the 6LoWPAN. Note that different values have been defined for these two parameters between the nodes having the function of server coap and the border router, for the differences of functions within the network.

2.1.1 Objective Function 0 and Trickle Parameters

An Objective Function (OF) defines the metric and limits for nodes to select and optimize routes within an RPL instance [3]. The definition of a OF separate from the protocol allows RPL to use different optimization criteria in relation to the application and the WSN. RPL OF uses to determine the rank of a node. This value represents the distance from the root of

the DODAG. The rank is exchanged between the nodes and is used to set routes and avoid loops. The Objective Function 0 (OF0) is a particular OF that operates on the parameters that are obtained from the configuration of the DODAG parameters and the DIO messages.
To set the OF0 the following command has been inserted in the file *project-conf.h*:

```
#undef RPL_OF_OCP
#define RPL_OF_OCP  RPL_OCP_OF0
```

In addition, we went to edit the **parameters of Trickle**.
The Trickle algorithm is the algorithm that regulates the transmission by nodes of the DIO messages necessary for the formation of the DODAG.
Our goal was to **speed up and optimize DODAG training**. It must be considered that the network creation performance also depends on the network topology. A more or less dense network can make performance vary considerably according to the parameters of Trickle. For our application we hypothesized that the nodes were positioned according to a uniform distribution around the border router, as shown in the figure below.

Figure 2.3: 6LoWPAN Topology

The Trickle parameters have been set as follows:

- RPL_CONF_DIO_REDUNDANCY (10 by default): the Trickle K redundancy constant, used to carry out the Broadcast Suppression if more than K messages have been received in the time period I. **Set to 2**.

- RPL_CONF_DIO_INTERVAL_MIN: used to define the minimum initial period of Trickle Imin = $2^{\text{RPL\_CONF\_DIO\_INTERVAL\_MIN}}$ (2^{12} by default). **Set to 4** (16 ms).

- RPL_CONF_DIO_INTERVAL_DOUBLINGS: used to define the maximum Trickle period.
  Imax = $2^{(\text{DIO\_INTERVAL\_MIN} + \text{DIO\_INTERVAL\_DOUBLINGS})}$ (2^{20} by default). **Set at 14** (4.37 minutes).

The setting was obtained experimentally by trying various configurations and based on the analysis of the performances shown during the lessons of the course. The values of K = 2 and Imin = 4 were chosen in order to improve the speed of formation of the DODAG.

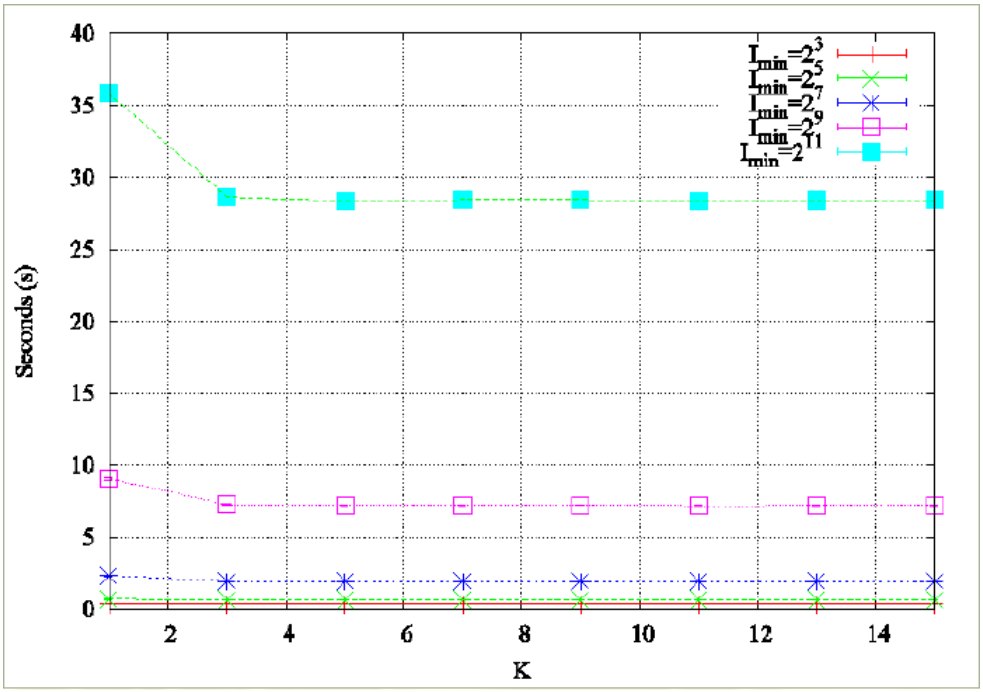

Figure 2.4: Seconds needed to the first DODAG formation by varying K[4]

3 | Application

The general schema of the application is the following:

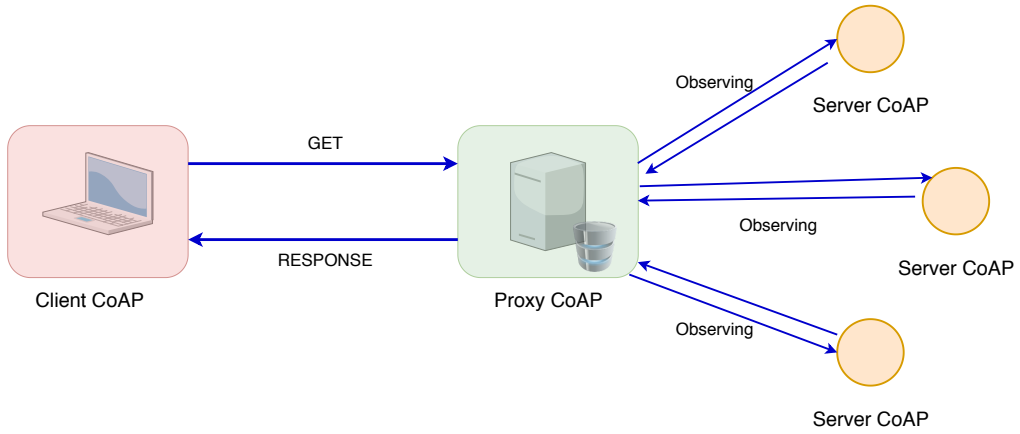

Figure 3.1: General Application Schema

From an application point of view, the 6LoWPAN network can be represented as a series of CoAP servers that each manage a resource. On power-up, the proxy starts observing every resource on all the servers and stores the result locally. Hence, the proxy performs a cache function for the client. In fact, when the CoAP client makes a request (GET) to the proxy by requesting a resource from a CoAP server, the proxy first processes the request, then replies (Response) to the client with the value of the requested resource.
The application level of the client and the proxy server was made in Java, while for the CoAP servers they were made using the C language.

3.1 Data Encoding

The stored data and the exchanged messages are encoded in JSON, using the SenML representation format [5].

```
[{"n":"value", //n rappresenta il nome della risorsa
  "v": 30    //v rappresenta un intero
}]

[{"n":"error",  //risorsa trasmessa in caso di errore
```

```
"sv":"Supporting content-types application/json"  //sv
    ↪ rappresenta una stringa
}]
```

Listing 3.1: SenML Example

3.2 Client CoAP

The CoAP client provides a command line interface that allows the proxy of a specific CoAP server to be requested from the proxy, identifiable with a unique ID. The power-up interface shows the following text:

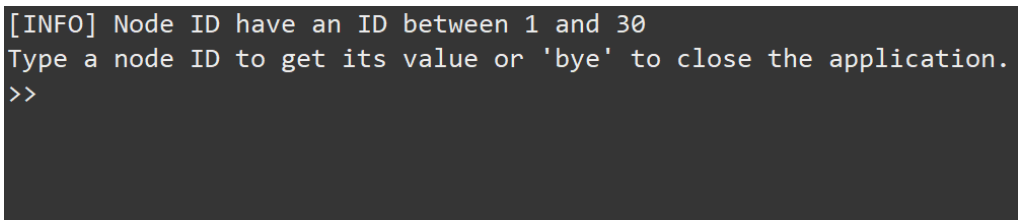

Figure 3.2: Client Interface

The user can then choose to type:

- server ID: of which it wants to know the last value of the resource stored in the proxy. The proxy will return the value of the resource to the client.
- 'bye': end client execution.

To run the client from the terminal, type:

```
java -jar client.jar [number_of_nodes]
```

3.3 Proxy CoAP

The CoAP proxy is the intermediate entity between the CoAP client and the CoAP servers of the 6LoWPAN network, therefore it acts as a server for the client (as it responds to GET requests) and as a client for the servers (as it "observes" on their resources).

The CoAP Proxy is divided into 4 modules:

- Proxy: this is the main module that initializes and starts the Server module to manage requests from the user client and initializes and starts as many instances of the Client module as there are CoAP Servers. It also allocates an array that performs the cache function where resources will be stored.

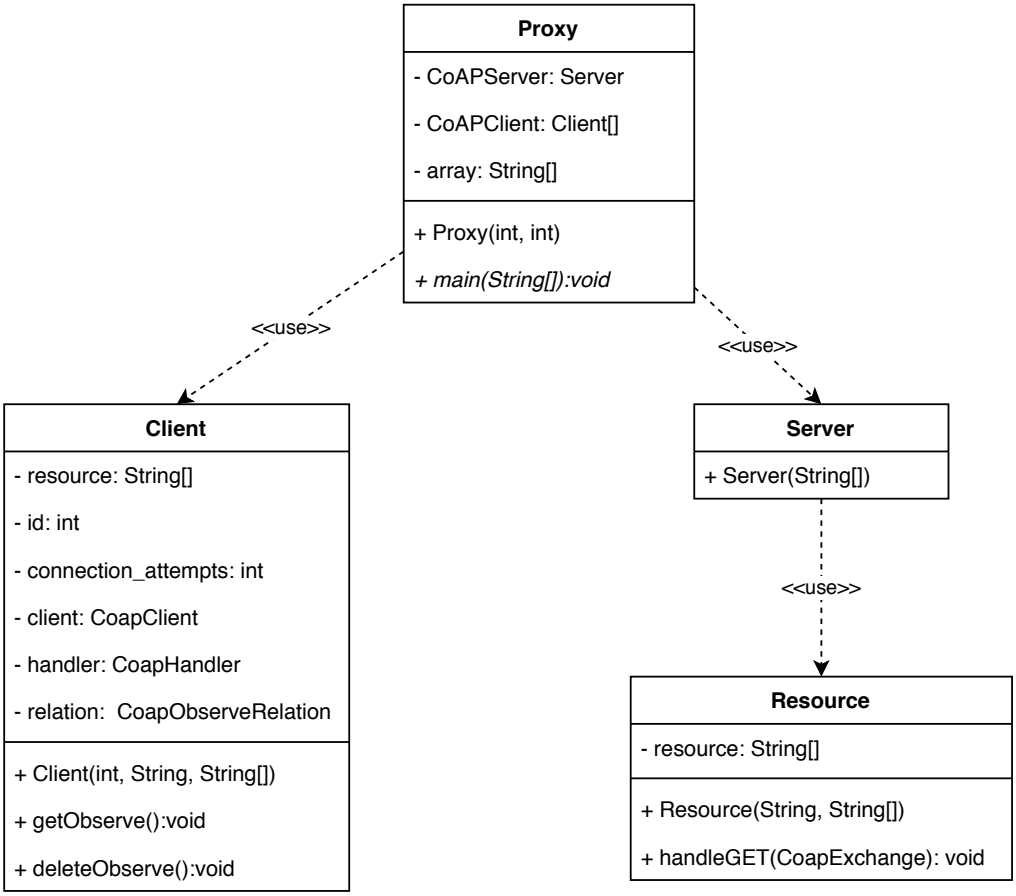

Figure 3.3: Class Diagram of Proxy CoAP

- Server: is the module that represents the server side of the proxy. On startup it initializes an instance of the Resource module.
- Resource: is the module that represents the resource that the CoAP client draws on to retrieve the values of the server resources. It then manages GET requests by returning a value contained in the array provided by the Proxy main module, using the ID passed by the client as an index or possibly returning an error.
- Client: is the module with which the proxy periodically "observes" the resources of the 6LowPAN servers. Resources are stored in the array provided by the main Proxy module. If the observer fails, for example because the CoAP server is unreachable, the client retries the connection for a maximum of 3 times. If 3 consecutive failures occur, the client disconnects.

To run the terminal proxy type:

```
1  java -jar proxy.jar [type_mote(z1/cooja)] [
   ↪ number_of_nodes]
```

type _mote identifies the type of mote that uses the 6LoWPAN (z1 or cooja), number _of _nodes indicates the number of nodes (server coaps) present in the 6LoWPAN.

3.4 Server CoAP

Each CoAP server in the 6LowPAN network manages a resource called "value" described by a random integer. CoAP servers respond to GET requests by returning the resource value. Once the Proxy starts observing a resource, it is sent periodically (every 30 seconds). Furthermore, since the application is performed on devices equipped with buttons (such as the Z1), pressing the button triggers the sending of the resource directly to the Proxy.

4 | Bibliography

1. RFC 6206, The Trickle Algorithm, March 2011
2. RFC 6550, RPL: IPv6 Routing Protocol for Low Power and Lossy Networks, March 2012
3. RFC 6552, Objective Function Zero for the Routing Protocol for Low Power and Lossy Networks (RPL), March 2012
4. C. Vallati, E. Mingozzi, Trickle F: fair broadcast suppression to improve energy efficient route formation with the RPL routing protocol, Proceedings of the 3rd IFIP Conference on Sustainable Internet and ICT for Sustainability (SustainIT 2013), Palermo, Italy, October 30-31, 2013
5. RFC 8428, Sensor Measurement Lists (SenML), August 2018