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 **2**

2 Architecture **3**
2.1 WSN 6LowPAN RPL 3
2.1.1 Objective Function 0 and Trickle Parameters 5

3 Application **8**
3.1 Data Encoding 8
3.2 Client CoAP 9
3.3 Proxy CoAP 9
3.4 Server CoAP 11

4 Bibliography **12**

1 | Overview

Lo scopo di questo progetto è la realizzazione di un'architettura Client-Proxy-Server che utilizza il protocollo di livello applicativo CoAP, con la funzione di server delegata a una Wireless Sensor Network (WSN) 6loWPAN multi-hop.

2 | Architecture

L'architettura generale è la seguente.

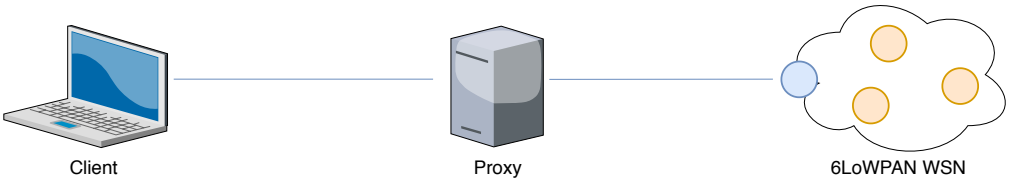

Figure 2.1: Schema generale architettura

Il client è connesso a un proxy, avente la funzione di cache, che richiede e immagazzina i dati dalla rete di sensori 6LoWPAN. Di seguito verrà descritta in modo più approfondito la 6LoWPAN, dichiarando e motivando le scelte implementative.

2.1 WSN 6LowPAN RPL

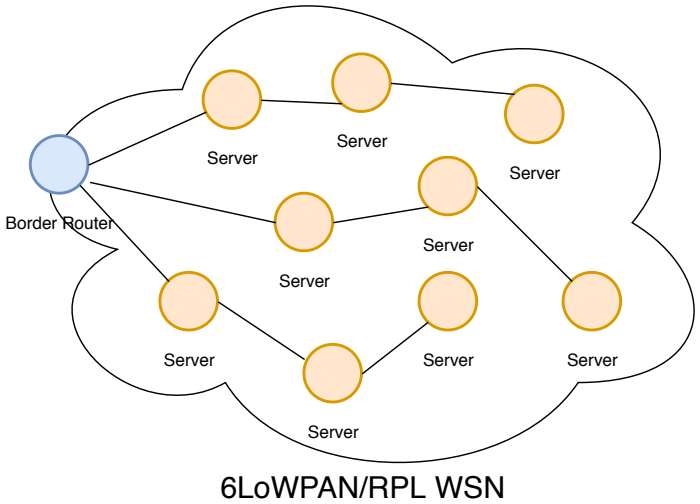

Figure 2.2: Schema generale 6LoWPAN

La WSN è una rete basata su 6loWPAN multi-hop che utilizza il protocollo di routing RPL. Per eseguire RPL sulla WSN è necessario un border

router che si occupa di costruire il DODAG e che permette l'indirizzamento dei nodi della rete al di fuori della WSN. Come da specifiche la rete supporta un numero di nodi intorno ai 30 raggiungibili dal border router con un massimo di 3-4 hop. I nodi utilizzati sono gli Zolertia Z1. Questo genere di dispositivi impongono dei vincoli dovute alle limitate capacità hardware, come quella relativa alla memoria utilizzabile. Per permettere il corretto funzionamento della rete, evitando l'overflow della memoria dei nodi, e allo stesso tempo una rapida formazione del DODAG di RPL, sono stati configurati nel file *project-conf.h* i seguenti parametri:

```
#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
/*************************************/
```

```
11
12 // Configuro Trickle /*******************/
13
14 #undef RPL_CONF_DIO_REDUNDANCY
15 #define RPL_CONF_DIO_REDUNDANCY   2
16
17 #undef RPL_CONF_DIO_INTERVAL_MIN
18 #define RPL_CONF_DIO_INTERVAL_MIN   4
19
20 #undef RPL_CONF_DIO_INTERVAL_DOUBLINGS
21 #define RPL_CONF_DIO_INTERVAL_DOUBLINGS 14
22 /*************************************/
23
24 // TABLES /*****************************/
25
26 #undef NBR_TABLE_CONF_MAX_NEIGHBORS
27 #define NBR_TABLE_CONF_MAX_NEIGHBORS   15
28
29 #undef UIP_CONF_MAX_ROUTES
30 #define UIP_CONF_MAX_ROUTES            30
31 /*************************************/
32
33 #ifndef UIP_FALLBACK_INTERFACE
34 #define UIP_FALLBACK_INTERFACE rpl_interface
35 #endif
36
37 #ifndef UIP_CONF_RECEIVE_WINDOW
38 #define UIP_CONF_RECEIVE_WINDOW  60
39 #endif
40
41 #ifndef WEBSERVER_CONF_CFS_CONNS
42 #define WEBSERVER_CONF_CFS_CONNS 2
43 #endif
44
45 #endif /* PROJECT_ROUTER_CONF_H_ */
```

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

UIP_CONF_MAX_ROUTES e NBR_TABLE_CONF_MAX_NEIGHBORS sono stati settati per garantire buone performance della 6LoWPAN. Si noti che si è definito valori diversi per questi due parametri tra i nodi aventi funzione di server coap e il border router, per le differenze di funzioni all'interno nella rete.

2.1.1 Objective Function 0 and Trickle Parameters

Un'Objective Function (OF) definisce la metrica e i limiti affinché i nodi selezionino e ottimizzino le rotte all'interno di un'istanza RPL [3]. La definizione di una OF separata dal protocollo permette a RPL di utilizzare differenti criteri di ottimizzazione in relazione all'applicazione e alla WSN. RPL usa OF per determinare il rank di un nodo. Questo valore rappresenta la distanza dalla radice del DODAG. Il rank è scambiato tra i nodi e serve

per settare le rotte ed evitare i loop. L' Objective Function 0 (OF0) è una particolare OF che opera sui parametri che sono ottenuti dalla configurazione dei parametri di DODAG e dei messaggi DIO.
Per settare la OF0 è stato inserito nel file *project-conf.h* il seguente comando:

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

Inoltre, si è andati a modificare i **parametri di Trickle**.
L'algoritmo di Trickle è l'algoritmo che regola l'invio in broadcast da parte dei nodi dei messaggi DIO, necessari per la formazione del DODAG.
Il nostro obiettivo è stato quello di **velocizzare e ottimizzare la formazione del DODAG**. Si deve considerare che le performance di creazione della rete dipendono anche dalla topologia della rete. Una rete più o meno densa può far variare in modo considerevole le performance in funzione dei parametri di Trickle. Per la nostra applicazione abbiamo ipotizzato che i nodi fossero posizionati secondo una distrubuzione uniforme intorno al border router, come visibile nella figura di seguito.

Figure 2.3: Topologia 6LoWPAN

I parametri di Trickle sono stati settati come di seguito:

- RPL_CONF_DIO_REDUNDANCY (10 di default): la costante di ridondanza Trickle K, utilizzata per effettuare la Broadcast Soppression se nel periodo di tempo I si è ricevuto più di K messaggi. **Settato a 2**.
- RPL_CONF_DIO_INTERVAL_MIN: si utilizza per definire il periodo minimo iniziale di Trickle Imin = $2^{\text{RPL\_CONF\_DIO\_INTERVAL\_MIN}}$ (2^{12} di default). **Settato a 4** (16 ms).

- RPL_CONF_DIO_INTERVAL_DOUBLINGS: si utilizza per definire il periodo massimo di Trickle
  $\text{Imax} = 2^{(\text{DIO\_INTERVAL\_MIN} + \text{DIO\_INTERVAL\_DOUBLINGS})}$ (2^{20} di default).
  Settato a 14 (4,37 minuti).

Il settaggio è stato ottenuto per via sperimentale provando varie configurazioni e basandosi sulle analisi delle performance mostrate durante le lezioni del corso. I valori di K = 2 e di Imin = 4 sono stati scelti al fine di migliorare la velocità di formazione del DODAG.

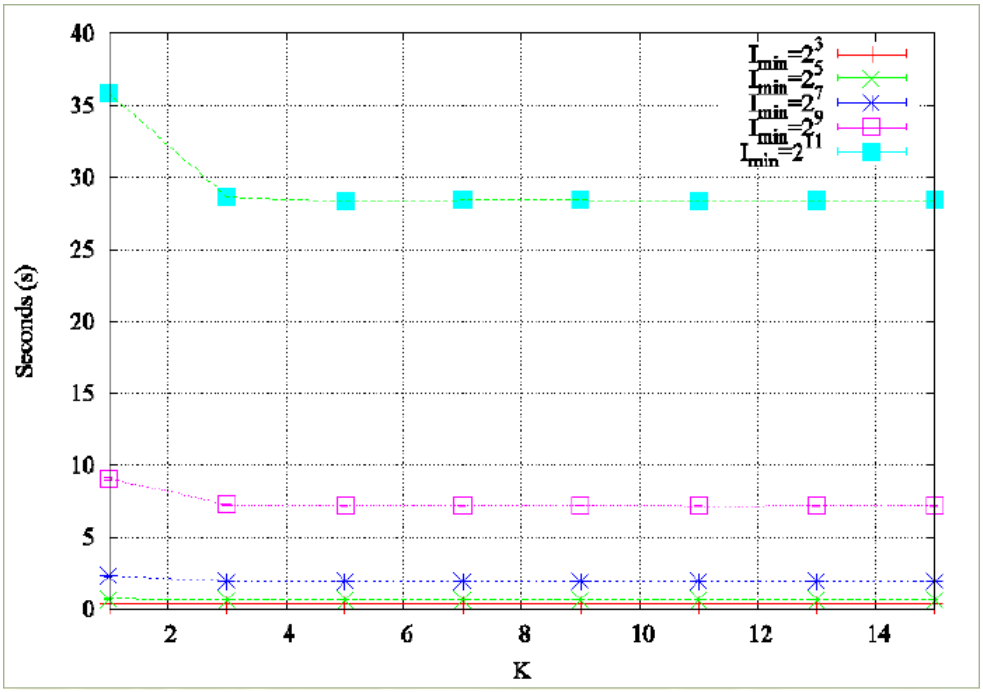

Figure 2.4: Secondi necessari alla formazione del primo DODAG al variare di K [4]

3 | Application

Lo schema generale dell'applicazione è il seguente:

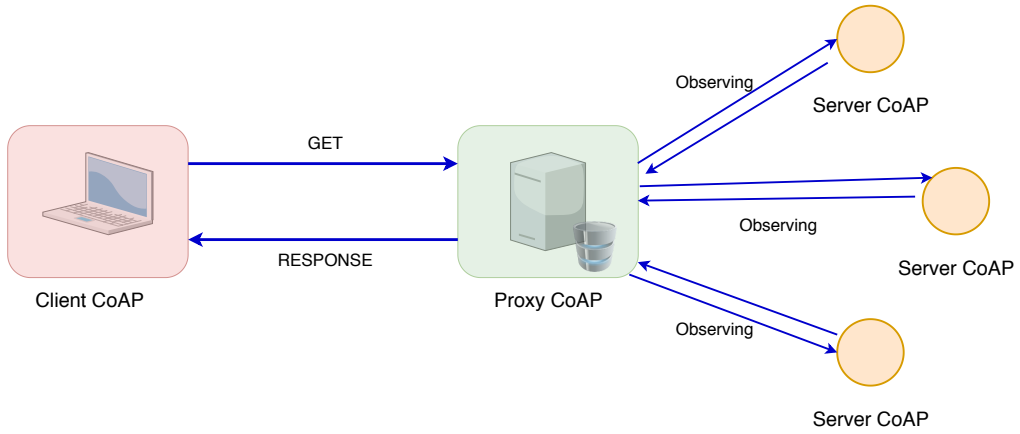

Figure 3.1: Schema generale dell'applicazione

Dal punto di vista applicativo la rete 6LoWPAN è rappresentabile come una serie di server CoAP che gestiscono ognuno una risorsa. Il proxy, all'accensione, inizia a fare "observing" su ogni risorsa di tutti i server e memorizza il risultato localmente. Quindi, il proxy svolge una funzione di cache per il client. Infatti, quando il client CoAP fa una richiesta (GET) al proxy richiedendo una risorsa da un server CoAP, il proxy prima elabora la richiesta, poi risponde (Response) al client con il valore della risorsa richiesta.
Il livello applicativo del client e del proxy server è stato realizzato in liguaggio Java, mentre per i server CoAP sono stati realizzati utilizzando il linguaggio C.

3.1 Data Encoding

I dati immagazzinati e i messaggi scambiati sono codificati in JSON, utilizzando il formato di rappresentazione SenML [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: Esempio SenML

3.2 Client CoAP

Il client CoAP fornisce una interfaccia a linea di comando che permette di richiedere al proxy la risorsa di uno specifico server CoAP, identificabile con un ID univoco. L'interfaccia all'accensione mostra il seguente testo:

```
[INFO] Node ID have an ID between 1 and 30
Type a node ID to get its value or 'bye' to close the application.
>>
```

Figure 3.2: Interfaccia Client

L'utente quindi può scegliere di digitare:

- ID di un server: di cui vuole conoscere l'ultimo valore della risorsa memorizzata nel proxy. Il proxy ritornerà al client il valore della risorsa.
- 'bye': terminare l'esecuzione del client.

Per eseguire il client da terminale digitare:

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

3.3 Proxy CoAP

Il proxy CoAP è l'entità intermedia fra il client CoAP e i server CoAP della rete 6LoWPAN, quindi si comporta da server per il client (in quanto risponde alle richieste GET) e da client per i server (in quanto fa "observing" sulle loro risorse).

Il Proxy CoAP è suddiviso in 4 moduli:

- Proxy: è il modulo principale che inizializza e avvia il modulo Server per gestire le richieste dal client utente e inizializza e avvia tante istanze del modulo Client quanti sono i Server CoAP. Inoltre alloca un array che svolge la funzione di cache dove verranno memorizzate le risorse.

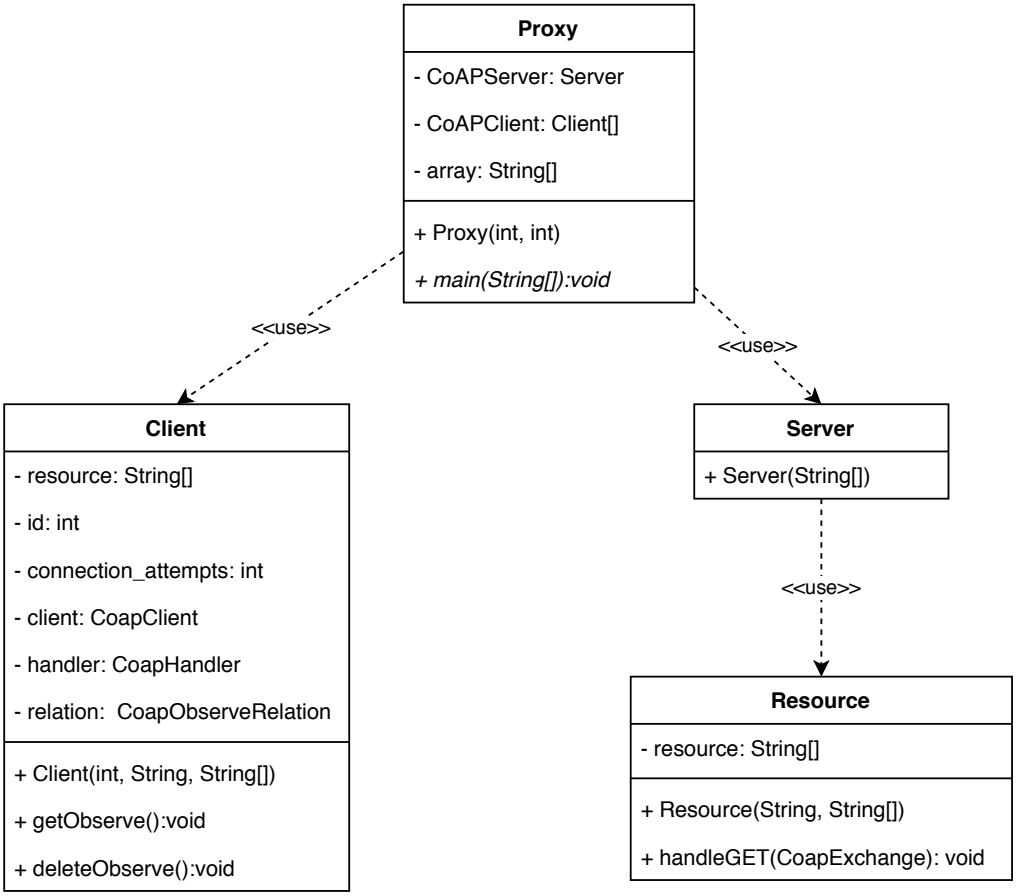

Figure 3.3: Diagramma delle classi del Proxy CoAP

- Server: è il modulo che rappresenta il lato server del proxy. All'avvio inizializza un'istanza del modulo Resource.
- Resource: è il modulo che rappresenta la risorsa al quale il client CoAP attinge per recuperare i valori delle risorse dei server. Gestisce quindi le richieste GET restituendo un valore contenuto nell'array fornito dal modulo principale Proxy, utilizzando come indice l'ID passato dal client o eventualmente restituendo un errore.
- Client: è il modulo con cui il proxy fa "observing" periodico sulle risorse dei server della 6LowPAN. Le risorse vengono immagazzinate nell'array fornito dal modulo principale Proxy. Se l'observing fallisce, per esempio perché il server CoAP non è raggiungibile, il client ritenta la connessione per un massimo di 3 volte. Nel caso avvengano 3 fallimenti consecutivi, il client si disconnette.

Per eseguire il proxy da terminale digitare:

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

type_mote identifica il tipo di mote che utilizza la 6LoWPAN (z1 o cooja), number_of_nodes indica il numero di nodi (server coap) presenti nella 6LoWPAN.

3.4 Server CoAP

Ogni server CoAP della rete 6LowPAN gestisce una risorsa denominata "value" descritta da un intero casuale. I server CoAP rispondo alle richieste GET restituendo il valore della risorsa. Una volta che il Proxy inizia ad osservare una risorsa, questa viene inviata periodicamente (ogni 30 secondi). Inoltre, essendo l'applicazione eseguita su dispositivi dotati di bottoni (come gli Z1), la pressione del bottone scatena l'invio della risorsa direttamente al 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