Corso di Programmazione di Reti
Relazione di Progetto

Davide Di Marco

Giugno 2021

Indice

Capitolo 1

Analisi

1.1 Requisiti

Per il mio progetto d'esame ho scelto la traccia 1. Riporto in seguito il testo della traccia fornita da implementare.
"Si immagini di avere uno scenario di Smart Meter IoT che rilevano la temperatura e umidità del terreno in cui sono posizionati. I 4 dispositivi sono indicati nella figura a fianco come DEVICE. Questi dispositivi si collegano 1 volta al giorno con una connessione UDP verso il Gateway. Tramite questa connessione i dispositivi inviano le misure che hanno raccolto durante le 24 ore precedenti. Le misure consistono di un file che contiene l'ora della misura e il dato di temperatura e umidità. Una volta che i pacchetti di tutti i dispositivi sono arrivati al Gateway, il gateway instaura una connessione TCP verso un server centrale dove i valori vengono visualizzati sulla console del server nel seguente modo: Ip_address_device_1 – ora misura – valore temperatura – valore umidità I 4 Dispositivi IoT hanno un indirizzamento appartenente ad una rete di Classe C del tipo 192.168.1.0/24 Il Gateway ha due interfacce di rete: quella verso i dispositivi il cui IP Address appartiene alla stessa network dei dispositivi mentre l'interfaccia che parla con il server ha indirizzo ip appartenente alla classe 10.10.10.0/24, classe a cui appartiene anche l'IP address del server centrale. Si realizzi un emulatore Python che sfruttando il concetto dei socket visti in laboratorio consenta di simulare, utilizzando l'interfaccia di loopback del proprio PC, il comportamento di questo sistema. Si devono simulare le conessioni UDP dei device verso il Gateway e la connessione TCP del Gateway verso il Server mostrando sulla Console del server la lista dei messaggi ricevuti nel formato indicato sopra. Inoltre indicare la dimensione dei buffer utilizzati su ciascun canale trasmissivo, il tempo impiegato per trasmettere il pacchetto UDP ed il tempo impiegato per trasmettere il pacchetto TCP."

1.2 Analisi e modello del dominio

La traccia specificata nella sezione precedente illustra bene i requisiti funzionali principali dell'applicazione. L'obiettivo principale dell'applicazione è lo storage in cloud(server) dei 4 messaggi inviati dai 4 diversi client. L'implementazione consiste in 4 client che si connettono, attraverso il protocollo UDP, a un gateway e ognuno di essi invia un messaggio espresso nel seguente formato:
Ip_address_device_1 – ora misura – valore temperatura – valore umidità

Il gateway una volta ricevuti tutti e 4 i messaggi stabilisce una connessione verso il server attraverso il protocollo TCP e invia i dati ricevuti dai client in modo tale da poterli memorizzare sul server.
Il gateway quindi possiede 2 interfaccie di rete: una client-gateway di tipo UDP e l'altra gateway-server di tipo TCP.
Per quanto riguarda l'indirizzamento degli IP si hanno:

- 4 client che appartengono alla rete di classe C di tipo 192.168.1.0/24
- 1 gateway la quale rete in ingresso fa parte della stessa dei client (con ip appartente a: 192.168.1.0/24) e la rete in uscita con ip appartente al tipo 10.10.10.0/24
- 1 server avente un indirizzo IP di tipo 10.10.10.0/24

Infine interessa conoscere i tempi di trasmissione dei vari pacchetti sia da client verso gateway sia da gateway verso il server.

1.2.1 Schema del dominio

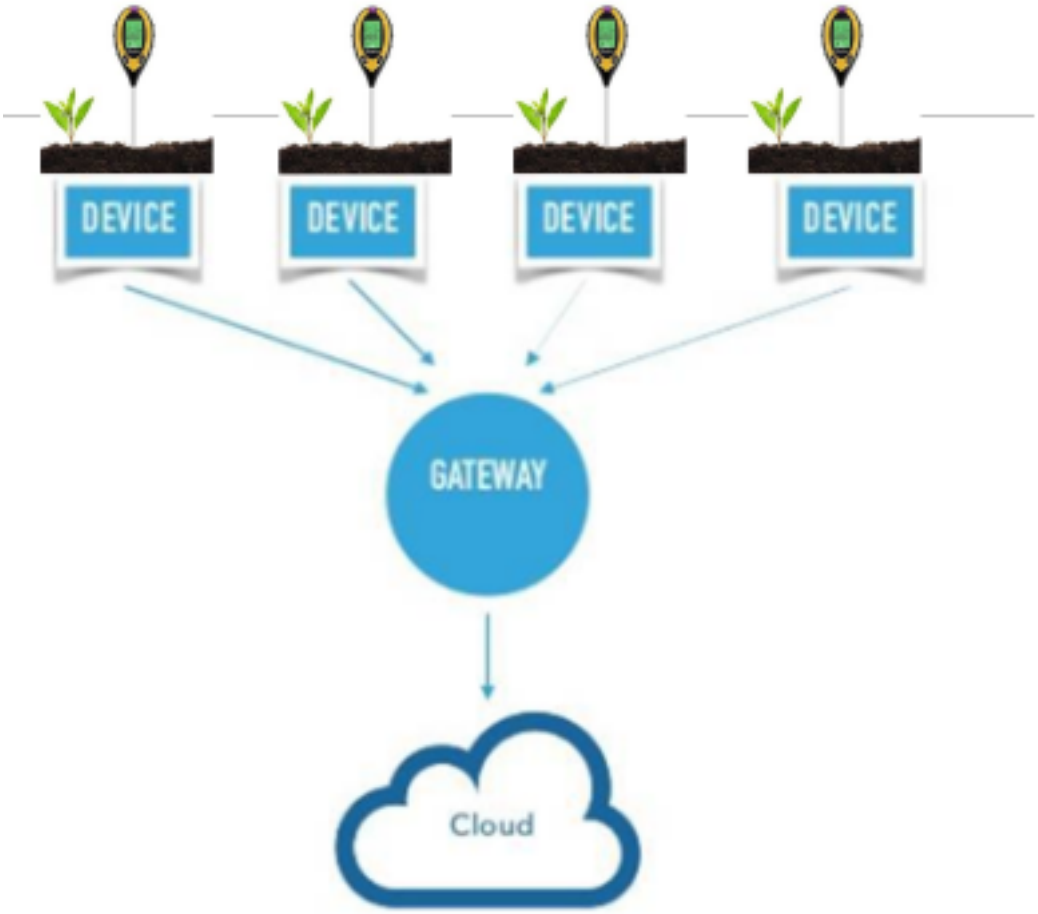

Figura 1.0: Figura rappresentante lo schema dello scenario da implementare

Capitolo 2

Sviluppo e funzionamento

L'applicazione è stata pensata per essere utilizzata in diversi scenari a seconda del numero di client che si devono connettere e tramite un gateway che può essere scelto a seconda delle preferenza. Il server su cui immagazzinare i dati invece rimane sempre lo stesso. Allo scopo sono state create 3 classi differenti: Gateway, Server, Client.

2.1 Classi

2.1.1 Classe Server

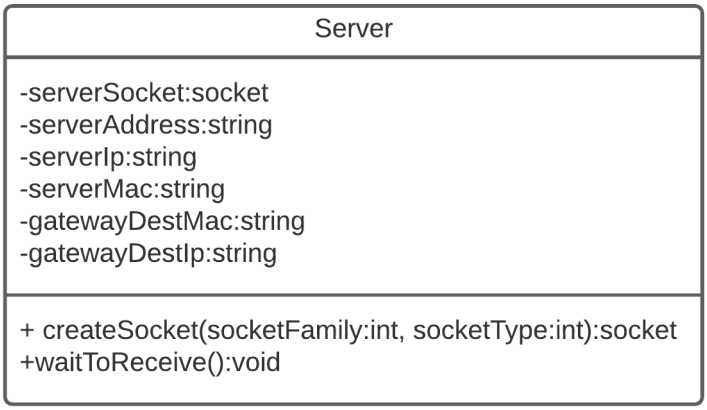

Figura 1.1: UML della classe Server

La classe Server è quella che contiene l'implementazione del Server. Essa è adibita alla ricezione del messaggio da parte del gateway. All'avvio dell'applicazione, il server si mette subito in ascolto sulla porta 8080 per la ricezione dei messaggi da parte del gateway. Quando un messaggio viene ricevuto, questo viene stampato a

video e salvato all'interno del server, infine viene inviato un messaggio di conferma al gateway dell'avvenuta ricezione del messaggio da parte del server.
Il protocollo utilizzato per la ricezione e l'invio dei messaggi è di tipo TCP. I suoi indirizzi sono:

- indirizzo IP: 10.10.10.1
- indirizzo MAC: 49:9D:D0:93:EB:B2

```
Web SERVER online sulla porta:  8080
Sono in attesa di ricevere il messaggio...
<socket.socket fd=74, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM,
proto=0, laddr=('127.0.0.1', 8080), raddr=('127.0.0.1', 35474)> ('127.0.0.1', 35474)

Ho ricevuto il messaggio:
 192.168.1.10 - 08.00 - 09°C - 27%
192.168.1.14 - 12.00 - 09°C - 65%
192.168.1.15 - 16.00 - 22°C - 55%
192.168.1.16 - 20.00 - 40°C - 40%
Tempo impiegato: 0.001042 ms
Sono in attesa di ricevere il messaggio...
```

Figura 1.2: Console del server alla ricezione del messaggio

2.1.2 Classe Gateway

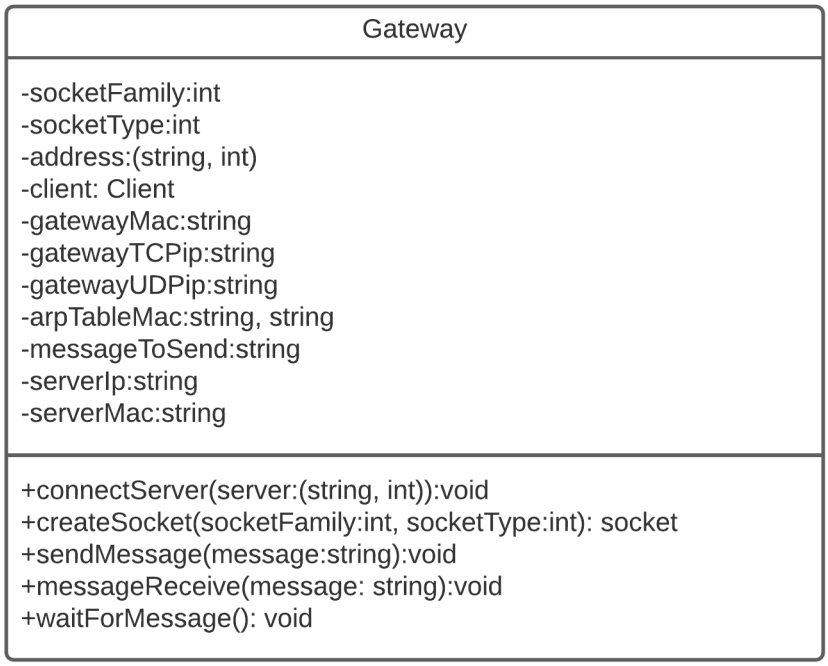

Figura 1.3: UML della classe Gateway

La classe Gateway è quella che contiene l'implementazione del gateway. Essa è adibita alla ricezione del messaggio da parte dei client e all'invio dei messaggi al server. All'avvio dell'applicazione, il gateway si mette subito in ascolto sulla porta 8400 per la ricezione dei messaggi da parte dei client. Il protocollo utilizzato per la ricezione dei messaggi è di tipo UDP mentre per l'invio dei messaggi al server è di tipo TCP. I suoi indirizzi sono:

- MAC:
  indirizzo MAC: 33:A1:8D:55:42:5E
- Interfaccia UDP:
  indirizzo IP: 192.168.1.20
- Interfaccia TCP:
  indirizzo IP: 10.10.10.5

Il gateway prima di comunicare con il server deve attendere la ricezione di tutti e 4 i messaggi dei client: ogni volta che un messaggio viene ricevuto viene controllato se il client che ha inviato il messaggio faccia parte dei client autorizzati all'invio

dei messaggi. I client autorizzati sono identificati da indirizzo ip e mac contenuti all'interno della arp table che mappa ogni ip dei client al relativo indirizzo MAC. Successivamente viene controllato se il client si è già connesso al gateway precedentemente e se questa verifica ha esito positivo, il messaggio non viene considerato dal gateway poichè non è possibile ricevere più messaggi dallo stesso client prima dell'invio dei messaggi al server. Il messaggio nel caso in cui non siano verificate le condizioni spiegate nei periodi precedenti, procede al salvataggio temporaneo dei messaggi ricevuti in un'unica stringa che sarà poi successivamente inviata al server tramite l'interfaccia di rete TCP del gateway.

```
Gateway: sono in attesa di ricevere messaggi
--------RICEVO UN MESSAGGIO DA MAC:  27:39:F9:D9:01:07

SOURCE MAC:  27:39:F9:D9:01:07

DESTINATION MAC:  33:A1:8D:55:42:5E

SOURCE IP:  192.168.1.16

DESTINATION IP:  192.168.1.20

MESSAGGIO RICEVUTO:  192.168.1.16 - 20.00 - 40°C - 40%

DIMENSIONE MESSAGGIO: 51 bytes

TIME ELAPSED: 0.002706 ms
```

Figura 1.4: Console del gateway quando riceve un messaggio

```
-----HO RICEVUTO TUTTI I MESSAGGI QUINDI MI CONNETTO AL SERVER E LI INVIO----

INVIO AL SERVER IL MESSAGGIO:
192.168.1.10 - 08.00 - 09°C - 27%
192.168.1.14 - 12.00 - 09°C - 65%
192.168.1.15 - 16.00 - 22°C - 55%
192.168.1.16 - 20.00 - 40°C - 40%
```

Figura 1.5: Console del gateway quando invia il messaggio al server

2.1.3 Classe Client

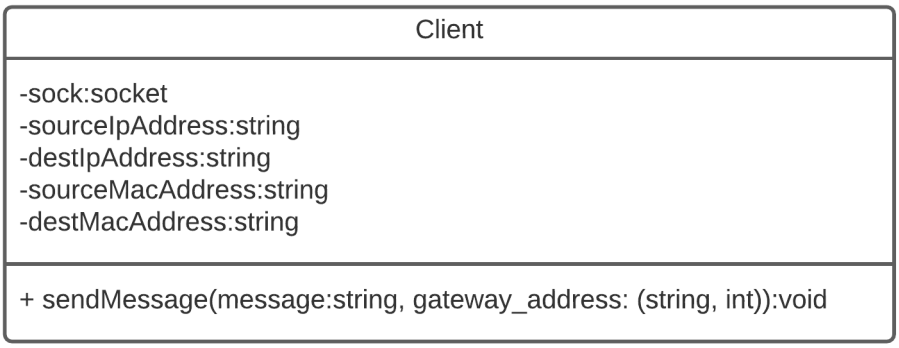

Figura 1.6: UML della classe Client

La classe Client è quella che contiene l'implementazione dei client. Essi sono adibiti all'invio dei messaggi al gateway. I 4 client hanno tutti lo stesso comportamento e quindi i client sono tutti instanze di questa stessa classe. Il protocollo utilizzato per l'invio dei messaggi è di tipo UDP. Gli indirizzi sono:

- CLIENT 1:
  indirizzo IP: 192.168.1.10
  indirizzo MAC: 5C:D4:85:A3:5D:AE
- CLIENT 2:
  indirizzo IP: 192.168.1.14
  indirizzo MAC: 02:79:6F:E5:91:24
- CLIENT 3:
  indirizzo IP: 192.168.1.15
  indirizzo MAC: 3D:0E:25:E3:CD:85
- CLIENT 4:
  indirizzo IP: 192.168.1.16
  indirizzo MAC: "27:39:F9:D9:01:07

```
Invio il messaggio: "192.168.1.10 - 08.00 - 09°C - 27%"
Ho inviato il messaggio, quindi chiudo la connessione con il gateway
Invio il messaggio: "192.168.1.14 - 12.00 - 09°C - 65%"
Ho inviato il messaggio, quindi chiudo la connessione con il gateway
Invio il messaggio: "192.168.1.15 - 16.00 - 22°C - 55%"
Ho inviato il messaggio, quindi chiudo la connessione con il gateway
Invio il messaggio: "192.168.1.16 - 20.00 - 40°C - 40%"
Ho inviato il messaggio, quindi chiudo la connessione con il gateway
```

Figura 1.7: Invio dei messaggi da parte dei client

2.2 Struttura dei pacchetti

L'elemento principale della comunicazione tra le varie reti sono i pacchetti di dati. La traccia chiede di conoscere i tempi di trasmissione dei vari pacchetti e per questo ho pensato di inserire all'interno dei pacchetti oltre che all'header e al messaggio, un ulteriore campo chiamato start time. Lo start time fa parte del messaggio e contiene un informazione riguardo l'istante (in millisecondi) di partenza del pacchetto dal client o dal gateway.

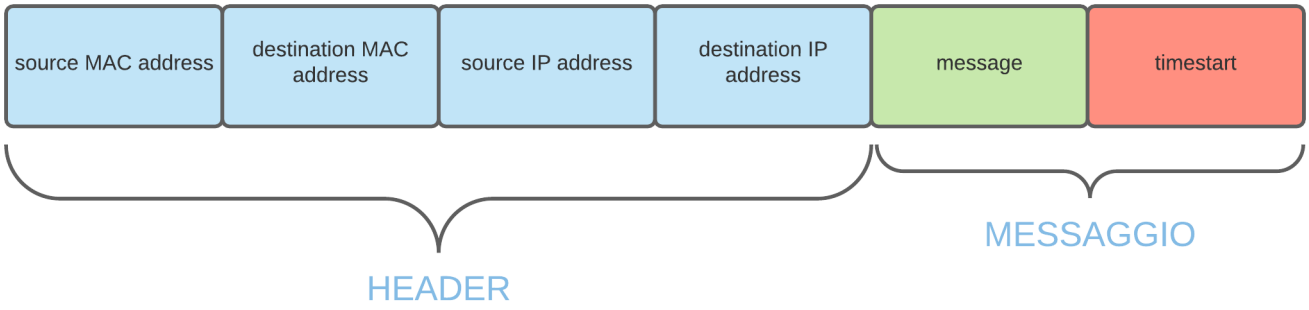

Figura 1.8: struttura dei pacchetti

2.3 Timing e buffersize

Quando un messaggio viene inviato viene inserito all'interno del pacchetto l'ora in millisecondi di partenza del messaggio. Questo permette di calcolare il tempo di trasmissione di ogni pacchetto eseguendo una semplice differenza tra l'ora di ricezione del pacchetto e l'ora di timestart. Il gateway o il server, quando ricevono il messaggio considerano quindi l'ora di partenza e l'ora di arrivo così da ottenere il tempo impiegato. Per utilizzare questa funzionalità è stata utilizzata la libreria `time` di python.
Per la dimensione dei buffer dei pacchetti sono stati utilizzati 4096 bytes.

Capitolo 3

Guida all'esecuzione dell'applicazione

3.1 Esecuzione dei file

- client.py, server.py e gateway.py sono le 3 classi principali e contengono le implementazioni degli oggetti di rete.
- clients.py contiene un piccolo script per eseguire i 4 client contemporaneamente

Per l'esecuzione è necessario eseguire in tre terminali diversi, i 3 file in sequenza:

1. server.py
2. gateway.py
3. clients.py

Per eseguire bisogna prima dare i permessi eseguendo nella directory dove sono contenuti i file il seguente comando:
`chmod u+x ./*.py`
successivamente aprire 3 terminali nella directory dei file ed eseguire i file come da sequenza sopra riportata eseguendo:
`./*nomefile*.py`

3.2 Repository github

Link al repository su github: https://github.com/davidedimarco00/ProgettoReti