Relazione progetto programmazione di reti

Manuel Luzietti

Matricola 0000937684

Indice

1 Introduzione **3**

2 Descrizione **4**

3 Istruzioni **10**

4 Immagini output **11**

1.Introduzione

Il progetto consiste nel simulare uno scenario di Smart Meter IoT che rilevano la temperatura e umidità del terreno in cui sono posizionati.

Quattro dispositivi si collegano una volta al giorno con una connessione UDP verso un gateway, al quale comunicano le misure effettuate durante le 24 ore precedenti.

I dati, che consistono dell' ora della misura e della temperatura e umidità rilevata, sono memorizzati in un file.

Una volta che I pacchetti di tutti I dispositivi sono arrivati al Gateway, questo instaura una connessione TCP verso un server centrale dove I valori vengono visualizzati su console.

2.Descrizione

Device:

I Device sono stati incorporati in un unico sorgente addottando una strategia multithread, al solo scopo della simulazione. In un contesto reale si può importare la classe dal sorgente device.py per creare device su dispositivi separati. La classe comprende sei campi interni:

- Ip_device : l' ip del device
- Socket_device: la socket UDP attraverso la quale invierà dati al gateway
- Running: stato del thread
- FileName: nome del file da cui leggerà I dati rilevati
- Gateway_address: tupla contenente ip e porta dell' applicativo gateway
- Secs: secondi tra due rivelazioni

Al momento della creazione di un' istanza della classe verranno richiesti come argomenti Ip_device e FileName. Sono opzionali gatewayAddress e secs se si vogliono impostazioni differenti. Nel contesto reale, I device dovrebbero inviare dati ogni 24 ore, ma per testare al meglio l'applicativo ho impostato il delay a 15 secondi, può comunque essere cambiato settando secs come argomento. Il gateway_address è settato di default all' indirizzo di loopback così da testare l' applicazione, può comunque essere cambiato a sua volta.

```
def __init__(self,ip,fileName,gatewayAddress=("localhost",8000),secs=15):
    Thread.__init__(self)
    self.ip_device = ip
    self.socket_device = socket(AF_INET,SOCK_DGRAM)
    self.running = True
    self.fileName = fileName
    self.gateway_address = gatewayAddress
    self.secs = secs
```

Fig 2.1: codice della funzione __init__ della classe DeviceClass

Il cuore del Device è all' interno della funzione run.

All' interno di un ciclo while che viene interrotto solo se lo stato del thread (`running`) è settato su false, vengono letti ogni `secs` secondi I dati dal relativo file attraverso la funzione readData, che legge la prima riga per poi cancellarla.

```
def readData(name):
    fd = open("data\\" + name + ".txt","r")
    lines = fd.readlines()
    fd.close()
    fd = open("data\\" + name + ".txt","w")
    fd.write("".join(lines[1:]))
    fd.close()
    return lines[0]
```

Fig 2.2: codice della funzione readData dal modulo device.py

Questi dati vengono poi inviati al gateway attraverso la socket UDP, creata all' inizializzazione dell' oggetto istanza della classe DeviceClass. Durante questo procedimento vengono calcolati i tempi di trasmissione usando la funzione perf_counter del modulo time. In fine vi è un ciclo for che itera una volta per ogni secondo del delay `secs`. Questa scelta è stata preferita al semplice sleep(secs) per fare in modo che non sia bloccante durante l' intera esecuzione.

```
def run(self):
    while self.running:
        #generazione dati
        try:
            message = self.ip_device + " - " + readData(self.fileName).rstrip('\n')
        except:
            sys.exit(0)
        #calcolo trasmission delay
        thread_lock.acquire()
        print(self.ip_device + "sending data to Gateway on interface 192.168.1.0")
        thread_lock.release()
        time0 = time.perf_counter()
        self.socket_device.sendto(message.encode(), self.gateway_address)
        time1 = time.perf_counter()
        #acquisisco lock così da stampare su terminale in modo ordinato
        thread_lock.acquire()
        print(self.ip_device , " message --trasmission time: ", time1-time0, " data: ",'\n'+message+'\n')
        thread_lock.release()
        #attesa a prossima generazione dati
        for i in range(self.secs):
            #ciclo di sleep(1) così da non bloccare device per tutta la durata dell' attesa
            time.sleep(1)
            if not self.running:
                break
```

Fig 2.3: funzione run della classe DeviceClass.

La funzione signal_handler del modulo device ci permette un' uscita pulita dall' apllicazione, digitando Ctrl+C da tastiera. Questo è possibile grazie alle scelte che hanno permesso di rendere l' applicativo non bloccante durante l' intera esecuzione. La funzione blocca tutti I thread, chiude gli eventuali socket aperti, fa il join dei thread e poi chiude l'applicativo.

```
def signal_handler(signal,frame):
    print("exiting")
    try:
        for device in DeviceThreads:
            device.stop()
            if device.socket_device:
                device.socket_device.close()
            device.join()
    finally:
        sys.exit()
```

Fig 2.4: funzione signal_handler per la chiusura pulita dei thread e dei relativi socket.

Gateway:

Nella prima parte del gateway sono create le liste device_message e deviceip_received che conterranno rispettivamente I messaggi arrivati dai vari dispositivi e gli indirizzi ip di questi. Viene creata la socket UDP, a cui verrà assegnato l' indirizzo locale con bind. La socket viene resa non bloccante attraverso la funzione settimeout che ci permette di impostare un tempo limite oltre il quale lancerà un' eccezione. La scelta di rendere il tutto non bloccante è stata fatta ancora una volta per permettere un' uscita pulita dall' applicazione in ogni momento. La variabile NUM_DEVICE indica il numero di messaggi di device diversi che il gateway dovrà aspettare prima di potersi collegare al server.

```
print("Gateway started")
signal.signal(signal.SIGINT,signal_handler)
#lista di messaggi arrivati tramite protocollo udp
device_message = []
#lista degli ip da cui ho ricevuto i messaggi
deviceip_received = []
server_address = ("localhost",8100)
gateway_socket_TCP = None
gateway_socket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
#setto timeout così da non rendere alcune funzioni completamente bloccanti
gateway_socket.settimeout(5)
gateway_socket.bind(("localhost",8000))
#numero di device di cui attendere messagio prima di inviare a server dati raccolti
NUM_DEVICE = 4
```

Fig 2.5: prima parte del gateway.

Una volta settati questi parametri si entra nel ciclo principale del gateway.

All' interno del ciclo while principale vi è un ciclo while al cui interno si aspetta la ricezione dei messaggi necessari. Attraverso la lista deviceip_received è possibile controllare che I messaggi raccolti siano tutti di diversi Device. Una volta raccolti il numero necessario di messaggi da ip diversi, si esce dal ciclo dedicato alla ricezione dei messaggi. Questa parte di ricezione messaggi è stata inserita all' interno di blocco try-except per via dell' eccezione che può venir lanciata da un eventuale timeout. In questo modo se non dovesserò arrivare messaggi entro un tempo limite l' eccezzione sbloccherà l' applicativo.

```
while True:
    while len(device_message) < NUM_DEVICE:
        try:
            #prova a ricevere dati, messo all' interno di blocco try perchè il socket non è bloccante
            #dopo 5 secondi da timeout exception, così da poter ancora bloccare applicazione all' occorrenza
            message, addr = gateway_socket.recvfrom(1024)
            message_ip = message.decode().split("-")[0]
            #controlla che ip di mittente messaggio non sia già in lista
            if message_ip not in deviceip_received:
                print("receiving data on interface 192.168.1.0 from " + message_ip)
                device_message.append(message.decode())
                deviceip_received.append(message_ip)
        except:
            pass
```

Fig 2.6: prima sezione del ciclo while principale responsabile della ricezione messaggi dai Device

Si instaura una connessione TCP verso il server tramite la quale verranno inviati I dati raccolti. Sempre attraverso la funzione perf_counter di time verranno calcolati I trasmission delay. Una volta inviati I dati vengono pulite le liste device_message e deviceip_received che saranno così di nuovo pronte a raccogliere nuovi messaggi e relativi ip.

```
    try:
        gateway_socket_TCP = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        #apre connessione tcp
        gateway_socket_TCP.connect(server_address)
        data = str("\n").join(device_message)
        #calcolo trasmission delay
        time0 = time.perf_counter()
        gateway_socket_TCP.send(data.encode())
        time1 = time.perf_counter()
        #chiude connessione tcp
        gateway_socket_TCP.close()
        print("data sent on interface 10.10.10.0 , trasmission delay: ", time1-time0)
        #pulisco lista di messaggi raccolti e di ip
        device_message.clear()
        deviceip_received.clear()
    except Exception as s:
        print("error occurs: ",s)
        if gateway_socket_TCP:
            gateway_socket_TCP.close()
```

Fig 2.7: Seconda parte del ciclo while, responsabile dell' inoltro dei dati raccolti al server.

Ancora una volta è presente la funzione signal_handler che si occuperà di chiudere l' applicativo in maniera pulita.

Server:

Viene creata la socket TCP che viene collegata all' indirizzo locale sulla porta 8100, e viene messe in ascolto. Ancora una volta imposto un timeout sulla socket così che non sia bloccante nel caso non dovessero arrivare messaggi.

All' interno di un ciclo while vengono accettate le connessioni TCP provenienti dal gateway, attraverso la quale si ricevono I dati che verranno poi visualizzati su terminale.

Ancora una volta è presente la funzione signal_handler che si occuperà di terminare l' applicativo alla pressione di CTRL+C su tastiera

```
print("Server started")
socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
socket.bind(("localhost",8100))
socket.listen(1)
#setto timeout così da rendere accept() e recv() non completamente bloccanti,
#così da poter ancora interrompere l'applicativo al bisogno
socket.settimeout(10)
sock = None
signal.signal(signal.SIGINT, signal_handler)
while True:
    try:
        #accetta connessione tcp e riceve dati
        sock, addr = socket.accept()
        message =  sock.recv(1024).decode()
        print(message)
        sock.close()
    except:
        pass
```

Fig 2.8: codice server

3.Istruzioni

1. (nel caso I file dati all' interno della cartella data siano vuoti, vanno generati eseguendo lo script `generator data.py` all' interno della stessa cartella data, posizionandosi nella cartella data prima di eseguirlo)
2. Avviare `server.py`
3. Avviare `gateway.py`
4. Avviare `device.py`
5. Per interrompere I singoli premere CTRL+D, questi termineranno nel giro di pochi secondi.

4.Immagini output

```
Device  192.168.1.2  started
Device  192.168.1.3  started
192.168.1.2sending data to Gateway on interface 192.168.1.0
Device  192.168.1.4  started
192.168.1.2  message --trasmission time:  0.00099149999999959  data:
192.168.1.2 - 01:32 - 16 - 64

192.168.1.3sending data to Gateway on interface 192.168.1.0
Device  192.168.1.5  started
192.168.1.3  message --trasmission time:  0.02343340000000005  data:
192.168.1.3 - 01:32 - 19 - 29

192.168.1.5sending data to Gateway on interface 192.168.1.0
192.168.1.4sending data to Gateway on interface 192.168.1.0
192.168.1.5  message --trasmission time:  0.029034900000000974  data:
192.168.1.5 - 01:32 - 30 - 32

192.168.1.4  message --trasmission time:  0.0644834999999997  data:
192.168.1.4 - 01:32 - 12 - 88
```

Fig 4.1: console di device.py con trasmission delay.

```
Gateway started
receiving data on interface 192.168.1.0 from 192.168.1.2
receiving data on interface 192.168.1.0 from 192.168.1.3
receiving data on interface 192.168.1.0 from 192.168.1.5
receiving data on interface 192.168.1.0 from 192.168.1.4
data sent on interface 10.10.10.0 , trasmission delay:  3.549999999918896e-05
```

Fig 4.2: console di gateway, riceve dati attraverso la socket UDP e trasmette su socket TCP una volta raccolti I dati di tutti I dispositivi.

```
Server started
192.168.1.2 - 01:32 - 16 - 64
192.168.1.3 - 01:32 - 19 - 29
192.168.1.5 - 01:32 - 30 - 32
192.168.1.4 - 01:32 - 12 - 88
```

Fig 4.3: console di server che riceve dati su connessione TCP.