

Relazione Progetto

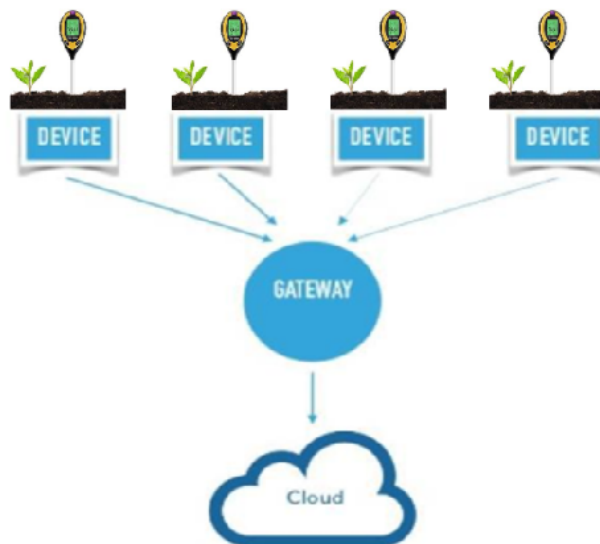
Traccia 1 - Progetto IoT

A.A 2020/2021

Pietro Pezzi
0000925022

1. Introduzione

Il progetto consiste nel realizzare un programma *python* che emuli 4 dispositivi IoT che comunicano letture ad un gateway; quest'ultimo, una volta ottenute le letture di tutti i dispositivi, le invia ad un server che si occupa di stamparle a schermo (sulla console). L'immagine seguente descrive al meglio lo scenario.



2. Istruzioni esecuzione

Per eseguire il programma è fondamentale rispettare l'ordine di esecuzione dei vari script. Innanzitutto bisogna eseguire *server.py*, successivamente eseguire *gateway.py* ed infine *run_devices.py*. Tutti e tre i programmi sopracitati sono eseguibili tramite il comando `python3 <nomeprogramma>` oppure `python <nomeprogramma>`

Note riguardo Spyder:

I programmi sopra indicati funzionano senza problemi anche all'interno di 3 console in *Spyder*, tuttavia vi sono comunque alcune scomodità dovute all'IDE. Riguardo ai thread di *run_devices.py*, pur essendo *daemon* thread ovvero thread che si dovrebbero chiudere una volta terminato il processo padre, continuano ad essere eseguiti anche terminato il processo. Inoltre gateway e server continuano ad occupare le porte anche se terminati. Per risolvere tutti questi problemi è necessario terminare ognuno dei 3 processi riavviando il kernel della console tramite **Ctrl + .**

3. Descrizione

Il codice del progetto consiste in 4 programmi:

1. *device.py*

device.py contiene il codice di un oggetto **Device** che emula il comportamento di un dispositivo IoT per la misurazione del terreno. Un device esegue un certo numero di letture, con un certo intervallo di tempo tra ogni lettura, per poi inviarlo al gateway. Le letture vengono memorizzate in un file txt all'interno della directory *reads*; una volta inviate le letture al gateway il file txt viene svuotato per poi “riempirlo” nuovamente. Per la creazione di un device sono necessari i seguenti argomenti: ip del device, id del device, numero di letture da effettuare prima di inviarle al gateway, intervallo di tempo tra ogni lettura.

2. *run_devices.py*

run_devices.py serve per configurare ed eseguire diversi device, è possibile modificare le diverse variabili al suo interno, in modo da eseguire più device con diverso numero di letture e intervallo di tempo. *(leggere i commenti all'interno del suo codice per maggiori informazioni.)*

3. *gateway.py*

gateway.py si occupa di collezionare le letture di ogni device, comunicate tramite socket UDP, per poi inviarle in un unico messaggio al server tramite una connessione TCP. Assicurarsi che il *deviceAmount* all'interno di *run_device.py* corrisponda con quello del gateway. Il gateway è in grado di scartare le letture che ha già ottenuto in una “sessione” da un device; perciò, nel caso in cui *deviceAmount* non corrispondesse a quello di *run_devices.py*, il gateway potrebbe non mandare mai un messaggio al server, oppure mandare letture disordinate.

4. *server.py*

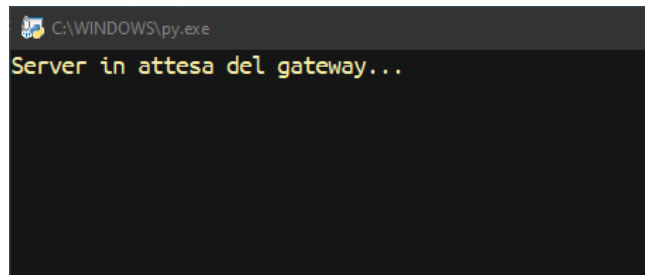
server.py si occupa solamente di ascoltare, tramite un socket TCP, un messaggio dal gateway; questo messaggio viene stampato a video nel seguente formato

```
Ip_address_device_1 - ora_misura - valore_temperatura -  
valore_umidità
```

per poi attendere un nuovo messaggio e così via.

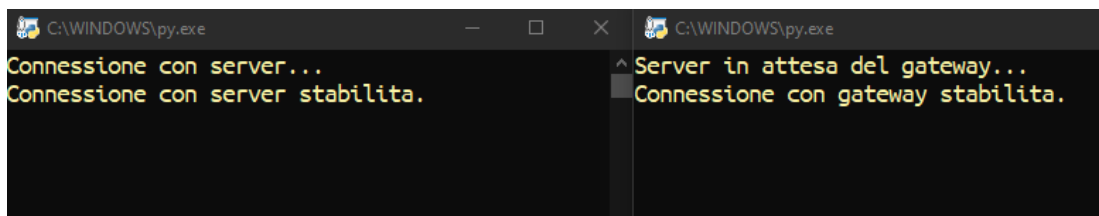
4. Descrizione esecuzione

Seguendo l'ordine riportato al punto “2. Istruzioni esecuzione”, avviando il server si metterà in attesa del gateway sull'indirizzo locale 127.0.0.1 sulla porta 8000.



```
C:\WINDOWS\py.exe
Server in attesa del gateway...
```

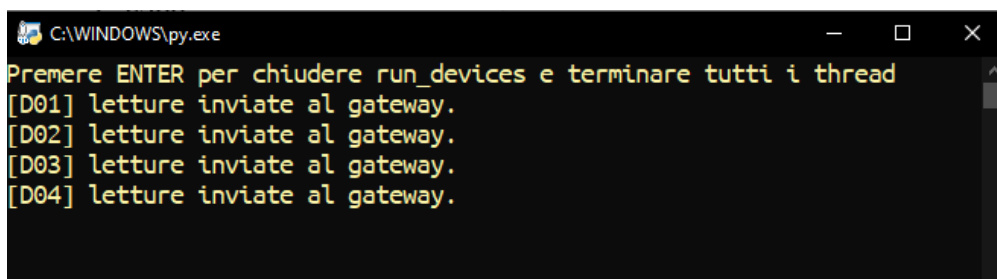
In seguito, una volta avviato il gateway, il server stabilirà una connessione TCP con il gateway, che si metterà in attesa di messaggi sul socket UDP sempre sull'indirizzo locale 127.0.0.1 ma su porta 10000.



```
C:\WINDOWS\py.exe
Connessione con server...
Connessione con server stabilita.

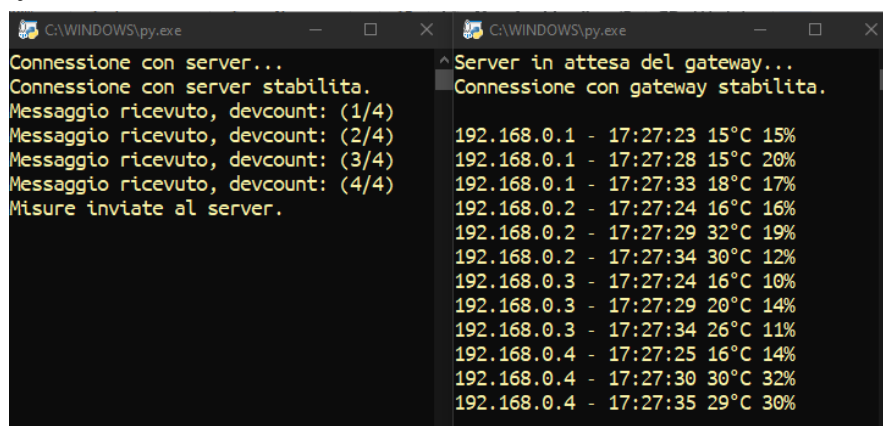
C:\WINDOWS\py.exe
Server in attesa del gateway...
Connessione con gateway stabilita.
```

Infine eseguire run_devices.py, che collegandosi tramite un socket UDP, invierà al gateway le sue letture. Attendendo qualche secondo si dovrebbe ottenere un output simile.



```
C:\WINDOWS\py.exe
Premere ENTER per chiudere run_devices e terminare tutti i thread
[D01] letture inviate al gateway.
[D02] letture inviate al gateway.
[D03] letture inviate al gateway.
[D04] letture inviate al gateway.
```

Su gateway e device otterremo.



```
C:\WINDOWS\py.exe
Connessione con server...
Connessione con server stabilita.
Messaggio ricevuto, devcount: (1/4)
Messaggio ricevuto, devcount: (2/4)
Messaggio ricevuto, devcount: (3/4)
Messaggio ricevuto, devcount: (4/4)
Misure inviate al server.

C:\WINDOWS\py.exe
Server in attesa del gateway...
Connessione con gateway stabilita.
192.168.0.1 - 17:27:23 15°C 15%
192.168.0.1 - 17:27:28 15°C 20%
192.168.0.1 - 17:27:33 18°C 17%
192.168.0.2 - 17:27:24 16°C 16%
192.168.0.2 - 17:27:29 32°C 19%
192.168.0.2 - 17:27:34 30°C 12%
192.168.0.3 - 17:27:24 16°C 10%
192.168.0.3 - 17:27:29 20°C 14%
192.168.0.3 - 17:27:34 26°C 11%
192.168.0.4 - 17:27:25 16°C 14%
192.168.0.4 - 17:27:30 30°C 32%
192.168.0.4 - 17:27:35 29°C 30%
```

5. Dettagli implementativi.

Come accennato precedentemente, è possibile modificare la quantità di device e le varie caratteristiche di ogni dispositivo aggiornando alcune variabili all'interno di *run_devices.py*. La modifica di queste variabili consente di cambiare facilmente il numero dei dispositivi e il comportamento di ogni dispositivo, è però essenziale che la variabile *deviceAmount* in *run_devices.py* combaci con quella in *gateway.py*.

Dimensioni buffer

il socket del gateway utilizza un buffer di **1024** byte, mentre il server utilizza un buffer di **4096** byte. Queste due dimensioni infatti vengono utilizzate all'interno di *run_devices.py* per controllare se i valori inseriti sono validi, infatti i valori dati devono rispettare le seguenti condizioni:

$\text{readAmount} * 34 \leq 1024$

$\text{readAmount} * 34 * \text{deviceAmount} \leq 4096$:

dove *readAmount* è il numero di letture che esegue un device prima di connettersi al gateway, *deviceAmount* è il numero dei device e **34** è la lunghezza in byte di ogni lettura.

Packet transfer time

La dimensione dei pacchetti è stata individuata tramite WireShark.

Formula per ottenere il PTT:

$$\text{Packet transfer time} = (\text{packet size})_{\text{bit}} / (\text{bit rate})_{\text{bit/s}}$$

utilizzando *deviceAmount*=4, *readAmount*=3.

Ptt pacchetto UDP:

udp && ip.addr == 127.0.0.1 && ip.dst == 127.0.0.1						
No.	Time	Source	Destination	Protocol	Length	Info
11094	0.192456	127.0.0.1	127.0.0.1	UDP	134	61693 → 10000 Len=102
11095	0.491093	127.0.0.1	127.0.0.1	UDP	134	61693 → 10000 Len=102
11120	0.045355	127.0.0.1	127			
11125	0.347891	127.0.0.1	127			

Lunghezza in byte → $134 * 8 = 1072 \text{ bit}$

Un pacchetto UDP ha una dimensione di **1072** bit, con un bit rate di **10** mbps otteniamo un tempo di **0,0001** secondi ovvero **0,1** ms

Ptt pacchetto TCP:

tcp && ip.addr == 127.0.0.1 && ip.dst == 127.0.0.1						
No.	Time	Source	Destination	Protocol	Length	Info
19389	0.000035	127.0.0.1	127.0.0.1	TCP	45	51710 → 51709 [PSH, ACK] Seq=9326 Ack=1 Win=65535 Len=1
19390	0.000007	127.0.0.1	127.0.0.1	TCP	44	51709 → 51710 [ACK] Seq=1 Ack=9327 Win=34049 Len=0
19406	0.000211	127.0.0.1	127.0.0.1	TCP	452	52435 → 8000 [PSH, ACK] Seq=2857 Ack=1 Win=10233 Len=408
19407	0.000015	127.0.0.1	127.0.0.1	TCP	45	8000 → 52435 [ACK] Seq=1 Ack=2857 Win=10219 Len=0
19408	0.022860	127.0.0.1	127.0.0.1	TCP	45	52435 → 8000 [PSH, ACK] Seq=2857 Ack=1 Win=65535 Len=1

Lunghezza in byte → $452 * 8 = 3616$ bit

Un pacchetto TCP ha una dimensione di **3616** bit, con un bit rate di **10** mbps otteniamo un tempo di circa **0,3** ms

6. Librerie utilizzate

- datetime
- time
- random
- socket
- os
- sys
- Thread from threading