# Relazione Progetto Programmazione di Reti

Daniel Guariglia - Shola Oshodi

August 2021

# Contents

# 1 Descrizione del Progetto

Il progetto realizzato si pone come obiettivo quello di simulare uno scenario di Smart Meter IOT, nel quale sono presenti quattro device che effettuano misurazioni circa la temperatura e l'umidità del terreno e, una volta al giorno trasmettono i dati raccolti al Gateway per mezzo di una connessione UDP. Il Gateway dovrà poi preoccuparsi di instradare i dati raccolti dai device a un Server Cloud (tramite una connessione TCP) che si occuperà poi della visualizzazione dei dati. Il Gateway presenterà due interfacce di rete differenti: una per comunicare con i Device [**con indirizzo IP 192.168.1.0/24**] e una per comunicare con il Server [ **con indirizzo IP 10.10.10.0/24**].

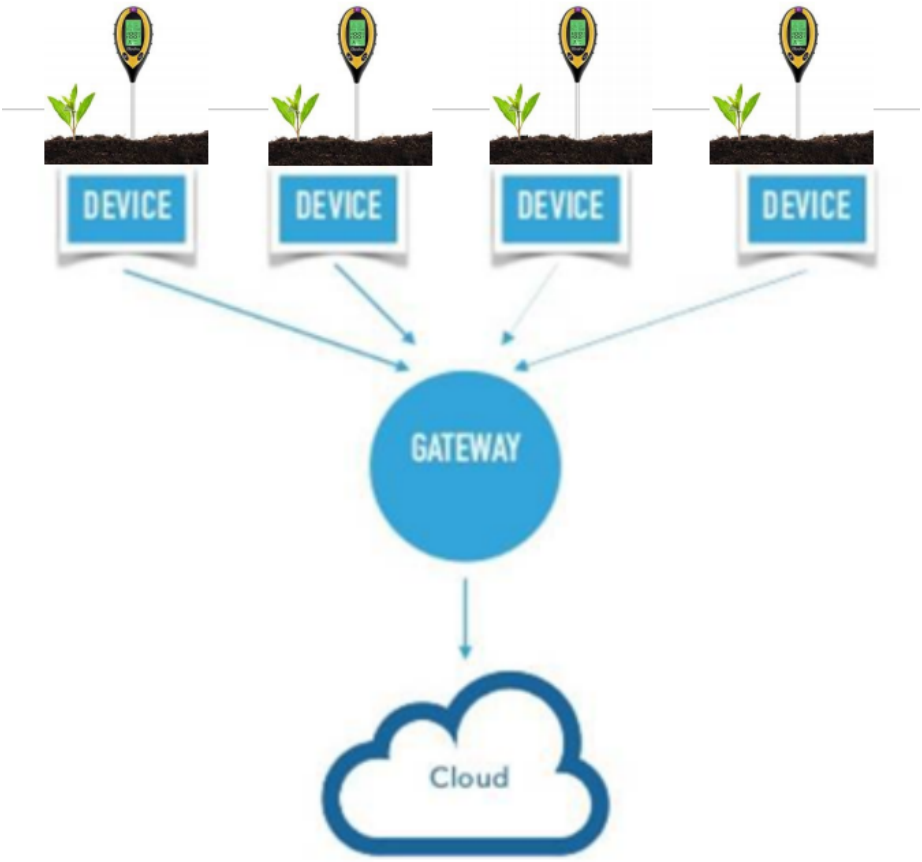


Figure 1.1: Descrizione del progetto

# 2 Analisi e Progettazione

Al fine di simulare al meglio la realtà proposta si è scelto di mantenere una divisione netta tra le principali componenti del progetto (Device, Gateway e Cloud), inoltre, con l'intenzione di rendere il codice quanto più riusabile possibile ogni componente verrà istanziato come un Thread completamente autonomo. Questo ci permetterà di estendere le componenti simulate nel nostro programma potendo così aumentare a nostro piacimento il numero di Device o di Server presenti. Abbiamo scelto di creare un ambiente di testing nel quale è possibile inserire l'intervallo di tempo nel quale si vogliono simulare le 24 ore necessarie a raccogliere i dati delle misurazioni. In questo modo è possibile verificare più agevolmente il corretto funzionamento di quanto realizzato. Ogni Device campionerà umidità e temperatura 4 volte in un giorno. Infine con l'obiettivo di emulare il salvataggio dei dati raccolti sul Server Cloud abbiamo inserito un file locale che fungerà da storico per le misurazioni.

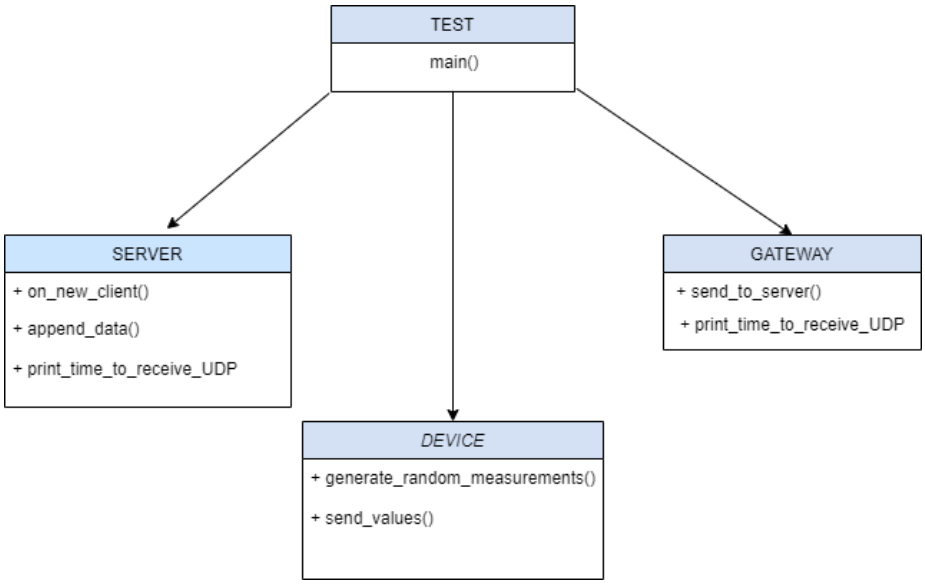


Figure 2.1: Descrizione delle classi

# 3 Socket

La comunicazione tra i diversi dispositivi è ovviamente possibile grazie all'utilizzo dei Socket. Ogni Device aprirà con il Gateway una connessione UDP, avendo il gateway due interfacce di rete e dovendo occuparsi anche di ritrasmettere al Cloud le informazioni ricevute dai device sarà necessario aprire anche una connessione TCP con quest'ultimo. Il Cloud sarà in ascolto sulla porta 8081 che, insieme al suo indirizzo IP sarà necessario affinchè il gateway possa stabilire una connessione. Le connessioni sono gestite all'interno di un try catch in modo che i diversi dispositivi siano in grado di gestire eventuali errori o eccezioni generate dall'utilizzo dei socket. La comunicazione tra i diversi socket avverrà quindi come riportato in seguito.

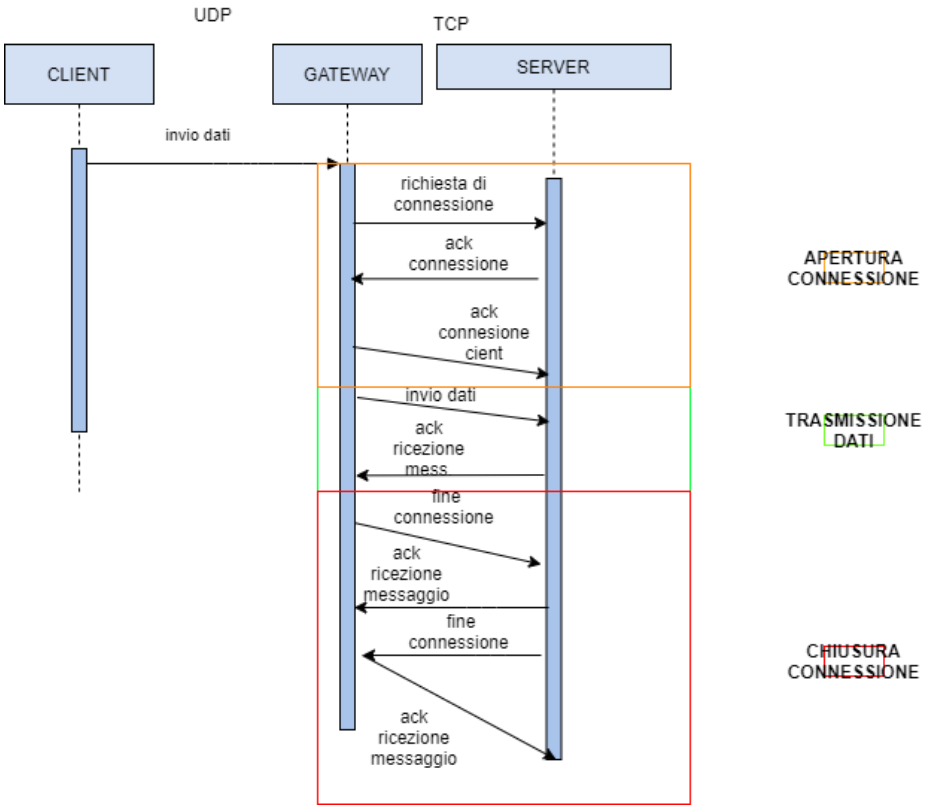


Figure 3.1: Comunicazione Socket

# 4 Strutture dati

All' interno del programma per trasmettere tutti i dati raccolti relativi alle misurazioni si è scelto di utilizzare una struttura dati JSON che ci permetterà di gestire in modo pratico le comunicazioni tra i diversi dispositivi.

Per il salvataggio dei dati si è scelto di utilizzare un file di testo sul quale verrà registrato ogni dato campionato, che fungerà quindi da storico delle misurazioni. Il file di storico come anticipato in precedenza è stato inserito per emulare in maniera quanto più realistica il funzionamento del Server Cloud.

# 5 Threads

Ogni device verrà gestito da un thread autonomo avremo quindi 6 thread totali: quattro riservati ai device, uno al gateway e uno al cloud.

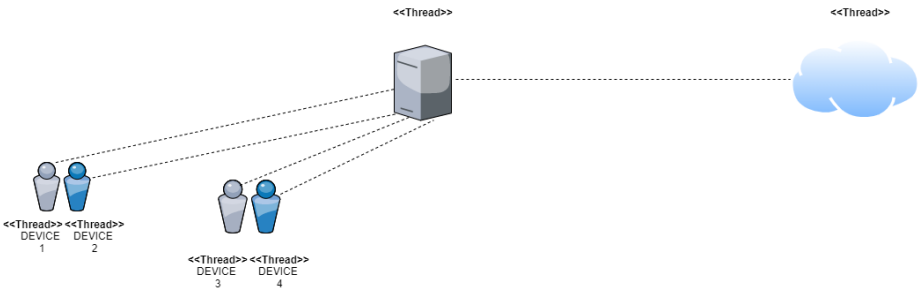


Figure 5.1: Schematizzazione dei Thread

Ogni Thread dispone di un metodo pubblico "stop" che blocca il loop e interrompe il thread.

# 6 Guida Utente

All'interno del progetto è presente il file test.py realizzato per facilitare la fase di testing e per verificare il corretto funzionamento del programma. Eseguendolo ci verrà chiesto in quando tempo desideriamo simulare le 24 ore nelle quali i dispositivi effettuano la misurazione (in modo che si possa osservare un invio dei dati più consistente senza dover attendere il passare di un'intera giornata).


Figure 6.1: console

Inserito questo dato verranno creati i diversi dispositivi ai quale verranno poi assegnati gli opportuni indirizzi Ip e numeri di porta.

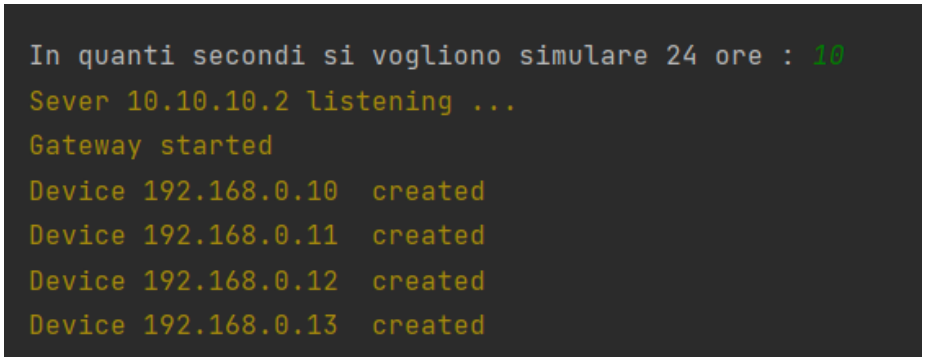


Figure 6.2: console

Quando quest'ultimi saranno avviati sarà sufficiente visualizzare quanto stampato in console per osservare la comunicazione. I server stamperanno i quanto ricevuto aggiungendo anche la dimensione dei buffer e il tempo di comunicazione necessario alla ricezione del messaggio. Per una migliore visualizzazione consigliamo di eseguire il programma utilizzando un IDE quale Spyder o PyCharm.

```
SERVER --> Messaggio ricevuto = 192.168.0.13 -  2021-08-17T15:55:02.028234 -  21 -  10

SERVER --> Messaggio ricevuto = 192.168.0.12 -  2021-08-17T15:54:57.012155 -  32 -  85

SERVER --> Messaggio ricevuto = 192.168.0.12 -  2021-08-17T15:54:59.520508 -  31 -  61

SERVER --> Messaggio ricevuto = 192.168.0.12 -  2021-08-17T15:55:02.028234 -  32 -  90

SERVER --> Messaggio ricevuto = 192.168.0.10 -  2021-08-17T15:54:57.011157 -  36 -  54
```

Figure 6.3: console

```
La trasmissione UDP ha richiesto 0.618  millisecondi

La trasmissione UDP ha richiesto 15.671999999999999  millisecondi

La trasmissione UDP ha richiesto 15.671999999999999  millisecondi

La trasmissione UDP ha richiesto 15.153  millisecondi

La trasmissione TCP ha richiesto 0.9990000000000001  millisecondi
```

Figure 6.4: console

# 7 Librerie utilizzate

Per la realizzazione del programma si è scelto di utilizzare le seguenti librerie:

- Random
- Socket
- Time
- Thread
- Colored (per colorare i messaggi stampati in console)
- DateTime
- Json