TRACCIA 1 – Progetto IoT

Relazione di progetto
Programmazione di Reti

Stefano Scolari 0000915930
Kelvin Oluwada Milare Obuneme Olaiya 0000921596

19 maggio 2021

Indice

Capitolo 1

Analisi

1.1 Requisiti

Alcuni device IOT si occupano di raccogliere dati ed effettuare misurazioni concernenti temperatura ed umidità, questo per riuscire a mantenere adeguata l'irrigazione di una coltivazione "smart".
Si vuole quindi realizzare un sistema di rete composto da un certo numero di device IOT ed un gateway a cui questi ultimi si possano collegare. Il gateway, quando abbia ricevuto le misurazioni da tutti i device, deve inviare i dati ad un server.
Quando il server ha ricevuto i dati si occuperà di visualizzarne il contenuto.
I dispositivi di cui si vuole tenere traccia sono 4, questi inviano aggiornamenti sui dati giornalieri al gateway una volta ogni 4 ore, per un totale di 6 rilevazioni giornaliere.
Come precedentemente accennato, i dati devono tracciare l'ora della rilevazione, la temperatura ed umidità rilevata.

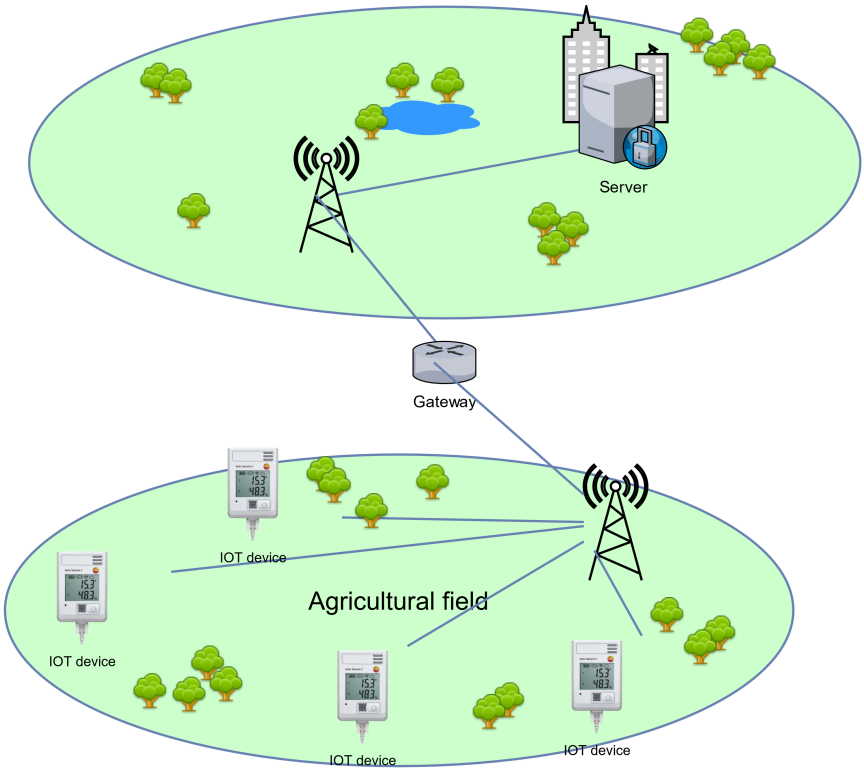

Figura 1.1: Rappresentazione generale della rete

Struttura di rete generale richiesta

Il commissionante richiede che gli indirizzi IP dei **Device** IOT appartengano ad una rete di classe C del tipo `192.168.1.0/24`.
Il **Gateway** deve possedere due interfacce di rete. Una si interfaccia coi dispositivi, appartenente alla loro stessa network. L'altra, che comunica con il **Server**, ha indirizzo IP appartenente alla classe `10.10.10.0/24`, a questa stessa classe appartiene anche l'indirizzo IP del server.
Si richiede inoltre che i device instaurino connessioni **UDP** con il Gateway per inviare i dati, mentre si avrà una connessione **TCP** tra Gateway e Server.

Capitolo 2

Design

2.1 Architettura di Rete

La fig. 2.1 mostra l'architettura di rete implementata per realizzare il sistema richiesto.

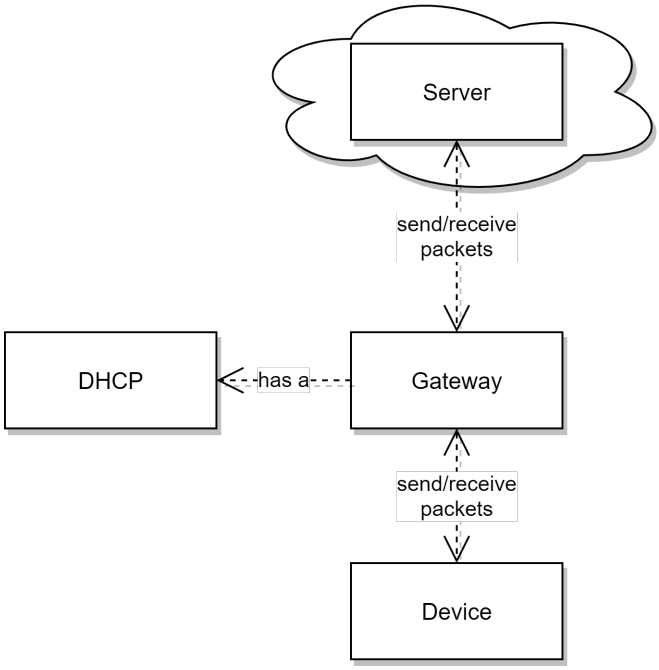

Figura 2.1: Schema UML architettura di rete generale

Device

I Device, alla loro creazione iniziale, con necessaria assegnazione dell'indirizzio MAC, richiedono automaticamente al server DHCP del Gateway di ricevere un indirizzo IP per la loro sottorete.
Quando un dispositivo ha raccolto i dati delle 24h, sempre attraverso una connessione UDP, si occuperà di inviare le misurazioni al Gateway di riferimento.

Gateway

Il Gateway si occupa di raccogliere i dati inviati dai vari dispositivi per mezzo di una connessione UDP.
Quando raccolte le misurazioni da parte dei 4 dispositivi, il Gateway deve occuparsi di instaurare una connessione TCP con il Server a cui vuole inviare i dati.

DHCP server

Abbiamo scelto di munire il Gateway di un proprio DHCP, per permettere un'assegnazione automatica degli indirizzi IP della sottorete di cui fanno parte i Device.

2.2 Design dettagliato

Abbiamo scelto di strutturare le varie entità e componenti attraverso la realizzazione di classi, così da poter operare con un'organizzazione di codice maggiormente espressiva e legata alle astrazioni concettuali ideate.

Device

L'oggetto **Device** ha il compito di raccogliere le varie misurazioni nell'arco della giornata. Ai fini della simulazione abbiamo deciso di far coincidere la durata di un giorno con il trascorrere di 24 secondi. Ogni 4 secondi verrà quindi aggiunta una nuova misurazione. Trascorsa una giornata verrà instaurata la comunicazione con il gateway per inviare l'elenco delle misurazioni.

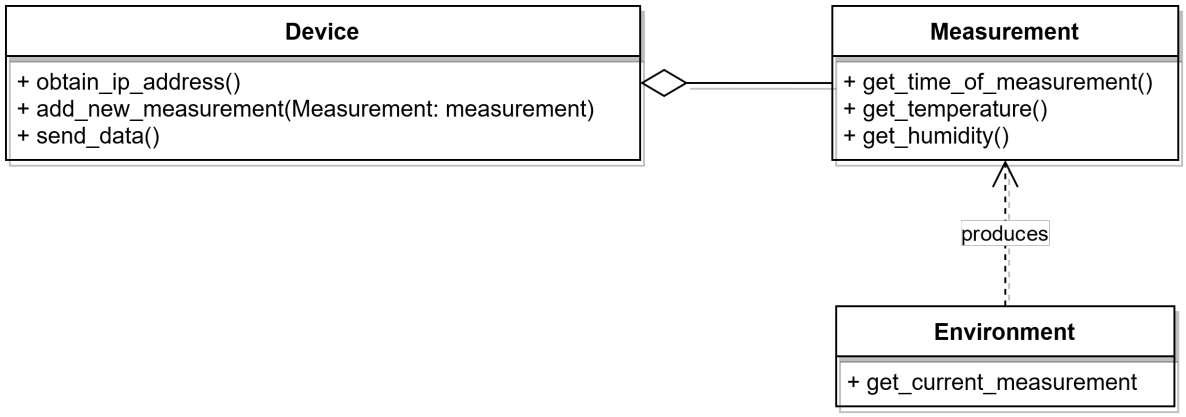

Figura 2.2: Device, measurement e environment

Measurement

Un oggetto della classe Measurement racchiude dentro di sé i dati relativi all'umidità e temperatura ed inoltre un timestamp di quando quella misurazione è stata acquisita.

Environment

Per rappresentare a livello astratto dove vengono effettuate le misurazioni, abbiamo scelto di creare una semplice classe Environment, che nel nostro caso, per semplicità, produce i dati in maniera casuale.

Packet

Lo scambio di informazione tra le diverse entità avviene per mezzo di pacchetti. Anche in questo caso abbiamo operato una semplificazione. Un pacchetto contiente le principali informazioni utili a formare un `header Ethernet`, `header IP` ed infine il `payload` con l'informazione vera e propria. Un pacchetto, inoltre, contiene anche un timestamp di quando questo è stato generato, in maniera tale che sarà poi eventualmente possibile calcolare i tempi di trasmissione.

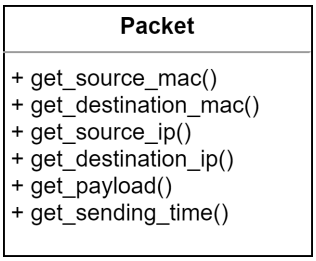

Figura 2.3: Packet

Gateway

Il gateway ha il compito di attendere di ricevere le misurazioni da tutti e 4 i dispositivi della rete, per poi successivamente inviarle tutte assieme al server.

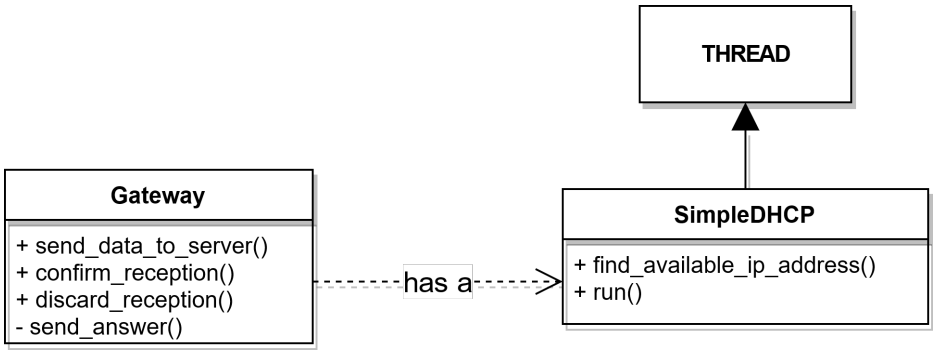

Figura 2.4: Gateway

DHCP

Come accennato in precedenza, un device deve essere collegato ad una rete di classe C di tipo `192.168.1.0/24`. Per far si che ciò avvenga, abbiamo deciso di aggiungere al Gateway un componente DHCP che ha il compito di fornire a chi ne fa richiesta un indirizzo IP appartenente alla suddetta rete.
Ai fini della simulazione, il comportamento di questo DHCP risulta semplificato. Il nostro protocollo prevede che il dispositivo faccia una richiesta UDP al DHCP inviando un pacchetto vuoto (Nell'intestazione del pacchetto l'indirizzo IP del mittente sarà `0.0.0.0`). Quest'ultimo analizzerà l'intestazione del pacchetto estraendone *l'indirizzo MAC*. Se per questo *indirizzo MAC* vi è una corrispondenza nell'*Arp Table* del DHCP, quest'ultimo risponderà al device con un pacchetto che conterrà nel corpo *l'indirizzo IP* che gli è stato precedentemente assegnato. Se invece nell'*Arp Table* non vi è alcuna corrispondenza, l'*indirizzo IP* assegnato avrà un *host number* successivo all'ultimo che è stato concesso nella rete.
Come si evince dalla fig. 2.4, il DHCP è un componente del Gateway, ed abbiamo scelto di renderlo un `Thread`, in modo da poter operare autonomamente dal Gateway stesso, evitando che DHCP e Gateway effettuino funzionalità bloccanti rispetto all' altro.

Server

Il ruolo del server è quello di attendere di ricevere le misurazioni dei 4 device e successivamente stamparle a video.

2.3 Schema di comunicazione

2.3.1 Comunicazione Device - DHCP server

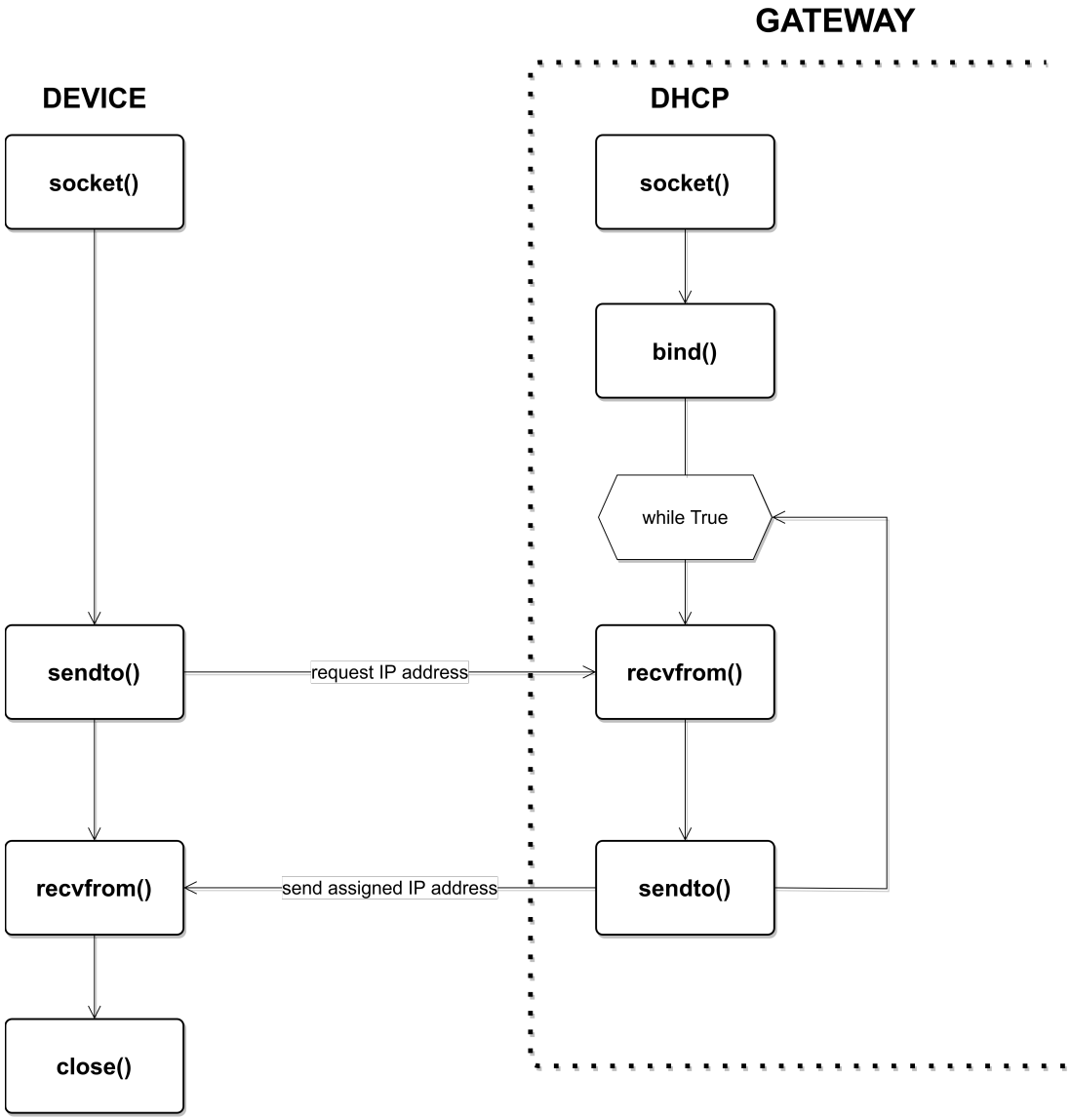

Figura 2.5: Schema rappresentante la comunicazione fra Device e DHCP

2.3.2 Comunicazione Device - Gateway

Dalla fig. 2.6 si può vedere che il device apre una nuova socket **UDP** ogni volta che intende inviare le proprie rilevazioni al Gateway.
Abbiamo optato per questa scelta in modo tale da evitare di tenere impiegate le risorse allocate per mantenere in essere la socket.
L'invio dei dati non è infatti continuo, e non richiede quindi l'apertura costante della socket.
Inoltre, dato che viene utilizzato il *protocollo UDP*, che offre un servizio non affidabile e non orientato alla connessione, abbiamo deciso di far attendere il device per un pacchetto di conferma che dovrà arrivare entro un certo tempo di timeout.
Nel caso non dovesse pervenire conferma allo scattare del timeout, il device provvederà ad inviare nuovamente le sue misurazioni.
In questo modo si cerca di evitare che il Gateway debba aspettare un altro giorno per poter inviare i dati al server.

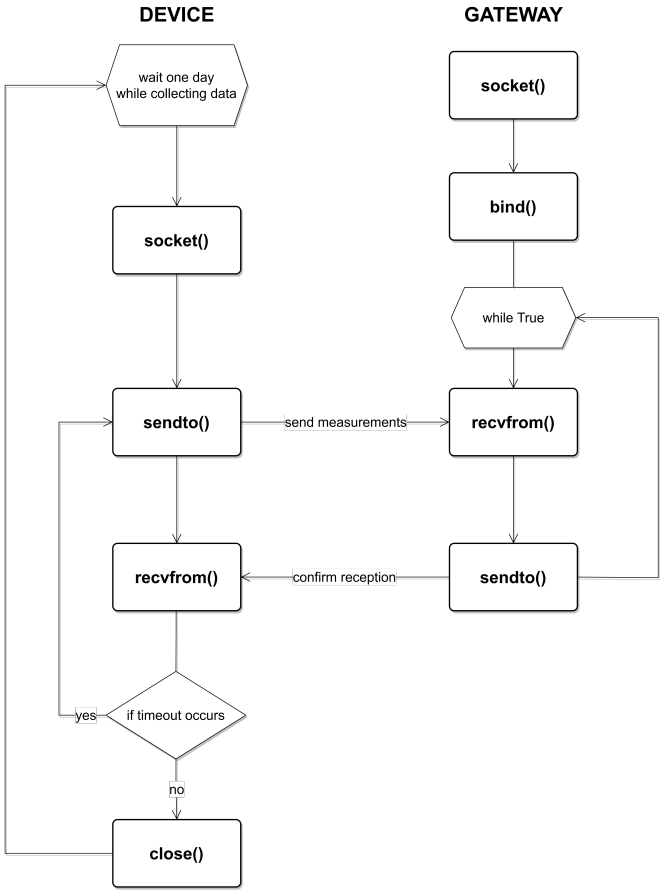

Figura 2.6: Schema rappresentante la comunicazione fra Device e Gateway

2.3.3 Comunicazione Gateway - Server

La creazione della socket **TCP** avviene solo quando il Gateway ha ricevuto le misurazioni da tutti e 4 i dispositivi.

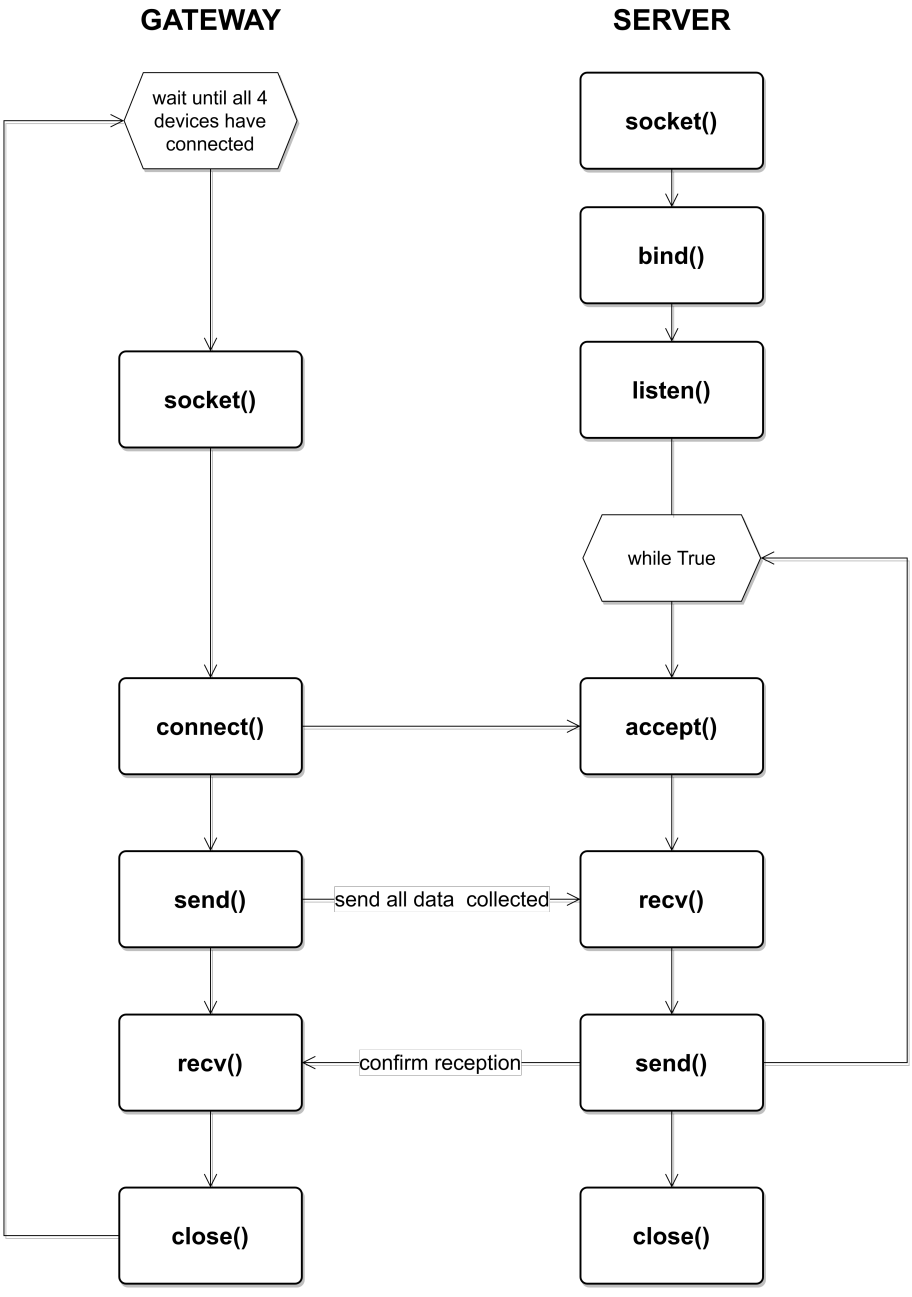

Figura 2.7: Schema rappresentante la comunicazione Gateway e Server

2.4 Dimensione dei buffer

Per il dimensionamento dei buffer abbiamo tenuto conto del fatto che per una migliore corrispondenza con l'hardware e le realtà di rete, la dimensione del buffer in bytes dovrebbe essere una potenza relativamente piccola di 2. In fase di costruzione dell'architettura di rete abbiamo inoltre stimato le dimensioni dei pacchetti scambiati dalle diverse entità. Viste queste considerazioni ecco le dimensioni che abbiamo assegnato ai buffer di ricezione:

- Per i **Device** in attesa di una risposta dal **DHCP** o dal **Gateway**, `1024 Bytes` risultano essere più che sufficienti;
- Per il **Gateway** in attesa di ricevere i dati dai device, tenuto conto che la dimensione di un pacchetto si aggira intorno ai `600-700 Bytes`, `1024 Bytes` sono sufficienti.
- Per il **Gateway** che attende la risposta da parte del **Server** `1024 Bytes` sono più che sufficienti;
- Per il **DHCP** che attende una richiesta di un `indirizzo IP`, `1024 Bytes` sono più che sufficienti.
- Per il **Server** che attende la ricezione delle misurazioni dei 4 dispositivi abbiamo assegnato una dimensione pari a `4096 Bytes` visto che la dimensione di un paccheto contente le misurazioni di un solo device si aggira intorno ai `600 Bytes` (Considerati i 4 device, sono ca. 2400, da cui `4096 Bytes` per renderla una potenza di 2)

Capitolo 3

Sviluppo

3.1 Metodologia di lavoro

3.1.1 GIT

Lavorando in coppia abbiamo scelto di utilizzare *Git* per condividere il lavoro ed il codice. Nonostante pratici con *branching* e *GitFlow*, in questo caso non abbiamo ritenuto necessario il loro utilizzo.

3.1.2 Visual Studio Live Share

Abbiamo deciso in diverse occassioni di approcciarci allo sviluppo ed alle fasi di programmazione seguendo quello che è il pair-programming, grazie all'utilizzo dell'estensione *Live Share* di *Visual Studio*. Questo ci ha permesso di adottare uno stile ibrido e doubly-checked, prevenendo possibili errori oppure incomprensioni causate da un approccio asincrono.

3.2 Note di sviluppo

Per la formattazione dell'output abbiamo utilizzato una libreria di Python, `beautifultable`, per la sua installazione basta lanciare il comand `pip install beautifultable`.
Per ulteriori informazioni si consulti il seguente link.
Mentre per l'invio effettivo attraverso le socket dei "pacchetti" abbiamo optato per la serializzazione con l'utilizzo del modulo pickle.

Appendice A

Guida utente

Per testare efficacemente la simulazione è necessario seguire i seguenti step:

- Lanciare il `Server` in un terminale dedicato con il comando:

  ```
  $ python IOT_server_TCP.py
  ```

- Lanciare il `Gateway` in un terminale dedicato con il comando:

  ```
  $ python IOT_gateway.py
  ```

- Lanciare i 4 `Device`, ciascuno in un terminale dedicato, con il comando:

  ```
  $ python IOT_device_UDP.py
  ```

  assegnare poi a ciascun `Device` un proprio *indirizzo MAC* quando richiesto da input.