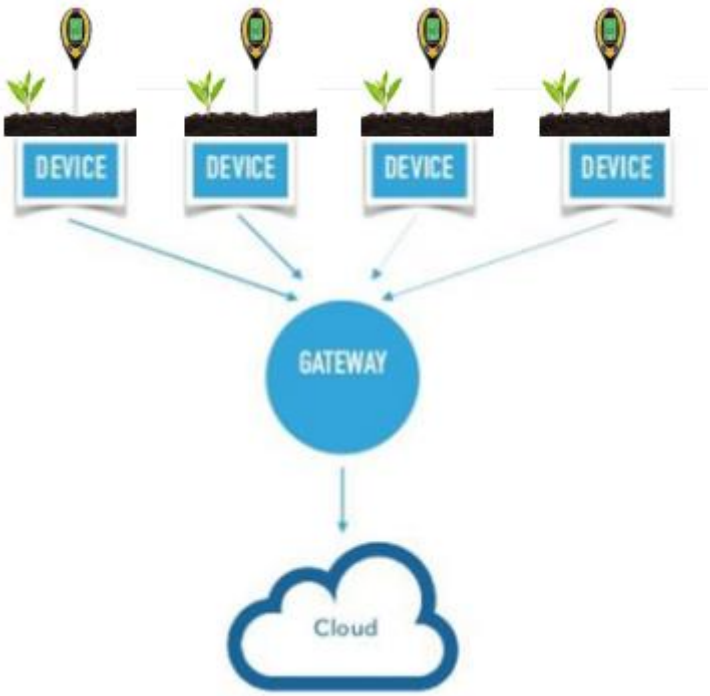
# PROGRAMMAZIONE DI RETI

Traccia 1 – Progetto IoT

Filippo Venturini

Matricola: 0000914966

2020/2021



# Istruzioni per l'avvio del sistema

Nel sistema progettato sono presenti 6 componenti a cui corrispondono 6 codici sorgenti Python, uno per ogni elemento.

Sono presenti:

- 1 Server
- 1 Gateway
- 4 Device

Per la corretta esecuzione dell'elaborato è necessario eseguire i vari codici nel seguente ordine:

- Avvio del server
- Avvio del gateway
- Avvio dei 4 device

Ad ogni avvio corrisponde un messaggio su console che certifica il funzionamento di ogni dispositivo.

# Funzionamento generale

Si procede ad una descrizione del funzionamento del sistema e delle scelte progettuali effettuate per la realizzazione dell'elaborato.

Per quanto riguarda la logica di funzionamento è stata rispettata la traccia, sono quindi presenti 4 device che simulano le misurazioni di temperatura e umidità e tramite il proprio indirizzo di rete privato instaurano una connessione di tipo UDP verso il gateway.

Il gateway è in ascolto su una porta in attesa della connessione di tutti e 4 i dispositivi (come richiesto nella traccia del progetto) e successivamente instaura con il server una connessione di tipo TCP tramite il proprio indirizzo pubblico, inoltrandogli un pacchetto unico contenente tutte e 4 le misurazioni che verrà poi mostrato sulla Console.

# Analisi dei componenti e strutture dati

## Device

Di seguito viene riportato il codice di un singolo device essendo gli altri tre identici ad esso, segue una descrizione delle scelte tecniche effettuate per la risoluzione del problema.

```
'''
------DEVICE 1-------

'''

import socket as sk
import time
import datetime
import random

device1_ip = "192.168.1.2"
device1_mac = "10:AF:CB:EF:19:CF"

gateway_ip = "192.168.1.1"
gateway_mac = "05:10:0A:CB:24:EF"
```

Il **device** contiene innanzitutto due indirizzi, il proprio *indirizzo IP* e il proprio *indirizzo MAC*. Contiene inoltre gli indirizzi del **gateway** (*IP* e *MAC*) a cui spedirà il pacchetto, queste coppie di indirizzi verranno utilizzate in seguito per la costruzione dell'*header ethernet* e dell'*header IP*.

```
gateway_port = 8100

gateway = ("localhost", gateway_port)
```

Si utilizza il *localhost* e la porta *8100* per identificare il **gateway** a cui il *socket* che verrà creato successivamente verrà connesso.

```
ethernet_header = device1_mac + gateway_mac
IP_header = device1_ip + gateway_ip

random.seed()
```

Vengono composti un *header IP* e un *header Ethernet* assemblando gli indirizzi *IP* e *MAC* di sorgente e destinazione e viene inizializzato un *seed* che verrà utilizzato in seguito per simulare casualmente le misurazioni effettuate dal device.

```
while True:
    try:
        device1Socket = sk.socket(sk.AF_INET, sk.SOCK_DGRAM)
        device1Socket.connect(gateway)
        device1_port = str(device1Socket.getsockname()[1])

        UDP_header = str(device1_port).zfill(5) + str(gateway_port).zfill(5)
```

Successivamente si entra in un ciclo infinito, in cui ad ogni iterazione viene creato un *socket* che utilizza indirizzi ipv4 ed è del tipo SOCK_DGRAM, utilizzato quindi per instaurare una connessione **UDP** con il **gateway**. Viene inoltre prelevata la porta assegnata in automatico al socket del device, sarà utile successivamente per costruire *l'header UDP*. Dato che la porta ha dimensione variabile, si utilizza l'istruzione *zfill(5)* per riempirla di 0 qualora fosse necessario, rendendo così la lunghezza di essa costante e pari a 5 caratteri.

```
        now = datetime.datetime.now()
        current_hour = str(now.hour) + ":" + str(now.minute) + ":" + str(now.second)

        message = current_hour +  " - " + str(random.randrange(0, 30)) +"°C - " + str(random.randrange(40, 60)) + "%"
        packet = ethernet_header + IP_header + UDP_header + message
```

In seguito si campiona l'ora attuale assumendola come orario della misurazione e si compone prima il messaggio (già nel formato richiesto dalla traccia) utilizzando valori casuali per temperatura e umidità, e infine il pacchetto aggiungendo *l'header* costituito dalle sorgenti e dalle destinazioni degli indirizzi MAC e IP e dalle porte.

```
        print("Source MAC address: " + device1_mac + ", destination MAC address: " + gateway_mac +
              "\nSource IP address: " + device1_ip + ", destination IP address: " + gateway_ip +
              "\nSource port: " + device1_port + ", destination port: " + str(gateway_port) + "\n")

        print ('Sending: "%s"\n' % message)

        start = time.time()
        device1Socket.sendto(packet.encode(), gateway)

        response, address = device1Socket.recvfrom(128)
        response = response.decode("utf-8")

        end = time.time()

        if(response != ""):
            print("UDP Trasmission time: ", end-start, " s.\n")
        else:
            print("Error in packet trasmission!")
```

Si procede codificando il pacchetto, inviandolo al **gateway** attraverso il socket **UDP** e attendendo la conferma di ricezione, il tempo totale impiegato viene campionato e stampato in output come **UDP Trasmission time**.

```
        device1Socket.close()
        time.sleep(30)
```

Viene poi chiusa la connessione del socket che sarà riaperta nel giorno successivo per la trasmissione delle nuove misurazioni.

Come ultima cosa viene effettuata una *sleep* di 30 secondi che ha lo scopo di simulare la durata di un giorno, dovendo i device connettersi al gateway solo una volta ogni 24 ore.

## Gateway

Il **gateway** è stato implementato considerando un numero variabile di dispositivi che viene memorizzato nella variabile *n_devices*.

```
"""
------GATEWAY-------

"""

import socket as sk
import time

n_devices = 4

gatewayUDPSocket = sk.socket(sk.AF_INET, sk.SOCK_DGRAM)

gateway_UDP_port = 8100
server_port = 8200

gatewayUDPSocket.bind(("localhost", gateway_UDP_port))

server = ("localhost", server_port)
```

Esso contiene due *socket*, il primo viene utilizzato per la connessione con i **device** utilizzando il protocollo **UDP** mentre il secondo ha lo scopo di interfacciarsi con il **server** utilizzando una connessione **TCP**. Il socket **UDP** viene subito messo in ascolto sulla porta 8100 pronto a ricevere le misurazioni dei device mentre quello **TCP** verrà creato in seguito.

```
gateway_private_ip = "192.168.1.1"
gateway_public_ip = "10.10.10.2"

gateway_mac = "05:10:0A:CB:24:EF"

server_ip = "10.10.10.1"
server_mac = "F3:04:73:EF:10:BF"
```

Il **gateway** possiede tre indirizzi: uno privato per la comunicazione con i device, uno pubblico che sarà il *source address* del pacchetto inviato al **server** e un indirizzo *MAC*. Sempre per quanto riguarda gli indirizzi esso ha inoltre il riferimento all'indirizzo *IP* e *MAC* del **server**.

Si procede a descrivere l'implementazione della logica della ricezione, della rielaborazione e dell'inoltro dei pacchetti con le misurazioni.

Vengono utilizzati due array per memorizzare rispettivamente gli indirizzi *IP* e i messaggi di tutti i device che si connettono al **gateway**, si effettua un ciclo finché nell'array degli *IP* non sono memorizzati tanti indirizzi quanti sono il numero di **device** attesi (settati inizialmente nella variabile *n_devices*).

```
arp_table = {}

connected_devices_ip = []
all_devices_data = []

while True:
    try:
        print('\n\rWaiting the devices connections...\n')
        message = ""

        while(len(connected_devices_ip) < n_devices):
            packet, address = gatewayUDPSocket.recvfrom(128)
            packet = packet.decode("utf-8")

            device_mac = packet[0:18]
            device_ip = packet[34:45]
            device_port = packet[56:61]
            device_data = packet [66:]

            UDP_header = str(gateway_UDP_port).zfill(5) + str(device_port).zfill(5)
            ethernet_header = gateway_mac + device_mac
            IP_header = gateway_private_ip + device_ip

            response_packet = ethernet_header + IP_header + UDP_header + "Packet received...\n"

            gatewayUDPSocket.sendto(bytes(response_packet,"utf-8"), ('localhost', int(device_port)))
```

All'interno del ciclo viene ricevuto e decodificato un pacchetto, si procede poi a prelevare *dall'header* l'indirizzo *MAC* e *IP* del device, la porta e il messaggio con le misurazioni. Si costruisce l'header del messaggio di risposta che viene inviato al device dal gateway attraverso il socket UDP.

```
            if(device_ip not in connected_devices_ip):
                arp_table[device_ip] = device_mac
                connected_devices_ip.append(device_ip)
                all_devices_data.append(device_data)
                print(device_ip + " connected")
```

Se l'indirizzo *IP* del **device** non è già presente nella lista di quelli connessi significa che è nuovo, quindi il suo indirizzo e il suo messaggio vengono memorizzati negli array. Viene inoltre implementata una *ARP table* utilizzando un dizionario, che mappa per ogni nuovo dispositivo che si connette al gateway il suo indirizzo *IP* all'indirizzo *MAC*.

```
        for i in range(n_devices):
            message += connected_devices_ip[i] + " - " + all_devices_data[i] + " \n"
```

Una volta connessi tutti e quattro i device viene fatto un ciclo che assembla il nuovo messaggio nel formato richiesto utilizzando gli *IP* e i messaggi memorizzati nei due array descritti precedentemente.

```
        gatewayTCPSocket = sk.socket(sk.AF_INET, sk.SOCK_STREAM)
        gatewayTCPSocket.connect(server)
        gateway_TCP_port = str(gatewayTCPSocket.getsockname()[1])

        ethernet_header = gateway_mac + server_mac
        IP_header = gateway_public_ip + server_ip
        TCP_header = str(gateway_TCP_port).zfill(5) + str(server_port).zfill(5)

        packet = ethernet_header + IP_header + TCP_header + message

        print('\n\rSending to the server...\n')
        start = time.time()
        gatewayTCPSocket.send(bytes(packet,"utf-8"))
```

In seguito il **gateway** si connette con il **server** attraverso un *socket TCP*, viene prelevata la porta assegnata in automatico e viene composto l'header con gli indirizzi IP, MAC e le porte rese a lunghezza fissa (con l'istruzione *zfill(5)*). L'header viene aggiunto al messaggio formando il pacchetto che viene codificato e inviato al **server.**

```
        response = gatewayTCPSocket.recv(128)
        response = response.decode("utf-8")
        end = time.time()

        if(response != ""):
            print("TCP Trasmission time: ", end-start, " s.\n")
        else:
            print("Error in packet trasmission!")

        gatewayTCPSocket.close()
        connected_devices_ip.clear()
        all_devices_data.clear()
```

Il gateway attende il pacchetto di conferma della ricezione e stampa il tempo impiegato per la trasmissione indicandolo come **TCP Trasmission time**.

Vengono infine resettati gli array che memorizzano gli IP e i messaggi dei device connessi e viene chiusa la connessione con il server che sarà riaperta quando saranno disponibili nuove misurazioni.

## Server

Il **server** utilizza per la ricezione del pacchetto da parte del **gateway** la porta *8200* e l'indirizzo di *localhost*. Contiene inoltre al proprio interno un socket che utilizza indirizzi ipv4, della famiglia SOCK_STREAM con lo scopo quindi di comunicare utilizzando **TCP**.

```
"""
SERVER
"""

import socket as sk
import time

serverTCPSocket = sk.socket(sk.AF_INET, sk.SOCK_STREAM)
server_address=('localhost', 8200)

server_ip = "10.10.10.1"
server_mac = "F3:04:73:EF:10:BF"
```

Il **server** è dotato di un indirizzo pubblico appartenente alla classe richiesta dal testo e un indirizzo *MAC*, che fungono da *destination address* per il **gateway**.

```
serverTCPSocket.bind(server_address)
serverTCPSocket.listen(1)
print ('The server is up on port:',8200)

while True:
    try:
        print ('Ready to serve...\n')
        connectionSocket, addr = serverTCPSocket.accept()

        packet = connectionSocket.recv(512)
        packet = packet.decode("utf-8")

        gateway_mac = packet [0:17]

        gateway_ip = packet [34:44]

        gateway_port = packet[54:59]

        data = packet[64:]

        print("Source MAC address: " + gateway_mac + ", destination MAC address: " + server_mac +
              "\nSource IP address: " + gateway_ip + ", destination IP address: " + server_ip +
              "\nSource port: " + gateway_port + ", destination port: " + str(server_port) +
              "\n\n" + data)
```

Successivamente ad essersi messo in ascolto sulla porta *8200* il **server** entra in un ciclo infinito. All'interno ad ogni iterazione accetta la connessione con il gateway, riceve il pacchetto con le misurazioni ed estrae: le misurazioni, l'indirizzo MAC, l'indirizzo IP e la porta del gateway. In seguito stampa a video le informazioni ricevute nel formato specificato dal testo.

```
ethernet_header = server_mac + gateway_mac
IP_header = server_ip + gateway_ip
TCP_header = str(server_port).zfill(5) + str(gateway_port).zfill(5)

response_packet = ethernet_header + IP_header + TCP_header + "Packet received...\n"

connectionSocket.send(bytes(response_packet,"utf-8"))
```

In conclusione il server procede a comporre l'header del pacchetto di conferma della ricezione e lo invia al gateway attraverso il socket della connessione.

# Considerazioni sulla dimensione dei buffer

I buffer utilizzati per la ricezione dei messaggi sono di due differenti dimensioni. Sia i buffer utilizzati per la ricezione dei pacchetti con le misurazioni nel gateway, che quelli utilizzati per la ricezione dei pacchetti di conferma, sono di 128 byte. Questo perché tale dimensione può permettere in una futura espansione (eventualmente più realistica) l'aggiunta di informazioni ulteriori nei messaggi inviati.

Il buffer di ricezione di tutte le misurazioni contenuto nel server è stato creato di 512 byte. Questo perché al suo interno conterrà sempre tutte le misurazioni dei device messe insieme dal gateway, di conseguenza necessita di una capienza 4 volte superiore a quella usata per ricevere una misurazione singola.