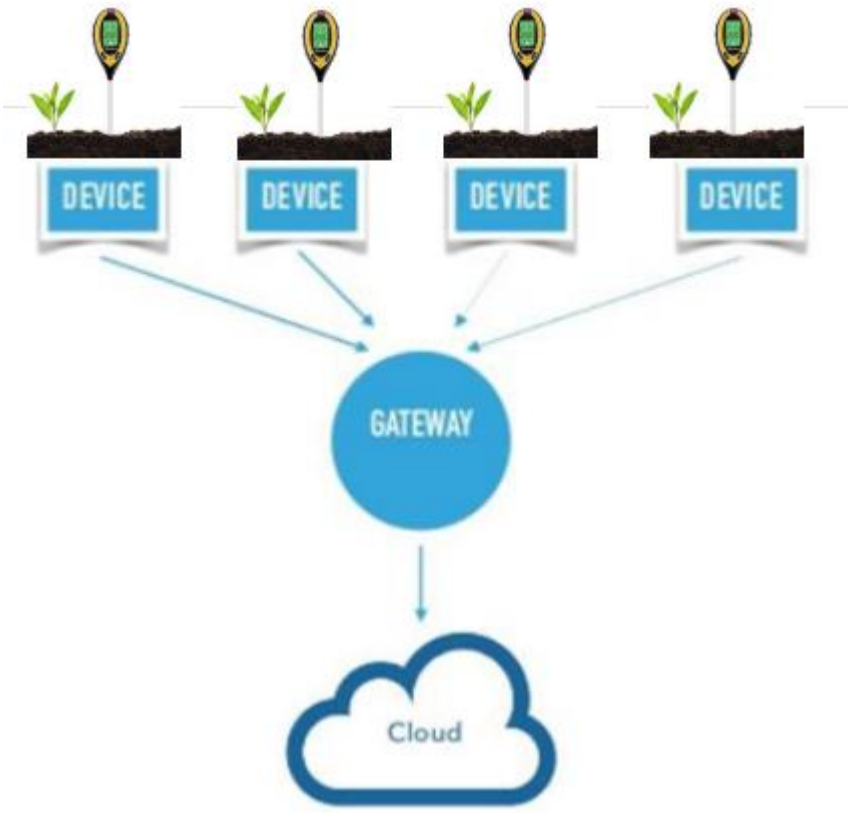
Relazione progetto Programmazione di Reti
Traccia 1

Malucelli Nicolò 0000914984



Sommario

Introduzione

Il progetto ha come obiettivo quello di simulare una rete in cui sono presenti quattro device, un gateway ed un cloud. I dispositivi effettuano misurazioni periodiche che inviano giornalmente al gateway. Una volta che il gateway possiede le misurazioni di tutti i dispositivi le invia al cloud che stampa a video tali informazioni.

Scelte di progetto

I dispositivi effettuano una misurazione "al giorno". Essendo questa una simulazione si è scelto di far durare un giorno 60 secondi, per cui ogni 60 secondi i dispositivi effettuano una misurazione e la spediscono al Gateway attraverso una connessione UDP. Prima di chiudere la connessione i dispositivi attendono il messaggio di avvenuto ricevimento da parte del Gateway.

Una volta che tutti i dispositivi hanno inviato le proprie misurazioni al Gateway, quest'ultimo apre una connessione TCP con il Cloud, che una volta ricevuto il pacchetto mostrerà le informazioni in esso contenute. Il Gateway non si limita a inoltrare i messaggi ricevuti ma li compatta in un unico pacchetto per una migliore ottimizzazione delle risorse di rete.

Analisi dettagliata delle componenti

Analisi dei Device

I 4 device hanno tutti lo stesso funzionamento, l'unica cosa che differisce tra essi sono l'indirizzo IP e l'indirizzo MAC:

	Indirizzo IP	Indirizzo MAC
Device 1	192.168.1.2	A4:67:3F:24:BF:63
Device 2	192.168.1.3	CF:18:56:92:13:CF
Device 3	192.168.1.4	56:24:48:54:06:CA
Device 4	192.168.1.5	3E:1E:C5:4C:A2:4B

Come richiesto dalla traccia del progetto i 4 dispositivi hanno un indirizzo IP di classe C del tipo 192.168.1.0/24, mentre l'indirizzo MAC è stato assegnato loro casualmente.

Avendo tutti i device lo stesso funzionamento sarà analizzato soltanto il codice di uno che varrà per tutti:

```
import socket as sk
import time
import dataCollector as dc

client_ip = "192.168.1.5"
gateway_ip = "192.168.1.1"

client_mac = "3E:1E:C5:4C:A2:4B"
gateway_mac = "AB:53:5E:2A:AA:C5"

client_port = ""
gateway_port = 8000

gateway = ("localhost", gateway_port)

data = []
```

Ogni device oltre a conoscere le proprie informazioni conosce anche gli indirizzi IP e MAC del gateway, nonché il numero di porta. Tali informazioni sono fondamentali per poter poi costruire i diversi header che compongono il pacchetto.

"data" è l'array contenente le misurazioni del singolo device ed inizialmente è vuoto.

All'interno del "while True" il dispositivo effettua prima di tutto una misurazione e stabilisce una connessione con il gateway.

```
try:
    while True:
        #data meameasurement
        print("measuring data")
        data.append((dc.getTime(),dc.getTemperature(),dc.getHumidity()))
        print("data measured, opening the connection with the gateway")
        #creating socket and connecting to the gateway
        socket = sk.socket(sk.AF_INET, sk.SOCK_DGRAM)
        socket.connect(gateway)
        client_port = str(socket.getsockname()[1])
        print("connection opened, sending data to gateway")
```

Il passo successivo consiste nel creare i tre header, ognuno contenente informazioni fondamentali per il corretto invio del pacchetto. Il messaggio viene poi assemblato concatenando agli header le misurazioni.

```
#creating headers
header_Ethernet = client_mac + gateway_mac
header_Ip = client_ip + gateway_ip
header_udp = str(client_port).zfill(5) + str(gateway_port).zfill(5)
message = header_Ethernet + header_Ip + header_udp

for measurement in data:
    message = message + measurement[0] + ";" + str(measurement[1])  + ";" + str(measurement[2]) + ";"
message = message[0:-1]
```

Una volta che il pacchetto è completo viene spedito al gateway. Prima dell'invio del pacchetto viene misurato il tempo, in modo da poter poi ricavare il tempo di trasmissione. Successivamente il device si pone in ascolto del pacchetto di risposta, che indica la corretta ricezione del pacchetto. Il **buffer** utilizzato per ricevere il messaggio di conferma è di **128 byte**. Una volta ricevuto il pacchetto di risposta, il dispositivo calcola il tempo di trasmissione facendo la differenza dei due tempi misurati precedentemente e chiude la connessione con il gateway.

La "sleep" finale di un minuto simula il trascorrere di un giorno.

```
start = time.time()
socket.sendto(bytes(message, "utf-8"), gateway) #send data to gateway
print("data sent, waiting for response message (128 bytes buffer)")

received_message, address = socket.recvfrom(128) #waiting for gateway response
received_message.decode()
if received_message[66:] == "received":
    print("response message received from gateway")

end = time.time()
print("transmission time: ", end - start, " seconds") #calculating transmission time

socket.close() #closing connection

print("connection closed. Wait a day to send data again\n\n\n")
time.sleep(60)
data.clear()
```

Analisi del Gateway

Similmente ai device, il gateway conosce gli indirizzi IP e MAC relativi al cloud, nonché la porta su cui questo è in ascolto. A differenza dei device, il gateway ha però due interfacce: la prima sulla stessa rete dei device è sempre in ascolto di un eventuale connessione mentre la seconda funge da client per la connessione con il Cloud ed è per questo sulla stessa rete di quest'ultimo.

	Indirizzo IP	Indirizzo MAC
Interfaccia lato device	192.168.1.1	AB:53:5E:2A:AA:C5
Interfaccia lato cloud	10.10.10.1	EF:32:32:AB:A1:C9

Il gateway contiene inoltre la “arp table” ed una “waiting list”. Quest’ultima è un dizionario che memorizza per ogni indirizzo IP dei device il messaggio ricevuto.

Nel caso dei device, il binding del socket avveniva il modo automatico una volta instaurata la connessione con il Gateway. Al contrario il Gateway deve effettuare il binding in maniera esplicita su un esatto numero di porta, altrimenti i dispositivi non saprebbero su quale porta inviare le proprie misurazioni. Questo vale solo per il socket che funge da server, ovvero quello che comunica con i device. Per quanto riguarda il socket che permette di comunicare con il cloud, esso opera in modo analogo a quello dei device e svolgendo il ruolo di client non necessita di binding manuale.

```
import socket as sk
import time

socket_lan = sk.socket(sk.AF_INET, sk.SOCK_DGRAM)
socket_cloud = ""

CLIENT_NUMBER = 4

gateway_lan_ip = '192.168.1.1'
gateway_cloudside_ip = '10.10.10.1'
cloud_ip = "10.10.10.2"

gateway_lanside_mac = "AB:53:5E:2A:AA:C5"
gateway_cloudside_mac = "EF:32:32:AB:A1:C9"
cloud_mac = "CF:4A:63:33:9B:44"

gateway_lanside_port = 8000
gateway_cloudside_port = ""
cloud_port = 8200

waiting_list = {}
arp_table = {}

socket_lan.bind(("localhost", gateway_lanside_port))
```

Il Gateway si mette in ascolto sul socket lato device utilizzando un **buffer** di **128 bytes**. Una volta instaurata la connessione e ricevuto un messaggio, il gateway legge gli header del pacchetto arrivato e compone gli header del pacchetto di risposta. L’istruzione “.zfill(5)” è importante poiché il numero di porta non ha lunghezza fissa ed una scorretta lettura porterebbe a leggere dati sbagliati.

“arp_table” e “waiting_list” sono aggiornate ad ogni nuovo messaggio ricevuto.

```
while True:
    print("Waiting for device connection... (128 bytes buffer)")
    received_message, address = socket_lan.recvfrom(128)
    print(len(received_message))
    received_message =  received_message.decode("utf-8")

    #reading headers
    header_Ethernet = received_message[0:34]
    header_Ip = received_message[34:56]
    header_Udp = received_message[56:66]
    device_ip = header_Ip[0:11]
    device_mac = header_Ethernet[0:17]
    device_port = header_Udp[0:5]

    #creating response headers
    header_Ethernet = gateway_lanside_mac + device_mac
    header_Ip = gateway_lan_ip + device_ip
    header_Udp = str(gateway_lanside_port).zfill(5) + str(device_port).zfill(5)

    print("received message from ", device_ip)
    print("sending response message to ", device_ip, "...\n")

    response = header_Ethernet + header_Ip + header_Udp + "received"
    socket_lan.sendto(bytes(response, "utf-8"), ("localhost", int(device_port)))

    #reading message
    message = received_message[66:]
    arp_table[device_ip] = device_mac
    waiting_list[device_ip] = message
```

Se tutti i device hanno inviato i loro dati, il Gateway si prepara a mandarli al Cloud. Come avviene per i device, anche il Gateway misura il tempo di trasmissione, misurando il tempo prima dell'invio del pacchetto e dopo aver ricevuto la conferma di corretta lettura di quest'ultimo da parte del Cloud. Il **buffer** utilizzato per la ricezione del messaggio di conferma è di **128 bytes**.

Una volta spediti i valori delle misurazioni, il gateway elimina i vecchi valori e si pone nuovamente in ascolto di nuove connessioni da parte dei device.

```
if len(waiting_list.keys()) == CLIENT_NUMBER:

    #creating socket to connect with cloud
    socket_cloud = sk.socket(sk.AF_INET, sk.SOCK_STREAM)
    print("opening connection with cloud...")
    socket_cloud.connect(("localhost", cloud_port))
    gateway_cloudside_port = str(socket_cloud.getsockname()[1])

    #creating message headers
    header_Ethernet = gateway_cloudside_mac + cloud_mac
    header_Ip = gateway_cloudside_ip + cloud_ip
    header_Tcp = str(gateway_cloudside_port).zfill(5) + str(cloud_port).zfill(5)
    cloud_message = header_Ethernet + header_Ip + header_Tcp

    #creating message
    for ip in waiting_list.keys():
        cloud_message = cloud_message + ip + ";" + waiting_list[ip] + " "
    cloud_message = cloud_message[0:-1]

    start = time.time()
    print("sending data to cloud...")
    socket_cloud.send(cloud_message.encode()) #send data to cloud
    print("data sent to cloud, waiting for cloud response (128 bytes buffer)")

    received_message = socket_cloud.recv(128).decode()
    if received_message[64:] == "received":
        print("response message received from cloud")

    end = time.time()
    print("transmission time: ", end - start, " seconds")

    print("closing connection with cloud...\n\n\n")

    socket_cloud.close()
    waiting_list = {}
```

Analisi del Cloud

A differenza di Device e Gateway, al Cloud non serve conoscere informazioni aggiuntive oltre alle proprie poiché svolge unicamente un ruolo da server.

	Indirizzo IP	Indirizzo MAC
Cloud	10.10.10.2	CF:4A:63:33:9B:44

```
import socket

cloud_ip = "10.10.10.2"
cloud_mac = "CF:4A:63:33:9B:44"
cloud_port = 8200

socket_cloud = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socket_cloud.bind(("localhost", cloud_port))
socket_cloud.listen(1)
print("Cloud online\n\n")
```

Il Cloud è sempre in attesa di una connessione da parte del Gateway. Una volta instaurata la connessione si prepara a ricevere i dati, utilizzando un **buffer** di **256 bytes**.

```
while True:
    print("Waiting for gateway connection...")
    conn, addr = socket_cloud.accept()
    print("Gateway connected, ready to receive data (256 bytes buffer)")
    message = conn.recv(256)
    print(len(message))
    message = message.decode()

    #reading headers
    header_Ethernet = message[0:34]
    header_Ip = message[34:54]
    header_Tcp = message[54:64]
    #reading message
    message = message[64:]
    #reading gateway info
    gateway_ip = header_Ip[0:10]
    gateway_mac = header_Ethernet[0:17]
    gateway_port = header_Tcp[0:5]

    print("Received message from ", gateway_ip)
```

Similmente a quanto accadeva tra Gateway e Device, il Cloud compone i nuovi header ed invia un messaggio di corretta ricezione dei dati. Una volta spedito il messaggio di risposta, il Cloud si appresta a chiudere la connessione con il Gateway e stampa i dati ricevuti.

```
#creating response header
header_Ethernet = cloud_mac + gateway_mac
header_Ip = cloud_ip + gateway_ip
header_Tcp = str(cloud_port).zfill(5) + str(gateway_port).zfill(5)

#creating response message
response = header_Ethernet + header_Ip + header_Tcp + "received"

print("Sending response message to ", gateway_ip, "...")
conn.send(bytes(response, "utf-8"))

print("Closing connection...\n")
conn.close()
```

Analisi DataCollector

Il file dataCollector.py contiene unicamente funzioni di utility ed è importato da ogni device. Le tre funzioni di utility restituiscono rispettivamente l'ora (in un formato ore:minuti:secondi), la temperatura e l'umidità. Queste ultime due funzioni restituiscono valori casuali all'interno di un range. In una applicazione alla realtà di questo progetto ovviamente tali valori non sarebbero casuali ma misurati attraverso appositi sensori.

```
from datetime import datetime
import random

def getTime():
    return datetime.now().strftime("%H:%M:%S")

def getTemperature():
    return random.randrange(16, 38, 1)

def getHumidity():
    return random.randrange(45, 80, 1)
```