# Elaborato per il corso di Programmazione di Reti A.A 2020/2021

Di Girolamo Alberto
alberto.digirolamo2@studio.unibo.it
0000914559

# Analisi e descrizione delle strutture dati utilizzate

**Server TCP**

Il server deve attendere nuove connessioni e stampare a console le informazioni ricevute. In seguito, deve poi mandare un messaggio di conferma di ricezione.

```
import socket

buffer_size = 512
server_ip = "10.10.10.1"
server_mac = "00:00:0A:BB:12:11"
server_port = 9100
```

Il codice del **server TCP** presenta in testa le sue caratteristiche tecniche . Come prima istruzione viene salvata la dimensione del *buffer* di ricezione. Viene scelto 512 byte poiché il server contiene tutte le misurazioni effettuate dai device. Questo valore dipenderà quindi dal numero di device, dal numero di rilevazioni che vengono effettuate e dalla quantità di dati raccolti da queste misurazioni.

Successivamente vengono impostati i rispettivi *indirizzi IP* e *MAC* seguiti dal numero della *porta* che il **server** userà per comunicare con il **gateway**.

```
server_TCP = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address=('localhost',server_port)
server_TCP.bind(server_address)
server_TCP.listen(1)

print('The Server is up on port: ',server_port)
```

Viene creato l'oggetto *socket* di tipologia *TCP* e si utilizza il *localhost* con la porta indicata precedentemente per identificare dove avverrà la connessione con il **gateway**. Successivamente si associa il *server_address* al socket creato in precedenza e lo si mette in ascolto di nuove connessioni.

```
while True:
    try:
        print ('Ready to serve...')
        connectionSocket, addr = server_TCP.accept()

        message = connectionSocket.recv(buffer_size)
        message = message.decode("utf-8")

        source_mac = message[0:17]
        source_ip = message[34:45]
        source_port = message[54 : 59]
        message = message[64 :]

        print('Size buffer of TCP transmission: ', buffer_size, '\n')
        print(message)
```

Quando si accetta una nuova connessione si indica la dimensione del *buffer* e successivamente si decodifica il messaggio.

Successivamente il messaggio viene suddiviso in tutte le sue parti che lo compongono poiché questo *socket* conterrà informazioni circa il mittente e il destinatario come l'indirizzo *MAC*, l'indirizzo *IP* e il numero di *porta* con la quale è avvenuta la connessione. In ultima posizione invece è presente il vero e proprio messaggio.

```
    header_mac = server_mac + source_mac
    header_ip  = server_ip  + source_ip
    TCP_header =  str(server_port).zfill(5) + str(source_port).zfill(5)

    responce_packet = header_mac + header_ip + TCP_header + 'Packet received..\n'
    connectionSocket.send(bytes(responce_packet, 'utf-8'))

except IOError:
    connectionSocket.close()
```

Per confermare la ricezione del messaggio viene creato un nuovo pacchetto con il rispettivo *header* che viene poi inviato al **gateway**.

**Gateway**

Il **gateway** si pone come intermediario tra il **server** i **client**. Ha lo scopo di attendere nuove connessioni attraverso una connessione *UDP* dai device e inoltrare i messaggi ricevuti al **server** attraverso una connessione *TCP*.

```
import socket
import time

server = (("localhost", 9100))

buffer_size = 128

n_device = 4
gateway_UDP_port = 9000

gateway_ip_device_interface = "192.168.1.1"
gateway_ip_server_interface = "10.10.10.1"
gateway_mac = "30:05:0A:EF:12:10"

server_ip = "10.10.10.1"
server_mac = "00:00:0A:BB:12:11"

gateway_UDP = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
gateway_UDP.bind(("localhost", gateway_UDP_port))

devices_ip = []
devices_data = []
arp_table = []
```

Le prime informazioni che troviamo nel **gateway** stabiliscono le caratteristiche di questo. Vengono impostati due differenti indirizzi *IP* a seconda dell'interfaccia a cui si riferiscono. Nel lato *client* l'indirizzo *IP* è del tipo *192.168.1.0/24*, mentre a lato *server* è del tipo *10.10.10.0/24.* Inoltre vengono salvati gli indirizzi *MAC* del **gateway** e del **server**.

Vengono stabilite due porte: una utilizzata per permettere la connessione dei **client** con il **gateway**, l'altra invece assieme all'indirizzo *localhost* viene utilizzata per connettere il **gateway** al **server**.

Poiché il **gateway** deve essere sempre disponibile per ricevere nuove connessioni da parte dei **device**, viene aperta una connessione *UDP* nella porta indicata.

La dimensione scelta del *buffer* è *128 byte*, poiché risulta sufficiente per mantenere le informazioni ottenute da un singolo device.

```
while True:

    try:
        print('\n\r waiting to devices connection...')
        message = ''

        while(len(devices_ip) < n_device):
            data, address = gateway_UDP.recvfrom(buffer_size)
            data = data.decode('utf8')
            source_mac = data[0: 17]
            source_ip  = data[34: 45]
            source_port = data[56 : 61]
            device_data = data[66:]

            UDP_header = str(gateway_UDP_port).zfill(5) + str(source_port).zfill(5)
            header_mac = gateway_mac + source_mac
            header_ip = gateway_ip_device_interface + source_ip

            if (source_ip not in devices_ip):
                response_packet = header_mac + header_ip + UDP_header + 'Packet received..\n'
                gateway_UDP.sendto(bytes(response_packet,"utf-8"), ('localhost', int(source_port)))
                devices_ip.append(source_ip)
                devices_data.append(device_data)
                print(source_ip + ' connected\n')
```

È presente una variabile (*n_device*) che indica il numero di **device** che devono stabilire una connessione e viene utilizzata per uscire dal ciclo.

Ogni qual volta che il **gateway** riceve un messaggio questo viene suddiviso in tutte le sue parti per estrapolare l'indirizzo *MAC* e *IP*, la *porta* utilizzata e il messaggio contenuto.

L'indirizzo *IP* viene utilizzato per controllare se un **device** è già connesso durante una sessione. Nel caso non lo sia, viene memorizzato il messaggio appena ricevuto e inviato un pacchetto di risposta che conferma l'avvenuta ricezione.

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

    TCP_header = str(gateway_TCP_port).zfill(5) + str(source_port).zfill(5)
    header_ip  = gateway_ip_server_interface + server_ip
    header_mac = gateway_mac + server_mac
    message = header_mac + header_ip + TCP_header

    for i in range(n_device):
        message += devices_data[i]

    start = time.time()

    gateway_TCP.sendto(bytes(message, "utf-8"), server)
    response = gateway_TCP.recv(buffer_size)
    response = response.decode("utf-8")

    end = time.time()
    print(response[65:] +"\nTCP Trasmission time: ", end-start, " s.\n")
    print('Size buffer of TCP and UDP trasmissions is: ', buffer_size, '\n')
    gateway_TCP.close()
    devices_ip.clear()
    devices_data.clear()

except IOError:
    gateway_UDP.close()
    gateway_TCP.close()
```

Quando tutti i **device** riescono a inviare il proprio messaggio al gateway questo esce da ciclo e stabilisce una connessione *TCP* con il **server**.

Viene formattato il pacchetto per la spedizione impostandogli un *header* formato dalle informazioni sia del sorgente (**gateway**) che del destinatario (**server**) circa lindirizzo *MAC*, *IP* e la *porta* utilizzata.

Per tenere traccia del tempo di trasmissione del pacchetto si avvia un timer che verrà poi fermato quando si riceverà un messaggio di risposta da parte del **server** di avvenuta ricezione.

Infine si chiudono le connessioni *TCP* verso il **server** e si svuotano gli array contenenti le informazioni di connessione della sessione attuale.

**Device**

A seguire viene riportato la descrizione di un singolo device poiché gli ulteriori tre sono identici.

Il **device** ha lo scopo di rilevare i dati (temperatura e umidità del terreno e orario della rilevazione) e di inviarli al **gateway** attraverso una connessione *UDP* attendendo successivamente un messaggio di conferma.

```python
import socket
import time
import datetime
import random as rn

def random_time(i):
    date = datetime.datetime.now()
    return f"{date.hour}:{date.minute}:{date.second}"
def random_temperature():
    t = rn.randint(20, 35)
    return f"{t}°C"
def random_humidity():
    u = rn.randint(25, 40)
    return f"{u}%"
```

Come prime istruzioni sono riportati tre semplici metodi per simulare la realtà. Riguardo la temperatura e l'umidità, questi generano due numeri random in un range preimpostato, mentre per l'orario della rilevazione viene utilizzato l'orario attuale del sistema.

```python
buffer_size = 128

device1_ip = "192.168.1.2"
device1_mac = "31:05:0B:EF:19:01"

gateway_ip = "192.168.1.1"
gateway_mac = "30:05:0A:EF:12:10"

header_ip  = device1_ip + gateway_ip
header_mac = device1_mac + gateway_mac

gateway_port = 9000
gateway = ("localhost",gateway_port)

number_measurements = 2
```

In base alla lunghezza del messaggio da inviare al **gateway** vi è scelto di utilizzare una dimensione di *128 byte* come dimensione del *buffer*. Se in futuro i **device** dovranno inviare più informazioni allora bisognerà adattare questo valore. Di conseguenza anche la dimensione del *buffer* del **gateway** andrà modificata e in proporzione anche quella del **server**.

Ad ogni **device** vengono assegnati gli indirizzi *MAC* e *IP* propri e del dispositivo con cui si devono connettere (**Gateway**). Vengono inizializzate le variabili che saranno poi utilizzate per costruire l' *header* del pacchetto. Successivamente viene stabilita la porta con la quale il **device** si deve

interfacciare per comunicare con il **gateway** associata a *localhost.* Di seguito viene stabilito il numero di misurazioni che ogni device deve compiere nell'arco di "una giornata".

```
while True:
    try:
        message = ''
        for i in range(number_measurements):
            time.sleep(24/number_measurements)
            msg = f"{device1_ip}-{random_time(i)}-{random_temperature()}-{random_humidity()}\n"
            message += msg
            print(msg)

        device1 = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        device1.connect(gateway)
        device1_port = str(device1.getsockname()[1])
        UDP_header = str(device1_port).zfill(5) + str(gateway_port).zfill(5)
        message = header_mac + header_ip + UDP_header + message
```

Il ciclo *while* permette di iterare (indefinitamente) il procedimento di seguito descritto simulando il funzionamento di un dispositivo IoT.

In questo codice si è scelto di simulare un giorno in 24 secondi, di conseguenza utilizzando una proporzione con il numero di ore e il numero di rilevamenti da effettuare, ogni misurazione viene presa sempre dopo uno stesso lasso di tempo dalla precedente.

Ogni volta che si ottiene una misurazione, viene formattato il messaggio seguendo questa logica: "IP_device"-"orario_rilevazione"-"temepratura"-"umidità"

Il **device** avvia una connessione *UDP* verso il **gateway** e salva il numero di *porta* utilizzato. Successivamente viene formattato il pacchetto per la comunicazione aggiungendo *l'header* formato dal numero di *porta* e dall'indirizzo *MAC* e *IP* sia del **device** che del destinatario (**gateway**).

```
        start = time.time()
        device1.sendto(message.encode(),gateway)
        response, address = device1.recvfrom(buffer_size)
        response = response.decode("utf-8")

        end = time.time()

        print(response[66:] +"\nUDP trasmission time: ", end-start, " s.\n")
        print('Size buffer of UDP trasmission is ', buffer_size, '\n')
        device1.close()
    except IOError:
        device1.close()
```

Viene fatto partire un timer per misurare il tempo totale della connessione. Successivamente viene inviato il messaggio al **gateway**, si attende un messaggio di risposta, lo si decodifica e si ferma il timer.

A console viene poi mostrato un messaggio con il tempo impiegato e la dimensione del *buffer*

Infine, si chiude la connessione e si ripete il ciclo.