

Smart Meter IoT

Relazione del progetto di Programmazione di
Reti

A.A. 2020-2021

Lucia Castellucci (MATRICOLA 915056)

11 giugno 2021

Indice

1	Introduzione	2
2	Descrizione	4
2.1	Devices	4
2.2	Gateway	7
2.3	Cloud	9
3	Funzionamento	12
4	Librerie utilizzate	18

Capitolo 1

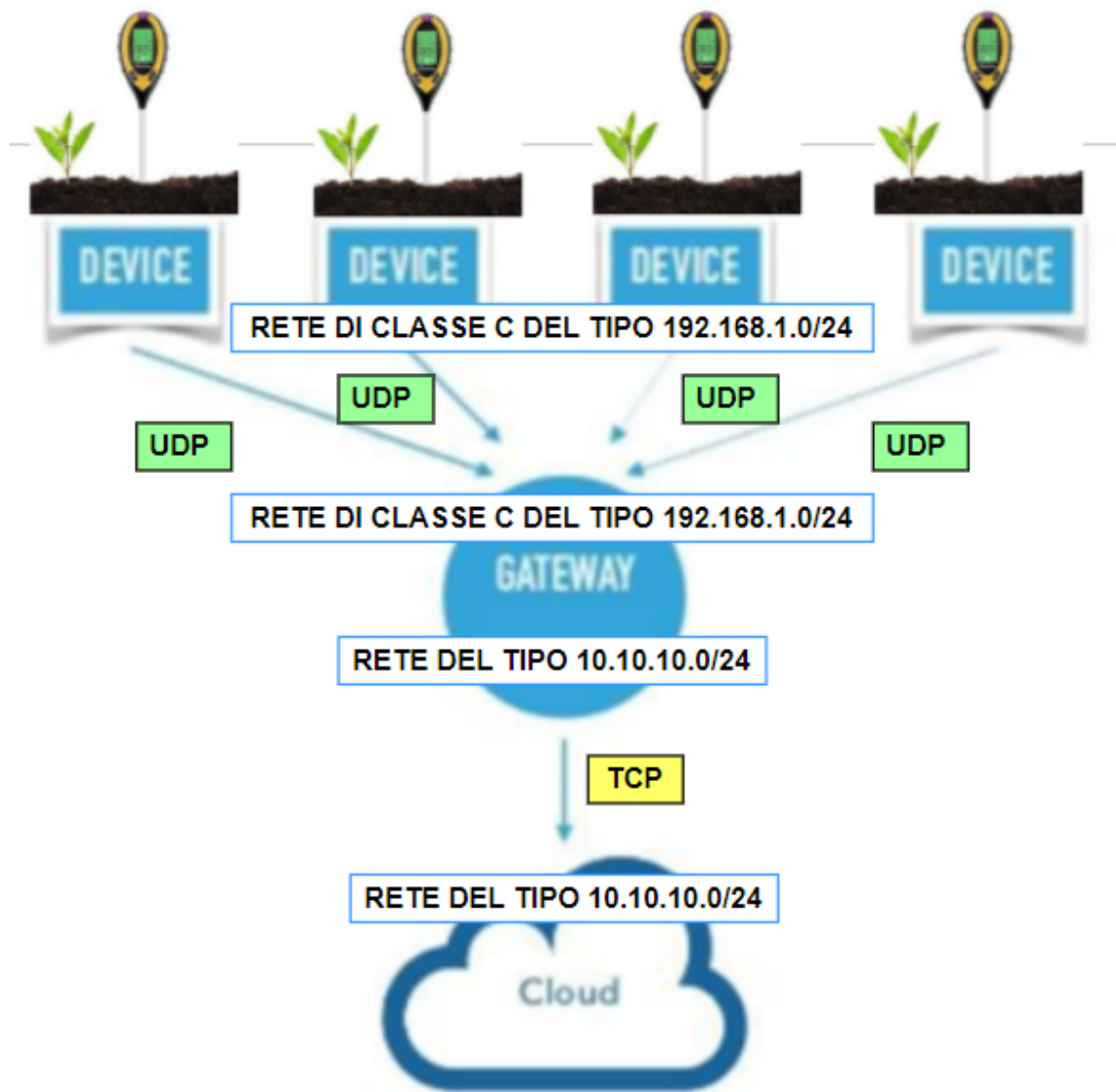
Introduzione

Il progetto d'esame consiste in un sistema, scritto in linguaggio Python, che gestisce uno scenario di Smart Meter IoT che rilevano la temperatura e umidità del terreno in cui sono posizionati. I 4 dispositivi sono indicati come Device. Questi dispositivi si collegano 1 volta al giorno con una connessione UDP verso il Gateway. Tramite questa connessione i dispositivi inviano le misure che hanno raccolto durante le 24 ore precedenti. Le misure consistono nel rilevare l'ora della misura e il dato di temperatura e umidità. Una volta che i pacchetti di tutti i dispositivi sono arrivati al Gateway, il gateway instaura una connessione TCP verso un server centrale dove i valori vengono visualizzati sulla console del server nel seguente modo:

Ip-address-device-1 – ora misura – valore temperatura – valore umidità

I 4 Dispositivi IoT hanno un indirizzamento appartenente ad una rete di Classe C del tipo 192.168.1.0/24. Il Gateway ha due interfacce di rete: quella verso i dispositivi il cui IP Address appartiene alla stessa network dei dispositivi mentre l'interfaccia che parla con il server ha indirizzo ip appartenente alla classe 10.10.10.0/24, classe a cui appartiene anche l'IP address del server centrale. Vengono dunque simulate le connessioni UDP dei device verso il Gateway e la connessione TCP del Gateway verso il Server mostrando sulla Console del server la lista dei messaggi ricevuti nel formato indicato sopra.

La struttura della rete può essere riassunta con il seguente schema:



Capitolo 2

Descrizione

Le entità principali del sistema possono essere individuate come:

2.1 Devices

I devices funzionano con il criterio del UDP Socket Client. Hanno il compito di eseguire le rilevazioni delle 24 ore precedenti e di inviarle, una volta al giorno, al Gateway. All'interno della rete le rilevazioni vengono simulate grazie ad un file per ogni device, contenente le rilevazioni della giornata precedente, già formattate correttamente. Per rendere il codice meno ripetitivo, e per permetterne l'estendibilità, ho delegato ad un "General Device" la gestione delle mansioni comuni a tutti i device, ossia:

- **readingSurveys:** alla funzione vengono passati come parametro l'indirizzo IP del client e il nome del file. Con quest'ultimo, come prima istruzione, viene individuato il file corretto delle rilevazioni del device, e viene aperto. Successivamente, vengono lette tutte le rilevazioni dal file, a meno che il file non sia vuoto, e in quel caso il messaggio resterà """. Le rilevazioni man mano che vengono lette vengono inserite all'interno del messaggio con il formato: client-IP - rilevazione. Non appena si termina la lettura il file viene chiuso e la funzione ritorna al chiamante il messaggio, ossia l'insieme di tutte le rilevazioni del device in questione. Nello specifico:

```

#Fuction which read surveys from file contained in the directory Surveys
def readingSurveys (file_name, client_IP):

    # Searching for file path
    filePath = "Surveys/" + file_name
    # Opening file witch contains surveys
    file = open(filePath, "r")
    print("Reading the detections from file...")
    # Waiting for the reading
    time.sleep(1)
    message = ""

    while True:
        # Reading line
        line = file.readline()
        # If the file is over or the file is empty
        if (line == ""):
            # Let's stop the searching of surveys
            break;
        # If line is not empty
        else:
            # Let's format the message
            message = message + client_IP + " " + line + "\n"

    # Closing the file
    file.close();
    print("Surveys readed!")

    return message

```

- **connectionToServerUDP**: alla funzione vengono passati come parametro il messaggio (le rilevazioni), l'indirizzo del server, e la dimensione del buffer. Questa funzione implementa il funzionamento di un UDP Socket Client, per cui: crea il socket UDP per l'invio della richiesta e lo associa alla variabile socket. Poi, siccome alla funzione è già stata passata la tupla del server address (che contiene l'indirizzo IP del nostro server e la porta su cui il server è in ascolto), il Client (device) può tentare l'invio delle rilevazioni codificate (message passato come parametro) al Server (gateway), ma solo dopo essersi salvato in una variabile il momento in cui è stato provato l'invio. Successivamente si mette in attesa di una risposta da parte del server. Se il server risponde, per cui era attivo e in ascolto sulla porta corretta, il messaggio di risposta viene stampato e la socket viene chiusa. Se, invece, le surveys non sono arrivate correttamente al Gateway, allora viene stampato il messaggio di errore corrispondente, e la porta viene chiusa comunque.

```

# Function which connect the client (device) with the server (gateway)
# With an UDP connection
def connectionToServerUDP (message, server_address, buffer_size):
    # UDP Server Pocket
    # Creating the object SOCKET
    socket = sk.socket(sk.AF_INET, sk.SOCK_DGRAM)
    # Sending surveys to Gateway
    try:
        print("Sending surveys to Gateway... ")
        # Waiting 2 second for sending the request
        time.sleep(2)
        # Taking note about the current time for calculate the total_time
        time_zero = time.time()
        # Sending the request
        socket.sendto(message.encode(), server_address)
        print('... Waiting to receive the response from the Gateway...')
        # Receiving the response from server
        data, server = socket.recvfrom(buffer_size)
        # Taking note about the current time for calculate the total_time
        time_n = time.time()
        # Calculating total time
        total_time = time_n - time_zero
        # Waiting 2 seconds for printing the OK message
        time.sleep(2)
        print ('Received message! The message contains: "%s" % data.decode('utf8'))
        print("UDP message's sending time {} and the size of used buffer is {}".format(total_time, buffer_size))

    except Exception as info:
        print(info)

    finally:
        print("\nClosing socket\n")
        socket.close()

```

A questo punto ai singoli device resta solo il compito di importare il GeneralDevice e di chiamare le funzioni di raccolta delle rilevazioni e di connessione al server UDP (gateway). Le funzioni verranno lanciate facendo attenzione a impostare correttamente le variabili che verranno passate come parametro alle funzioni, quali indirizzo IP del Client, tupla del server address, dimensione del buffer, e nome del file corrispondente alle rilevazioni del singolo Device. La porzione di codice sotto riportata è stata estrapolata dal Device1, ma è identica a quella degli altri Devices, fatta eccezione per il nome del file, che acquisirà lo stesso numero identificativo del dispositivo corrispondente.

```

import GeneralDevice as device

# Taking note about Device informations
# Device IP
client_IP = "192.168.1.2"
# Gateway address
server_address = ("localhost", 10005)
# Buffer size
buffer_size = 4096
# Surveys file name
file_name = "Surveys1.txt"
# Reading surveys from file which contains surveys taken by Device 1
message = device.readingSurveys(file_name, client_IP)
# Connecting the client (device) with the server (gateway)
device.connectionToServerUDP(message, server_address, buffer_size)

```

2.2 Gateway

Il Gateway ha due interfacce di rete. Una delle due appartiene alla stessa rete dei device, e funziona con il criterio del UDP Socket Server. Essa ha il compito di prelevare le rilevazioni da tutti e 4 i device. L'altra, appartiene alla stessa rete del cloud, e funge da TCP Socket Client. Quest'ultima ha il compito di connettersi col Cloud e di trasmettergli i dati delle rilevazioni.

Vediamo dunque nel dettaglio le funzioni del Gateway:

- **connectionToClientUDP**: alla funzione vengono passati come parametro il messaggio (le rilevazioni), l'indirizzo del server, e la dimensione del buffer. Questa funzione implementa il funzionamento di un UDP Socket Server, per cui: crea il socket UDP per l'invio della richiesta dentro la variabile socket, che poi associa alla tupla server address passata come parametro. A questo punto, tante volte quante il numero dei dispositivi, il server si mette in ascolto. Non appena un client invia un messaggio al server, viene stampata una stringa che comunica che sono stati ricevuti dei dati da un Device. La rilevazione appena ricevuta viene aggiunta al messaggio (previa decodifica) e poi stampata. Infine, viene mandato dal gateway al device un messaggio, codificato, che comunica il corretto ricevimento delle rilevazioni. La porta viene dunque chiusa e la funzione ritorna il messaggio ottenuto al termine del ricevimento della rilevazione.

```
# Fuction which moves surveys from devices to gateway
def connectionToClientUDP (message, server_address, buffer_size):

    # UDP Server Pocket
    # Creating the object SOCKET
    socket_devices = sk.socket(sk.AF_INET, sk.SOCK_DGRAM)
    print ('\nStarting up on %s port %s' % server_address)
    # Associating the socket with the server_address
    socket_devices.bind(server_address)

    for i in range(4):
        # Waiting for receiving message
        print('\nWaiting to receive message...')
        data, address = socket_devices.recvfrom(4096)
        # Surveys received
        print('Received %s bytes from a Device' % (len(data)))
        message = message + data.decode('utf-8') + "\n"
        # Printing surveys
        print(message)
        # Waiting 2 seconds for reply
        time.sleep(2)
        # Reply with OK message to Device
        sent = socket_devices.sendto("Surveys arrived".encode(), address)
        print ('Sent %s bytes back to Device' % (sent))

    print ('\nClosing socket\n')
    socket_devices.close()
    return message
```


- **connectionToServerTCP**: alla funzione vengono passati come parametro il messaggio (le rilevazioni), l'indirizzo del server, e la dimensione del buffer. Questa funzione implementa il funzionamento di un TCP Socket Client, per cui: crea il socket UDP per l'invio della richiesta e lo associa alla variabile socket-cloud. Differentemente rispetto ad una connessione UDP, non è possibile per il Client (Gateway), tentare l'invio delle rilevazioni codificate (message passato come parametro) al Server (Cloud), prima di stabilire una connessione col server. Dunque, come primo step, sarà necessario tentare una connessione al server. Se questa riesce, dopo aver salvato l'istante in cui si è instaurata la connessione, si potrà con tutta sicurezza inviare il messaggio codificato, e aspettare la risposta del server, per poi stampare il messaggio di corretto ricevimento delle rilevazioni e calcolare il tempo di invio del messaggio. L'ultimo step sarà ovviamente la chiusura della connessione. Se invece la connessione non dovesse riuscire, verrà lanciata un'eccezione, verrà chiusa la porta, e l'applicazione verrà istantaneamente terminata.

```
# Fuction which moves surveys from gateway to cloud
def connectionToServerTCP (message, server_address, buffer_size):

    # TCP Server Pocket
    # Creating the object SOCKET
    socket_cloud = sk.socket(sk.AF_INET, sk.SOCK_STREAM)

    # Connecting to Cloud
    try:
        socket_cloud.connect(server_address)
        print("Connecting with Cloud... ")
        print("... Sending surveys to Cloud... ")
        #Waiting 2 second for sending the request
        time.sleep(2)
        #Taking note about the current time for calculate the total_time
        time_zero = time.time()

    # If connection failed printing an error message
    except Exception as data:
        print (Exception,":",data)
        print ("Connection failed, try again.\r\n")
        sys.exit(0)

    # Sending surveys to cloud
    socket_cloud.send(message.encode())
    # Print message
    print(message.encode())
    # Waiting for a response from the cloud
    print("Waiting the server's response...")
    response = socket_cloud.recv(buffer_size)
    # Taking note about the current time for calculate the total_time
    time_n = time.time()
    # Calculating the total time for sending surveys from gateway to cloud
    total_time = time_n - time_zero
    print("Received Message: {}".format(response.decode("utf8")))
    print("TCP message's sending time {} and the size of used buffer is {}".format(total_time, buffer_size))
    print("\nClosing connection\n")
    socket_cloud.close()
```

A questo punto al Gateway resta solo il compito di chiamare la funzione di connessione ai client (devices) per il ricevimento delle rilevazioni e la funzione di connessione al server (cloud) per l'invio delle rilevazioni, previa impostazione delle variabili che verranno passate come parametro alle funzioni, quali tuple dei server address (Gateway, come server UDP per i devices e Cloud, come server TCP per il Gateway) e dimensione del buffer.

```
# Information about the gateway in each interfaces
server_gateway_address = ('localhost', 10005)
buffer_size = 4096
message = ""
# Moving surveys from devices to gateway
message = connectionToClientUDP(message, server_gateway_address, buffer_size)
server_cloud_address = ('localhost', 8001)
# Moving surveys from gateway to cloud
connectionToServerTCP(message, server_cloud_address, buffer_size)
```

2.3 Cloud

Il Cloud funziona con il criterio del TCP Socket Server, e ha il compito di connettersi col Gateway per poter ricevere tutte le rilevazioni della giornata precedente porvenienti dai device. Per cui l'unica funzione che implementa è:

- **connectionToClientTCP**: la funzione prende in ingresso l'indirizzo del server e la dimensione del buffer. Siccome implementa il funzionamento di un TCP Socket Server: crea il socket TCP per l'invio della richiesta dentro la variabile socket, che poi associa alla tupla server address passata come parametro. A questo punto, il server si mette in ascolto, e se trova una richiesta di connessione da parte del client, la accetta. Ora che il server è connesso col client, tenta la ricezione delle rilevazioni. Se le rilevazioni arrivano correttamente, vengono stampate con un messaggio che informi della ricezione delle stesse, e la connessione viene chiusa. Diversamente, se c'è stato un errore nella ricezione, questo viene stampato e la connessione viene chiusa.

```

def connectionToClientTCP(server_address, buffer_size):

    # TCP Server Pocket
    # Creating the object SOCKET
    serverSocket = sk.socket(sk.AF_INET, sk.SOCK_STREAM)
    # Associating the socket with the server_address
    serverSocket.bind(server_address)

    # Define the backlog queue
    serverSocket.listen(1)

    # The socket is enabled to receive connections
    print('Cloud server is ready to serve...')
    # As soon as the connection is up, it returns a tuple
    connectionGateway, addr = serverSocket.accept()
    print('The Gateway is connected!')

    try:
        # Receiving surveys from socket
        print('... And it is uploading the surveys ...')
        message = connectionGateway.recv(buffer_size)
        # Printing surveys
        print(message.decode("utf8"))
        # Reply with OK message to Gateway
        print("Surveys received!")
        connectionGateway.send("Surveys received".encode())
        # Closing connections with Gateway
        print("\nClosing connection\n")
        connectionGateway.close()

    except IOError:
        # Sending error message for file not found
        connectionGateway.send(bytes("An error is occurred", "UTF-8"))
        print("\nClosing connection closed\n")
        connectionGateway.close()

```

A questo punto al cloud resta solo il compito di chiamare la funzione di connessione ai client (gateway) per il ricevimento delle rilevazioni, previa impostazione delle variabili che verranno passate come parametro alle funzioni, quali tuple del server address (del cloud) e dimensione del buffer.

```

def connectionToClientTCP(server_address, buffer_size):

    # TCP Server Pocket
    # Creating the object SOCKET
    serverSocket = sk.socket(sk.AF_INET, sk.SOCK_STREAM)
    # Associating the socket with the server_address
    serverSocket.bind(server_address)

    # Define the backlog queue
    serverSocket.listen(1)

    # The socket is enabled to receive connections
    print('Cloud server is ready to serve...')
    # As soon as the connection is up, it returns a tuple
    connectionGateway, addr = serverSocket.accept()
    print('The Gateway is connected!')

    try:
        # Receiving surveys from socket
        print('... And it is uploading the surveys ...')
        message = connectionGateway.recv(buffer_size)
        # Printing surveys
        print(message.decode("utf8"))
        # Reply with OK message to Gateway
        print("Surveys received!")
        connectionGateway.send("Surveys received".encode())
        # Closing connections with Gateway
        print("\nClosing connection\n")
        connectionGateway.close()

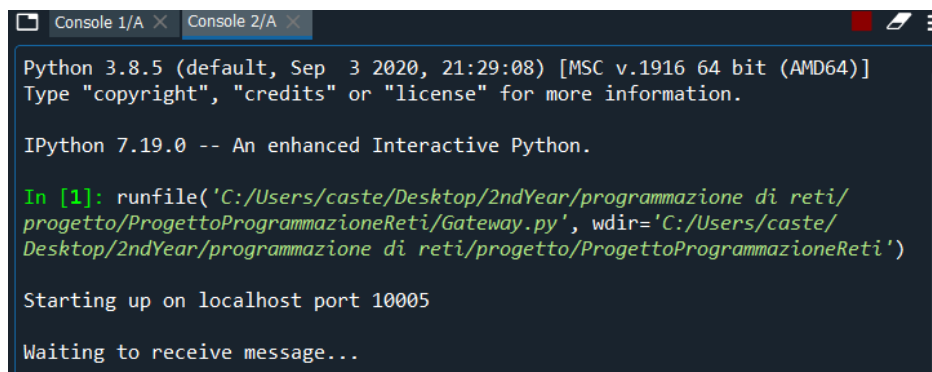
    except IOError:
        # Sending error message for file not found
        connectionGateway.send(bytes("An error is occurred", "UTF-8"))
        print("\nClosing connection closed\n")
        connectionGateway.close()

```

Capitolo 3

Funzionamento

Prima di tutto viene lanciato il Gateway, che si metterà in attesa di ricevere messaggi dai Device:



```
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

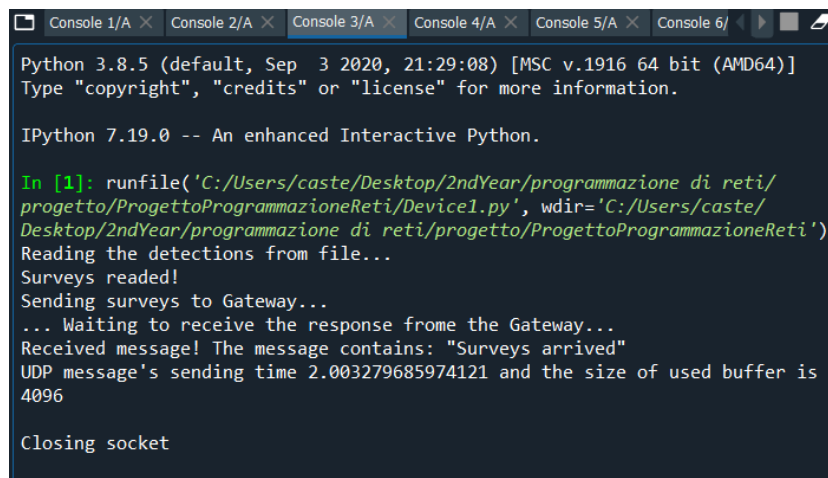
IPython 7.19.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/caste/Desktop/2ndYear/programmazione di reti/
progetto/ProgettoProgrammazioneReti/Gateway.py', wdir='C:/Users/caste/
Desktop/2ndYear/programmazione di reti/progetto/ProgettoProgrammazioneReti')

Starting up on localhost port 10005

Waiting to receive message...
```

Successivamente vengono lanciati i Devices, che inviano immediatamente il messaggio e chiudono la connessione, indicando il tempo che è stato impiegato nell'invio del messaggio e la dimensione del buffer:



```
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.19.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/caste/Desktop/2ndYear/programmazione di reti/
progetto/ProgettoProgrammazioneReti/Device1.py', wdir='C:/Users/caste/
Desktop/2ndYear/programmazione di reti/progetto/ProgettoProgrammazioneReti')
Reading the detections from file...
Surveys readed!
Sending surveys to Gateway...
... Waiting to receive the response from the Gateway...
Received message! The message contains: "Surveys arrived"
UDP message's sending time 2.003279685974121 and the size of used buffer is
4096

Closing socket
```

```
Console 1/A X Console 2/A X Console 3/A X Console 4/A X Console 5/A X Console 6/ X
Python 3.8.5 (default, Sep 3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.19.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/caste/Desktop/2ndYear/programmazione di reti/
progetto/ProgettoProgrammazioneReti/Device2.py', wdir='C:/Users/caste/
Desktop/2ndYear/programmazione di reti/progetto/ProgettoProgrammazioneReti')
Reading the detections from file...
Surveys readed!
Sending surveys to Gateway...
... Waiting to receive the response frome the Gateway...
Received message! The message contains: "Surveys arrived"
UDP message's sending time 2.01224946975708 and the size of used buffer is
4096

Closing socket
```

```
Console 1/A X Console 2/A X Console 3/A X Console 4/A X Console 5/A X Console 6/ X
Python 3.8.5 (default, Sep 3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.19.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/caste/Desktop/2ndYear/programmazione di reti/
progetto/ProgettoProgrammazioneReti/Device3.py', wdir='C:/Users/caste/
Desktop/2ndYear/programmazione di reti/progetto/ProgettoProgrammazioneReti')
Reading the detections from file...
Surveys readed!
Sending surveys to Gateway...
... Waiting to receive the response frome the Gateway...
Received message! The message contains: "Surveys arrived"
UDP message's sending time 2.0108277797698975 and the size of used buffer is
4096

Closing socket
```

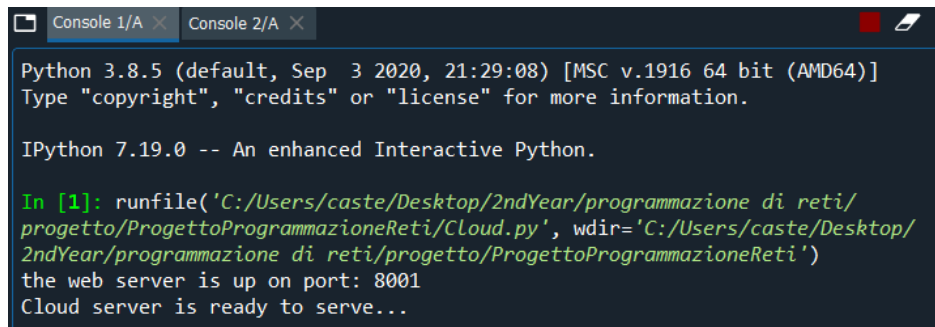
```
file 1/A X Console 2/A X Console 3/A X Console 4/A X Console 5/A X Console 6/A X
Python 3.8.5 (default, Sep 3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.19.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/caste/Desktop/2ndYear/programmazione di reti/
progetto/ProgettoProgrammazioneReti/Device4.py', wdir='C:/Users/caste/
Desktop/2ndYear/programmazione di reti/progetto/ProgettoProgrammazioneReti')
Reading the detections from file...
Surveys readed!
Sending surveys to Gateway...
... Waiting to receive the response frome the Gateway...
Received message! The message contains: "Surveys arrived"
UDP message's sending time 2.0082740783691406 and the size of used buffer is
4096

Closing socket
```

Simultaneamente viene aperta la connessione TCP del Cloud, che sarà in attesa che il Gateway provi a connettersi ad esso:

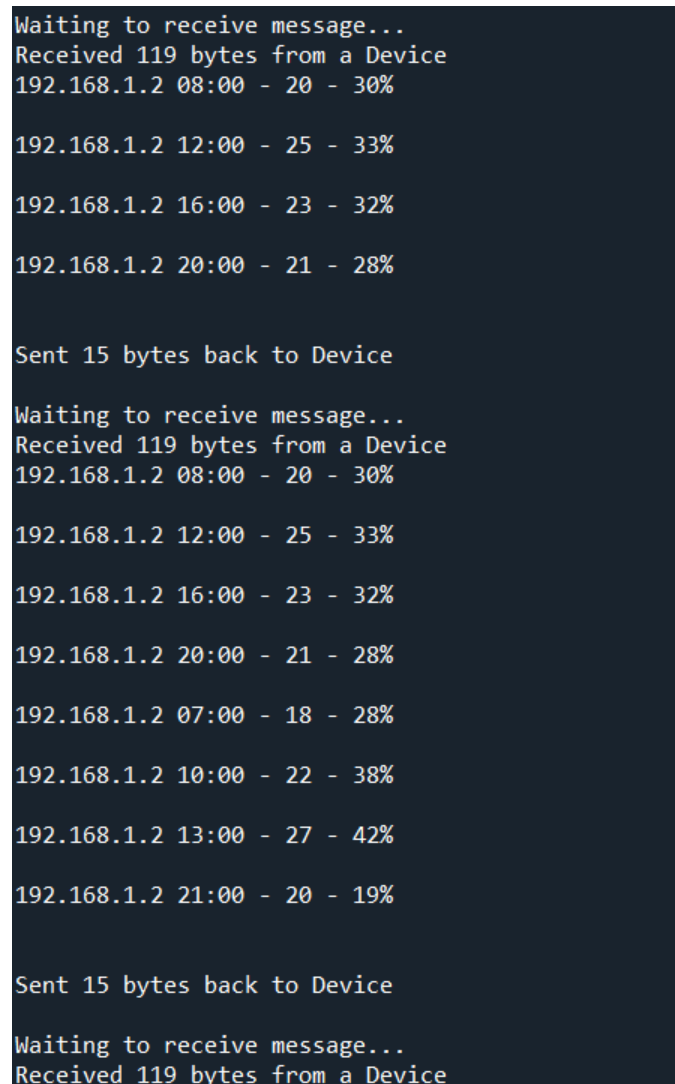


```
Console 1/A x Console 2/A x
Python 3.8.5 (default, Sep 3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.19.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/caste/Desktop/2ndYear/programmazione di reti/
progetto/ProgettoProgrammazioneReti/Cloud.py', wdir='C:/Users/caste/Desktop/
2ndYear/programmazione di reti/progetto/ProgettoProgrammazioneReti')
the web server is up on port: 8001
Cloud server is ready to serve...
```

Una Volta che il Gateway ha ricevuto tutte le rilevazioni dai 4 Devices:



```
Waiting to receive message...
Received 119 bytes from a Device
192.168.1.2 08:00 - 20 - 30%

192.168.1.2 12:00 - 25 - 33%

192.168.1.2 16:00 - 23 - 32%

192.168.1.2 20:00 - 21 - 28%

Sent 15 bytes back to Device

Waiting to receive message...
Received 119 bytes from a Device
192.168.1.2 08:00 - 20 - 30%

192.168.1.2 12:00 - 25 - 33%

192.168.1.2 16:00 - 23 - 32%

192.168.1.2 20:00 - 21 - 28%

192.168.1.2 07:00 - 18 - 28%

192.168.1.2 10:00 - 22 - 38%

192.168.1.2 13:00 - 27 - 42%

192.168.1.2 21:00 - 20 - 19%

Sent 15 bytes back to Device

Waiting to receive message...
Received 119 bytes from a Device
```

... chiude la socket UDP... :

```
192.168.1.2 12:00 - 25 - 33%
192.168.1.2 16:00 - 23 - 32%
192.168.1.2 20:00 - 21 - 28%
192.168.1.2 07:00 - 18 - 28%
192.168.1.2 10:00 - 22 - 38%
192.168.1.2 13:00 - 27 - 42%
192.168.1.2 21:00 - 20 - 19%
192.168.1.2 02:00 - 17 - 35%
192.168.1.2 09:00 - 19 - 32%
192.168.1.2 16:00 - 24 - 36%
192.168.1.2 23:00 - 21 - 30%
192.168.1.2 05:00 - 18 - 25%
192.168.1.2 11:00 - 24 - 28%
192.168.1.2 17:00 - 26 - 40%
192.168.1.2 23:00 - 20 - 32%

Sent 15 bytes back to Device
Closing socket
```


... tenta una connessione TCP verso il Cloud, gli invia le rilevazioni, e chiude la connessione, indicando il tempo che è stato impiegato nell'invio del messaggio e la dimensione del buffer:

```
Connecting with Cloud...
... Sending surveys to Cloud...
b'192.168.1.2 08:00 - 20 - 30%\n192.168.1.2 12:00 - 25 - 33%\n192.168.1.2
16:00 - 23 - 32%\n192.168.1.2 20:00 - 21 - 28%\n192.168.1.2 07:00 - 18 -
28%\n192.168.1.2 10:00 - 22 - 38%\n192.168.1.2 13:00 - 27 - 42%\n
\n192.168.1.2 21:00 - 20 - 19%\n192.168.1.2 02:00 - 17 - 35%\n192.168.1.2
09:00 - 19 - 32%\n192.168.1.2 16:00 - 24 - 36%\n192.168.1.2 23:00 - 21 -
30%\n192.168.1.2 05:00 - 18 - 25%\n192.168.1.2 11:00 - 24 - 28%\n
\n192.168.1.2 17:00 - 26 - 40%\n192.168.1.2 23:00 - 20 - 32%\n\n'
Waiting the server's response...
Received Message: Surveys received
TCP message's sending time 0.0 and the size of used buffer is 4096

Closing connection
```

A questo punto il Cloud ha ricevuto tutte le rilevazioni...

```
Cloud server is ready to serve...
The Gateway is connected!
... And it is uploading the surveys ...
192.168.1.2 08:00 - 20 - 30%

192.168.1.2 12:00 - 25 - 33%

192.168.1.2 16:00 - 23 - 32%

192.168.1.2 20:00 - 21 - 28%

192.168.1.2 07:00 - 18 - 28%

192.168.1.2 10:00 - 22 - 38%

192.168.1.2 13:00 - 27 - 42%

192.168.1.2 21:00 - 20 - 19%

192.168.1.2 02:00 - 17 - 35%

192.168.1.2 09:00 - 19 - 32%

192.168.1.2 16:00 - 24 - 36%

192.168.1.2 23:00 - 21 - 30%

192.168.1.2 05:00 - 18 - 25%

192.168.1.2 11:00 - 24 - 28%

192.168.1.2 17:00 - 26 - 40%

192.168.1.2 23:00 - 20 - 32%

Surveys received!
```

... e chiude la connessione TCP.

```
Surveys received!  
Closing connection
```

Capitolo 4

Librerie utilizzate

- Socket
- Time