Multimedia

Homework 2

Alessandro Trigolo

7 Giugno 2024

Indice

Elenco delle figure

Todo list

1 Introduzione

Fai introduzione

1.1 Utilizzo dello script

Le richieste dell'homework sono state soddisfatte attraverso uno script scritto con il linguaggio *Python* ed eseguito sul sistema operativo *Windows 10* in lingua italiana. Quest'ultima specifica è significativamente importante in quanto se la lingua del sistema operativo non è in italiano, lo script non è in grado di effettuare il *parsing* corretto una volta effettuate le richieste. Si osserva inoltre che tali richieste vengono effettuate tramite il comando `ping`, preinstallato in Windows.

Prima di lanciare lo script, è necessario aprire il file *script.py* e impostare correttamente il percorso dello script modificando la costante `PATH_TO_SCRIPT`.

```
PATH_TO_SCRIPT = os.path.join("multimedia", "hw-2", "script")
```

In secondo luogo è possibile impostare le specifiche di progetto mediante delle costanti che indicano il nome del server su cui inviare le richieste, il numero di instanze da mandare ad ogni richiesta ping e lo step che ci sarà tra la lunghezza del pacchetto durante l'iterazione i e la successiva $i + 1$. Per esempio, nell'esempio seguente, si manderanno le richieste al server in Atltanta. Si prenderanno poi tutte le lunghezze partendo da 10 e aggiungendo ogni volta il valore di `STEP_BETWEEN_LENGTHS` fino ad arrivare a 1470. Per ognuna di queste lunghezze si effettureanno 30 istanze di richiesta. In altre parole si faranno 30 richieste di lunghezza 10 byte al server di Atlanta, poi altre 30 di lunghezza $10 + 25 = 35$ byte, poi altre 30 di lunghezza $35 + 25 = 60$ bytes e così via fino ad arrivare alla lunghezza 1470.

```
# project specification
SELECTED_SERVER_CITY = "Atlanta"
INSTANCES = 30
STEP_BETWEEN_LENGTHS = 25
```

indica che hai usato subprocess per invocare i cmd

1.2 Parametri di progetto

- Los Angeles
- K = 250
- Dim = 10 → 1470, step di 1

Di che nalle task 1 hai usato ping mentre enella task 2 haiu suato psiong

2 Performance di rete

fai piccolo riassunto

parla di come funzione il comando ping

2.1 Numero di link attraversati

Il primo valore che andremo a calcolare è il numero di connessioni — dette anche *links* — che ci separa dal server di Los Angeles. In particolare ci occuperemo di recuperare tale valore in due modi diversi. Il primo modo per farlo è tramite l'utilizzo del comando Windows `tracert` che permetti di tracciare interamente il percorso che un pacchetto IP[1] compie al fine di raggiungere la destinazione, che in questo caso è il server di Los Angeles `la.speedtest.clouvider.net`. Una volta esguito il comando si ottiene sul terminale la lista numerata di tutti i link attraversati, come mostrato nel frammento di codice seguente.

Se vuoi inserisci come funzione comando tracert: una serie di ping con ttl incrementale fino a che non raghgiungi il server

```
Traccia instradamento verso la.speedtest.clouvider.net [77.247.126.223]
su un massimo di 30 punti di passaggio:

1    <1 ms    <1 ms    <1 ms  CLOUD-STORAGE [192.168.1.1]
2    36 ms    17 ms     8 ms  151.6.141.50
3    11 ms     7 ms     8 ms  151.6.141.34
4    14 ms    14 ms    14 ms  151.6.0.68
5    13 ms    15 ms    12 ms  151.6.1.182
6    15 ms    14 ms    14 ms  mno-b3-link.ip.twelve99.net [62.115.36.84]
7    30 ms    29 ms    29 ms  prs-bb1-link.ip.twelve99.net [62.115.135.224]
8   113 ms   113 ms   127 ms  ash-bb2-link.ip.twelve99.net [62.115.112.242]
9   172 ms   172 ms   171 ms  lax-b22-link.ip.twelve99.net [62.115.121.220]
10  173 ms   173 ms   173 ms  clouvider-ic-355873.ip.twelve99-cust.net
    [213.248.74.63]
11  176 ms     *      203 ms  77.247.126.1
12  172 ms   171 ms   172 ms  77.247.126.223

Traccia completata.
```

Tale risultato si traduce nel seguente codice Python dove, dopo aver memorizzato il risultato del comando nella stringa `result`, quest'ultima viene suddivisa in righe e viene analizzata in modo tale da restituire l'ultimo numero del link, che coincide con il numero totale di connessioni attraversate per raggiungere il server. Mediante tale funzione si ottiene che il numero di links necessari per raggiungere il server di Los Angeles corrisponde a 12.

```python
def get_links_from_tracert(server: str) -> int:

    # get number of links from tracert
    cmd = f"tracert {server}"
    result = subprocess.run(cmd, shell=True,
        stdout=subprocess.PIPE, stderr=subprocess.PIPE,
        text=True)

    last_link_line = result.stdout.split("\n")[-4]

    return int(last_link_line.split(" ")[1])
```

Per contare il numero di connessioni, oltre all'utilizzo del comando `tracert`, è possibile impiegare direttamente il comando `ping`. In particolare, è sufficiente iterare

[1] IP è l'acronimo di *Internet Protocol*

attraverso una serie di ping, diminuendo ad ogni iterazione il parametro TTL (Time-To-Live), che rappresenta il numero massimo di link che il pacchetto può attraversare prima di essere eliminato. La seguente funzione in Python itera sui valori di TTL da 20 a 0. Ad ogni iterazione viene inviata una richiesta ping predefinita, con 4 pacchetti di dimensione di 32 byte. Quando il ping non ottiene una risposta, significa che il valore di TTL è diminuito al punto da non essere più sufficiente per raggiungere la destinazione finale. Pertanto, viene restituito il valore di TTL incrementato di 1, che è il minimo TTL necessario per raggiungere il server.

```
1    def get_links_from_ping(server: str) -> int:
2
3        for ttl in range(20, 0, -1):
4
5            cmd = f"ping {server} -i {ttl}"
6            result = subprocess.run(cmd, shell=True,
                 stdout=subprocess.PIPE, stderr=subprocess.PIPE,
                 text=True)
7
8            if "TTL scaduto durante il passaggio" in result.stdout:
9                return (ttl + 1)
```

Tramite questa funzione il numero di link ottenuti coincide con quello in precedenza, che è 12.

2.2 Analisi del *Round Trip Time*

La seconda parte, volta ad analizzare il *RTT*, necessita uno script Python più complesso, anche volto a migliorare le prestazioni computazionali dello script. Si è quindi scelto di utilizzare un approccio **multi-threading** ed il comando `psping`, che è più prestante rispetto al comnue ping, inoltre permetto di ottnere delle latenze più precise (fino al centesimo di millisecondo).

Per permettere l'utilizzo di diversi thread, è necessario racchiudere le richieste ping in una funzione, chiamata `ping_server`. In questa funzione la richiesta di ping ha diverse proprietà specificate:

- → `-n`, che specifica il numero di istaze per ogni richiesta ping;
- → `-l`, indica la lunghezza del *payload* di ciascuna richiesta;
- → `-i`, specifica l'intervallo di secondo tra un ping e l'altro; viene impostato a zero per aumentare la rapidità dello script;
- → `-w`, che indica il numero di *warmup request* da fare prima di iniziare la sequenza di ping.

Dopo aver eseguito il comando, il risultato viene analizzato e ne sono estratte le latenze, che sono aggiunte in una lista (o vettore). Infine sono ritornati due valori, la lunghezza corrente e la lista con tutte le latenze.

```
1    def ping_server(server, instances, length):
2
3        cmd = f"psping -n {instances} -l {length} -i 0 -w 0
             {server}"
```

```
        result = subprocess.run(cmd, shell=True,
            stdout=subprocess.PIPE, stderr=subprocess.PIPE,
            text=True)

        millisecs_vector = []
        lines = result.stdout.split("\n")

        for line in lines:
            if "Reply from" in line:
                duration = float(line.split(": ")[1].split("ms")[0])
                millisecs_vector.append(duration)

        return length, millisecs_vector
```

La funzione descritta sopra viene utilizzata nel seguente script dove, mediante il pacchetto `ThreadPoolExecutor`, viene eseguita da ciascun thread. In questo modo molte più richieste sono mandate contemporaneamente migliorando notevolmente le prestazioni del programma. Senza l'utilizzo di un algoritmo multithreading, ottenere tempistiche di esecuzione superiori a una o due ore era inevitabile, anche con un numero limitato di istanze (ad esempio, 50 istanze con incrementi di 75). Infine, la coppia restituita dalla funzione `ping_server`, viene aggiunta al dizionario `stats` dove sono immagazzinate tutte latenze per poi essere analizzate.

```
    stats = {}

    # using ThreadPoolExecutor to imporve computational capabilites
    with concurrent.futures.ThreadPoolExecutor(max_workers=100) as
        executor:

        futures = []
        for length in payload_lengths:
            # submit a task to the executor to ping the server with the
                specified parameters
            future = executor.submit(ping_server, server,
                INSTANCES, length)
            futures.append(future)

        # iterate over completed futures as they become available
        for future in concurrent.futures.as_completed(futures):
            length, millisecs_vector = future.result()
            stats[length] = millisecs_vector

    # order stats by length
    stats = dict(sorted(stats.items()))
```

Al fine di calcolare tutti i massimi, i minimi, le medie e le varianze è sufficiente iterare lungo il dizionario creato. Lo seguente script si occupa proprio di questo, anche se in modo poco efficiente ma leggibile.

```
    max_values = {}
    min_values = {}
```

```
    average_values = {}
    standard_deviations = {}

    for key, value in stats.items():
        max_values[key] = max(value)
        min_values[key] = min(value)
        average_values[key] = sum(value) / len(value)
        standard_deviations[key] = math.sqrt(sum((x -
            average_values[key]) ** 2 for x in value) / len(value))
```

Grafici

Dopo aver eseguito gli script, è quindi possibile mostrare tutte le latenze raccolte, i loro massimi, i minimi, le latenze medie per ciascuna lunghezza e la loro varianza. Sono presentati di seguito tre immagini contenenti i risultati delle richieste; cioascune delle tre immagini fa riferimento ad una diversa scelta di istanze e di step.

L'immagine 1 di seguito rappresenta tutte le latenza raccolte durante l'esecuzione dei ping con 20 istanze per richiesta e con uno step di 10 tra una lunghezza e l'altra, per un totale di 2940 ping effettuati in circa un minuti. Si può osservare che in genere tutte le latenze sono attorno ai 170 millisecondi, con alcuni picchi tra i 750 e gli 800 bytes. Non a caso, nei grafici rappresentati in figura 2, si osserva infatti che i

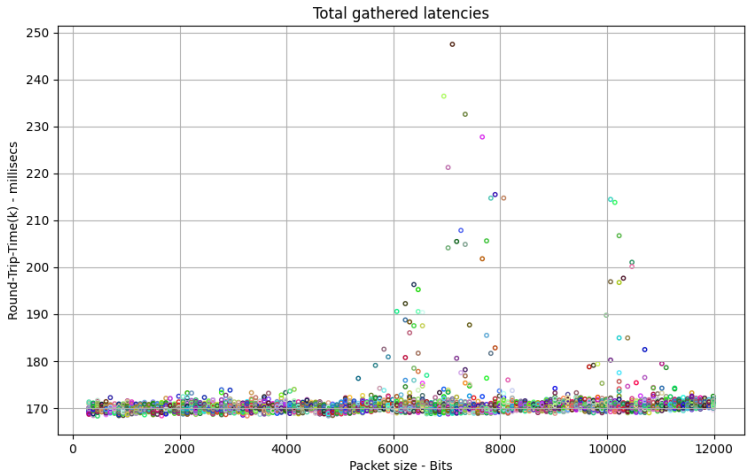

Figura 1: Tutte le ottenute con 20 istanze ed uno step di 10.

picchi sono rispecchiati nella rappresentazione dei massimi (in alto a destra), delle media (in basso a sinistra) e delle varianze (in basso a destra). Si osserva inoltre che i minimi variano tra 168 e 170 millisecondi e che crescono al crescere della dimensione del pacchetto; questo aspetto sarà dettagliatamente esplorato nel prossimo paragrafo, dedicato al *throughput* (2.3).

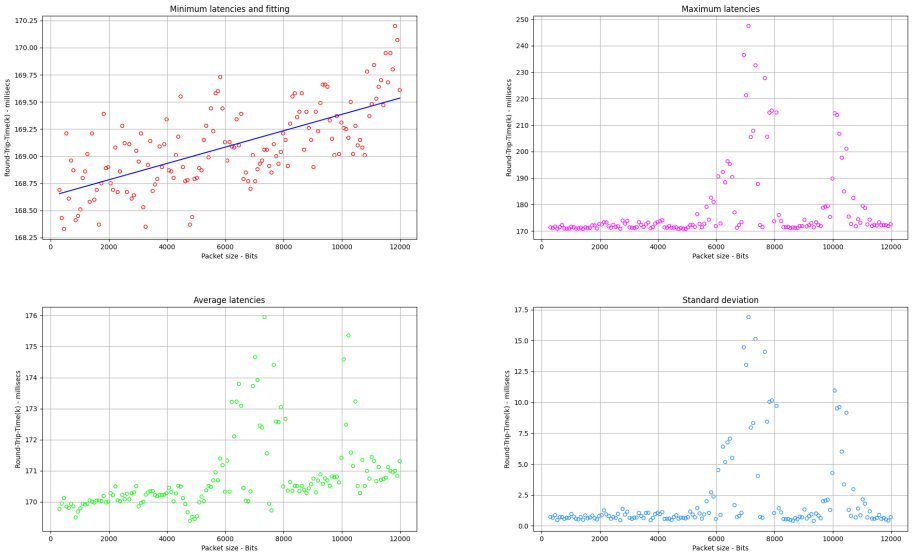

Figura 2: Grafici in riferimento all'immagine1.

Aumentando il numero di ping effettuati, in particolare aumentonado il numero di istanze a 100 e riducendo l'incremento delle lunghezze a 2, otteniamo un grafico come quello della figura 3, che contiene 73100 valori di latenza, ottenuti in circa 6 minuti. Anche in questo caso possiamo notare dei picchi che sono distribuiti lungo tutte le lunghezze. In particolare, i più elevati si aggirano attorno a 600 bytes di lunghezza del payload. Osservando infatti i grafici nell'immagine 4, notiamo una forte varianza (fino a 30 millisecondi) proprio in queste zone. In questa immagine si rende ancora più evidente la crescita dei minimi e anche delle medie con il crescere della lunghezza dei pacchetti.

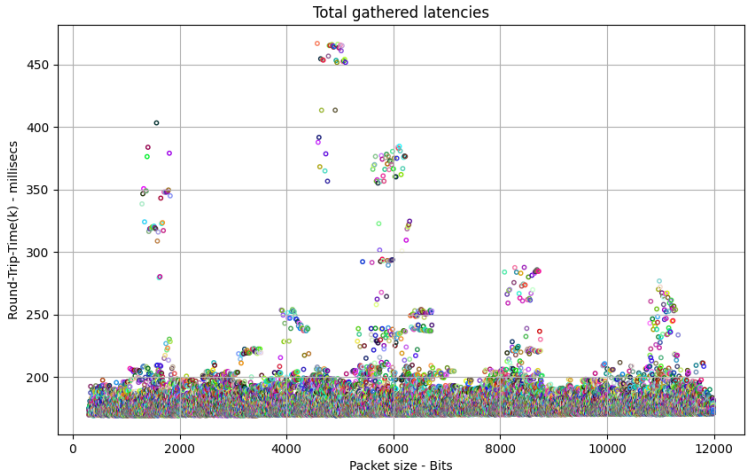

Figura 3: Grafici ottenuti con 100 istanze ed uno step di 2.

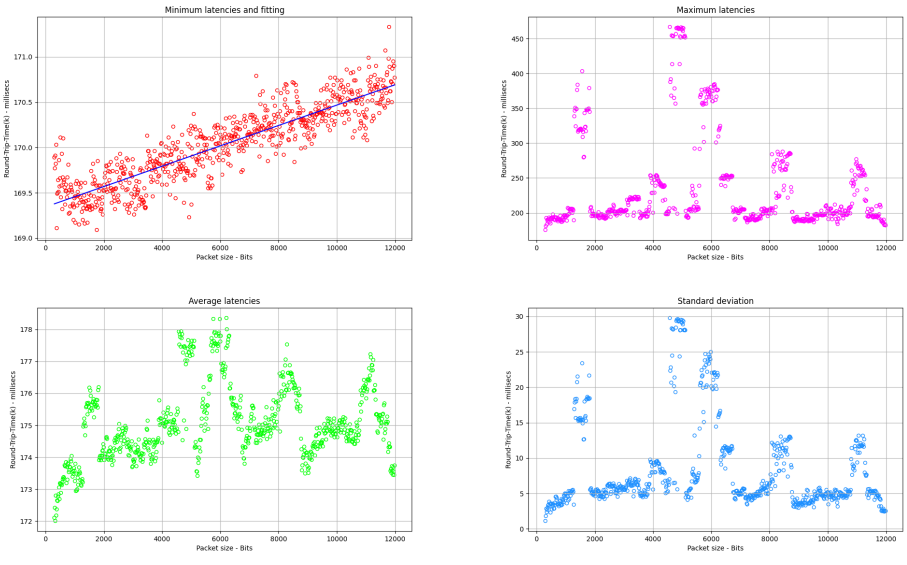

Figura 4: Grafici in riferimento all'immagine 3.

Infine, nell'immagine 5 sono state processate un totale di 365250 richieste in circa un quarto d'ora, risultato di 250 istanze per lunghezza ed uno step di 1 tra una lunghezza e l'altra. Si può infatti notare che il numero di dati in quest'ultimo grafico è molto più elevato rispetto ai due precedenti garantendo quindi che i dati analizzati sono più simili a quelli reali. In questo caso è interssante osservare che sia la media che la

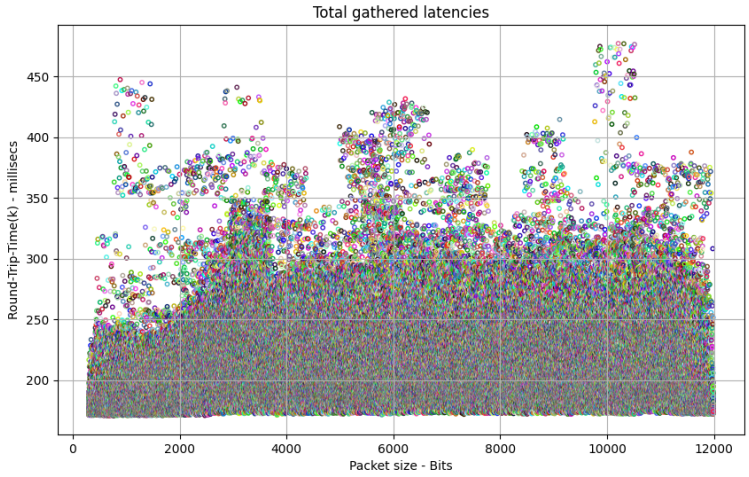

Figura 5: Grafici ottenuti con 250 istanze ed uno step di 1.

varianza sono elevate per la maggior parte delle lunghezze per decrescere solo quando il pacchetto è di dimensione ridotta oppure molto elevata (figuar 6). Anche il grafico che rappresenta il minimo non segue più una retta definita (a differenza di quello mostrato nei grafici 4), ma dopo i 350 bytes si hanno minimi che arrivano fino a 178 millisecondi.

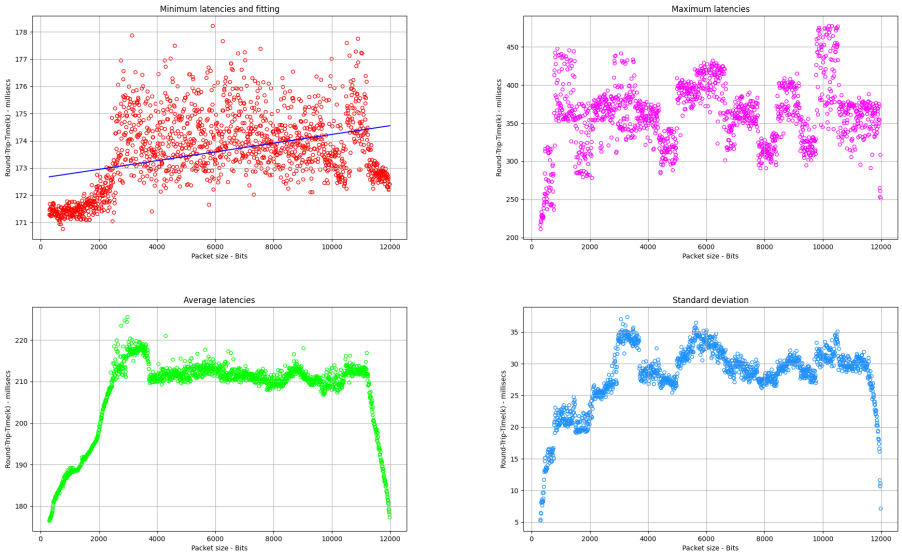

Figura 6: Grafici in riferimento all'immagine 5.

scrivi qualcosa qua

2.3 Throughput

Attraverso tutti i dati raccolti è quindi possibile

continua qui

```
    payload_lengths_bit = [8 * (length + 28) for length in
        payload_lengths]

    # use linear regression to retrive alpha, need to transform list into
        np.arrays
    reg = LinearRegression().fit(
        np.array(payload_lengths_bit).reshape(-1, 1), # transpose of
            payload lengths
        np.array(list(min_values.values()))
    )

    alpha = reg.coef_[0]

    # compute throughput
    throughput_identical_link = 2 * tracert_links / alpha
    throughput_bottleneck = 2 / alpha
```

3 Conclusioni