



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Programmazione di Reti Laboratorio 3

Andrea Piroddi

Dipartimento di Informatica, Scienza e Ingegneria

Analisi di dettaglio del Web server Python





404

404 Error: requested doughnut does not exists!

URL: <http://www.danluu.com/404>

You have reached this page because the requested doughnut does not exist on this server.
Please check that the URL is correct or inform the webmaster of the website of an incorrect link.

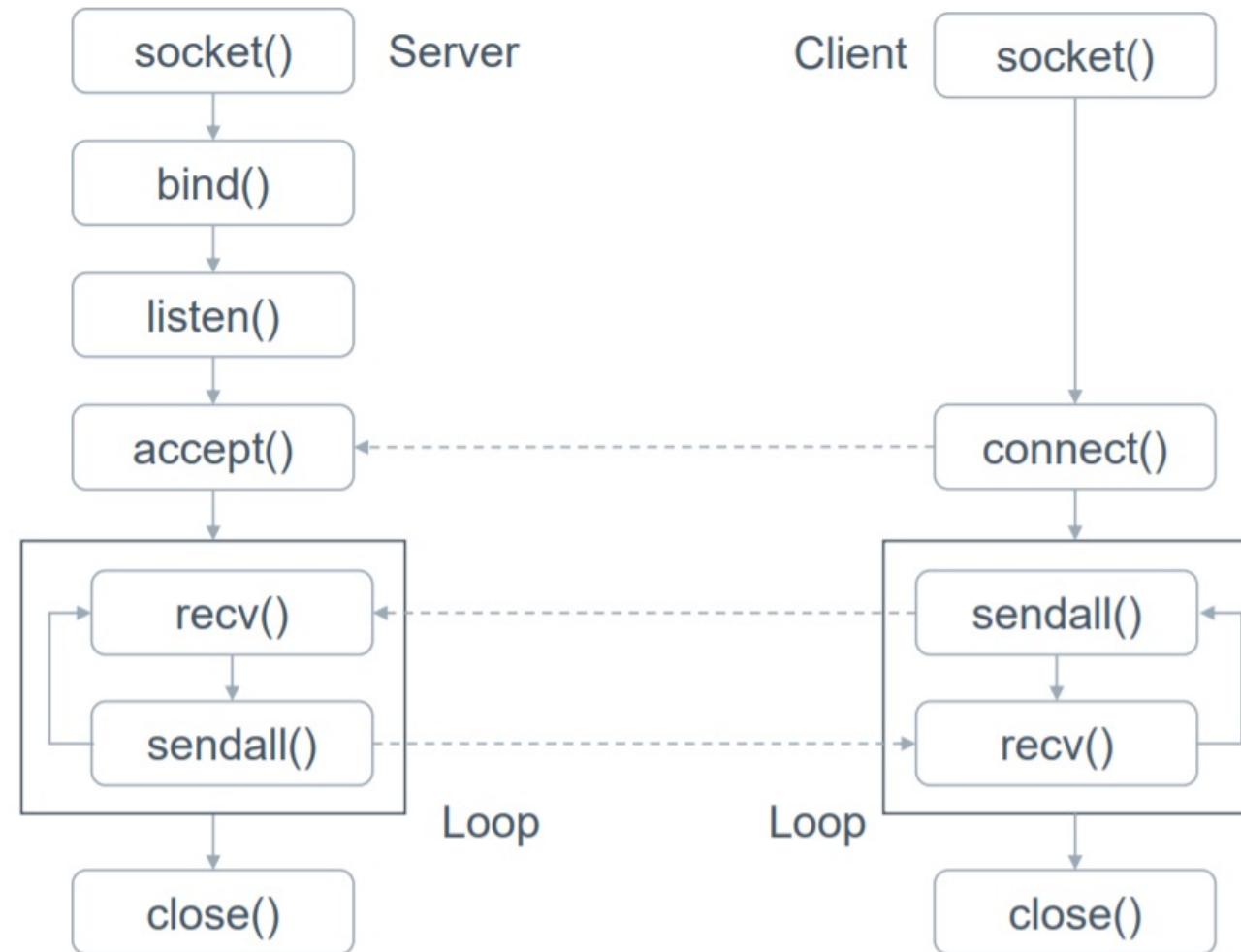
[Go to website home page](#)



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Analisi di dettaglio del Web server Python

Schema generale TCP



Analisi di dettaglio del Web server Python

Assegniamo alla variabile **serverPort** il valore della porta che vogliamo dare al socket

```
from socket import *
```

```
serverPort=82
```

```
serverSocket = socket(AF_INET, SOCK_STREAM)
```

```
serverSocket.bind(('',serverPort))
```

```
serverSocket.listen(1)
```

`socket.listen([backlog])` Abilita il socket a ricevere connessioni. **backlog** riguarda le connessioni in sospeso, non stabilite. Se assente, viene scelto un valore automaticamente.

Creiamo l'oggetto **socket** che ha la sintassi seguente:

`s = socket(socket_family, socket_type):`

socket_family corrisponde alla tipologia di indirizzo con cui vogliamo lavorare, di default abbiamo **AF_INET** che sta per gli indirizzi IPv4 (**AF_INET6** per IPV6) e a

socket_type corrisponde invece la tipologia di socket che vogliamo creare, di default abbiamo **SOCK_STREAM** perché stiamo creando un socket di tipo TCP (se volessimo usare il socket UDP qui dovremmo indicare **SOCK_DGRAM**)

Associa l'oggetto socket appena creato all'indirizzo **address**. Il socket non deve essere già stato associato ad altri indirizzi. Il formato di address dipende dalla famiglia di indirizzi, per **AF_INET** o **AF_INET6** è una tupla (hostname, port). Nel caso specifico con '' stiamo sottintendendo l'ip 127.0.0.1.



Analisi di dettaglio del Web server Python

Inizia il loop

while True:

connectionSocket, addr = serverSocket.accept()

Se provate a visualizzare il contenuto di questo oggetto (con print) vedreste qualcosa del genere (**ovviamente solo nel momento in cui arriva la richiesta dal client**)

```
<socket.socket fd=416, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 82),  
raddr=('127.0.0.1', 64481)>, ('127.0.0.1', 64481)
```

remote address

"fd" è un'abbreviazione di File Descriptor. Un descrittore di file è un numero che identifica in modo univoco un file aperto nel sistema operativo di un computer. Descrive una risorsa di dati e come è possibile accedervi. Il descrittore è identificato da un numero intero non negativo univoco. Esiste almeno un descrittore di file per ogni file aperto sul sistema.

È il Protocol number
ed è=0 quando si usa
AF_INET

local address



Analisi di dettaglio del Web server Python

* Considerazioni sulla dimensione del buffer
sono nelle ultime slide

try:

```
    message = connectionSocket.recv(1024)
```

socket.recv(bufsize[, flags])

Riceve dati dal socket connesso, restituendo una stringa binaria. **bufsize** specifica la dimensione del buffer di ricezione. Valori consigliati di bufsize sono potenze di 2 relativamente basse, come 1024, 4096. **flags** sono delle flag platform-dependent che modificano il comportamento della funzione.

Se provate a leggere il contenuto del messaggio vedreste qualcosa del genere

b'GET /index.html HTTP/1.1\r\nHost: 127.0.0.1:82\r\nConnection: keep-alive\r\nUpgrade-Insecure-Requests: 1\r\nUser-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.149 Safari/537.36\r\nSec-Fetch-Dest: document\r\nAccept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9\r\nSec-Fetch-Site: none\r\nSec-Fetch-Mode: navigate\r\nSec-Fetch-User: ?1\r\nAccept-Encoding: gzip, deflate, br\r\nAccept-Language: it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7\r\n\r\n'

il tipo di software utilizzato dal client

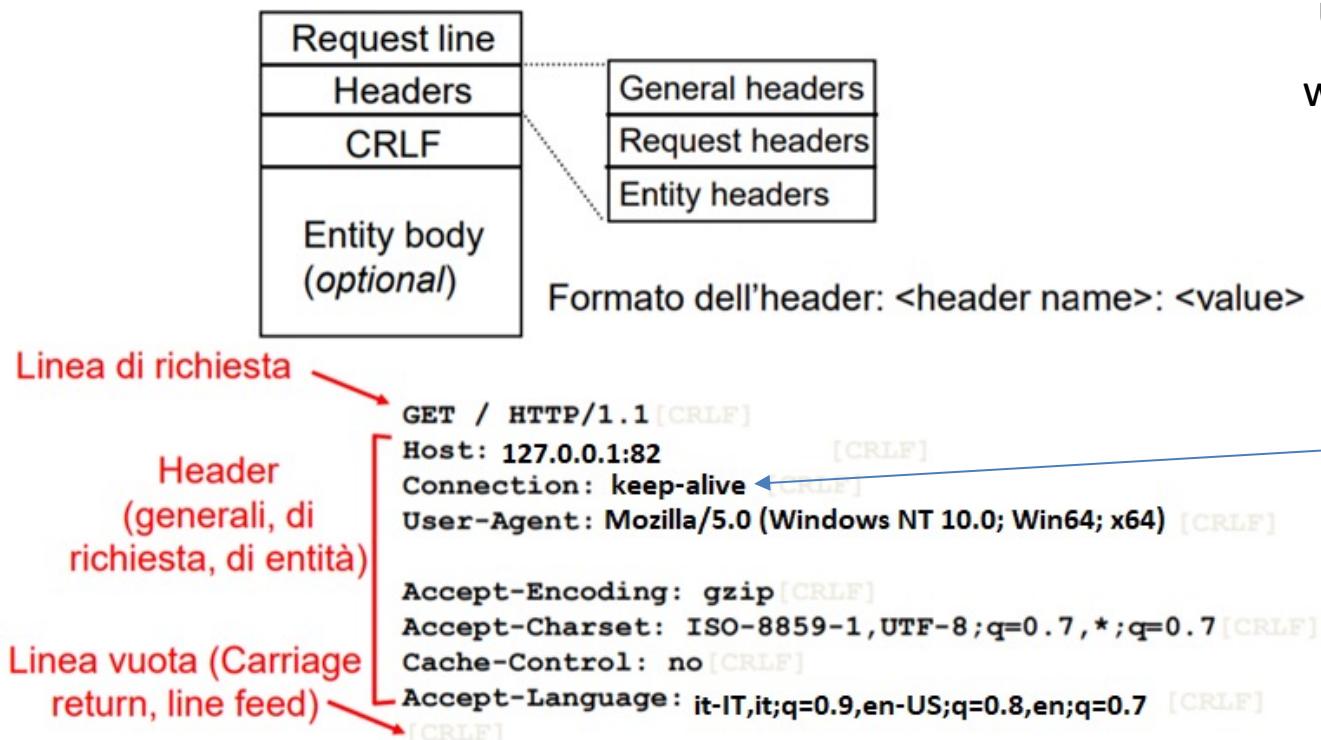
La b' indica che ciò che segue è un data type di tipo bytes



Analisi di dettaglio del Web server Python

Messaggio di richiesta HTTP

- Formato generale



GET è il metodo con cui vengono richieste la maggior parte delle informazioni ad un Web server, tali richieste vengono veicolate tramite query string, cioè la parte di un URL che contiene dei parametri da passare in input ad un'applicazione; ad esempio:

www.ilmiosito.it:82/pagina-richiesta?id=123&page=3

La connessione persistente HTTP, chiamata anche keep-alive HTTP sottintende l'idea di utilizzare una singola connessione TCP per inviare e ricevere più richieste / risposte HTTP, anziché aprire una nuova connessione per ogni singola coppia richiesta / risposta.

Chi vuole approfondire → <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Connection>



Analisi di dettaglio del Web server Python

```
if len(message.split())>0:
```

Questa istruzione controlla che il messaggio di richiesta non sia vuoto

La funzione `len` ritorna il numero di caratteri di una stringa:

```
>>> Frutto = "banana"  
>>> len(Frutto)  
6
```

La funzione `split` spezza una stringa in una lista di parole singole, **considerando un qualsiasi carattere di spazio bianco** come punto di interruzione tra parole consecutive:

```
>>> import string  
>>> Verso = "Nel mezzo del cammin..."  
>>> string.split(Verso)  
['Nel', 'mezzo', 'del', 'cammin...']
```

Può anche **essere usato un argomento opzionale per specificare quale debba essere il delimitatore** da considerare. In questo esempio usiamo la stringa `\n` come delimitatore:

```
>>> string.split(Verso, '\n')  
['N', ' mezzo d', ' cammin...']
```

Il delimitatore non appare nella lista.



Analisi di dettaglio del Web server Python

Se supponessimo di voler vedere a cosa corrisponde `message.split()` vedremo questo:

```
[b'GET', b'/index.html', b'HTTP/1.1', b'Host:', b'127.0.0.1:82', b'Connection:', b'keep-alive', b'Upgrade-Insecure-Requests:', b'1', b'User-Agent:', b'Mozilla/5.0', b'(Windows', b'NT', b'10.0;', b'Win64;', b'x64)', b'AppleWebKit/537.36', b'(KHTML,', b'like', b'Gecko)', b'Chrome/80.0.3987.149', b'Safari/537.36', b'Sec-Fetch-Dest:', b'document', b'Accept:', b'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9', b'Purpose:', b'prefetch', b'Sec-Fetch-Site:', b'none', b'Sec-Fetch-Mode:', b'navigate', b'Sec-Fetch-User:', b'?1', b'Accept-Encoding:', b'gzip,', b'deflate,', b'br', b'Accept-Language:', b'it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7']
```

`message.split[0]`

`message.split[1]`



Analisi di dettaglio del Web server Python

```
filename = message.split()[1]
```

Assegno ad una variabile che chiamo filename il contenuto di *message.split()[1]* ossia '/index.html'

```
f = open(filename[1:], 'r+')
```

Considero la variabile filename **a partire dalla seconda lettera in poi** (extended slices), ossia escludo lo '/'

open () restituisce un oggetto file e viene comunemente utilizzato con due argomenti: open (nome file, modalità). Non ho ancora letto il contenuto del file, ho solo creato uno spazio in memoria dove ho inserito il contenuto del file *index.html*

Modalità	Descrizione
'r'	Apre un file di testo in lettura. Modo di apertura di default dei file.
'w'	Apre un file di testo in scrittura. Se il file non esiste lo crea, altrimenti cancella il contenuto del file.
'a'	Apre un file di testo in <i>append</i> . Il contenuto viene scritto alla fine del file, senza modificare il contenuto esistente.
'x'	Apre un file di testo in creazione esclusiva. Se il file non esiste, restituisce un errore, altrimenti apre in scrittura cancellando il contenuto del file.
'r+'	Apre un file di testo in modifica. Permette di leggere e scrivere contemporaneamente.
'w+'	Apre un file di testo in modifica. Permette di leggere e scrivere contemporaneamente. Cancella il contenuto del file.



Analisi di dettaglio del Web server Python

```
outputdata = f.read()
```

Una volta aperto il file in modalità lettura, sarà necessario leggerne il contenuto. Se il file è di tipo testuale, si potrà utilizzare il metodo **read()**. Il contenuto lo assegniamo alla variabile che chiamiamo **outputdata**.

Se facessimo il print di outputdata vedremmo qualcosa del tipo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">  
<html><body><h1>It works!</h1>  
<p>This is the default web page for this server.</p>  
<p>The web server software is running but no content has been added, yet.</p>  
</body></html>
```



Analisi di dettaglio del Web server Python

Risposta HTTP

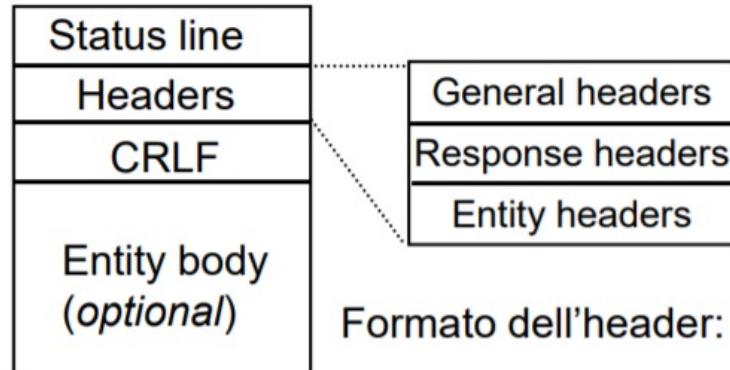
- Una *risposta HTTP* comprende
 - l'identificativo della versione del protocollo HTTP
 - il codice di stato e l'informazione di stato in forma testuale
 - un insieme di possibili altre informazioni riguardanti la risposta
 - l'eventuale contenuto della risorsa richiesta
- Se la pagina Web richiesta dall'utente è composta da molteplici risorse, ciascuna di esse è identificata da un URL differente: **il browser deve inviare un messaggio di richiesta HTTP per ognuna delle risorse incorporate nella pagina**



Analisi di dettaglio del Web server Python

Messaggio di risposta HTTP

- Formato generale



Formato dell'header: <header name>: <value>

Linea di stato → HTTP/1.1 200 OK[CRLF]
Intestazioni (general, di risposta, di entità) → Date: Sun, 19 Mar 2020 16:02:06 GMT[CRLF]
Server: Apache[CRLF]
Last-Modified: Thu, 29 Sep 2005 12:51:51 GMT[CRLF]
ETag: "2a7c3-15b9-92a267c0" [CRLF]
Accept-Ranges:bytes[CRLF]
Content-Length:5561[CRLF]
Connection:close[CRLF] ←
Content-Type:text/html; charset=ISO-8859-1[CRLF]
[CRLF]
Corpo del messaggio (es. file HTML richiesto) → data data data data data ...



Analisi di dettaglio del Web server Python

```
connectionSocket.send("HTTP/1.1 200 OK\r\n\r\n".encode())
connectionSocket.send(outputdata.encode())
```

socket.send(bytes[, flags])

Invia dati all'indirizzo cui il socket è connesso. **bytes** è una stringa binaria che contiene i dati da inviare. **flags** sono delle flag platform-dependent che modificano il comportamento della funzione. ...

```
>>> s.send(b'Hello...')
```

```
>>> s.send('...world!'.encode())
```

In Python 2 non è necessario codificare la stringa in quanto viene fatto automaticamente.



Analisi di dettaglio del Web server Python

```
connectionSocket.send("\r\n".encode())
```

Indica che il messaggio da inviare è terminato



```
connectionSocket.close()
```

```
socket.close()
```

Chiude un socket e ne rilascia le risorse allocate dal sistema operativo.

```
s = sck.socket(...)
```

```
s.close()
```



Analisi di dettaglio del Web server Python

I metodi di invio e ricezione dati del socket operano di norma con stringhe binarie (bytes).

Dichiarare una stringa binaria:

- ss = "Hello, world!"
- sb = b"Hello, world! "

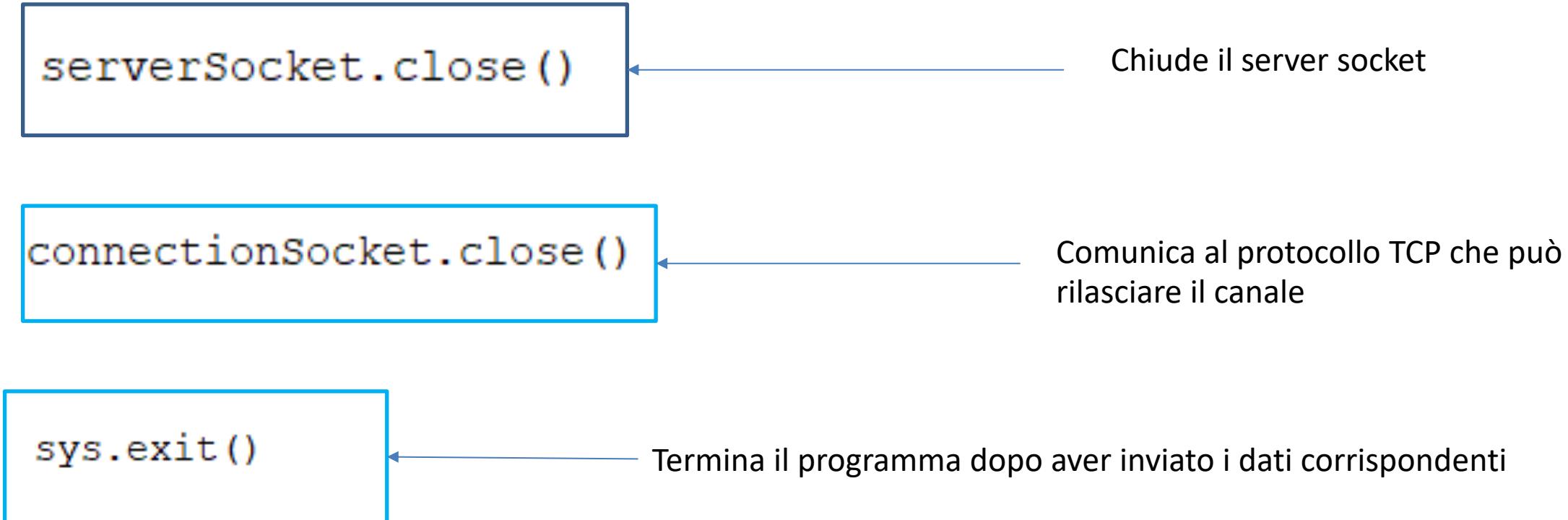
Metodi di conversione:

- ss.encode(), da stringa (ss) a stringa binaria
- sb.decode(), da stringa binaria (sb) a stringa

In Python 3 la distinzione tra stringa e stringa binaria è più marcata rispetto a Python 2, quindi bisogna fare particolare attenzione alle conversioni.



Analisi di dettaglio del Web server Python



Programmazione di un Socket - Web Client



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA



THE BIG BANG THEORY

THE FRIENDSHIP ALGORITHM

DR. SHELDON COOPER, PH.D



Esercizio 2: Programmazione di un Socket - Web Client

Invece di utilizzare un browser, scriviamo il nostro **client HTTP** per testare il server.

Il client si conterà al server utilizzando una connessione TCP, invierà una richiesta HTTP al server e visualizzerà la risposta del server come output.

Si può presumere che la richiesta HTTP inviata sia un metodo GET.

Il client deve accettare gli argomenti dalla riga di comando in cui l'utente specifica l'indirizzo IP o il nome host del server, la porta su cui è in ascolto il server e il nome del file archiviato sul server (eventualmente comprensivo di path).

Di seguito è riportato un formato del comando di input per eseguire il client:

```
>>>client.py server_host server_port filename nome_file
```



Esercizio 2: Programmazione di un Socket - Web Client

Lato server

Lato client

```
Documenti -- zsh -- 115x34
(base) apirodd@MacBook-Pro-di-andrea Documents % python client.py 127.0.0.1 8080 index.html
```

```
Documenti — python client.py 127.0.0.1 8080 index.html — 115x34
(base) apirodd@MacBook-Pro-di-andrea Documents % python client.py 127.0.0.1 8080 index.html
b'HTTP/1.1 200 OK\r\n\r\n'
```

Console 1/A

```
In [12]: runfile('/Users/apirodd/OneDrive - Alma Mater Studiorum Università di Bologna/programmazione di reti/Lezioni/Laboratorio 4/codice python Laboratorio 4/Socket_Programming_Assignment_1.py', wdir='/Users/apirodd/OneDrive - Alma Mater Studiorum Università di Bologna/programmazione di reti/Lezioni/Laboratorio 4/codice python Laboratorio 4')
the web server is up on port: 8080
Ready to serve...
```

Console 1/A

```
Ready to serve...
b'GET /index.html HTTP/1.1' :: b'GET' : b'/index.html'
b'/index.html' || b'index.html'
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<meta http-equiv="Content-Style-Type" content="text/css">
<title></title>
<meta name="Generator" content="Cocoa HTML Writer">
<meta name="CocoaVersion" content="2022.3">
<style type="text/css">
    p.p2 {margin: 0.0px 0.0px 12.0px 0.0px; font: 12.0px Times; color: #000000; -webkit-text-stroke: #000000}
    span.s1 {font-kerning: none}
</style>
</head>
<body>
<h1 style="margin: 0.0px 0.0px 16.1px 0.0px; font: 24.0px Times; color: #000000; -webkit-text-stroke: #000000"><span class="s1"><b>Funziona!</b></span></h1>
<p class="p2"><span class="s1">Corso di PROGRAMMAZIONE di RETI.</span></p>
<p class="p2"><span class="s1">Laboratorio 4.</span></p>
</body>
</html>
```



Esercizio 2: Programmazione di un Socket - Web Client

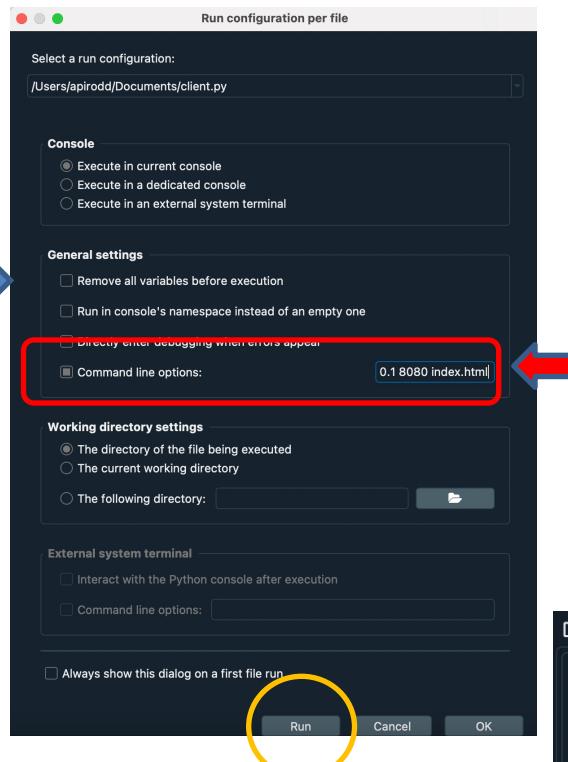
Lato server

E' possibile ottenere lo stesso risultato usando la funzione di Spyder **Run→ Configuration per file**

Lato client

```
python File Edit Search Source Run Debug Consoles Projects Tools View Help
Run configuration per file
Select a run configuration: /Users/apirodd/Documents/client.py
Console
Execute in current console
Execute in a dedicated console
Execute in an external system terminal
General settings
Remove all variables before execution
Run in console's namespace instead of an empty one
Directly enter debugging when errors appear
Command line options: 0.1 8080 index.html
Working directory settings
The directory of the file being executed
The current working directory
The following directory: [button]
External system terminal
Interact with the Python console after execution
Always show this dialog on a first file run
Run Cancel OK
```

```
'''Client:
After you run the server,
use command line "client.py localhost 8080"
Profile
Profile F10
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
from socket import *
import sys
clientsocket = socket(AF_INET, SOCK_STREAM)
if len(sys.argv) != 4:
    print(len(sys.argv))
    print("Your command is not right. Please be in this format:client.py server_host server_port filename")
    sys.exit(0)
host = str(sys.argv[1])
port = int(sys.argv[2])
request = str(sys.argv[3])
request = "GET /" + request + " HTTP/1.1"
try:
    clientsocket.connect((host,port))
except Exception as data:
```



```
In [12]: runfile('/Users/apirodd/OneDrive - Alma Mater Studiorum Università di Bologna/programmazione di reti/Lezioni/Laboratorio 4/codice python Laboratorio 4/Socket_Programming_Assignment_1.py', wdir='/Users/apirodd/OneDrive - Alma Mater Studiorum Università di Bologna/programmazione di reti/Lezioni/Laboratorio 4/codice python Laboratorio 4')
the web server is up on port: 8080
Ready to serve...
```

Inserite nella casella Command line options:
127.0.0.1 8080 index.html
E cliccate su RUN

```
Python 3.8.1 (default, Jan 8 2020, 16:15:59)
Type "copyright", "credits" or "license" for more information.

IPython 7.19.0 -- An enhanced Interactive Python.

In [1]: runfile('/Users/apirodd/Documents/client.py', args='127.0.0.1 8080
index.html', wdir='/Users/apirodd/Documents')
b'HTTP/1.1 200 OK\r\n\r\n'

Input Host Address:
```

```
Ready to serve...
b'GET /index.html HTTP/1.1' :: b'GET' : b'/index.html'
b'/index.html' || b'index.html'
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<meta http-equiv="Content-Style-Type" content="text/css">
<title></title>
<meta name="Generator" content="Cocoa HTML Writer">
<meta name="CocoaVersion" content="2022.3">
<style type="text/css">
p.p2 {margin: 0.0px 0.0px 12.0px 0.0px; font: 12.0px Times; color: #000000; -webkit-text-stroke: #000000}
span.s1 {font-kerning: none}
</style>
</head>
<body>
<h1 style="margin: 0.0px 0.0px 16.1px 0.0px; font: 24.0px Times; color: #000000; -webkit-text-stroke: #000000"><span class="s1"><b>Funziona!</b></span></h1>
<p class="p2"><span class="s1">Corso di PROGRAMMAZIONE di RETI.</span></p>
<p class="p2"><span class="s1">Laboratorio 4.</span></p>
</body>
</h1>
```



Esercizio 2: Programmazione di un Socket - Web Client

Come passare parametri al nostro script tramite riga di comando?

Usando la funzione **argv** del modulo **sys**.

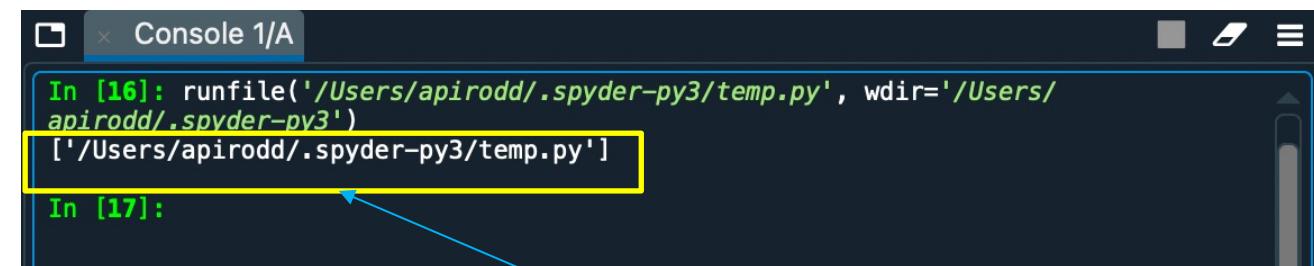
argv sta per "**argument variable**": si tratta semplicemente di una **lista di stringhe contenente i parametri che vengono passati** allo script quando lo state richiamando.

Quindi dobbiamo innanzitutto **importare il modulo sys** e vedere cosa fa il comando **argv**.

```
import sys
```

Immaginiamo ora di eseguire il seguente codice:

```
import sys  
print(sys.argv)
```



```
In [16]: runfile('/Users/apirodd/.spyder-py3/temp.py', wdir='/Users/apirodd/.spyder-py3')  
['/Users/apirodd/.spyder-py3/temp.py']  
In [17]:
```

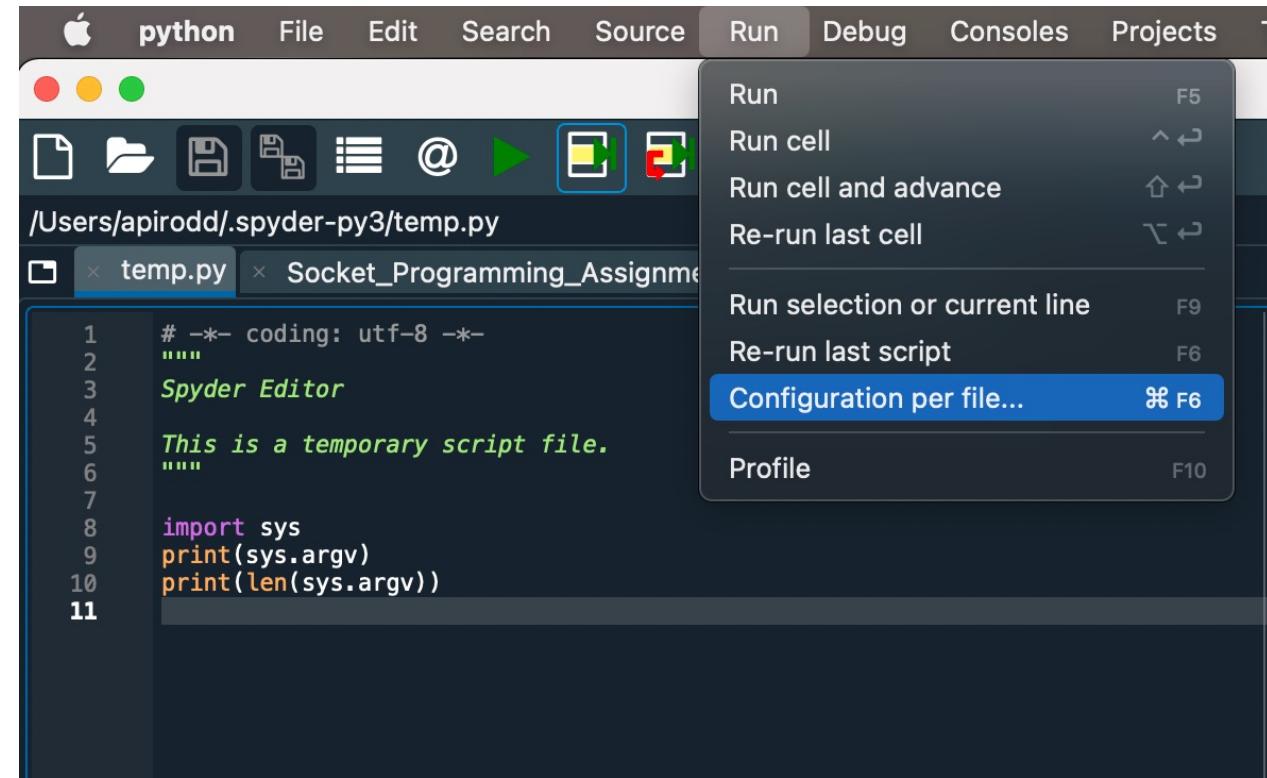
Restituisce una tupla (in questo caso di 1 elemento) composta dalla directory dove si trova il vostro file e dal nome del vostro file



Esercizio 2: Programmazione di un Socket - Web Client

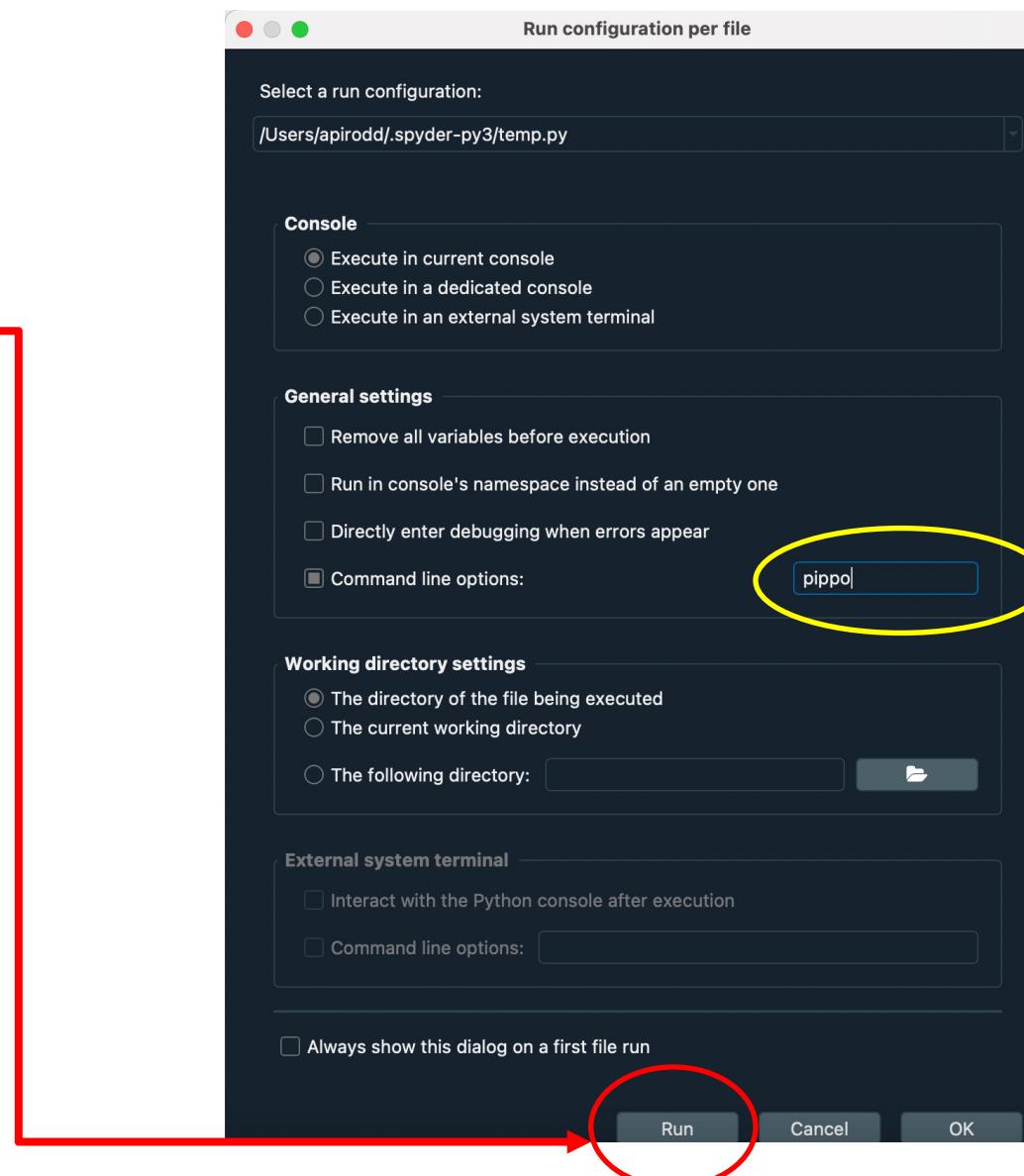
Proviamo ora ad eseguire il seguente codice usando **Run → Configuration per File** dal menu a tendina di IDLE, inserendo la parola «**pioppo**» nella finestra che si apre:

```
import sys  
print(sys.argv)  
print(len(sys.argv))
```



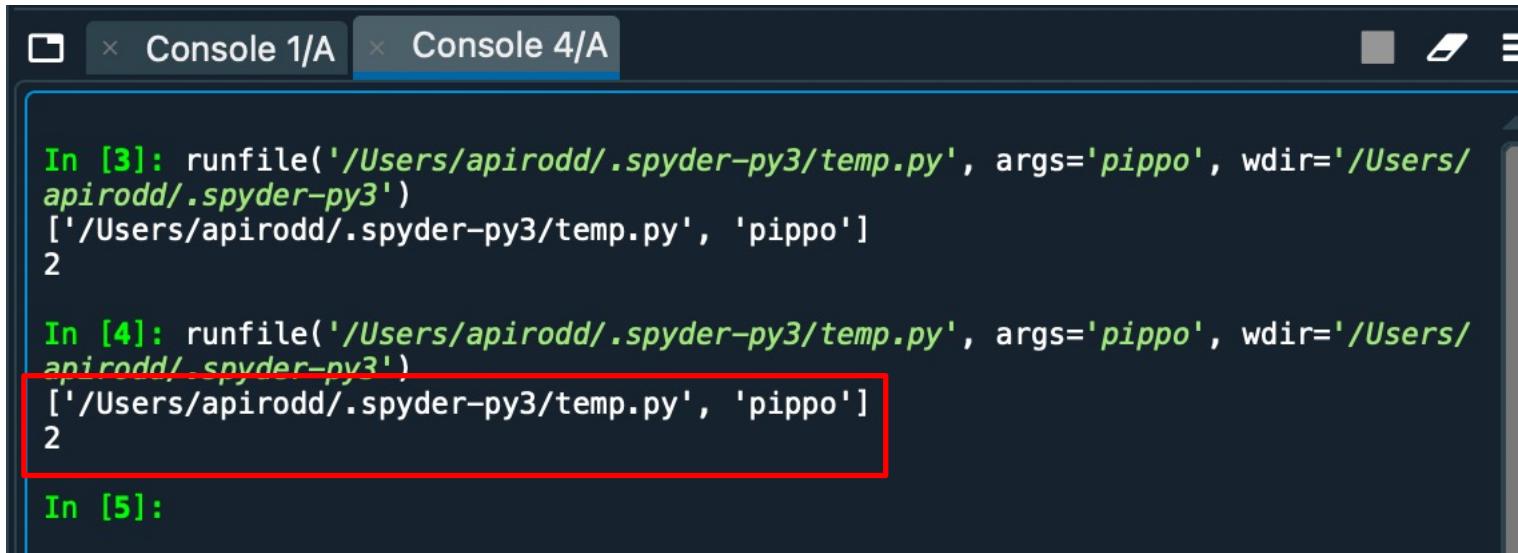
Esercizio 2: Programmazione di un Socket - Web Client

...inserendo la parola «**pioppo**» nella finestra che si apre, e cliccando su **RUN**



Esercizio 2: Programmazione di un Socket - Web Client

Il risultato sarà:



```
In [3]: runfile('/Users/apirodd/.spyder-py3/temp.py', args='pippo', wdir='/Users/apirodd/.spyder-py3')
['/Users/apirodd/.spyder-py3/temp.py', 'pippo']
2

In [4]: runfile('/Users/apirodd/.spyder-py3/temp.py', args='pippo', wdir='/Users/apirodd/.spyder-py3')
['/Users/apirodd/.spyder-py3/temp.py', 'pippo']
2

In [5]:
```

Restituisce una lista (in questo caso di 2 elementi ossia di lunghezza 2) composta dalla directory dove si trova il vostro file e il nome del vostro file, e come secondo elemento il nome **pippo**



Esercizio 2: Programmazione di un Socket - Web Client

Le Eccezioni

Anche se un'istruzione, o un'espressione, è sintatticamente corretta, può causare un errore quando si tenta di eseguirla. Gli errori rilevati durante l'esecuzione sono chiamati **eccezioni** e possono essere in gran parte gestite dal vostro eseguibile.

Un modo è utilizzare l'istruzione

try:

....

except Exception as e:

print(e)

In questo modo è possibile accedere agli attributi dell'oggetto eccezione ed evidenziare all'utente quale sia l'errore che ha interrotto l'esecuzione.



Considerazioni sul Buffer Size nel comando `clientsocket.recv(1024)`

In genere, le **dimensioni del buffer** di ricezione vengono aumentate quando l'autore del codice sta cercando di **ridurre la probabilità di riempire** completamente il buffer di ricezione del socket ed evitare che il sistema operativo debba eliminare alcuni pacchetti in entrata perché non ha spazio per inserire i dati. In un'applicazione basata su TCP, tale condizione farà arrestare temporaneamente il flusso fino a quando i pacchetti rilasciati non verranno reinviati correttamente; in un'applicazione basata su UDP, tale condizione farà sì che i pacchetti UDP in arrivo vengano eliminati silenziosamente.

La necessità o meno di farlo **dipende da due fattori**: la **velocità** con cui i dati devono riempire il buffer di ricezione del socket e la **velocità** con cui l'applicazione può svuotare il buffer di ricezione del socket tramite le chiamate a `recv()`. Se l'applicazione è in grado di svuotare il buffer in modo affidabile più velocemente della ricezione dei dati, la dimensione del buffer predefinita va bene; d'altra parte se vedete che non è sempre in grado di farlo, allora una dimensione del buffer di ricezione più grande potrebbe aiutarlo a gestire in modo più elegante raffiche improvvise di dati in arrivo.



Esercizio 5 – Ritardi di Trasferimento



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Esercizio 5 – Ritardi di Trasferimento

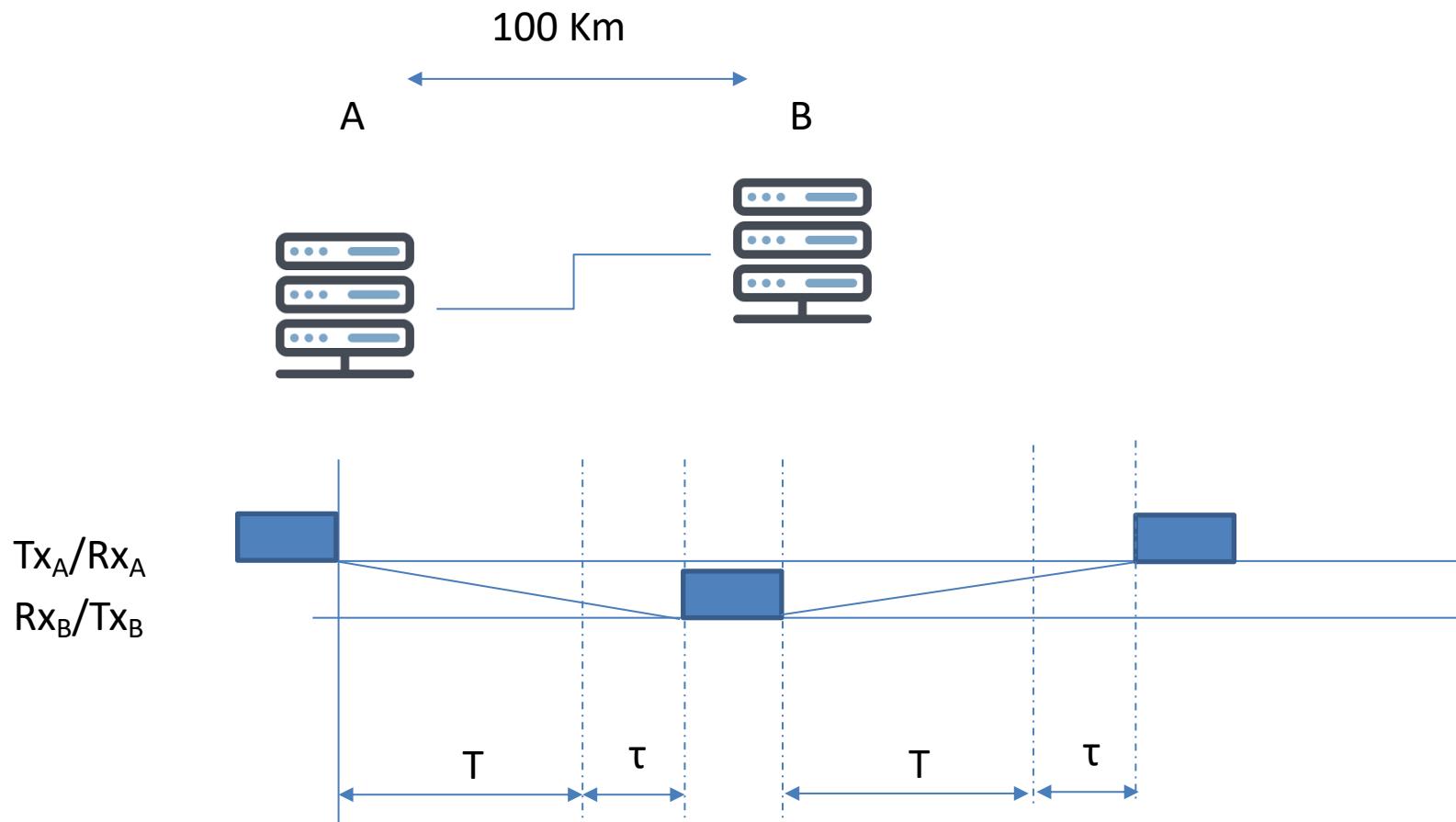
Un pacchetto di 10000 bit viene inviato dal server A alla velocità di 100 kb/s su un collegamento di 100 km. Il pacchetto viene ricevuto tutto in B e poi viene rimandato al mittente A alla stessa velocità di trasmissione.

Quesito:

Si calcoli l'intervallo di tempo che intercorre fra la **trasmissione** del primo bit in A e la **ricezione** dell'ultimo bit, sempre in A, assumendo che la velocità del segnale sia di 200000 km/s.



Esercizio 5 – Ritardi di Trasferimento – Soluzione 1/2



Esercizio 5 – Ritardi di Trasferimento – Soluzione 2/2

$$T_{tot} = T + \tau + T + \tau = 2 \cdot (T + \tau)$$

$$2 \cdot \left(\frac{L}{R} + \frac{\text{distanza}}{\text{velocità di propagazione}} \right) = 2 \cdot \left(\frac{10000 \text{ bit}}{100 \cdot \frac{10^3 \text{ bit}}{\text{s}}} + \frac{100 \text{ Km}}{200 \cdot \frac{10^3 \text{ Km}}{\text{s}}} \right)$$

$$2 \cdot (0.1 + 0.5 \cdot 10^{-3}) = 0.2 + 0.001 = 0.201 \text{ s} = 201 \text{ ms}$$



Esercizio 6 – Ritardi di Trasferimento



Esercizio 6 – Ritardi di Trasferimento

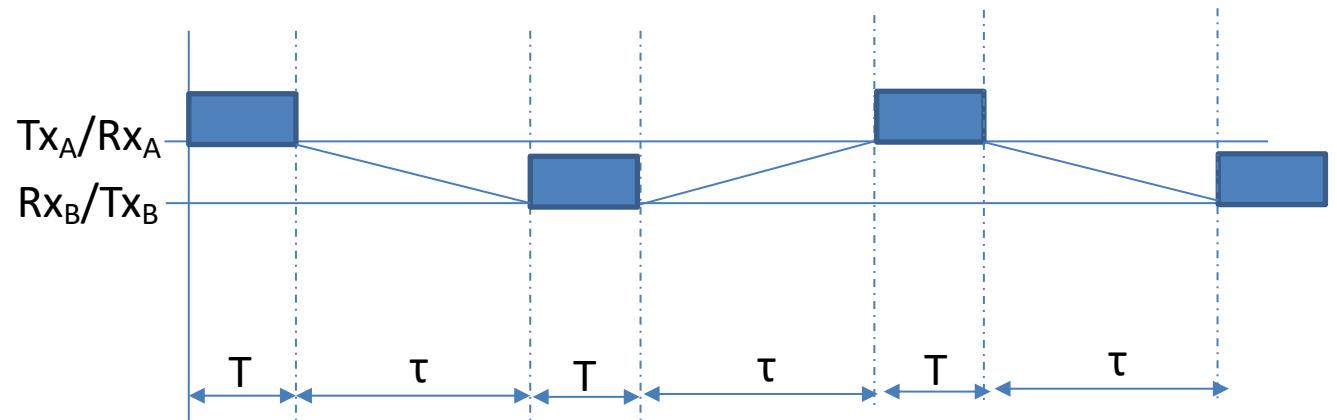
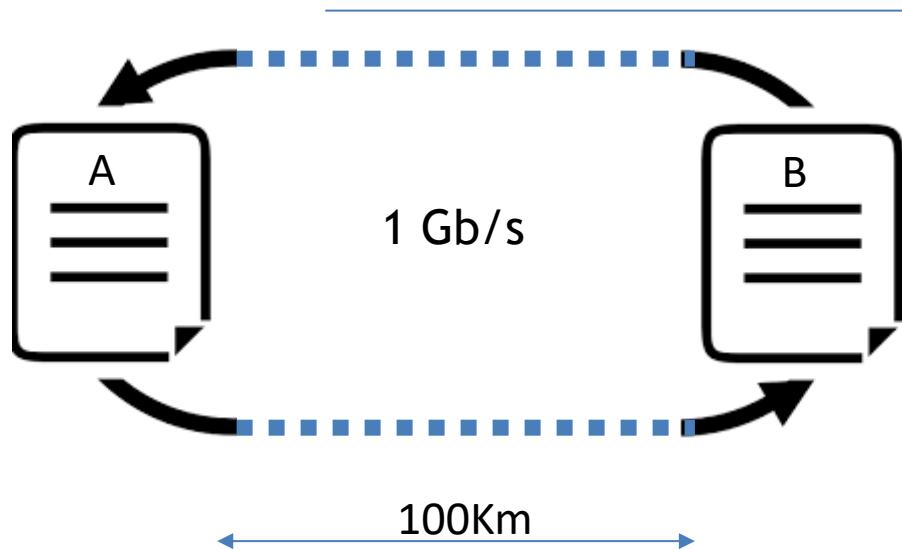
Due clienti di un internet service provider stanno utilizzando una connessione su un canale half-duplex lungo 100 km alla velocità di trasmissione di 1 Gb/s. I due utenti debbono trasferirsi l'un l'altro dei files, ed eseguono la trasmissione in *half-duplex* commutando il canale ogni 10000 bit ricevuti.

Quesiti:

- Assumendo che la commutazione del canale avvenga in un tempo nullo e che la velocità del segnale sia di 200000 km/s, a che velocità effettiva trasmettono i files?
- Si calcoli il totale tempo di trasmissione di un file di 1Gbyte.



Esercizio 6 – Ritardi di Trasferimento – soluzione 1/2



Tempo di Trasmissione

$$T = \frac{L}{R} = \frac{10000 \text{ bit}}{10^9 \text{ bit/s}} = 10^{-6} \text{ s} = 10 \mu\text{s}$$

Tempo di Propagazione

$$\tau = \frac{100 \text{ km}}{200 \cdot 10^3 \text{ km/s}} = 0.5 \cdot 10^{-3} = 500 \mu\text{s}$$

«A» trasmette il suo blocco di dati, aspetta che sia arrivato, commuta il canale ed aspetta il blocco trasmesso da «B»

Ogni utente trasmette 10000 bit in un periodo di $2 \times 510 \mu\text{s} = 1,02 \text{ ms}$

Il rate di trasferimento effettivo è: $R = 10 \text{ kbit}/1,02 \text{ ms} = 9,8 \text{ Mbit/s}$



Esercizio 6 – Ritardi di Trasferimento – soluzione 2/2

File da trasferire 1Gbyte = 8Gbit

Tempo complessivo necessario a trasferire il file da 1 Gbyte

$$\rightarrow T = \frac{8 \text{ Gbit}}{9.8 \cdot 10^{-3} \text{ Gbit/s}} = 0.816 \cdot 10^3 \text{ s} = 816 \text{ s}$$

Oppure, si considerino il numero complessivo di trasmissioni necessarie per inviare un file di 8Gbit

$$N = \frac{8 \cdot 10^9 \text{ bit}}{10 \cdot 10^3 \text{ bit}} = 0.8 \cdot 10^6 = 800000$$

Ogni trasmissione dura 1,02ms

$$\rightarrow T = 800000 \cdot 1.02 \text{ ms} = 816000 \text{ ms} = 816 \text{ s}$$



UTILIZZO DI TRACEROUTE



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Esercizio 8 – Utilizzo di Traceroute

TRACEROUTE è un Programma diagnostico: <http://www.traceroute.org>

The screenshot shows a web browser window with the URL [traceroute.org](http://www.traceroute.org). The page header includes a navigation bar with links like 'App', 'ARUBA', 'Importati', 'UOPEOPLE', 'ELSEVIER', 'Musica Classica', and 'UniBo'. Below the header is the site's logo 'traceroute.org' and a message from the maintainer, Thomas Kernen, asking for updates and corrections. The page is sponsored by 'kpn'. A large yellow arrow points from the text 'Selezionate il paese di appartenenza del server originante' to the list of countries.

By country:

Argentina	Armenia	Australia	Austria	Bangladesh	Belarus	Belgium	Bolivia
Brasil	Bulgaria	Canada	Chile	China	Colombia	Costa Rica	Croatia
Cyprus	Czech Republic	Denmark	Ecuador	Estonia	Faroe Islands	Finland	France
Germany	Greece	Grenada	Hong Kong	Hungary	Iceland	India	Indonesia
Iran	Ireland	Israel	Italy	Japan	Kenya	Korea	Kyrgyzstan
Latvia	Lithuania	Luxembourg	Macau	Malaysia	Malta	Mexico	The Netherlands
New Zealand	Pakistan	Paraguay	Philippines	Poland	Portugal	Romania	Russia
Saint Kitts	Saint Vincent	Singapore	Slovakia	South Africa	Spain	Sweden	Switzerland
Taiwan	Thailand	Togo	Turkey	Turkmenistan	Ukraine	United Kingdom	USA
...							

Selezionate il paese di appartenenza del server originante



Esercizio 8 – Utilizzo di Traceroute

Scegliete il server

Australia

- [Telstra \(AS1221\)](#)
- [Connect \(AS2764\)](#)
- [Optus \(AS4804\)](#)
- [ZipWorld \(AS7543\)](#)
- [TPG \(AS7545\)](#)
- [SPT \(AS9942\)](#)
- [Hafey.Org \(AS10145\)](#)
- [Fast Hit \(AS24381\)](#)

[To the top](#)

Austria

- [traceroute.at \(AS3248\)](#)
- [CoreTEC \(AS3248\)](#)
- [waldner.priv.at \(AS3248\)](#)
- [LeoXNet \(AS3330\)](#)
- [Tele2 \(AS8437\)](#)
- [Telekom Austria \(AS8447\)](#)
- [hotze.com \(AS8596\)](#)
- [Globedom \(AS12401\)](#)
- [net.toolkit \(AS12547\)](#)
- [max4eu \(AS20751\)](#)
- [nemox.net \(AS31394\)](#)
- [Crossip Communications GmbH \(AS44286\)](#)
- [MakeNewMedia Communications GmbH \(AS44765\)](#)



Esercizio 8 – Utilizzo di Traceroute

This traceroute commences from www.telstra.net, within AS 1221.

Enter the desired destination host.domain or IPv4 or IPv6 address: Inserite IP address che volete testare

There are other traceroute sites listed [here](#).

The traceroute CGI source can be found via:



Una volta inserito l'ip address dell'host di destinazione e dato invio, il server **AS1221** invia un certo numero di pacchetti **speciali** verso tale destinazione; questi pacchetti passano attraverso vari router. Quando un router ne riceve uno invia un breve messaggio che torna all'origine. **Questo messaggio contiene il nome e ip address del router che l'ha inviato.**



Esercizio 8 – Utilizzo di Traceroute

Prima di eseguire un comando traceroute, cerchiamo di comprendere il meccanismo di rete chiamato "**time to live**" (**TTL**).

TTL limita la durata della "vita" dei dati in una rete IP. Ad ogni pacchetto di dati viene assegnato un valore TTL. Ogni volta che un pacchetto di dati raggiunge un salto, il valore TTL viene diminuito di uno.

Un altro elemento chiave da comprendere è il "tempo di andata e ritorno" (**RTT – Round Trip Time**).

Traceroute garantisce che ogni salto nel percorso verso un dispositivo di destinazione rilasci un pacchetto e restituisca un messaggio di errore ICMP.

Ciò significa che traceroute può misurare il tempo che intercorre tra il momento in cui i dati vengono inviati e il momento in cui il messaggio ICMP viene ricevuto di nuovo per ogni hop, fornendo il valore RTT per ogni hop.



Esercizio 8 – Utilizzo di Traceroute

Per illustrare meglio questo aspetto, supponiamo di eseguire un traceroute e di specificare un massimo di 30 hop. Traceroute invierà pacchetti con un TTL pari a uno al server di destinazione. Il primo dispositivo di rete attraverso cui passano i dati ridurrà il TTL al valore zero e verrà inviato un messaggio che informa che i pacchetti sono stati ignorati. Questo ci dà l'RTT per il salto numero uno.

Da lì, i pacchetti di dati vengono inviati al server di destinazione con un TTL di due. Quando i pacchetti passano attraverso il primo hop, il TTL diminuisce a uno. Quando passano attraverso il secondo salto, diminuisce a zero. Il messaggio viene inviato di nuovo. Questo ci dà l'RTT per il salto numero due.

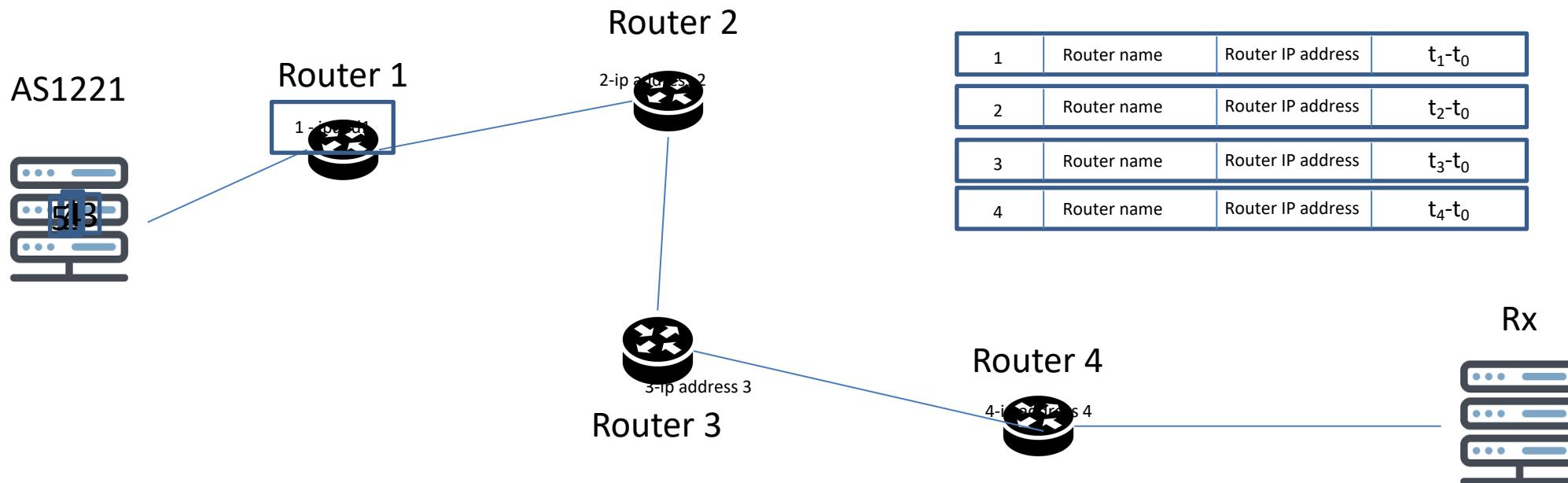
Questo processo si ripeterà fino a quando i pacchetti di dati **raggiungeranno il dispositivo di destinazione o raggiungeranno il numero massimo di hop**.

Alla fine di questo test, avremo il numero di hop al dispositivo di destinazione, la lunghezza RTT per ogni hop e il nome del dispositivo e l'indirizzo IP per ogni hop.



Esercizio 8 – Utilizzo di Traceroute

Supponiamo che tra il server Tx e la destinazione vi siano $N-1$ router (nel nostro esempio sono 4), l'AS1221 invia N pacchetti speciali nella rete (ossia 5). I pacchetti sono etichettati da 1 a N in sequenza. Quando l' i -esimo router riceve il pacchetto marcato con i , invece di mandarlo avanti, invia un messaggio verso l'origine. Quando l'host di destinazione riceve il pacchetto speciale N -esimo, anche esso restituisce un messaggio all'origine.



L'Host di origine quindi registra il tempo intercorso tra l'invio di un pacchetto e la ricezione del corrispondente messaggio di ritorno e memorizza anche il nome e l'indirizzo del router (o del destinatario) che restituisce il messaggio. In questo modo, l'AS1221 può ricostruire il percorso intrapreso dai pacchetti ed è inoltre in grado di determinare i ritardi di andata e ritorno di tutte le tratte.



Esercizio 8 – Utilizzo di Traceroute

Traceroute

Traceroute ripete l'operazione per tre volte consecutive, motivo per cui il risultato prevede 6 colonne e non 4.

This traceroute commences from www.telstra.net, within AS 1221.

Enter the desired destination host.domain or IPv4 or IPv6 address:

Valore dell'etichetta assegnata al pacchetto ossia il numero del router lungo il percorso	Nome del router	Indirizzo IP del router	Prima misura del ritardo di andata e ritorno				
1	gigabitethernet3-3.exi2.melbourne.telstra.net	(203.50.77.53)	0.368 ms	0.201 ms	0.239 ms		
2	bundle-ether3-100.win-core10.melbourne.telstra.net	(203.50.80.129)	2.609 ms	1.727 ms	2.240 ms		
3	bundle-ether12.ken-core10.sydney.telstra.net	(203.50.11.122)	12.856 ms	11.848 ms	12.610 ms		
4	bundle-ether1.ken-edge903.sydney.telstra.net	(203.50.11.173)	11.732 ms	11.723 ms	11.857 ms		
5	72.14.212.22 (72.14.212.22)	12.479 ms 12.474 ms 12.482 ms					

There are other traceroute sites listed [here](#).

The traceroute CGI source can be found via:



Se l'origine riceve meno di tre messaggi da ogni router
Traceroute pone un asterisco subito dopo il numero del router

Esercizio 8 – Utilizzo di Traceroute

Traceroute è presente come comando sia su Windows che su Linux e Mac

Su **Windows** è fruibile aprendo una shell di DOS,
E lanciando il comando «**tracert xxx.xxx.xxx.XXX**»



Su **Linux e Mac** è fruibile aprendo una Command Line Interface,
E lanciando il comando «**traceroute xxx.xxx.xxx.XXX**»



```
apirodd@ubuntunet2008:~$ traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  10.0.0.2 (10.0.0.2)  25.464 ms  25.822 ms  26.342 ms
 2  10.0.0.1 (10.0.0.1)  34.868 ms  35.367 ms  35.895 ms
 3  * * *
 4  172.17.54.56 (172.17.54.56)  37.117 ms  172.17.54.72 (172.17.54.72)  38.336 ms  172.17.54.58 (172.17.54.58)  38.979 ms
 5  172.17.54.154 (172.17.54.154)  40.417 ms  172.17.54.144 (172.17.54.144)  40.940 ms  172.17.54.146 (172.17.54.146)  41.496 ms
 6  172.19.177.36 (172.19.177.36)  42.593 ms  11.310 ms  172.19.177.42 (172.19.177.42)  33.373 ms
 7  172.19.177.2 (172.19.177.2)  43.640 ms  172.19.177.4 (172.19.177.4)  48.458 ms  172.19.177.2 (172.19.177.2)  46.018 ms
 8  etrunk49.milano50.mil.seabone.net (195.22.205.116)  46.524 ms  etrunk49.milano1.mil.seabone.net (195.22.205.98)  41.217 ms  44.553 ms
 9  74.125.51.148 (74.125.51.148)  50.457 ms  74.125.146.168 (74.125.146.168)  51.004 ms  72.14.195.206 (72.14.195.206)  49.847 ms
10  * 108.170.245.81 (108.170.245.81)  53.499 ms  108.170.245.65 (108.170.245.65)  52.928 ms
11  172.253.79.29 (172.253.79.29)  47.588 ms  dns.google (8.8.8.8)  51.396 ms  53.892 ms
```

```
Prompt dei comandi
Microsoft Windows [Versione 10.0.18363.592]
(c) 2019 Microsoft Corporation. Tutti i diritti sono riservati.

C:\Users\apirodd>tracert 8.8.8.8

Traccia instradamento verso dns.google [8.8.8.8]
su un massimo di 30 punti di passaggio:

 1  4 ms   5 ms   5 ms  DESKTOP-9SK0BHP [10.0.0.2]
 2  6 ms   6 ms   6 ms  10.0.0.1
 3  *      *      *      Richiesta scaduta.
 4  12 ms  12 ms  11 ms  172.17.54.60
 5  12 ms  12 ms  12 ms  172.17.54.144
 6  12 ms  12 ms  12 ms  172.19.177.42
 7  29 ms  25 ms  26 ms  172.19.177.4
 8  26 ms  24 ms  24 ms  etrunk49.milano50.mil.seabone.net [195.22.205.116]
 9  21 ms  21 ms  22 ms  72.14.221.64
10  25 ms  24 ms  24 ms  74.125.245.241
11  24 ms  23 ms  23 ms  172.253.79.35
12  24 ms  24 ms  24 ms  dns.google [8.8.8.8]

Traccia completata.

C:\Users\apirodd>
```



E-MAIL



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

E-Mail

La posta elettronica viene introdotta per la prima volta negli anni '60, tuttavia è diventata disponibile nella struttura attuale a partire dagli anni '70.

Protocolli utilizzati nei sistemi di posta elettronica

La comunicazione e-mail viene effettuata tramite tre protocolli in generale:

- IMAP
- POP
- SMTP



E-Mail - IMAP

IMAP è l'acronimo di **Internet Mail Access Protocol**.

Questo protocollo viene utilizzato durante la ricezione di un'e-mail.

Quando si utilizza IMAP:

- le e-mail saranno presenti nel server
- Le email non verranno scaricate nella casella di posta dell'utente
- Le email verranno successivamente eliminate dal server. Questo aiuta ad avere meno memoria utilizzata nel computer locale.



E-Mail - POP

POP è l'acronimo di Post Office Protocol.

Anche questo protocollo viene utilizzato per le e-mail in arrivo.

La differenza principale tra i due protocolli (imap e pop) è che POP scarica l'intera e-mail nel computer locale ed elimina i dati sul server una volta scaricati.

Questo è utile quando il server ha poca memoria libera.

La versione corrente di POP è POP3.



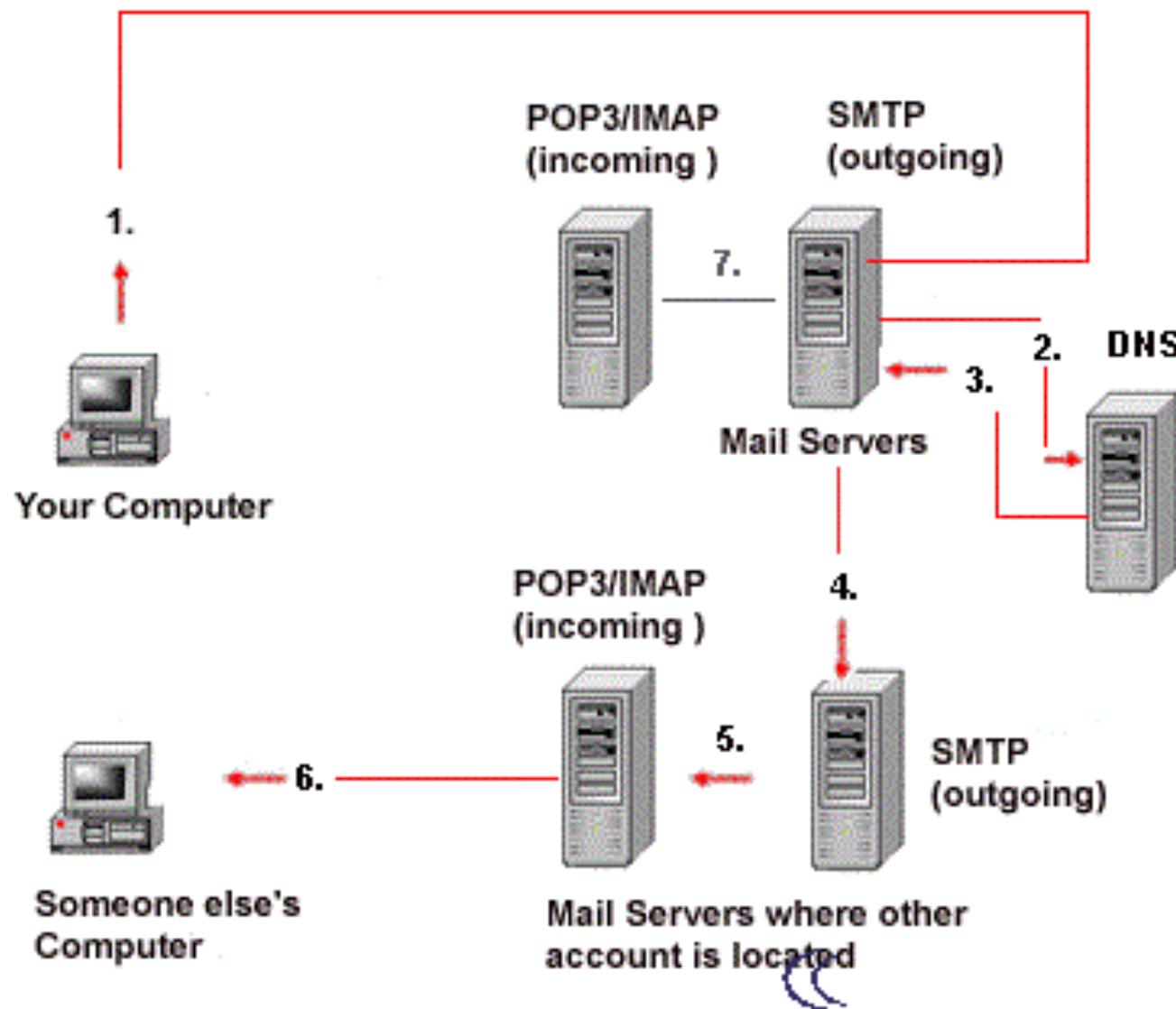
E-Mail - SMTP

SMTP sta per Simple Mail Transfer Protocol.

L'e-mail vengono inviate utilizzando questo protocollo.



E-Mail - architettura



E-Mail - Server

Un server di posta è un software.

Il software gestisce la ricezione delle e-mail in arrivo dagli utenti locali (persone all'interno dello stesso dominio) e dai mittenti remoti e inoltra le e-mail in uscita per la consegna.

Un computer su cui è installata un'applicazione di questo tipo può anche essere chiamato server di posta.

Nella figura precedente si vedono due server di posta.

I due server di posta utilizzati per le e-mail in uscita sono chiamati **MTA**, Mail Transfer Agent.

Gli altri due server di posta utilizzati per la posta in arrivo, che utilizzano i protocolli POP3/IMAP, sono chiamati **MDA**, Mail Delivery Agent.



E-Mail – HOW it Works

Il mittente inserisce l'indirizzo e-mail del destinatario insieme al messaggio utilizzando un'applicazione di posta elettronica (client SMTP). L'E-mail quindi verrà inviata all'MTA (Mail Transfer Agent). Questa comunicazione avviene tramite il protocollo SMTP.

Il passaggio successivo è la ricerca DNS. Il sistema invia una richiesta per conoscere l'MTA corrispondente del destinatario. Questo sarà fatto con l'aiuto del record MX. Nel DNS, in corrispondenza del dominio del destinatario, sarà presente un record MX (Mail Exchanger record). Questo specifica il server di posta di un dominio. Quindi, dopo la ricerca DNS, viene data una risposta al server di posta contenente l'indirizzo IP del server di posta del destinatario.

Il passaggio successivo è il trasferimento del messaggio tra i server di posta (server SMTP). Per questa comunicazione viene utilizzato il protocollo SMTP.

Ora, questo messaggio viene trasferito al MDA di destinazione e quindi viene trasferito al computer locale del destinatario. Qui possono essere utilizzati due protocolli. Se utilizziamo POP3, l'intera e-mail verrà scaricata sul computer locale e la copia sul server verrà eliminata. Se il protocollo utilizzato è IMAP, il messaggio e-mail viene archiviato nel server di posta stesso.

Se si è verificato un errore nell'invio dell'e-mail, le e-mail subiranno un ritardo. C'è una coda di posta in ogni server di posta. Queste mail saranno in sospeso nella coda della posta. Il server di posta continuerà a tentare di inviare nuovamente l'e-mail. Una volta che l'invio dell'e-mail fallisce in modo permanente, il server di posta può inviare un messaggio e-mail di ritorno all'indirizzo e-mail del mittente.



E-Mail – SMTP comandi e significato

Command	Action taken
HELO	Accepts the greeting from the client and stores it in <code>seen_greeting</code> . Sets server to base command mode.
EHLO	Accepts the greeting from the client and stores it in <code>seen_greeting</code> . Sets server to extended command mode.
NOOP	Takes no action.
QUIT	Closes the connection cleanly.
MAIL	Accepts the “MAIL FROM:” syntax and stores the supplied address as <code>mailfrom</code> . In extended command mode, accepts the RFC 1870 SIZE attribute and responds appropriately based on the value of <code>data_size_limit</code> .
RCPT	Accepts the “RCPT TO:” syntax and stores the supplied addresses in the <code>rcpttos</code> list.
RSET	Resets the <code>mailfrom</code> , <code>rcpttos</code> , and <code>received_data</code> , but not the greeting.
DATA	Sets the internal state to DATA and stores remaining lines from the client in <code>received_data</code> until the terminator <code>\r\n.\r\n</code> is received.
HELP	Returns minimal information on command syntax
VRFY	Returns code 252 (the server doesn’t know if the address is valid)
EXPN	Reports that the command is not implemented.



E-Mail – Esempio client SMTP

```
import smtplib
import email.utils
from email.mime.text import MIMEText

# Creiamo il messaggio mail
msg = MIMEText('Questo è il corpo del messaggio.')
msg['To'] = email.utils.formataddr(( 'DESTINATARIO', 'destinatario@example.com'))
msg['From'] = email.utils.formataddr(( 'MITTENTE', 'mittente@example.com'))
msg['Subject'] = 'Prova Invio Mail'

server = smtplib.SMTP('127.0.0.1', 1027)

server.set_debuglevel(True) # mostriamo le comunicazioni con il server
try:
    server.sendmail('mittente@example.com', [ 'destinatario@example.com'], msg.as_string())
finally:
    server.quit()
```



E-Mail – server SMTP

Documentazione la trovate: <https://docs.python.org/3/library/smtpd.html>

peer è l'indirizzo dell'host remoto

mailfrom è l'originatore della mail

rcpttos sono i destinatari della mail

```
from datetime import datetime
import asyncore
from smtpd import SMTPServer

class EmlServer(SMTPServer):
    no = 0
    def process_message(self, peer, mailfrom, rcpttos, data, mail_options=None, rcpt_options=None):
        filename = '%s-%d.eml' % (datetime.now().strftime('%Y-%m-%d-%H-%M-%S'),
                                   self.no)
        f = open(filename, 'wb')
        f.write(data)
        f.close
        print ('%s saved.' % filename)
        self.no += 1

def run():
    EmlServer(('127.0.0.1', 1027), None)
    try:
        asyncore.loop()
    except KeyboardInterrupt:
        pass

if __name__ == '__main__':
    run()
```

data è una stringa contenente il contenuto dell'e-mail



E-Mail – esempio

- Lanciate il codice smtp_server.py
- Lanciate il codice smtp_client.py

Nella console del client vedremo il seguente scambio di pacchetti

```
send: 'ehlo 87.43.168.192.in-addr.arpa\r\n'
reply: b'250-87.43.168.192.in-addr.arpa\r\n'
reply: b'250-SIZE 33554432\r\n'
reply: b'250-8BITMIME\r\n'
reply: b'250 HELP\r\n'
reply: retcode (250); Msg: b'87.43.168.192.in-addr.arpa\nSIZE 33554432\n8BITMIME\nHELP'
send: 'mail FROM:<mittente@example.com> size=256\r\n'
reply: b'250 OK\r\n'
reply: retcode (250); Msg: b'OK'
send: 'rcpt TO:<destinatario@example.com>\r\n'
reply: b'250 OK\r\n'
reply: retcode (250); Msg: b'OK'
send: 'data\r\n'
reply: b'354 End data with <CR><LF>.<CR><LF>\r\n'
reply: retcode (354); Msg: b'End data with <CR><LF>.<CR><LF>'
data: (354, b'End data with <CR><LF>.<CR><LF>')
send: b'Content-Type: text/plain; charset="utf-8"\r\nMIME-Version: 1.0\r\nContent-
Transfer-Encoding: base64\r\nTo: DESTINATARIO <destinatario@example.com>\r\nFrom: MITTENTE
<mittente@example.com>\r\nSubject: Prova Invio Mail\r\n\r\n
\nUXVlc3RvIMOoIGlsIGNvcnBvIGRlbCBtZXNzYWdnaw8u\r\n.\r\n.\r\n'
reply: b'250 OK\r\n'
reply: retcode (250); Msg: b'OK'
data: (250, b'OK')
send: 'quit\r\n'
reply: b'221 Bye\r\n'
reply: retcode (221); Msg: b'Bye'
```



Ritardi di Trasferimento



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Esercizio 7 – Ritardi di Trasferimento

Un sistema trasmissivo della velocità di 100 kb/s presenta una lunghezza di 600 km tra Tx ed Rx. Fra i due elementi di testa sono presenti due router, ciascuno dei quali presenta una latenza (latency), ossia un tempo di accodamento in uscita, pari al tempo di trasmissione, ed una capacità di 100kb/s. Si consideri trascurabile il tempo di elaborazione.

Quesiti:

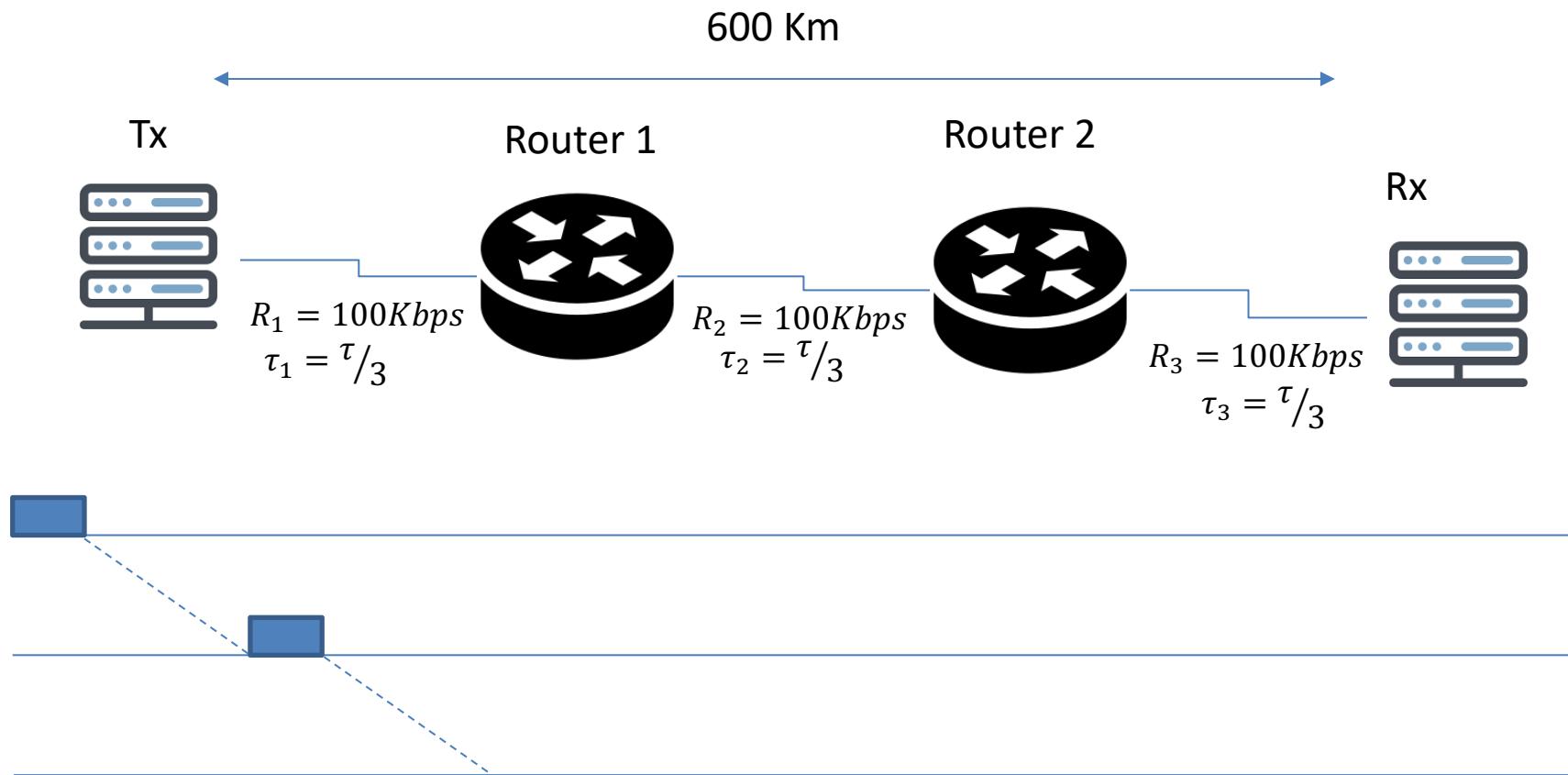
Si chiede di calcolare il ritardo totale nella trasmissione di un pacchetto di 3000 bit, assumendo un ritardo di propagazione di $5\mu s$: nei due casi in cui nei router si applichi:

- la modalità *store and forward*
- la modalità *cut-through*, considerando che la lunghezza dell'*header* sia di 200 bit (a parità di dimensione totale del pacchetto).



Esercizio 7 – Ritardi di Trasferimento – Soluzione 1/5

Scenario: **STORE & FORWARD** → il pacchetto deve essere completamente ricevuto prima di essere ritrasmesso



Esercizio 7 – Ritardi di Trasferimento – Soluzione 2/5

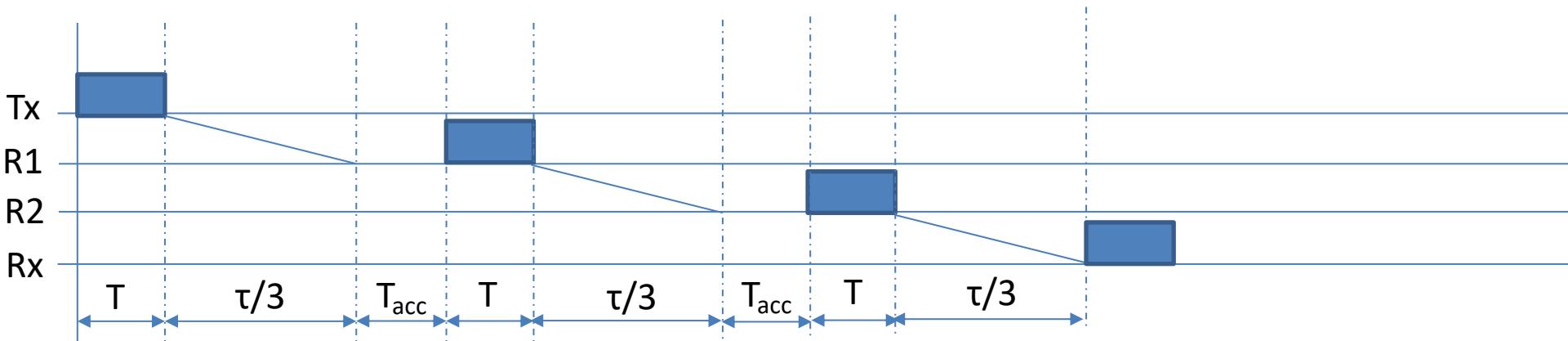
Scenario: STORE & FORWARD → il pacchetto deve essere completamente ricevuto prima di essere ritrasmesso

Osservazioni preliminari:

- Abbiamo 3 elementi trasmissivi quindi 3 tempi di trasmissione → $3 \cdot T = 3 \cdot \frac{L}{R} = 3 \cdot \frac{3000 \text{ bit}}{100 \cdot 10^3 \text{ bps}} = 3 \cdot 30 \cdot 10^{-3} \text{ s} = 90 \text{ ms}$
- Abbiamo 2 elementi che introducono latenza per accodamento e ritrasmissione → $2 \cdot T_{acc} = 2 \cdot T = 60 \text{ ms}$
- Abbiamo il tempo di propagazione fisico → $\tau = \frac{5 \mu\text{s}}{\text{Km}} \cdot 600 \text{ Km} = 3000 \mu\text{s} = 3.0 \text{ ms}$

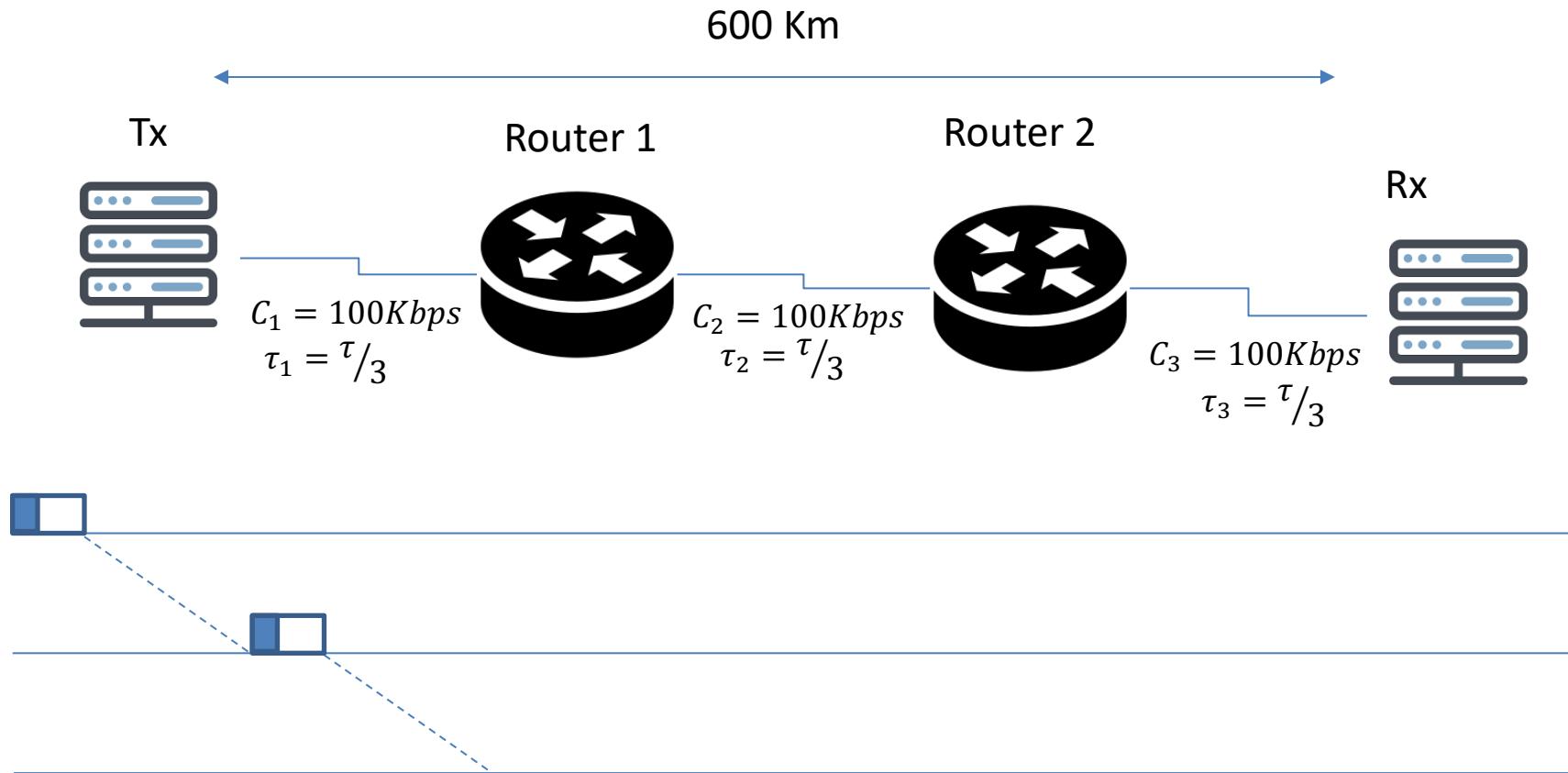
→ Sappiamo che il tempo di accodamento $L = T$, quindi facendo la somma otteniamo che:

$$T_{tot} = 3 \cdot T + 2 \cdot T + \tau = 5 \cdot T + \tau = 5 \cdot 30 \text{ ms} + 3 \text{ ms} = 153 \text{ ms}$$



Esercizio 7 – Ritardi di Trasferimento – Soluzione 3/5

Scenario: Cut & Through → il pacchetto viene ritrasmesso alla completa ricezione dell'header

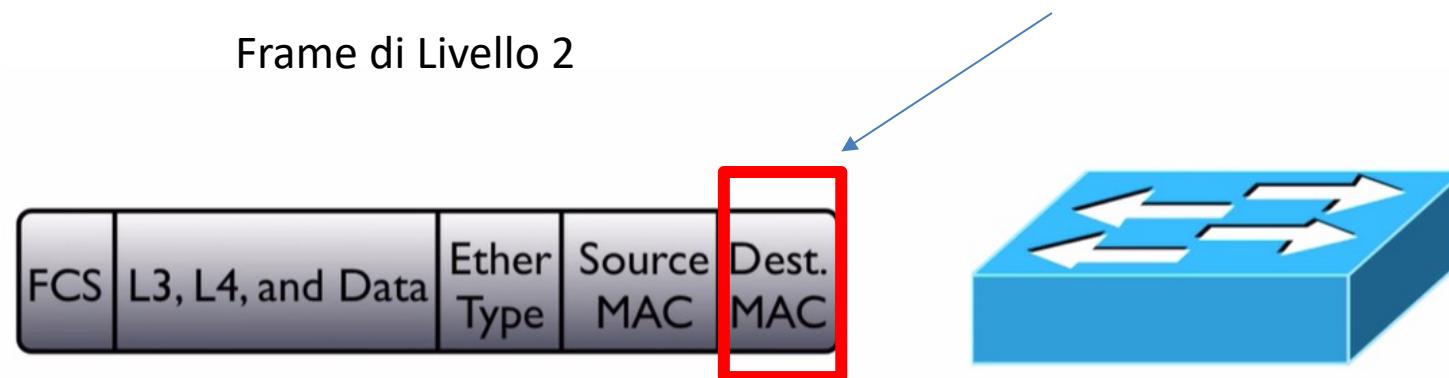


Esercizio 7 – Ritardi di Trasferimento – Soluzione 4/5

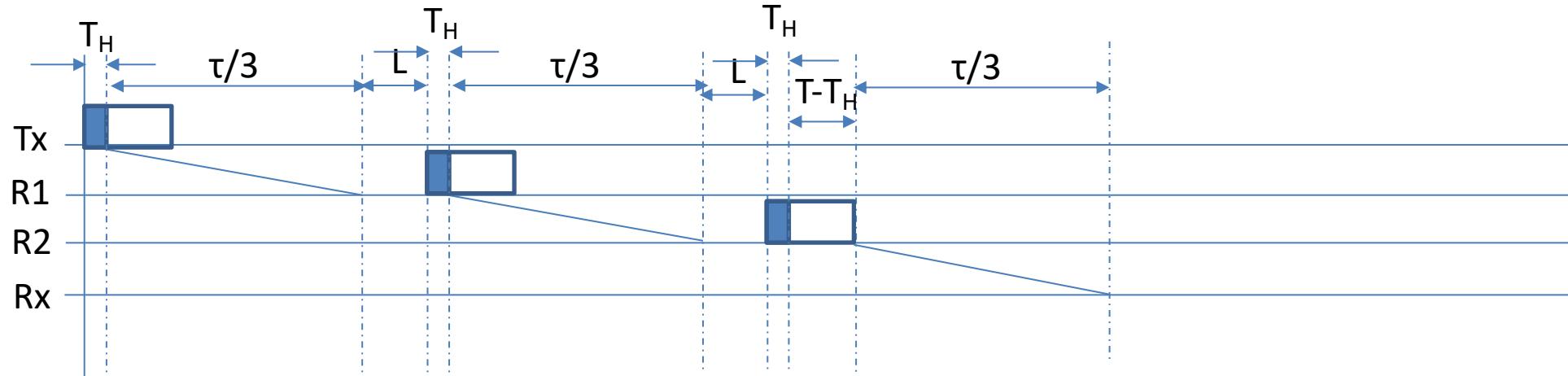
Scenario: Cut & Through → il pacchetto viene ritrasmesso alla completa ricezione dell'header

Osservazioni preliminari:

Nel caso *cut-through* il tempo di trasmissione dell'*header* si paga tre volte (l'*header* è sempre trasmesso con modalità *store and forward*) mentre il tempo di trasmissione del resto si paga una sola volta. Il tempo totale è dunque



Esercizio 7 – Ritardi di Trasferimento – Soluzione 5/5



$$T_{tot} = T_H + (T - T_H) + T_H + \tau$$

Annotations below the equation identify the components of the total transmission time:

- Tempo di trasmissione dell'Header (Header transmission time)
- Tempo di trasmissione del resto del pacchetto (Transmission time of the rest of the packet)
- Tempo di accodamento dell'Header (Header queuing time)
- Tempo di ritrasmissione del resto del pacchetto (Retransmission time of the rest of the packet)
- Tempo di ritrasmissione dell'Header (Header retransmission time)
- Tempo di accodamento dell'Header (Header queuing time)
- Tempo di ritrasmissione del resto del pacchetto (Retransmission time of the rest of the packet)
- Tempo di ritrasmissione dell'Header (Header retransmission time)

Below the annotations, the nodes are labeled: Server 1, router 1, and router 2.



Esercizio 7 – Ritardi di Trasferimento – Soluzione 5/5

$$\cancel{T_H} + \boxed{(T - T_H)} + \cancel{T_H} + \boxed{(T - T_H)} + \circled{T_H} + \cancel{T_H} + \boxed{(T - T_H)} + \circled{T_H} + \tau$$

$$T = \frac{L}{R} = \frac{3000 \text{ bit}}{100 \cdot 10^3 \text{ bps}} = 30 \cdot 10^{-3} \text{ s} = 30 \text{ ms}$$

$$\tau = \frac{5 \mu\text{s}}{\text{Km}} \cdot 600 \text{ Km} = 3000 \mu\text{s} = 3.0 \text{ ms}$$

$$T_{tot} = \boxed{3 \cdot T} + 2 \cdot T_H + \tau = 90 \text{ ms} + 4 \text{ ms} + 3 \text{ ms} = 97 \text{ ms}$$

$$T_H = \frac{200 \text{ bit}}{100 \cdot 10^3 \text{ bit/sec}} = 2 \cdot 10^{-3} \text{ sec} = 2 \text{ ms}$$

