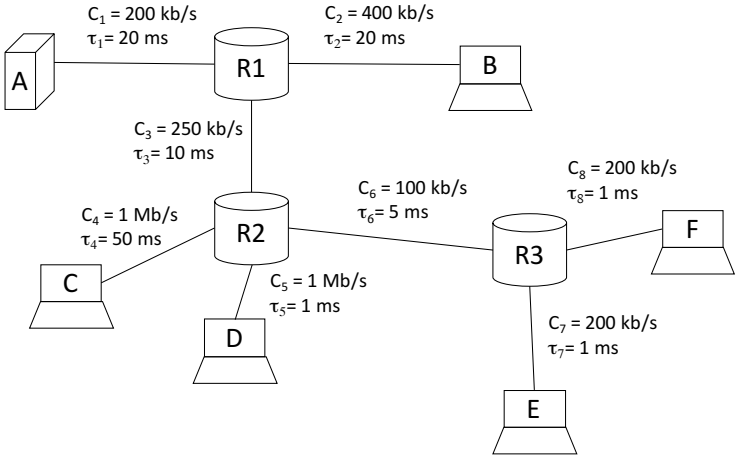
Prova in itinere – 4 Maggio 2016

Cognome	
Nome	
Matricola	

Tempo complessivo a disposizione per lo svolgimento: 1h45m
Usare lo spazio dopo ogni Esercizio/Quesito per la risposta.

Es1 (10 pt)	Es2 (8 pt)	Ques (9 pt)	Lab (6pt)

Esercizio 1 (10 punti)



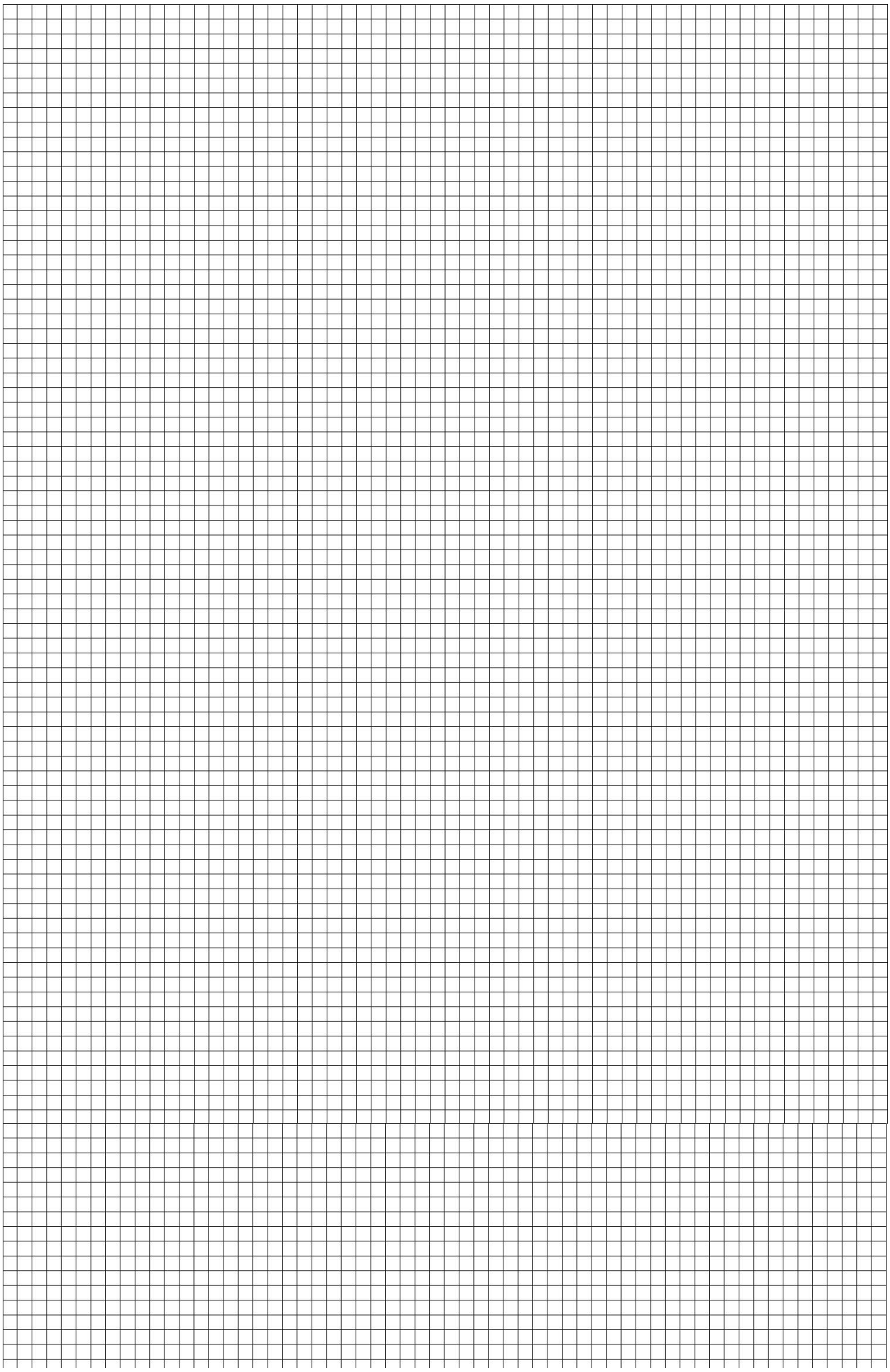
a) Una connessione TCP tra l'host A e l'host B nella rete in figura è caratterizzata dai seguenti parametri:

- Link bidirezionali e simmetrici
- MSS = 200 byte
- Lunghezza header complessivo (tutti i livelli), H = 50 byte
- Lunghezza ACK e segmenti di apertura, L_{ACK} = 250 byte
- RCWND = 1000 byte, SSTHRESH = 1600 byte

a.1) Si *calcoli* il tempo necessario a trasferire un file di dimensione F = 5 kbyte (dall'apertura della connessione alla ricezione dell'ultimo ACK)

a.2) Si *indichi* il rate medio di trasferimento del file da A a B

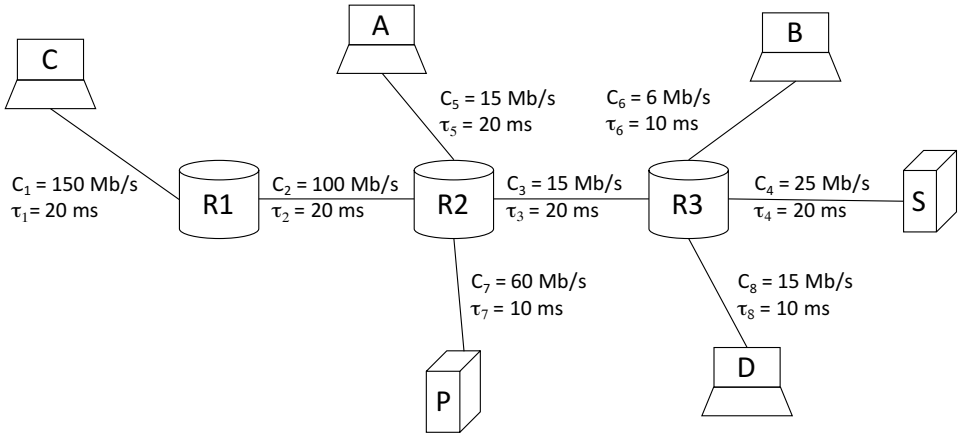
b) Nella rete a commutazione di pacchetto in figura, al tempo t=0 sono presenti 5 pacchetti in A diretti rispettivamente alle seguenti destinazioni: C, D, E, F, E. *Calcolare* l'istante di fine ricezione degli ultimi 3 pacchetti a destinazione assumendo che i pacchetti abbiamo le seguenti dimensioni: pacchetti verso C, L_C = 375 byte; pacchetti verso D, L_D = 250 byte; pacchetti verso E, L_E= 375 byte; pacchetti verso F, L_F= 125 byte.



Esercizio 2 (8 punti)

Nella rete in figura sono rappresentati 4 client (A, B, C e D), 3 router (R1, R2 e R3), un server HTTP (S) e un proxy HTTP (P). Il client C vuole trasferire un documento formato da una pagina HTML di dimensione L_{HTML} = 30 kbyte che richiama 11 oggetti di dimensione L_{OGG} = 75 kbyte. Nella rete sono presenti anche 4 flussi interferenti di lunga durata: 2 da A a D, 2 da A a B. Si supponga che: i) i messaggi di apertura della connessione e di richiesta HTTP siano di lunghezza trascurabile, ii) la connessione TCP tra proxy HTTP e server HTTP sia sempre aperta e non occorra mandare una richiesta di apertura.
Si calcoli:

a) il tempo di trasferimento del documento nel caso in cui il client C senza proxy configurato apra in parallelo in modalità non-persistente tutte le connessioni TCP necessarie
b) il tempo di trasferimento del documento nel caso in cui il client C con proxy configurato apra un'unica connessione TCP persistente per scaricare tutti gli oggetti, ipotizzando che la pagina HTML e solo i primi 6 degli 11 oggetti siano presenti nella cache del proxy
c) nel caso b), il numero minimo di oggetti che occorre trovare nella cache del proxy per avere un tempo di trasferimento minore di 2 s.



Quesiti (9 punti)

Q1

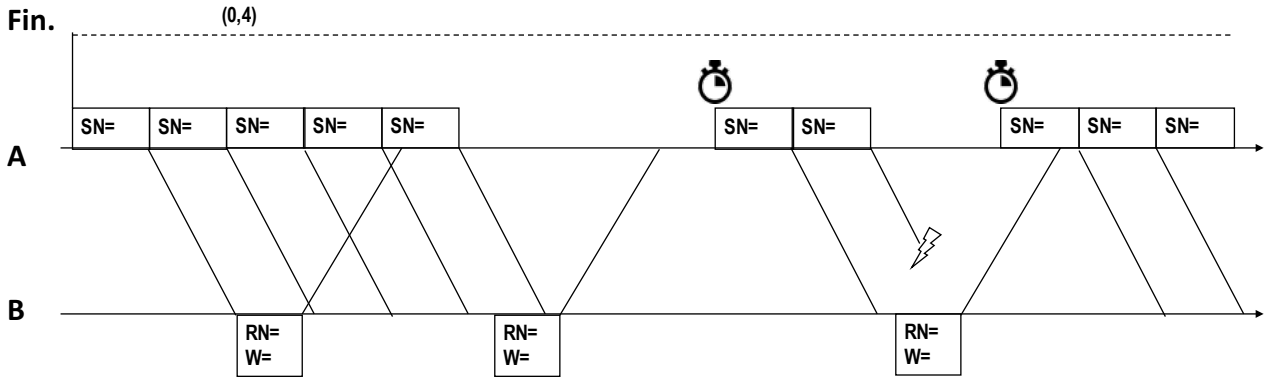
Nell'ambito del servizio DNS, *si illustrino* le differenze tra la risoluzione di un nome simbolico (*www.polimi.it*) in modalità iterativa ed in modalità ricorsiva.

Q2

Si completi la figura in cui è rappresentato un colloquio governato dai meccanismi Go-Back-N e controllo di flusso con il campo W. Il buffer in ricezione ha una capacità massima di 2 pacchetti, l'applicazione svuota il buffer ad ogni lettura ed invia un ACK. Si assuma una finestra iniziale al trasmettitore di 5 pacchetti.

Nel completamento si indichino con chiarezza:

- i valori corretti di SN, RN e W,
- gli estremi della finestra al trasmettitore, precisando gli instanti in cui essi cambiano sulla linea tratteggiata,
- i pacchetti accettati ed eventuali pacchetti scartati



Q3

Durante una sessione TCP, l'algoritmo di Jacobson stima valor medio e deviazione standard del RTT come $SRTT^0 = 10$ ms e $SDEV^0 = 2$ ms. I due segmenti successivi registrano un RTT di $RTT^1 = 16$ ms e $RTT^2 = 32$ ms. *Si indichino* nella tabella i valori di SRTT, SDEV, DEV e del Timeout alla ricezione di ciascuno dei due segmenti considerando $(1 - \alpha) = 7/8$ come peso della stima precedente di RTT e $(1 - \beta) = 3/4$ come peso della stima precedente di SDEV.
Si usi la tabella per indicare i risultati finali e lo spazio sottostante per mostrare i conti fatti.

	RTT	**SRTT**	**DEV**	**SDEV**	**Timeout**
$SRTT^0 = 10$ $SDEV^0 = 2$	$RTT^1 = 16$	$SRTT^1 =$	$DEV^1 =$	$SDEV^1 =$	$T^1 =$
	$RTT^2 = 32$	$SRTT^2 =$	$DEV^2 =$	$SDEV^2 =$	$T^2 =$

Laboratorio (6 punti)

Q1

L'utente con indirizzo di posta *studente@labfir.lan* si connette al proprio server POP per scaricare la posta ricevuta con credenziali username: *studente* e password: *bravo*. *Si completi* la seguente sequenza di comandi

```
user@pc:~$ telnet localhost ____
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
+OK Hello Jedi, may the Force be with
you!

_____ _________
+OK

_____ ______
+OK Logged in.

_____
1 370
2 458
3 498
.

_____ _
+OK 498 octects
Return-Path: professore@corsofir.it
Delivered-To: ____________________
Received: from server_corsofir.it
(localhost [127.0.0.1])
       by www.labfir.lan (Postfix) with
SMTP id E99C683BB
       for <destination>; Thu, 28 Apr
2016 13:55:33 +0200 (CEST)
Subject: Esame FIR
Message-Id: 201604.E99@www.labfir.lan
Date: Thu, 28 Apr 2016 13:55:33 +0200
(CEST)
From: professore@corsofir.it

In teoria non c'è differenza tra teoria e
pratica, in pratica c'è.
.

_____ _
+OK Marked to be deleted.

_____
1 370
2 458
.

_____
+OK

_____
1 370
2 458
3 498
.

_____
+OK Logging out.
Connection closed by foreign host.
```

Q2

Si vuole scrivere un'applicazione client/server UDP per il calcolo dei quadrati. Il client chiede all'utente di inserire un numero, il server risponde con il quadrato del numero e infine il client stampa la risposta. (**Hint!** Utilizzare le funzioni int() e str(): int() converte una stringa ricevuta in un numero intero per effettuare operazioni aritmetiche, str() converte un intero in una stringa da trasmettere). *Si completi* il codice del server

UDP client

```
from socket import *

serverName = 'localhost'
serverPort = 12000

clientSocket = socket(AF_INET, SOCK_DGRAM)

message = raw_input('Inserisci un numero')
clientSocket.sendto(message, (serverName,
serverPort))

reply, serverAddress =
clientSocket.recvfrom(2048)
print reply

clientSocket.close()
```

UDP server

```
from socket import *

serverPort = 12000

serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))

print "The server is ready to receive"

...DA COMPLETARE
```

Q3

Data la seguente coppia di script si indichi:

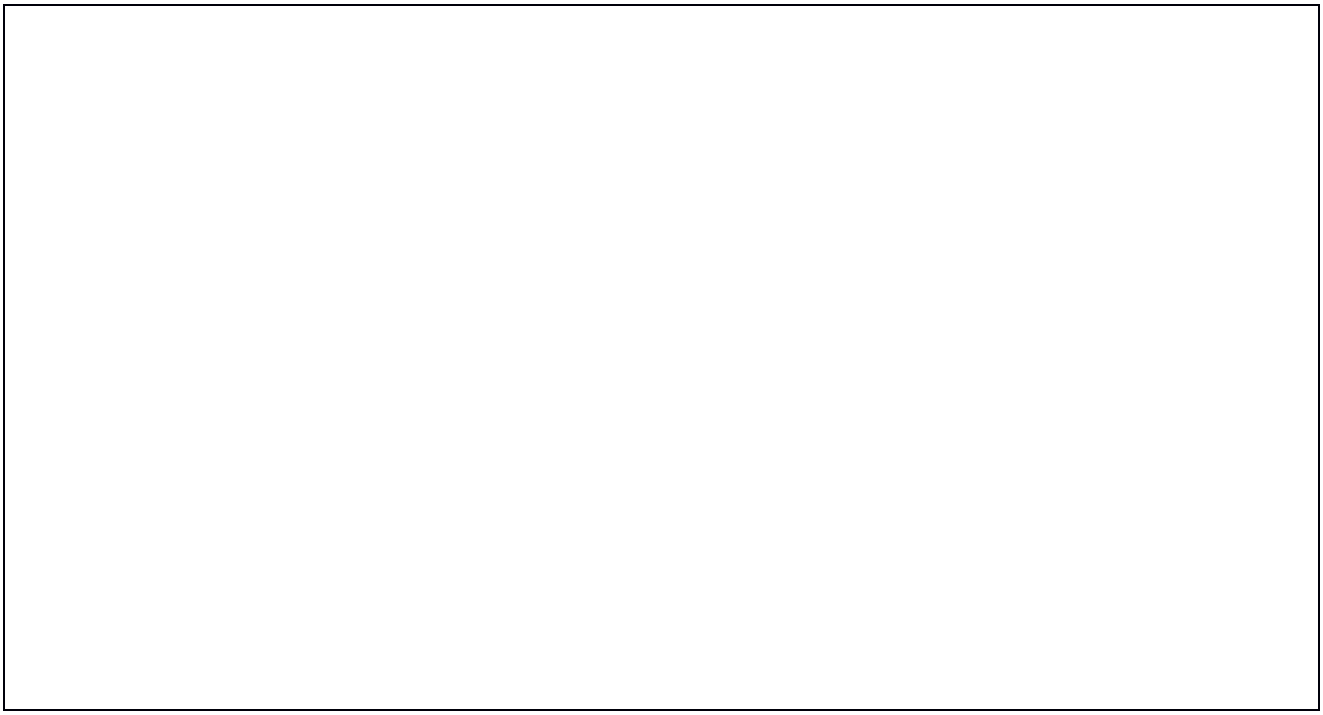
a) quale è il client e quale il server
b) cosa fa l'applicazione implementata

Script A

```
from socket import *

serverName = 'localhost'
serverPort = 12000

clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))

clientSocket.send('Ho vinto?')

reply = clientSocket.recv(1024)
print 'From Server:', reply

clientSocket.close()
```

Script B

```
from socket import *

serverPort = 12000

serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
client_num = 0
while True:
    connectionSocket,  clientAddress =
serverSocket.accept()
    client_num = client_num + 1
    print "Connection from: ", clientAddress
    sentence = connectionSocket.recv(1024)
    if client_num == 1337:
        connectionSocket.send('Hai vinto!')
        client_num = 0
    else:
        connectionSocket.send('Ritenta! :(')
connectionSocket.close()
```



Codice esercizi laboratorio

UDP client

```
from socket import *
serverName = 'localhost'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message, (serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print modifiedMessage
clientSocket.close()
```

UDP server

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
while 1:
    message, clientAddress = serverSocket.recvfrom(2048)
    print "Datagram from: ", clientAddress
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)
```

UDP error management

```
from socket import *
serverName = 'localhost'
serverPort = 12001
clientSocket = socket(AF_INET, SOCK_DGRAM)
clientSocket.settimeout(5)
message = raw_input('Input lowercase sentence:')
try:
    clientSocket.sendto(message, (serverName, serverPort))
    modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
    # in case of error blocks forever
    print modifiedMessage
except error, v:
    print "Failure"
    print v
finally:
    clientSocket.close()
```

TCP client

```
from socket import *
serverName = 'localhost'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

TCP server

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))
serverSocket.listen(1)
```

```
print 'The server is ready to receive'
while True:
    connectionSocket,  clientAddress = serverSocket.accept()
    print "Connection form: ", clientAddress
    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

TCP client persistent

```
from socket import *
serverName = 'localhost'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
while True:
    sentence = raw_input('Input lowercase sentence ( . to stop):')
    clientSocket.send(sentence)
    if sentence == '.':
        break
    modifiedSentence = clientSocket.recv(1024)
    print 'From Server:', modifiedSentence
clientSocket.close()
```

TCP server persistent

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))
serverSocket.listen(1)
while True:
    print 'The server is ready to receive'
    connectionSocket,  clientAddress = serverSocket.accept()
    print "Connection form: ", clientAddress
    while True:
        sentence = connectionSocket.recv(1024)
        if sentence == '.':
            break
        capitalizedSentence = sentence.upper()
        connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

TCP auto client

```
from socket import *
import time
serverName = 'localhost'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
for a in range(100):
    clientSocket.send('A')
time.sleep(1)
clientSocket.send('.')
#clientSocket.recv(1024)
clientSocket.close()
```

TCP auto server

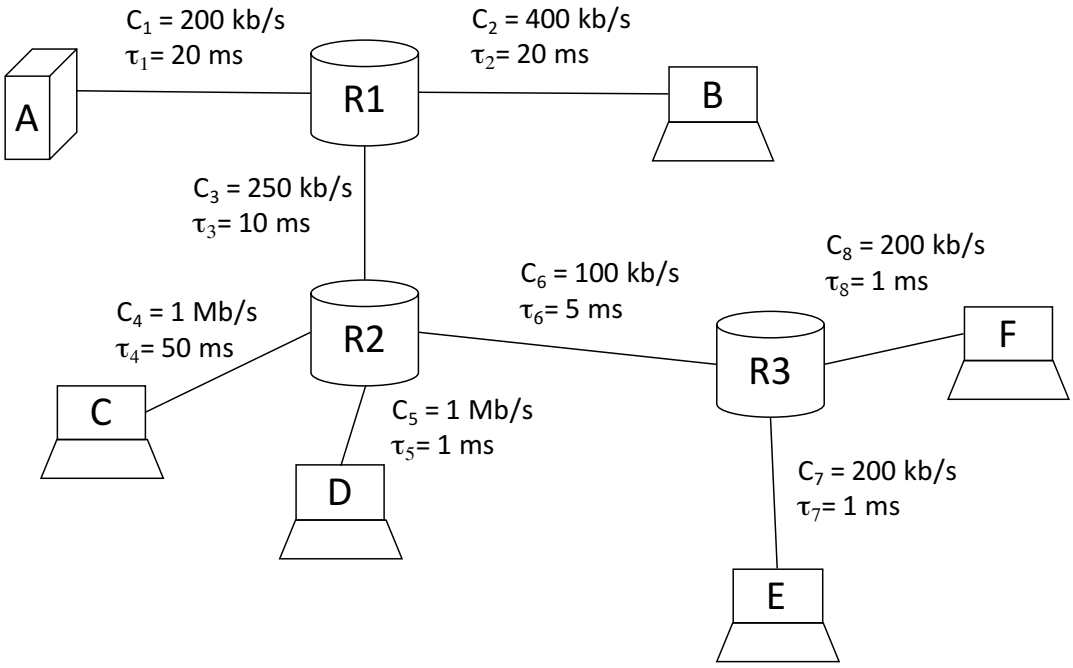
```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))
serverSocket.listen(1)
```

```
while True:
    print 'The server is ready to receive'
    connectionSocket,  clientAddress = serverSocket.accept()
    print "Connection form: ", clientAddress
    while True:
        sentence = connectionSocket.recv(1024)
        if sentence == '.':
            break
        print len(sentence)
#        connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

TCP server thread

```
from socket import *
import thread
def handler(connectionSocket):
    while True:
        sentence = connectionSocket.recv(1024)
        if sentence == '.':
            break
        capitalizedSentence = sentence.upper()
        connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
serverSocket.bind(('', serverPort))
serverSocket.listen(1)
while True:
    print 'The server is ready to receive'
    newSocket,  addr = serverSocket.accept()
    thread.start_new_thread(handler, (newSocket,))
```

Rete Esercizio 1



Rete Esercizio 2