Esame Completo - 26 Luglio 2017

Cognome	
Nome	
Matricola	

Tempo complessivo a disposizione per lo svolgimento: 2 ore 15 minuti
Si usi lo spazio bianco dopo ogni esercizio per la risoluzione

E1	E2	E3	Quesiti	Lab

1 - Esercizio (8 punti)

La rete di un ISP è riportata in figura. L'ISP possiede lo spazio di indirizzamento: 2.14.26.0/23 Definire un piano di indirizzamento in grado di supportare il numero di *host* indicato nella figura.

a) Indicare le sottoreti IP graficamente nella figura, mettendo in evidenza i confini tra le reti IP ed assegnando una lettera identificativa a ciascuna rete. Assegnare le lettere in ordine alfabetico iniziando dalla rete più grande e procedendo per dimensione decrescente. Per ciascuna sottorete definire l'indirizzo di rete, la *netmask* (in formato decimale puntato), e l'indirizzo di broadcast diretto, usando la tabella 1. Assegnare gli indirizzi alle sottoreti a partire da quelli più bassi del blocco 2.14.26.0/23.

b) Scrivere nella tabella 2 la tabella di instradamento del router R5 nel modo più compatto possibile dopo aver assegnato opportunamente degli indirizzi ai router a cui R5 è connesso.

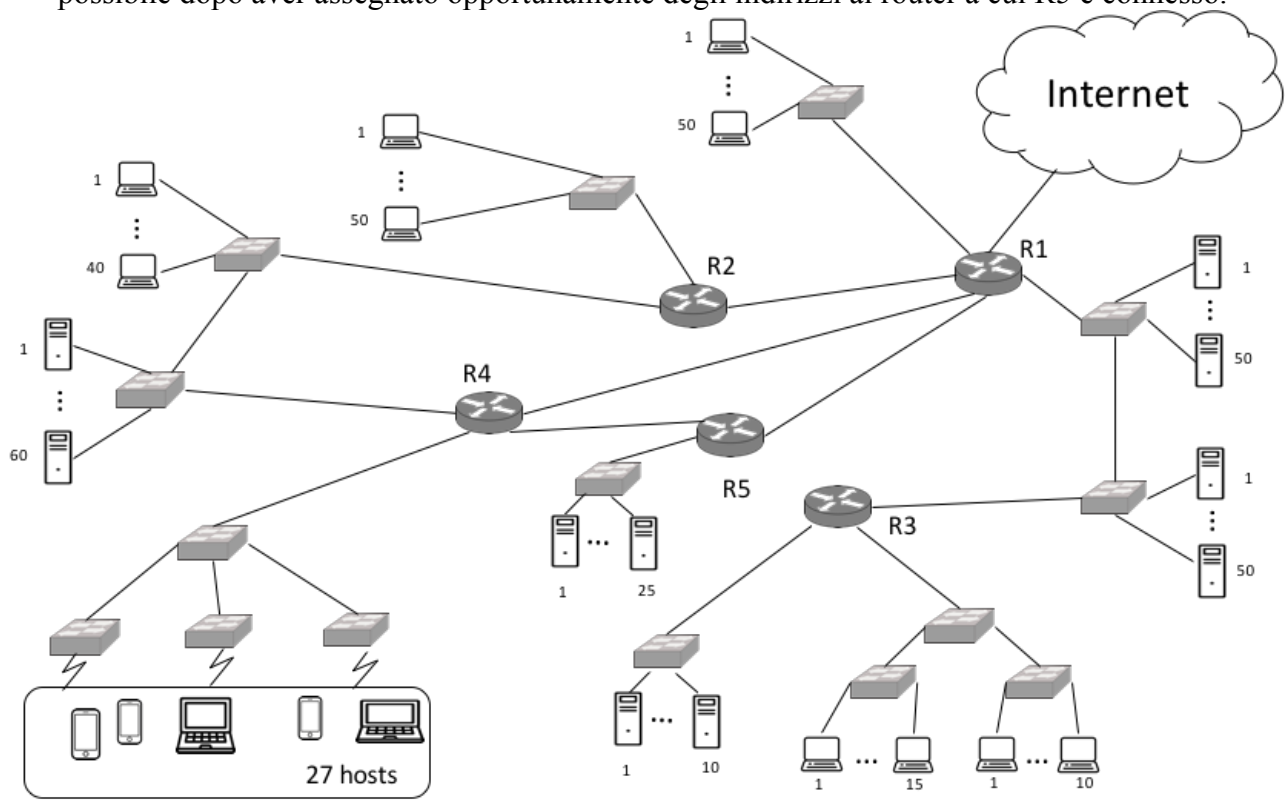
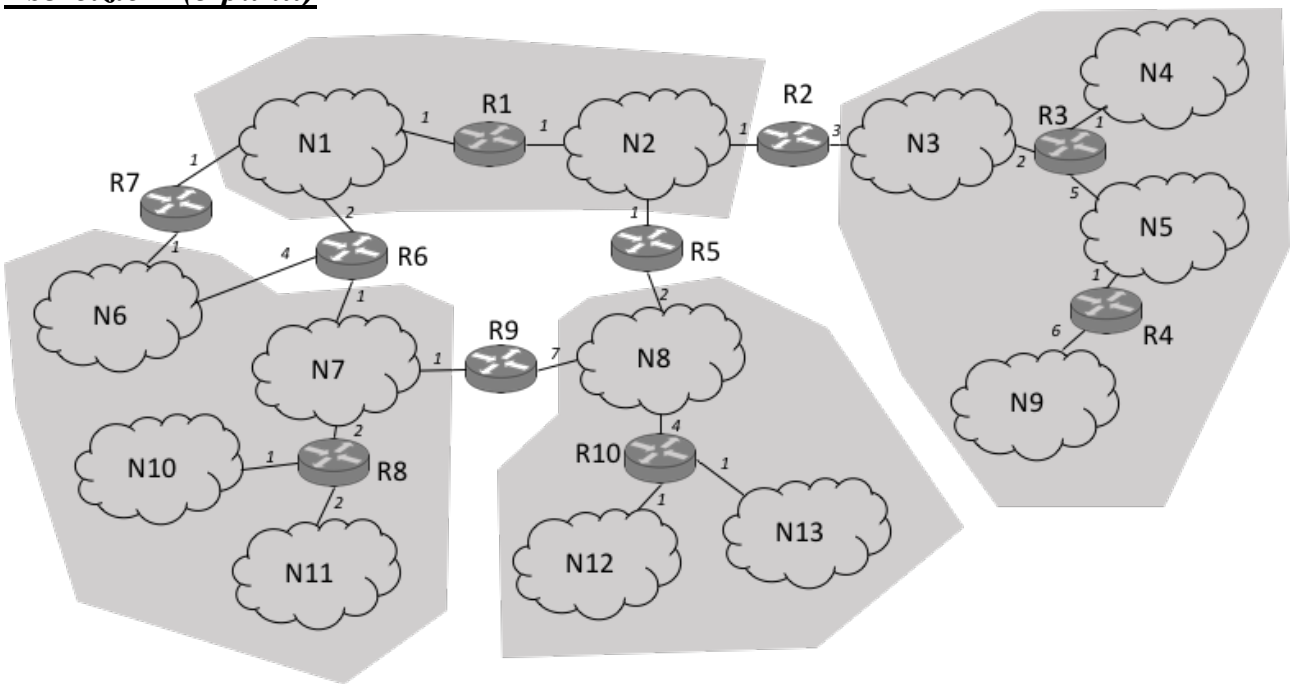


Tabella 1

Rete	Indirizzo di rete	Netmask	Ind. broadcast diretto
A			
B			

Tabella 2

Esercizio 2 (5 punti)



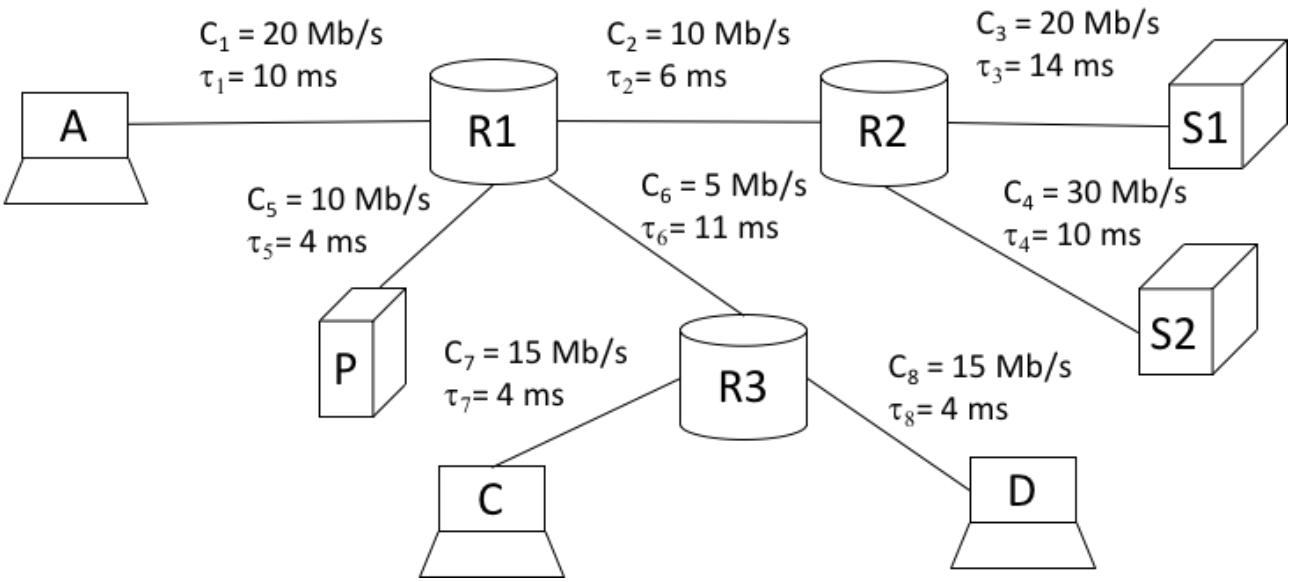
Si consideri la rete in figura dove sono indicati *router*, reti e costo associato alle interfacce dei *router*. Si supponga di utilizzare il protocollo di *routing* OSPF. Si divida come mostrato in figura la rete in 4 aree e si disegnino i grafi che rappresentano la rete vista dal *router* R1, R3, ed R5.

Rete vista da R1

Rete vista da R3

Rete vista da R5

Esercizio 3 (5 punti)



Si assuma A, P e S1 siano rispettivamente client, proxy e server HTTP. A chiede un contenuto web costituito da un documento HTML base di 12.5 KB e 6 immagini di 250 KB ciascuna.
Si calcoli il tempo di trasferimento necessario, nei seguenti casi:

a) A utilizza il proxy P che ha sia il documento base che le immagini con connessione HTTP persistente (una singola connessione)
b) A utilizza il proxy P che ha solo il documento base e a sua volta chiede le immagini al server S1; sia A che P usano connessione HTTP persistente (una singola connessione)
c) A utilizza il proxy P che ha solo il documento base e a sua volta chiede le immagini al server S1; A usa connessione HTTP persistente (una singola connessione), mentre P usa connessione HTTP non persistente (con trasmissione in parallelo delle immagini)

Per tutti casi si assuma che:

- Per tutto il tempo del trasferimento siano presenti dei flussi TCP interferenti: 2 flussi tra C ed S2, e due flussi tra D ed S2
- i messaggi di apertura connessione TCP e richieste HTTP sono di lunghezza trascurabile,
- nel calcolo dei **Round Trip Time (RTT)** si trascurino tempi di accodamento nei nodi,
- la trasmissione dei file avviene a un **ritmo medio di trasmissione** pari al valore di condivisione equa delle risorse **R**. Si indichino con **T** i **tempi di trasmissione dei messaggi applicativi** al ritmo R
- per le variabili RTT, R, T si utilizzino i pedici $_{\mathbf{P}}$ o $_{\mathbf{S}}$ per indicare se ci si riferisce al flusso A-P o al flusso P-S1. Per le variabili R e T si usino gli ulteriori pedici $_{\mathbf{HTML}}$ e $_{\mathbf{OBJ}}$ per indicare se ci si riferisce al file del documento HTML o ai file delle 6 immagini.

		Flusso A-P	Flusso P-S1	Unità mis.
	RTT			
Caso a	$Bott.neck_{HTML}$			*Indice link*
	R_{HTML}			
	T_{HTML}			
	$Bott.neck_{OBJ}$			*Indice link*
	R_{OBJ}			
	T_{OBJ}			
Caso b	$Bott.neck_{HTML}$			*Indice link*
	R_{HTML}			
	T_{HTML}			
	$Bott.neck_{OBJ}$			*Indice link*
	R_{OBJ}			
	T_{OBJ}			
Caso c	$Bott.neck_{HTML}$			*Indice link*
	R_{HTML}			
	T_{HTML}			
	$Bott.neck_{OBJ}$			*Indice link*
	R_{OBJ}			
	T_{OBJ}			

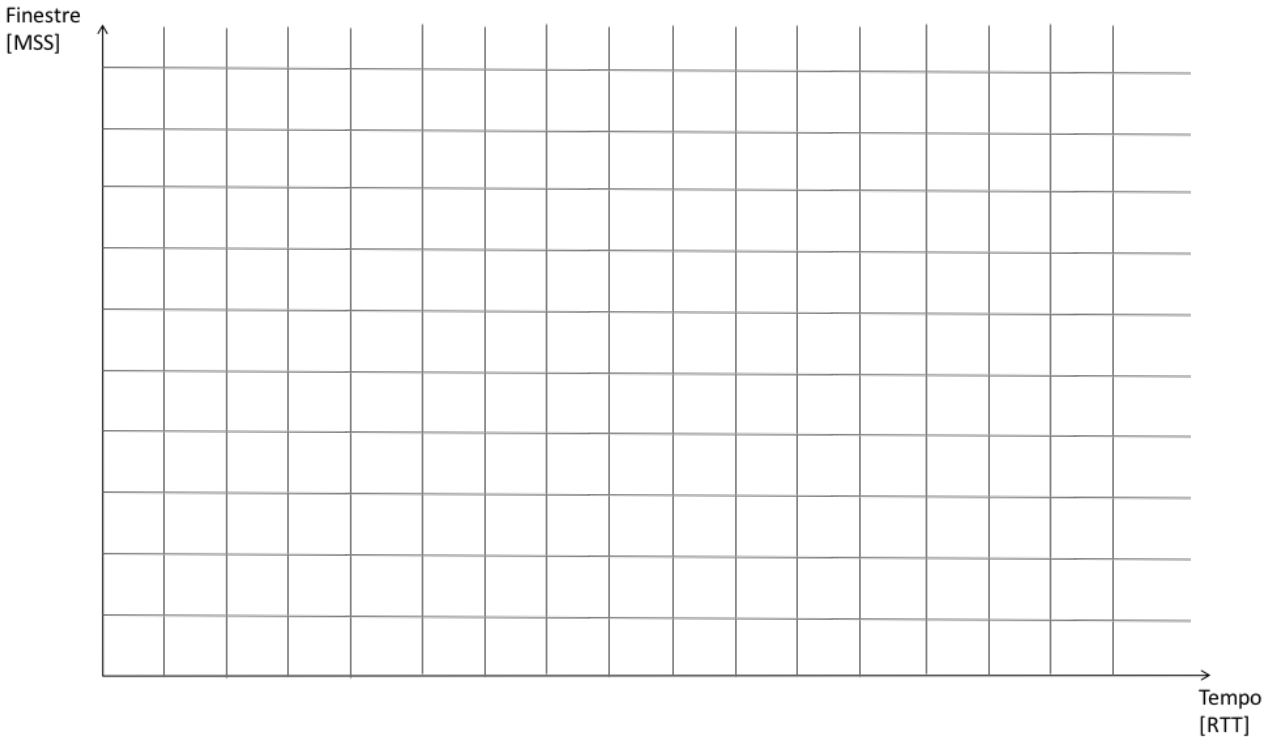
Caso a) tempo di trasferimento (fornire l'espressione simbolica prima del risultato numerico)
$\mathbf{T_{tot}}$ =

Caso b) tempo di trasferimento (fornire l'espressione simbolica prima del risultato numerico)
$\mathbf{T_{tot}}$ =

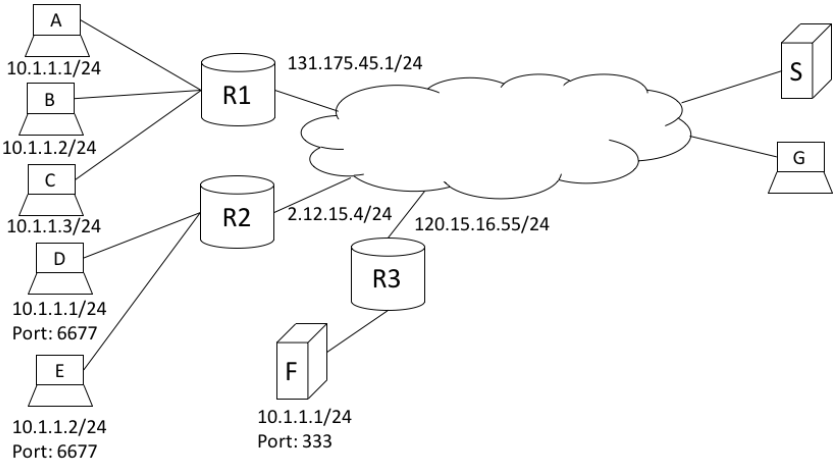
Caso c) tempo di trasferimento (fornire l'espressione simbolica prima del risultato numerico)
$\mathbf{T_{tot}}$ =

4–Domande (9 punti)

D1 - Una connessione TCP è usata per trasmettere un file da *120 KB* utilizzando i seguenti parametri: MSS = *1500 B,* RTT = *100 ms,* timeout pari a 3 RTT. Si assuma che le condizioni iniziali delle finestre siano: RCWND = *12 KB,* SSTHRESH = *6 KB,* CWND = 1*500 B.*
Si assuma inoltre che si verifichi un errore sulla connessione all'istante *1 s* (tutti i segmenti in trasmissione vengano persi).
Si tracci l'andamento nel tempo usando il diagramma riportato sotto di: CWND, SSTHRESH, e RCWND. Si calcoli il tempo di trasmissione del file utilizzando multipli di RTT come base temporale.



D2 - Si consideri la rete in figura dove nei router R1, R2 ed R3 sono configurati dei NAT.



a) Le connessioni degli host A, B e C verso il server S sono visti dal server con indirizzi sorgente rispettivamente 131.175.21.1, 131.175.21.2, 131.175.21.3. Quale tipo di NAT è configurato in R1? Perché?
b) Le connessioni degli host D ed E verso il server S sono visti dal server con indirizzi sorgente rispettivamente 2.12.15.4 e porte 6677 e 6678 rispettivamente. Quale tipo di NAT è configurato in R2? Perché?
c) Il server web F è raggiungibile dal client G con indirizzo 120.15.16.55 e porta 80. Quale tipo di NAT è configurato in R3? Perché?

D3 - Si illustri come opera il comando di traceroute.

Laboratorio (6 punti)

Si vuole scrivere un'applicazione client/server TCP per un servizio di antifurto biciclette che permette all'utente di verificare la posizione della propria bici.

Per semplicità la componente client/server relativa all'upload della posizione è stata omessa.

Descrizione Server

Il server, accessibile all'indirizzo www.trytostealmybike.com, una volta che il client ha instaurato la connessione TCP, chiede all'utente di autenticarsi attraverso la funzione *Authorized_user, user_ID = User_Authentication(connectionSocket),* che ha il compito di chiedere e verificare le credenziali utente. Le variabili restituite sono:

- la variabile booleana *Authorized_user,* il cui valore è *True* se l'utente ha inserito credenziali valide, *False* altrimenti.
- La variabile *user_ID,* che contiene l'identità dell'utente.

La verifica della validità delle credenziali avviene attraverso la funzione *User_Identity_Verification(user_name , Pass),* la quale restituisce una variabile booleana il cui valore è *True* se l'utente ha inserito credenziali valide, *False* altrimenti.

L'utente, nel caso inserisca credenziali errate, ha a disposizione un numero finito di tentativi, superato tale limite l'autenticazione avrà esito negativo.

Se l'autenticazione è andata a buon fine il server invia il messaggio 'OK' al client, per poi fornirgli la posizione della bicicletta e chiudere la connessione.

La funzione *Location = Get_bike_location(user_ID)* fornisce, dato l'ID di un utente registrato al servizio, le coordinate GPS della bicicletta in formato testuale.

Descrizione Client

Il client deve rispondere alle richieste del server, durante la fase di autenticazione, fino alla ricezione del messaggio 'OK'. Dopodiché si mette in attesa della ricezione delle coordinate della bicicletta da parte del server. Infine la connessione viene chiusa.

- Completare lo script "TCP server" date le seguenti specifiche:
    i. Utilizzare indirizzi IPv4
    ii. Lunghezza buffer di ricezione: 2048 byte.
    iii. Numero massimo di tentativi consentiti nella fase di login utente: 10
- Completare lo script "TCP client"
- L'utente può vedere il messaggio 'OK' del server? _____
- Il server riesce a gestire più utenti contemporaneamente? In caso positivo quanti utenti possono essere gestiti contemporaneamente? In caso negativo quanti utenti possono essere messi in attesa?
  _____
  _____

TCP client

```
from socket import *

serverAddress = _____
serverPort = _____
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect(_____)
```

```
#Authentication phase
while True:
   server_message = clientSocket.recv(1024)
   if server_message != 'OK':
      print server_message
      user_message = raw_input()
      clientSocket.send(user_message)
   else:
      break

server_message = clientSocket.recv(1024)
print server_message
clientSocket.close()
```

TCP server

```
def User_Authentication(connectionSocket):
   N_Attempts = _____________________________________________________

   for _ in range(N_Attempts):
      connectionSocket.send('User Name: ')
      user_name = connectionSocket.recv(_____________________)
      connectionSocket.send('Password: ')
      Pass = connectionSocket.recv(_____________________)

      Authorized_user = User_Identity_Verification(user_name , Pass)
      if Authorized_user:
        return True, user_name
      else:
         connectionSocket.send('Wrong User Name and Password')

   connectionSocket.send('Maximum number of attempts exceeded. Access Denied!')
   return False, ''


serverPort = 1200
serverSocket = socket(_____________________________)
serverSocket.bind(('',_____________________________))
serverSocket.listen(10)


while True:
   print 'The server is ready'
   connectionSocket, addr = serverSocket.accept()

   Authorized_user, user_ID = User_Authentication(connectionSocket)
   if Authorized_user:
      connectionSocket.send('OK')
      Location = Get_bike_location(user_ID)
      Message = 'Your bike is here: ' + Location
      connectionSocket.send(Message)

   connectionSocket.close()
```

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,))
```