Esame completo - 25 Luglio 2016

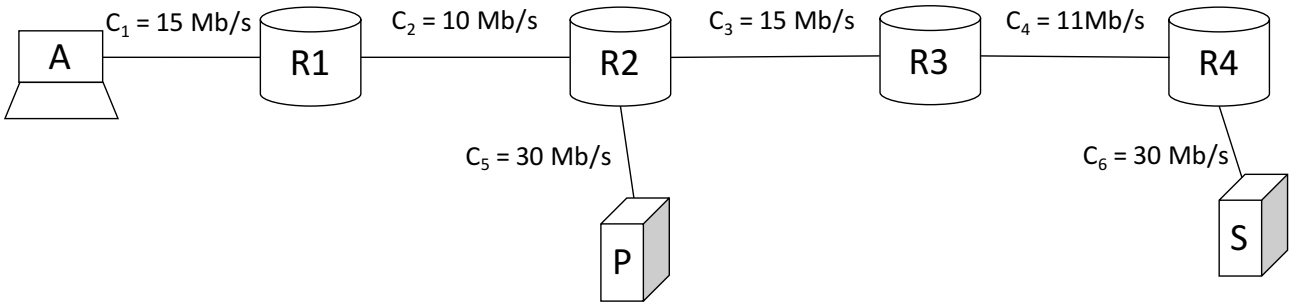
Cognome	**STUDENTE**
Nome	**BRAVO**
Matricola	**SOLUZIONI**

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

E1	E2	E3	Quesiti	Lab

1 - Esercizio (6 punti)

Nella rete in figura

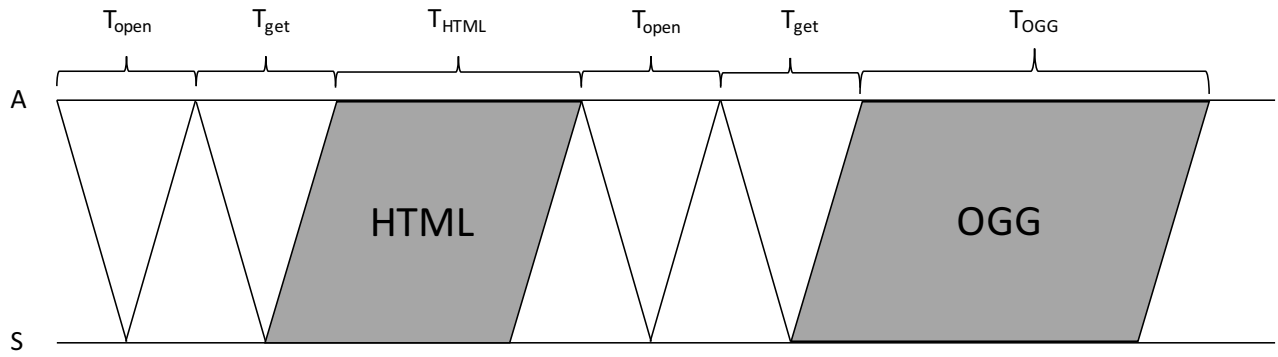


sono rappresentati 4 router (R1, R2, R3 e R4), in client A, un HTTP proxy P e un HTTP server S. Accanto ad ogni collegamento è indicata la propria capacità, mentre il tempo di propagazione è pari a 10 [ms] su ciascun collegamento.
Il client vuole scaricare del server un sito web composto da 1 pagina HTML di dimensione L_{HTML}=80 [kbyte] e 6 oggetti JPEG richiamati nella pagina HTML di dimensione L_{OGG}=500 [kbyte]. Nella rete sono presenti flussi interferenti di lunga durata: 4 tra R1 e R2, 10 tra R3 e R4.
Si chiede di calcolare il tempo di trasferimento del sito web a livello applicativo nei seguenti casi:

a) Il client A non ha proxy configurato, apre connessioni non-persistent in parallello (quando possibile e nel massimo numero possibile)
b) Il client A utilizza il proxy P, apre al massimo una connessione alla volta in modalità non-persistent. Solo la pagina HTML ed i primi 2 oggetti JPEG sono presenti nella cache del proxy.

Soluzione

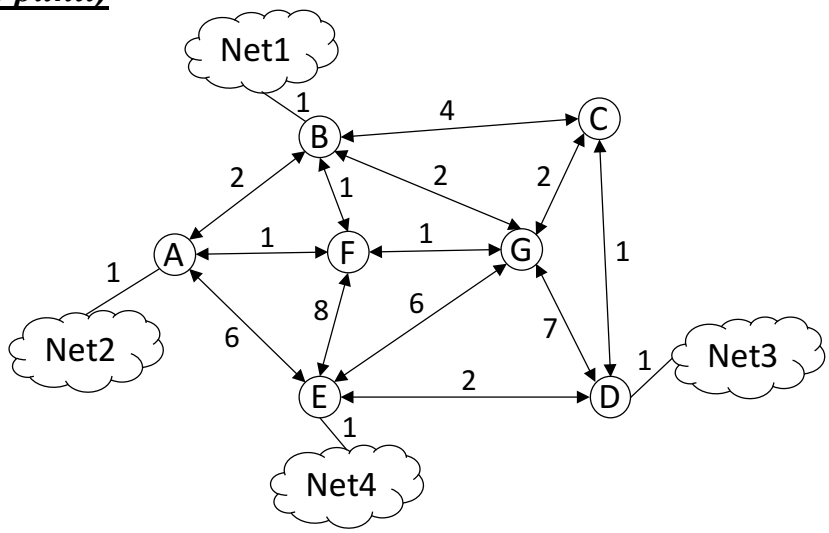
Punto a)
Il client C colloquia direttamente con il server S.
Per la pagina HTML, i collegamenti attraversati hanno le seguenti capacità effettive: A-R1 15 Mb/s, R1-R2 2 Mb/s (4+1 flussi), R2-R3 15Mb/s, R3-R4 1Mb/s (10+1 flussi). Dunque il trasferimento è governato dal collo di bottiglia R3-R4 a 1 Mb/s. Il tempo di trasferimento della pagina HTML è pari a T_{HTML} = *8*80 [kbit] / 1 [Mb/s] = 640 ms*
Per gli oggetti JPEG, i collegamenti attraversati sono i medesimi, dunque il collo di bottiglia sarà il link R3-R4 con una capacità effettiva di 687.5 kb/s (10+6 flussi). Il tempo di trasferimento di ogni oggetto JPEG è pari a T_{OGG} = *8*500 [kbit] / 687.5 [kb/s] = 5.82 s*.
Il tempo totale di trasferimento è pari a $T = T_{open} + T_{get} + T_{HTML} + T_{open} + T_{get} + T_{OGG}$ in accordo con la seguente figura. Per i tempi di propagazione, $T_{open} = T_{get} = 100\ [ms]$



T=6.86s

Punto b)
Il client C colloquia con il proxy P.
Il colloquio client-proxy è definito dal collo di bottiglia R1-R2 con una capacità di 2 Mb/s (4+1 flussi), mentre il colloquio proxy-server ha R3-R4 come collo di bottiglia, dunque una capacità di 1 Mb/s (10+1 flussi). I tempi di trasferimento sono $T_{HTML}^{CP} = \frac{640[kbit]}{2\left[\frac{Mbit}{s}\right]} = 320[ms], T_{OGG}^{CP} = \frac{4[Mbit]}{2\left[\frac{Mbit}{s}\right]} = 2[s]$ *e* $T_{OGG}^{PS} = \frac{4[Mbit]}{1\left[\frac{Mbit}{s}\right]} = 4[s]$, rispettivamente, per il trasferimento della pagina HTML dal proxy al client, di un oggetto JPEG dal proxy al client e di un oggetto JPEG dal server al proxy.
I tempi di propagazione sono $T_{open}^{CP} = T_{get}^{CP} = 60[ms]$ per la connessione client-proxy e $T_{open}^{PS} = T_{get}^{PS} = 80[ms]$ per la connessione proxy-client.
Il tempo totale di trasferimento è pari a $T = T_{open}^{CP} + T_{get}^{CP} + T_{HTML}^{CP} + 2\left(T_{open}^{CP} + T_{get}^{CP} + T_{OGG}^{CP}\right) + 4\left(T_{open}^{CP} + T_{get}^{CP} + T_{OGG}^{CP} + T_{open}^{PS} + T_{get}^{PS} + T_{OGG}^{PS}\right)$=29.8s

2 - Esercizio (6 punti)



Nella rete in figura è rappresentato il grafo di una rete in cui sono presenti dei router (A, B, C, D, E, F, G) e 4 reti (Net1, Net2, Net3, Net4). I costi di attraversamento sono indicati accanto ad ogni link, i link sono bidirezionali e simmetrici.
Si chiede di:

a) Calcolare mediante l'algoritmo di Dijkstra l'albero dei cammini minimi con sorgente F e destinazioni tutti gli altri router (si omettano la reti nel grafo). Indicare:
   - nella Tabella A, il valore dell'etichetta ad ogni step in cui il nodo viene analizzato: nel caso lo step successivo non modifichi l'etichetta dello step precedente occorre riscrivere l'etichetta dello step precedente. In caso di parità del valore di etichetta da fissare, si fissi quella del nodo con lettera minore (A < B < C, etc.)
   - nella figura sopra, l'albero trovato

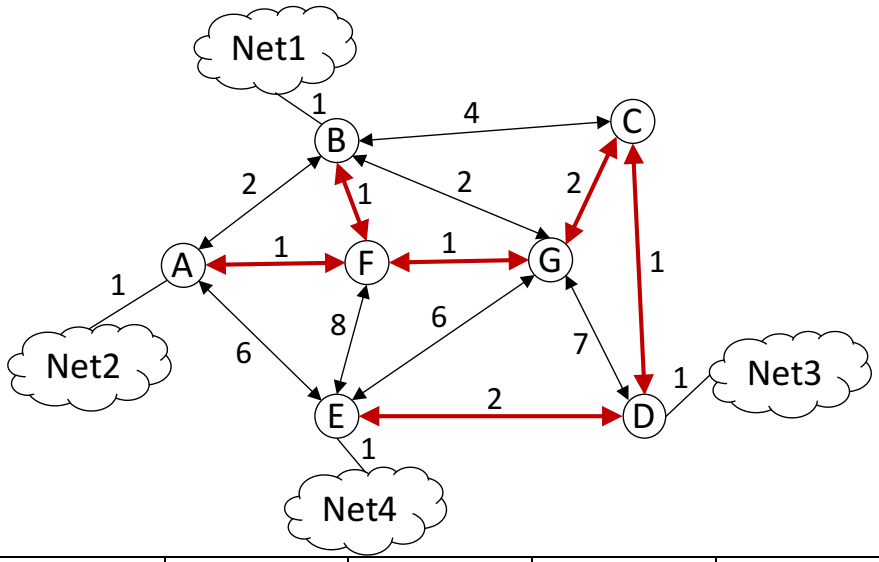
b) Sulla base dell'albero dei cammini calcolato al punto precedente, indicare i Distance Vector (DV) inviati dal router F ai propri vicini nella modalità Split Horizon (no Poisonous Reverse). Per ogni DV inviato indicare chiaramente le reti raggiunte ed il destinatario.

Tabella A

Nodo A	Nodo B	Nodo C	Nodo D	Nodo E	Nodo F	Nodo G

Soluzione

Punto a)



Nodo A	Nodo B	Nodo C	Nodo D	Nodo E	Nodo F	Nodo G
(F,1)	(F,1)	(B,5)	(G,8)	(F,8)	(-,0)	(F,1)
	(F,1)	(G,3)	(C,4)	(A,7)		(F,1)
				(A,7)		
				(D,6)		

Punto b)

DV inviato al nodo A: (Net1, 2), (Net3, 5), (Net4, 7)
DV inviato al nodo B: (Net2, 2), (Net3, 5), (Net4, 7)
DV inviato al nodo G: (Net1, 2), (Net2, 2)
DV inviato al nodo E: (Net1, 2), (Net2, 2), (Net3, 5), (Net4, 7)

Esercizio 3 (6 punti)

Un router ha le seguenti interfacce e tabella di routing

Rete	**Indirizzo**	**Netmask**
eth0	192.168.1.254	255.255.255.0
eth1	131.175.23.13	255.255.255.128
eth2	123.17.4.5	255.255.255.0
eth3	17.7.4.27	255.255.255.128

	Destination	**Netmask**	**Next Hop**
#1	13.14.190.0	255.255.255.128	123.17.4.34
#2	12.13.0.0	255.255.128.0	131.175.23.27
#3	12.13.192.0	255.255.192.0	123.17.4.34
#4	0.0.0.0	0.0.0.0	17.7.4.93

Il router ha configurato un NAPT che assegna agli indirizzi privati della rete eth0, l'indirizzo pubblico del router sulla rete eth2, 123.17.4.5. Inoltre, è configurato un Port Forwarding che mappa (123.17.4.5,80) in (192.168.1.3,80) sulla rete eth0.

Si chiede di indicare come verranno gestiti i seguenti pacchetti, in cui sono indicati. IP e porta sorgente, IP e porta destinazione e porta d'ingresso. Occorre indicare la tipologia di inoltro (scartato, diretto o indiretto), l'eventuale riga della tabella utilizzata, l'interfaccia d'uscita, l'eventuale modifica agli indirizzi IP sorgente o destinazione subita dal pacchetto in transito.

A) SRC: 192.168.1.5, 2345 **DST: 192.168.1.8, 2346** **da eth0**

Inoltro: SCARTATO Riga tabella routing: Interfaccia:

Eventuale motivo scarto: Inoltro diretto in cui l'interfaccia d'ingresso è uguale all'interfaccia d'uscita

Modifiche indirizzi IP:

B) SRC: 192.168.1.6, 4356 **DST: 12.13.205.7, 1234** **da eth0**

Inoltro: INDIRETTO Riga tabella routing: 3 Interfaccia: eth2

Eventuale motivo scarto:

Modifiche indirizzi IP: IP sorgente diventa 123.17.4.5

C) SRC: 137.12.5.3, 1234 **DST: 12.13.129.11, 80** **da eth2**

Inoltro: INDIRETTO Riga tabella routing: 4 Interfaccia: eth3

Eventuale motivo scarto:

Modifiche indirizzi IP: Nessuna

D) SRC: 137.15.7.2, 2345 **DST: 123.17.4.5, 80** **da eth2**

Inoltro: DIRETTO – PORT FORW Riga tabella routing: Interfaccia: eth0

Eventuale motivo scarto:

Modifiche indirizzi IP: IP destinazione diventa 192.168.1.3

E) SRC: 137.23.8.1, 25 **DST: 123.17.4.7, 1026** **da eth1**

Inoltro: DIRETTO Riga tabella routing: Interfaccia: eth2

Eventuale motivo scarto:

Modifiche indirizzi IP: Nessuna

F) SRC: 192.168.1.17, 115 **DST: 131.175.23.195, 6534** **da eth0**

Inoltro: INDIRETTO Riga tabella routing: 4 Interfaccia: eth3

Eventuale motivo scarto:

Modifiche indirizzi IP: IP sorgente diventa 123.17.4.5

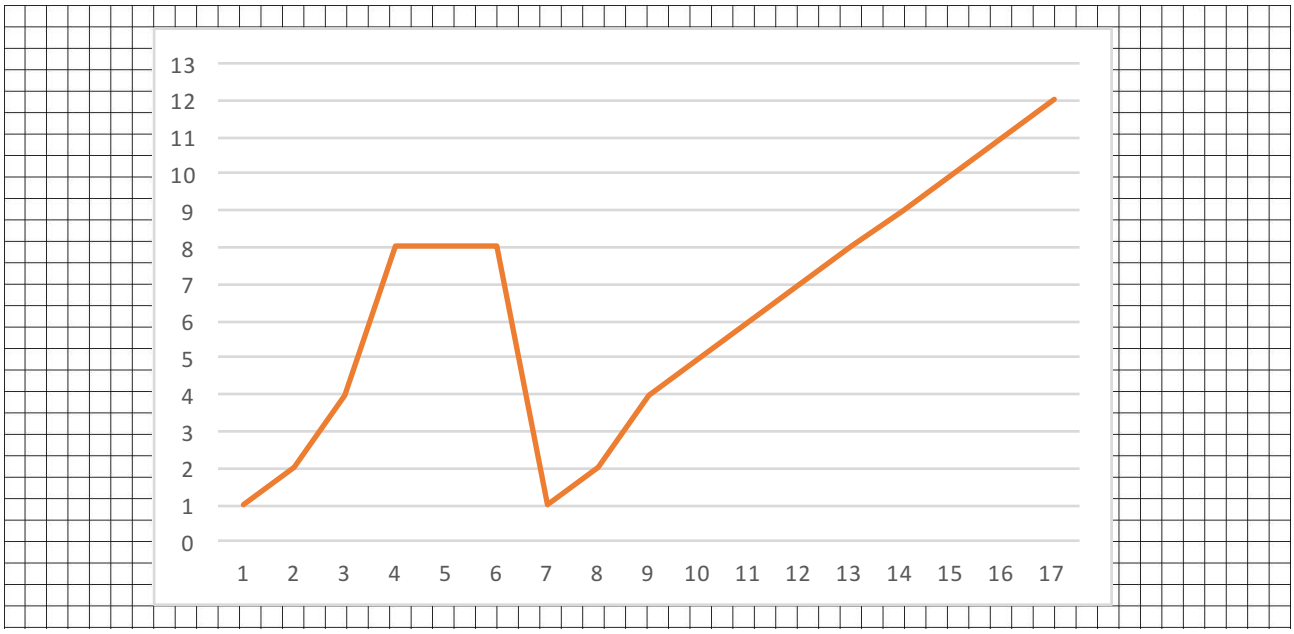
4–Domande (9 punti)

1) Sia dato il pool di indirizzi 131.175.17.0/25. Da questo pool occorre ricavare 2 sottoreti per le LAN aziendali S1 e S2 di 50 postazioni ciascuna
Si indichi:
   a) L'indirizzo di rete (con netmask) di ciascuna LAN
   b) Dopo 10 anni, a partire dalle LAN S1 e S2 si vuole gestire una nuova LAN da 110 postazioni. Esiste una tecnica che lo consenta? Se sì, indicare il nome della tecnica usata e l'indirizzo di rete (con netmask) della nuova LAN

   a) S1: 131.175.17.0/26, S2: 131.175.17.64/26 con ciascuna 62 indirizzi a disposizione
   b) Con il supernetting posso unire le due reti in 131.175.17.0/25 con 126 indirizzi a disposizione

2) Una connessione TCP deve trasferire in file da 8000 [byte] ed è caratterizzata dai seguenti parametri:
- MSS = 100 byte
- SSTHRESH = 800 byte
- RCWND = 1200 byte
- Timeout = 3 RTT

Si tracci l'andamento ad ogni RTT del valore della finestra di trasmissione nel caso in cui tutti i segmenti della quarta finestra (4° RTT) vengono persi. Si ignori la fase inziale di apertura della connessione



3) In una rete sono presenti un Host, un HTTP Server e un DNS Server.
Si assuma che le ARP cache siano a regime (no scambio messaggi ARP), le DNS cache vuote e siano date le seguenti configurazioni:

- Host –
  - IP: 1.1.1.1/24
  - gw: 1.1.1.23
  - DNS server: 1.1.1.25
- HTTP Server (*www.dominio.it)* –
  - IP: 1.1.1.2/24
  - gw: 1.1.1.23
  - DNS server: 1.1.1.25
- DNS Server –
  - IP: 1.1.1.25/24
  - gw: 1.1.1.23
  - con le seguenti corrispondenze (nome DNS, indirizzo IP):
    - www.dominio.it ⇔ 1.1.1.2
    - dns.dominio.it ⇔ 1.1.1.25

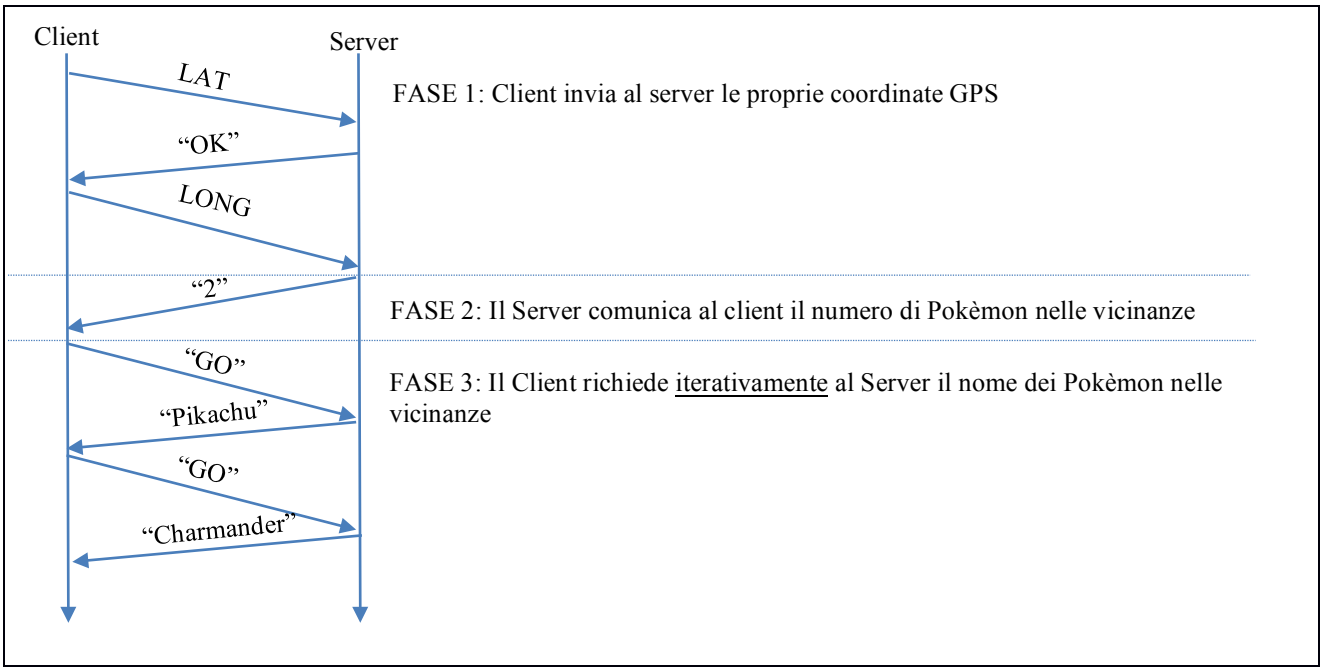
L'host richiede tramite browser una pagina HTML dal nome *index.html* al server *www.dominio.it*. Si elenchi la sequenza dei messaggi HTTP e DNS scambiati in assenza di errori, in particolare, per ogni messaggio si indichi:

- Tipologia (es. DNS request, DNS response, GET, 200 OK, etc.)
- Sintesi del contenuto: per i messaggi DNS, il nome da tradurre e la traduzione, per i messaggi HTTP, server e file richiesto
- Indirizzi IP sorgente e destinazione

1. DNS request (www.dominio.it, ?): src 1.1.1.1, dst 1.1.1.25
2. DNS reply (www.dominio.it, 1.1.1.2): src 1.1.1.25, dst 1.1.1.1
3. GET (www.dominio.it, index.html):): src 1.1.1.1, dst 1.1.1.2
4. 200 OK (index.html): src 1.1.1.2, dst 1.1.1.1

5–Laboratorio (6 punti)

Il codice sottoriportato rappresenta una versione semplificata del videogioco Pokèmon GO.
Periodicamente l'applicazione dell'utente (client) invia le proprie coordinate GPS al server, che risponde con l'elenco dei Pokèmon presenti nelle vicinanze. Il diagramma in figura mostra il protocollo applicativo, in 3 fasi, tra Client e Server.



Script client

```
from socket import *

serverName = 'localhost'
serverPort = 12000

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

# Invia coordinate GPS al server
(latitudine,longitudine) = (5,10)
clSock.send(str(latitudine))
message = clSock.recv(2)
if message == 'OK':
        clSock.send(str(longitudine))

# Legge dal server il numero di Pokemon nelle
vicinanze da richiedere poi al server
numero_pokemon = int( clSock.recv(100) )
lista_pokemon = []

# Richiede, uno alla volta, la lista dei pokemon
al server
for i in range(numero_pokemon):
        clSock.send('GO')
        pokemon = clSock.recv(100)
        lista_pokemon.append( pokemon )

print lista_pokemon

clSock.close()
```

Script server

```
from socket import *

serverPort = 12000
servSock = socket(AF_INET, SOCK_STREAM)

servSock.bind(('', serverPort))
servSock.listen(5)

print 'Server Pokemon GO pronto!'

while True:
        clSock,  clAddr= servSock.accept()
        print "Connection form: ", clAddr

        # Riceve dal client le coordinate GPS
        lat = int( clSock.recv(100) )
        clSock.send("OK")
        long = int( clSock.recv(100) )

        # Definisce la lista dei Pokemon…
        lista_pkmn = ["Pikachu","Charmander"]
        #…e ne invia la lunghezza
        clSock.send(str(len(lista_pokemon)))

        for pokemon in lista_pkmn:
                message = clSock.recv(2)
                if message == 'GO':
                        clSock.send(pokemon)

        clSock.close()
```

Q1) Completare il codice mancante nel Server e nel Client per implementare la fase 3 del protocollo.
Q2) Quanti utenti si possono accodare nel Server in attesa di essere serviti?

La coda è di 5 utenti

3) Che protocollo di trasporto è utilizzato? Perché la comunicazione tra server e client richiede tale protocollo?

E' utilizzato TCP. La modalità connection-oriented consente di mantenere memoria dello stato della connessione e inviare la lista corretta all'utente che ha aperto la connessione. Inoltre, TCP garantisce la consegna dei messaggi

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,))
```