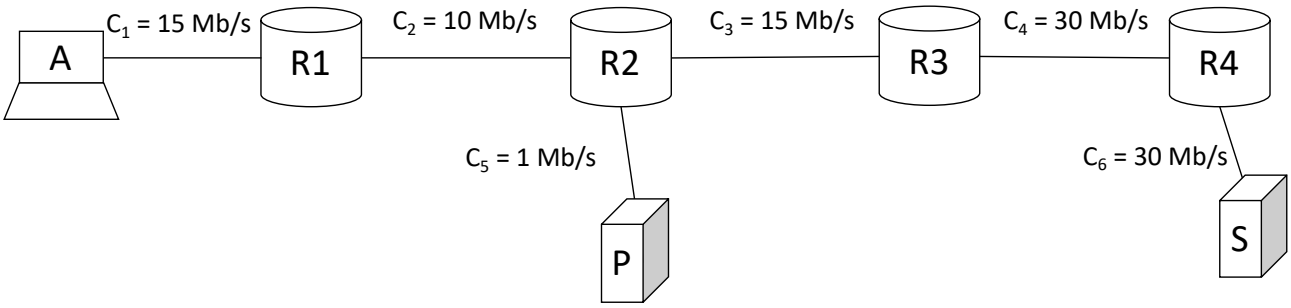
Esame completo – 13 Febbraio 2018

Cognome	
Nome	
Matricola	

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

E1	E2	E3	Quesiti	Lab

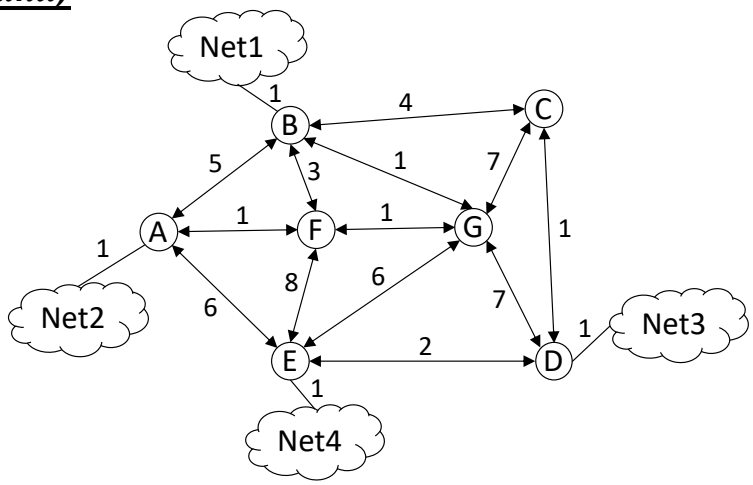
<u>1 - Esercizio (6 punti)</u>



Nella rete in figura sono rappresentati 4 router (R1, R2, R3 e R4), un client A, un HTTP proxy P e un HTTP server S. Accanto ad ogni collegamento è indicata la rispettiva capacità, mentre il tempo di propagazione è pari a 10 [ms] su ciascun collegamento.
Il client vuole scaricare dal 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. La probabilità di trovare una pagina HTML o un oggetto JPEG nella cache del proxy è pari a *0.6*. Si chiede di calcolare il tempo <u>medio</u> di trasferimento.

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 il Distance Vector (DV) (rispetto alle reti rappresentate) inviato dal router F al router G nella modalità Split Horizon con Poisonous Reverse, con lunghezza massima dei cammini consentita di *25*.

Tabella A

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

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: 192.168.1.6, 4356 DST: 12.13.130.7, 1234 da eth0

Inoltro: Riga tabella routing: Interfaccia:

Eventuale motivo scarto:

Modifiche indirizzi IP:

B) SRC: 23.12.55.254, 4356 DST: 123.17.4.12 da eth2

Inoltro: Riga tabella routing: Interfaccia:

Eventuale motivo scarto:

Modifiche indirizzi IP:

C) SRC: 131.175.23.63, 2345 **DST: 123.17.4.5, 80** **da eth2**

Inoltro: Riga tabella routing: Interfaccia:

Eventuale motivo scarto:

Modifiche indirizzi IP:

D) SRC: 137.12.5.3, 1234 **DST: 12.13.7.11, 80** **da eth2**

Inoltro: Riga tabella routing: Interfaccia:

Eventuale motivo scarto:

Modifiche indirizzi IP:

E) SRC: 137.23.8.1, 25 **DST: 17.7.4.35, 1026** **da eth1**

Inoltro: Riga tabella routing: Interfaccia:

Eventuale motivo scarto:

Modifiche indirizzi IP:

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

Inoltro: Riga tabella routing: Interfaccia:

Eventuale motivo scarto:

Modifiche indirizzi IP:

4–Domande (9 punti)

1) Con riferimento ai sistemi P2P a directory centralizzata. Si indichi in che cosa differiscono la fase di ricerca dei contenuti e quella di trasferimento dei file.

2) Un sistema di accesso multiplo centralizzato a divisione di tempo (TDMA) è caratterizzato da una trama con slot di durata complessiva TSLOT=10[us], con un tempo di guardia TG=2[us]. Il sistema serve 8 utenti e ha una rate trasmissivo del segnale multiplato di C=2 [Mb/s]. Si chiede di: 1) indicare il numero di bit di ciascun tributario trasmessi in ogni slot 2) indicare il massimo rate possibile per ciascun tributario in ingresso

3) In una rete sono presenti un Host, due HTTP Server e due 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:

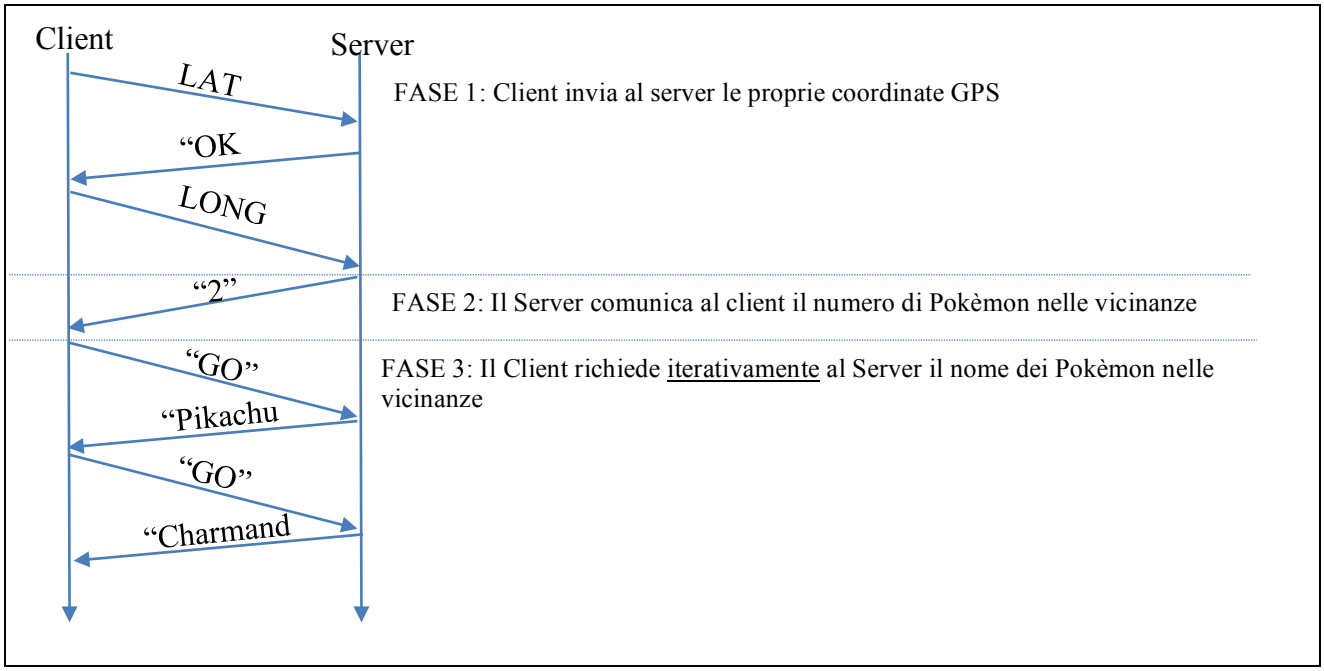
- Host –
  - IP: 1.1.1.1/24
  - gw: 1.1.1.23
  - DNS server: 1.1.1.25
- HTTP Server 1 (*www.dominio.it)* –
  - IP: 1.1.1.2/24
  - gw: 1.1.1.23
  - DNS server: 1.1.1.25
- HTTP Server 2 (*www.dominio.it)* –
  - IP: 1.1.1.13/24
  - gw: 1.1.1.23
  - DNS server: 1.1.1.25
- DNS Server 1 –
  - IP: 1.1.1.27/24
  - gw: 1.1.1.23
  - con le seguenti corrispondenze (nome DNS, indirizzo IP):
    - www.dominio.it ⇔ 1.1.1.13
    - dns.dominio.it ⇔ 1.1.1.27
- DNS Server 2 –
  - 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

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._____________________________

# 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):

        ____________________________________
        ____________________________________
        lista_pokemon.append( pokemon )

print lista_pokemon

clSock.close()
```

Script server

```
from socket import *

serverPort = 12000
servSock = socket(AF_INET, SOCK_STREAM)


servSock._____________________________________
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:

                _____________________________
                if message == 'GO':

                        ____________________

        clSock.close()
```

Q1) Completare il codice mancante nel Server e nel Client per implementare lo scambio del protocollo applicativo descritto dalla figura

Q2) Come dovrebbe essere modificato lo script del client se il server rispondesse con la stringa "OKAY" invece che "OK"?

3) Che protocollo di trasporto è utilizzato?

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,))
```