**Esame Completo - 27 Giugno 2018**

|  |  |
| --- | --- |
| **Cognome** |  |
| **Nome** |  |
| **Matricola** |  |

##### Tempo complessivo a disposizione per lo svolgimento: 2 ore

***Si usi lo spazio bianco dopo ogni esercizio per la risoluzione***

|  |  |  |  |
| --- | --- | --- | --- |
| **E1** | **E2** | **Quesiti** | **Lab** |
|  |  |  |  |

## 1 - Esercizio (7 punti)

La rete di un ISP è riportata in figura. L’ISP possiede lo spazio di indirizzamento: 93.71.176.0/21 Definire un piano di indirizzamento in grado di supportare il numero di *host* indicato nella figura.

1. 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 (# indirizzi rete A ≤ # indirizzi rete B ≤ …..). 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 93.71.176.0/21.
2. Scrivere nella tabella 2 la tabella di instradamento del router R1 nel modo più compatto possibile dopo aver assegnato opportunamente degli indirizzi ai router a cui R1 è connesso direttamente.



Tabella 1: Piano di indirizzamento

## NB: Nella rete sono presenti due VLAN, una usata per AP con SSID “company” e l’altra per AP con SSID “guests” che occorre separare con NETID diversi. Tutti gli indirizzi IP iniziano per 93.71

|  |  |  |  |
| --- | --- | --- | --- |
| **Rete** | **Indirizzo di rete** | **Netmask** | **Ind. broadcast diretto** |
| A | 176.0 | 255.255.254.0 | 177.255 |
| B | 178.0 | 255.255.254.0 | 179.255 |
| C | 180.0 | 255.255.255.0 | 180.255 |
| D | 181.0 | 255.255.255.0 | 181.255 |
| E | 182.0 | 255.255.255.128 | 182.127 |
| F | 182.128 | 255.255.255.128 | 182.255 |
| G | 183.0 | 255.255.255.128 | 183.127 |
| H | 183.128 | 255.255.255.192 | 183.191 |
| I | 183.192 | 255.255.255.224 | 183.223 |
| L | 183.224 | 255.255.255.252 | 183.227 |
| M | 183.228 | 255.255.255.252 | 183.231 |
| N | 183.232 | 255.255.255.252 | 183.235 |
| O | 183.236 | 255.255.255.252 | 183.239 |
| P | 183.240 | 255.255.255.252 | 183.243 |

Tabella 2: Tabella di routing di R1

NB: All’interfaccia del router R5 viene assegnato 183.233/30, mentre all’interfaccia del router R2 183.237/30

|  |  |  |
| --- | --- | --- |
| **Rete** | **Netmask** | **Next hop** |
| 176.0 | 255.255.254.0 | 183.233 |
| 183.0 | 255.255.255.0 | 183.233 |
| 182.128 | 255.255.255.128 | 183.233 |
| 178.0 | 255.255.254.0 | 183.237 |
| 180.0 | 255.255.252.0 | 183.237 |
| 183.240 | 255.255.255.252 | 183.237 |
| 0.0.0.0 | 0.0.0.0 | Itfc\_Rx |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Esercizio 2 (7 punti)

Nella rete in figura è attiva una connessione TCP tra il dispositivo A ed il dispositivo B. I parametri iniziali della connessione TCP siano i seguenti: MSS=500 [byte], RCWND=10MSS, SSTHRESH=8MSS, dimensione segmenti di riscontro e di apertura della connessione trascurabile. Supponendo che:

* i segmenti 8, 9 e 10 in trasmissione vanno persi (supporre che i pacchetti fuori ordine vengano scartati)
* il valore del time out iniziale sia tout=500 [ms]
* all’istante di tempo t=510,32 [ms] a partire dall’invio del primo segmento, il dispositivo B segnala un campo di *window* nei segmenti di riscontro pari a 2000[byte],

dire se la trasmissione sulla connessione TCP arriva a saturare la capacità del collegamento 1 e, in caso positivo, indicare il tempo oltre cui la trasmissione diventa continua sul collegamento 1; trovare il tempo di trasferimento di un file di 505 [Mbyte].



SOLUZIONE

La trasmissione non diventa mai continua perché RCWND < Wc



## Esercizio 3 (4 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 3 aree e si disegni il grafo che rappresentano la rete vista dal *router* R1e R3. Si indichino chiaramente cosa rappresenta ciascun nodo ed i costi associati agli archi.



SOLUZIONE

Visto da R1



Visto da R3

Dato che R3 è router di bordo per tutte le 3 aree, la topologia vista da R3 è la rete disegnata in figura.

## 4–Domande (8 punti)

**Q1**

Un sistema di accesso multiplo a divisione di tempo (TDMA) è caratterizzato da un rate trasmissivo sul canale di W=2.5 Mb/s e da una velocità netta per ciascun sotto-canale (tributario) V=200 kb/s. Sapendo che in ciascuno slot vengono trasmessi D=200 bit di dati e H=50 bit di overhead, e che il tempo di guardia Tg è di 25 s, calcolare il tempo di slot TS, il tempo di trama TT, e il numero N di sotto-canali.

TS = TB + Tg

TB = (D+H) / W = 100 us

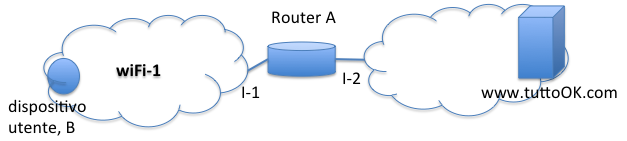
TS = 100 + 25 = 125 us

TT = D / V = 1 ms

N = TT / TS = 8

**Q2**

Si consideri la configurazione di rete in figura in cui un dispositivo d’utente, B, è parte della rete locale WiFi-1. Il router A, oltre alle normali funzionalità di inoltro/instradamento, fornisce anche un servizio di DHCP alla rete WiFi-1, tramite l’interfaccia I-1. Il dispositivo d’utente vuole collegarsi al web server www.tuttoOK.com. Si indichino le sequenze di messaggi che vengono scambiati tra i tre dispositivi in figura supponendo che il dispositivo di utente non abbia inizialmente nessuna configurazione IP e tabelle ARP vuote (si supponga invece che il router abbia tabelle di routing ed inoltro già a convergenza).



Da B in BC: DHCP Discover

Da Router in BC: DHCP Offer

Da B in BC: DCHP Request

Da Router in BC: DHCP Ack

(Da B a LNS: DNS Request)

(Da LNS a B: DNS Reply)

Da B in BC: ARP Request

Da Router a B: ARP Reply

Da B a Router: TCP SYN

Da Router a Server: TCP SYN

Da Server a Router: TCP SYN-ACK

Da Router a B: TCP SYN-ACK

Da B a Router: TCP ACK + http GET

Da Router a Server: TCP ACK + http GET

**Q3**

Un router che implementa il protocollo RIP, ha la tabella di instradamento indicata sotto, e riceve il distance vector riportata accanto dal router con indirizzo 2.35.2.254. Indicare come cambia la tabella di instradamento e i relativi costi.





## 6 -Laboratorio (6 punti)

Il codice sotto riportato rappresenta una versione semplificata di un sistema di prenotazione di posti per eventi. Il client richiede e visualizza la lista degli eventi disponibili e chiede all’utente di il numero dell’evento per cui vuole prenotare un posto. Gli eventi possono essere pieni, il server deve verificare se ci sono posti disponibili per l’evento richiesto prima di dare conferma al client.

**Q.** Completare il codice del Client e del Server

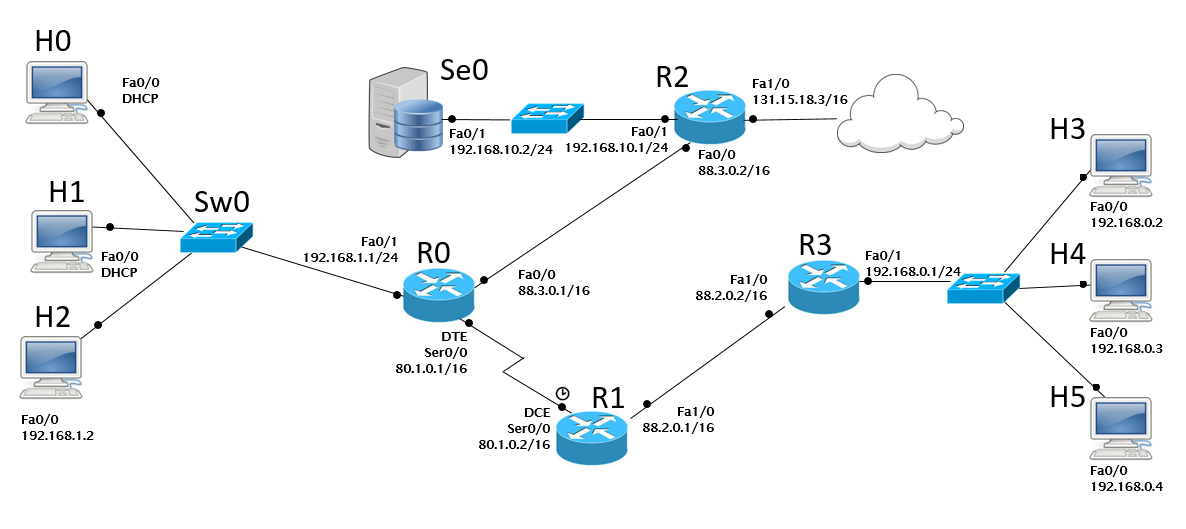
**Server**

|  |
| --- |
| from socket import \*  serverPort = 12000  listaEventi = ["1 - Evento 1", "2 - Evento 2", "3 - Evento 3"]  postiEventi = [0, 1, 2]  serverSocket = socket(AF\_INET, SOCK\_STREAM) # Welcome socket  serverSocket.bind(('', serverPort))  serverSocket.listen(1)  while 1:  print "Server in attesa di connessioni"  connectionSocket, addr = serverSocket.accept()  request = ""  while request != ".":  request = connectionSocket.recv(2048)  if request == "EVENTI":  reply = ""  for ev in listaEventi:  reply += ev + "\n"  connectionSocket.send(reply)  # v.isdigit() restituisce True/False se la var. v è una  # stringa che rappresenta un numero  elif request.isdigit() and int(request) in range(1,len(listaEventi)+1):  ev\_num = int(request)  if postiEventi[ev\_num-1] > 0:  postiEventi[ev\_num-1] -= 1  **reply = “OK” (o altro)**  else:  **reply = “NO\_SPACE”**  connectionSocket.send(reply)  elif request != ".":  reply = "KO"  connectionSocket.send(reply)  connectionSocket.close() |

**Client**

|  |
| --- |
| from socket import \*  serverName = "localhost"  serverPort = 12000  clientSocket = socket(AF\_INET, SOCK\_STREAM)  clientSocket.connect((serverName,serverPort)  clientSocket.send("EVENTI")  elenco\_eventi = clientSocket.recv(50)  reply = "KO"  while reply == "KO" or reply == 'NO\_SPACE':  print "Elenco degli eventi disponibili:"  print elenco\_eventi  request = raw\_input("Inserisci la tua scelta: ")  clientSocket.send(request)  **reply = clientSocket.recv(2048)**  if reply == "KO":  print "Scelta non valida."  elif reply == "NO\_SPACE":  print "Non ci sono posti disponibili per l'evento selezionato"  else:  print "Prenotazione andata a buon fine!"  clientSocket.send(".")  **clientSocket.close(2048)** |

Si consideri la rete in figura

****

**Attenzione:**

* **Indirizzi IP e gateway sono già stati configurati per i 6 host.**
* **Le interfacce dei router R0, R1 e R3 sono già state configurate ed attivate come in figura.**
* **Le reti /24 sono reti private**
* **Indicare sempre prima del comando il prompt visualizzato dal sistema, prestando attenzione alla modalità di partenza in ciascuna richiesta**

**Q1)** Configurare ed attivare l’interfaccia Fa0/0 del router **R2** come indicato in figura.

R2> enable

R2# configure terminal

R2(config)# interface FastEthernet0/0

R2(config-if)# ip address 80.3.0.2 255.255.0.0

R2(config-if)# no shutdown

**Q2)** Configurare il port forwarding sul router **R2** in modo che sia effettuato il seguente mapping:

|  |  |  |  |
| --- | --- | --- | --- |
| IP | Port | IP | Port |
| 88.3.0.2 | 18120 | 192.168.10.2 | 12000 |
| 88.3.0.2 | 18121 | 192.168.10.2 | 12001 |
| 88.3.0.2 | 18122 | 192.168.10.2 | 12002 |

R2>enable

R2#configure terminal

R2(config)#interface FastEthernet 0/1

R2(config-if)# ip nat inside

R2(config)# exit

R2(config)#interface FastEthernet 0/0

R2(config-if)# ip nat outside

R2(config)# exit

R2(config)# ip nat inside source static tcp 192.168.10.2 12000 88.3.0.2 18120

R2(config)#ip nat inside source static tcp 192.168.10.2 12001 88.3.0.2 18121

R2(config)#ip nat inside source static tcp 192.168.10.2 12002 88.3.0.2 18122

**Codice socket programming**

**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))

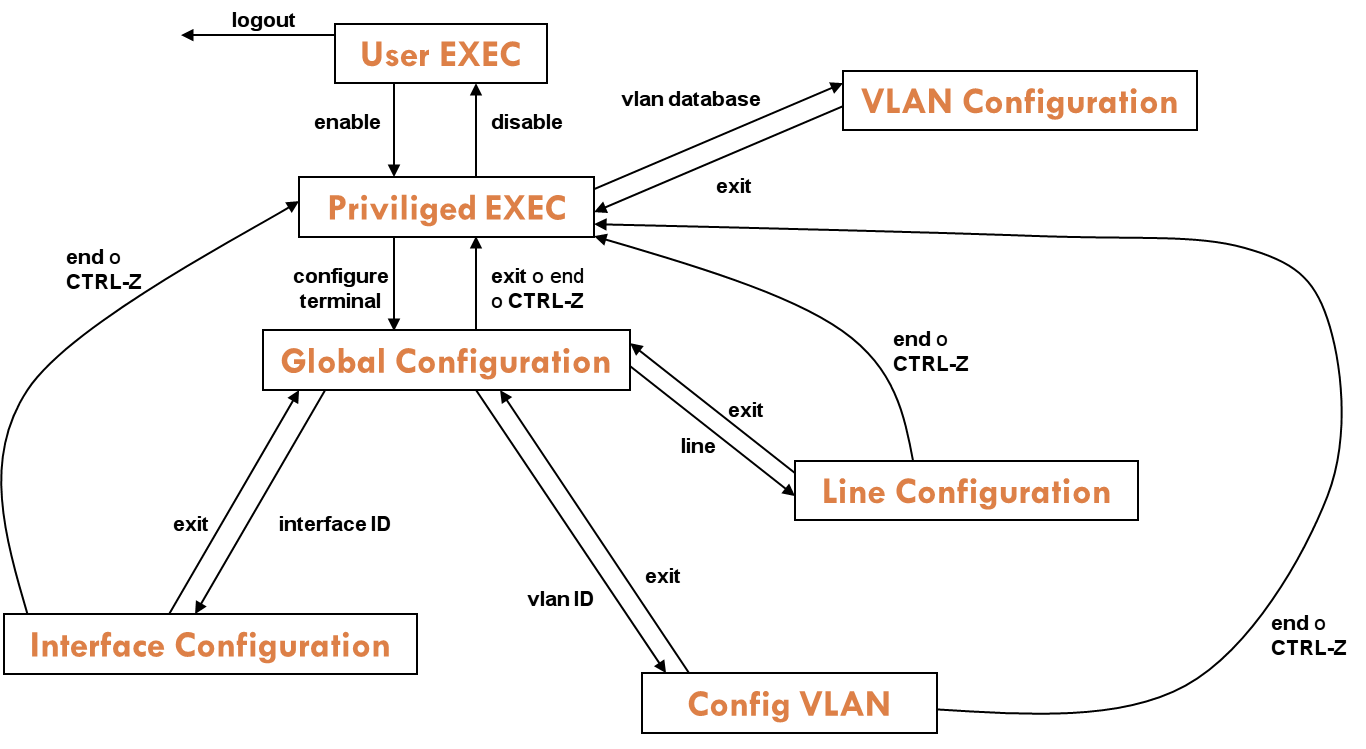
serverSocket.listen(1)

while True:

print 'The server is ready to receive'

newSocket, addr = serverSocket.accept()

thread.start\_new\_thread(handler, (newSocket,))



**Comandi**

|  |  |
| --- | --- |
| Router>  Router> show   cdp  clock  controllers  frame-relay  history  interfaces  ip  version | **Modalità User EXEC**  -CDP information  -Display the system clock  -Interface controllers status  -Frame-Relay information  -Display the session command history  -Interface status and configuration  -IP information  -System hardware and software |
| Router> enable  Router#  Router# show  access-lists  arp  cdp  clock  controllers  frame-relay  history  interfaces  ip  running-config  startup-config  version  Router# copy running-config startup-config | **Modalità Privileged EXEC**  -List access lists  -Arp table  -CDP information  -Display the system clock  -Interface controllers status  -Frame-Relay information  -Display the session command history  -Interface status and configuration  -IP information  -Current operating configuration  -Contents of startup configuration  -System hardware and software status  -Salvare la configurazione corrente |
| Router# configure terminal  Router(config)#  Router(config)# hostname HOST*NAME*  Router(config)# banner motd  Router(config)# enable secret *PASSWORD*  Router(config)# no enable secret | **Modalità Global Configuration**  -Cambiare nome al router  -Impostare messaggio del giorno  -Impostare password  -Disabilitare password |
| Router(config)# interface *TYPE SLOT/PORT*  Router(config-if)# no shutdown  Router(config-if)# shutdown  Router(config-if)# ip address *IP\_ADDRESS NETMASK*  Router(config-if)# clock rate *CLOCK\_RATE* | **Configurare interfaccia**  *-Attivare interfaccia*  *-Disattivare interfaccia*  *-Assegnare IP*  *-Clock seriale* |
| Router(config)# line vty 0 4  Router(config-line)# password *PASSWORD*  Router(config-line)# login  Router(config-line)# ^Z | **-Accesso via rete (remoto).**  -Impostare la password per l’accesso via rete |
| Router(config)# line console 0 | **Accesso via porta console** |
| Router(config)# ip dhcp pool *NAME\_POOL*  Router(dhcp-config)# default-router *ROUTER\_IP\_ADDRESS*  Router(dhcp-config)# network *NETWORK\_IP\_ADDRESS NETMASK*  Router(dhcp-config)# ip dhcp excluded-address *EXCLUDED\_IP\_ADDRESS* | **DHCP**  -Nome pool indirizzi  -Assegnare il default gateway al pool  -Definire la rete a cui appartengono gli indirizzi  -Escludere un indirizzo dal pool |
| Router(config)# ip route *DEST\_PREFIX* *DEST\_NETMASK NEXTHOP/INTERFACE*  Router(config)# no ip route *DEST\_PREFIX* *DEST\_NETMASK NEXTHOP/INTERFACE* | -Aggiungere una **rotta statica**  -Rimuovere una rotta statica |
| Router(config)# router rip  Router(config)# no router rip  Router(config-router)# version N  Router(config-router)# network A.B.C.D  Router(config-router)# passive-interface TYPE SLOT/PORT  Router# debug ip rip  Router# no debug ip rip  Router# show ip route  Router# show ip route rip  Router# show ip protocols  Router# show ip rip database | *-Abilitare* ***RIP***  *-Disabilitare RIP*  *-Scegliere la versione*  *-Definire le reti che usano RIP*  *-Configurare un’interfaccia in modalità passiva.*  *-Abilitare/disabilitare il debug per il protocollo RIP*  *-* *Ottenere la tabella di routing*  *-Visualizzare le entry nella tabella di routing ottenute con RIP*  *-* *Ottenere l'elenco dei protocolli di routing attivi e il loro stato*  *-* *Visualizzare le informazione raccolte dal routing RIP* |
| Router(config)# router ospf *ID-PROCESS*  Router(config)# no router ospf *ID-PROCESS*  Router(config-router)# network *A.B.C.D* *NET\_WILDCARD* area *N*  Router(config-router)# auto-cost reference-bandwidth *BANDWIDTH\_VALUE*  Router(config)# interface *TYPE SLOT/PORT*  Router(config-if)# ip ospf cost *COST\_VALUE* | -Abilitare **OSPF**  -Disabilitare OSPF  -Definire le reti che usano OSPF  -Modificare il valore di banda di riferimento  -Modificare la metrica costo |
| Router(config)# router eigrp *N* Router(config)# no router eigrp N  Router(config-router)# network *A.B.C.D* Router(config-router)# metric weights *TOS K1 K2*  *K3 K4 K5* | -Abilitare **EIGRP**  -Disabilitare OSPF  -Definire le reti che usano EIGRP  -Modificare i pesi delle metriche |
| Router(config)# interface *TYPE* *PORT*/*SLOT*  Router(config-if)# ip nat inside  Router(config-if)# ip nat outside  Router(config)# access-list *LIST\_NUM* permit *NET\_ADDR* *NET\_WILDCARD*  Router(config)# ip nat inside source list *LIST\_NUM* interface *OUTSIDE\_INTERFACE\_NAME* overload | **Configurazione NAT**  -definizione ruolo porte  - Creare una lista di indirizzi a cui sarà permesso il NAT  - Associare il NAT alla lista indicata prima |
| Router(config)# interface *TYPE* *PORT*/*SLOT*  Router(config-if)# ip nat inside  Router(config-if)# ip nat outside  Router(config)# ip nat inside source static tcp *IP\_INSIDE* PORT\_*INSIDE* *IP\_OUTSIDE PORT\_OUTSIDE* | **Configurazione Port Forwarding**  -definizione ruolo porte  - Associare staticamente l'indirizzo e la porta esterna a quelli interni |
| Switch> enable  Switch# show spanning-tree  Switch> enable  Switch# config  Switch(config)# spanning-tree vlan 1 priority 0 | **SPANNING TREE**  -Controllare lo stato del protocollo STP  -Impostazione di uno switch come Root Bridge |