Escuela Superio de Cómputo
Instituto Politécnico Nacional
Administración de Servicios en Red
Practica 3
Curso impartido por: Ricardo Martinez Rosales

Adrian González Pardo

25 de abril de 2021

1. Descripción y Desarrollo

Para desarrollar esta practica, previamente se estudio y se leyo acerca de los siguientes temas:

- Protocolos de Enrutamiento Estatico y Dinamico (RIP,OSPF)
- Enrutamiento multiple de resdistribución en protocolos Dinamicos
- Configuración de usuarios y de sesiones remotas
- Conexión SSH o Telnet vía Python

Para realizar esta practica las precondiciones de la misma es inicialmente configurar los routers, levantar al menos un usuario en telnet o ssh para conexión remota en todos los routers (Considerarlo usuario por defecto), encender y configurar las interfaces necesarias para trabajar. guardar el archivo y continuar con la aplicación

1.1. Aplicación desarrollada

```python
#!/usr/bin/env python3
from module_scan import *

"""
    @author:        Adrian González Pardo
    @date_update:   23/04/2021
    @github:        AdrianPardo99
"""


# Modulo que permite escanear todos los datos de una interfaz en especifico
scan_by_interface("tap0","admin","admin","1234")
```

```python
#!/usr/bin/env python3
from detecta import *
from ssh_connect import *
import os
import re
import netifaces as ni
import json

"""
    @author:        Adrian González Pardo
    @date_update:   25/04/2021
    @github:        AdrianPardo99
"""

```

```
"""
    @args:
        <interface_name> Es el nombre de la interfaz que va a trabajar para escanear toda la red
        Para comunicación SSH v2
        <user> Es el usuario por defecto en los routers a la hora de realizar todos los escaners
        <password> Es el password por defecto de los routers
        <secret> Es la clave secret a la hora de conectarse al router
"""
def scan_by_interface(interface_name="tap0",user="admin",password="admin",secret="1234"):
    # Prototipo de conexión a router cisco
    cisco={
        "device_type":"cisco_xe",
        "ip":"",
        "username":user,
        "password":password,
        "secret":secret
    }
    # Obtienen el disccionario de los datos de la red
    dic_data=ni.ifaddresses(interface_name)
    if 2 not in dic_data:
        print("No hay una dirección IPv4 en la interfaz")
        return [-1,-1]
    dic_data=dic_data[2][0]
    print(f"\n---------About---------\n{interface_name}:{dic_data}")
    addr=list(map(int,dic_data["addr"].split(".")))
    net=list(map(int,dic_data["netmask"].split(".")))

    c=determinate_prefix(net)
    # Se obtiene el identificador de la subred
    idnet=get_id_net(addr,net)
    # Se obtiene la dirección de broadcast
    range_net=get_broadcast_ip(idnet,net)

    print(f"-------Scan Network:-------\n\tID: {arr_to_ip(idnet)}/{c}\n\tNetmask: {arr_to_ip(net)
    }\n\tBroadcast: {arr_to_ip(range_net)}")

    # Se prepara para hacer is_host_up
    ips=[idnet[0],idnet[1],idnet[2],idnet[3]+1]
    responde=scan_range(ips,range_net)


    # Se filtra por primera vez que solo los elementos que sean Cisco

    ciscos=[]
    for i in range(len(responde)):
        for k,v in responde[i].items():
            if "Cisco_Router_IOS" in v:
                ciscos.append(responde[i])
    #print(f"Solo routers cisco: {ciscos}")

    # Despues de todo lo que hace el modulo hay que conectarse por ssh o telnet
    #   a los dispositivos cisco
    cmd=["sh ip int | i Internet address","sh ip int br | include up","sh run | include hostname"]
    c=0
    red={}
    net_router={}
    for i in ciscos:
        flag=False
        # Los datos del router (Interfaces)
        for k,v in i.items():
            print(f"-------Enviando comandos a router con ip: {k}-------")
            cisco["ip"]=k
            output=conectar(cisco,cmd)
            dir=re.split("\n|  Internet address is | ",output[0])
            inte=re.split("\n|      YES NVRAM  up                    up      |      YES manual up
                      up  | ",output[1])
            host_cmd=output[2].split("hostname ")[1]
            direcciones=[]
            interf=[]
            for j in dir:
                if j!="":
```

```
                    direcciones.append(j)
            for j in inte:
                if j!="":
                    interf.append(j)
            if host_cmd in red.keys():
                flag=False
            else:
                flag=True
            if flag:
                iter={}
                for j in range(len(direcciones)):
                    iter[interf[(j*2)]]=direcciones[j]
                    sub=direcciones[j].split("/")
                    pr=sub[1]
                    sub=list(map(int,sub[0].split(".")))
                    sub=arr_to_ip(get_id_net(sub,create_masc_by_prefix(int(pr))))
                    iter[f"{interf[(j*2)]}-sub"]=sub
                red[host_cmd]=iter
            dir.clear()
            inte.clear()
            direcciones.clear()
    print("\n\n\n")
    cmd=["ssh -l admin ","admin","ena","1234","sh ip int | i Internet address","sh ip int br |
    include up","sh run | include hostname","exit"]
    re_n={}
    # Obtiene los datos de la interfaz y se intenta conectar a la ip-1 a la que esta conectada
    for i in ciscos:
        for k,v in i.items():
            cisco["ip"]=k
            for l,m in red.items():
                for n,o in m.items():
                    ip_r=o.split("/")
                    if ip_r[0]!=k and "-sub" not in n:
                        ip_r=list(map(int,ip_r[0].split(".")))
                        ip_r[3]-=1
                        ip_r=arr_to_ip(ip_r)
                        # Hace la conexión puentada de un Router a otro router
                        print(f"Realizando conexión bridge entre {k} y {ip_r}")
                        cmd[0]=f"ssh -l admin {ip_r}"
                        output=conectar_bridge(cisco,cmd)
                        # Se obtienen sus datos como tabla de enrutamiento para realizar las
    configuraciones más tarde
                        host=re.split("#|\n| ",output[-2])[1]
                        dir=re.split("\n|      YES NVRAM  up                    up      |      YES
     manual up                    up  | ",output[-3])
                        inte=re.split("\n|  Internet address is | ",output[-4])
                        direcciones=[]
                        interfaces=[]
                        sub_n=[]
                        for i in range(len(dir)):
                            if ""!=dir[i] and "R" not in dir[i]:
                                direcciones.append(dir[i])
                        for i in range(len(inte)):
                            if ""!=inte[i] and "R" not in inte[i]:
                                interfaces.append(inte[i])
                                sub=inte[i].split("/")
                                pr=sub[1]
                                sub=list(map(int,sub[0].split(".")))
                                sub=arr_to_ip(get_id_net(sub,create_masc_by_prefix(int(pr))))
                                sub_n.append(sub)
                        it={}
                        for i in range(int(len(direcciones)/2)):
                            it[direcciones[i*2]]=interfaces[i]
                            it[f"{direcciones[i*2]}-sub"]=sub_n[i]
                        re_n[host]=it
    for k,v in re_n.items():
        red[k]=v
    json_routers=json.dumps(red,sort_keys=True,indent=4)
    print(f"Diccionario de routers:\n{json_routers}")

    route=[]
```

```
    conexiones=verifica_conectividad(red)
    # Se realiza las configuraciones de los routers permitiendo redistribución entre protocolos dinamicos y el estatico
    for i,j in red.items():
        route=[]
        if "1" in i:
            print(f"\nEnrutamiento estatico, preparando demas tablas de enrutamiento {i}")
            for k,v in red.items():
                if "1" not in k:
                    for l,m in v.items():
                        if "-sub" in l and m not in route and n not in v.values():
                            route.append(m)
            resultado=conexiones[verifica_index(conexiones,i)]
            parser=resultado.split(":")
            routers=parser[0].split("-")
            net=parser[1]
            route_c=[]
            for k,v in red.items():
                if "1" in k:
                    for l,m in v.items():
                        if "-sub" in l and m not in route:
                            route_c.append(m)
            route.remove(net)
            print(f"{routers[0]} enruta hacia {routers[1]} con net {route_c}")
            print(f"{routers[1]} enruta hacia {routers[0]} con net {route}")
            # Aca desarrollamos el comando en conjunto de las IP's que estan interconectadas
            # Obtenemos ip del R[0] hacia que ip salen la redirección de datos de R[1]

            ip_r1=list(red[routers[1]].values())
            ip=ip_r1.index(net)-1
            ip_r1=ip_r1[ip].split("/")[0]
            # Obtenemos ip del R[1] hacia que ip salen la redirección de datos de R[0]
            ip_r2=list(red[routers[0]].values())
            ip=ip_r2.index(net)-1
            ip_r2=ip_r2[ip].split("/")[0]

            cmd=["conf t"]
            for a in route_c:
                cmd.append(f"ip route {a} 255.255.255.0 {ip_r1}")
            cmd.append("end")
            print(f"{routers[0]} manda comandos hacia si mismo con configuracion= {cmd}")
            output=conectar_bridge(cisco,cmd)
            cmd=[f"ssh -l admin {ip_r1}","admin","ena","1234","conf t"]
            for a in route:
                cmd.append(f"ip route {a} 255.255.255.0 {ip_r2}")
            cmd.append("end")
            cmd.append("exit")
            print(f"{routers[0]} manda comandos hacia {routers[1]} con configuracion= {cmd}")
            output=conectar_bridge(cisco,cmd)
        elif "2" in i:
            print(f"\nEnrutamiento RIP {i}")
            resultado=conexiones[verifica_index(conexiones,i)]
            parser=resultado.split(":")
            routers=parser[0].split("-")
            net=parser[1]
            print(f"Conexion entre {routers[0]} y {routers[1]} con el identificador {net}")
            routes_r1=[]
            routes_r2=[]
            ip_r1=list(red[routers[0]].values())
            for i in ip_r1:
                if "/" not in i:
                    routes_r1.append(i)
            ip_r1=list(red[routers[1]].values())
            for i in ip_r1:
                if "/" not in i:
                    routes_r2.append(i)
            cmd=["conf t","router rip","ver 2","redistribute static","redistribute ospf 1","default-metric 1"]
            for i in routes_r1:
                cmd.append(f"net {i}")
            cmd.append("end")
```

```
            print(f"{routers[0]} manda comandos hacia si mismo con configuracion= {cmd}")
            output=conectar_bridge(cisco,cmd)
            # Sale la IP R[1]
            ip_r1=list(red[routers[1]].values())
            ip=ip_r1.index(net)-1
            ip_r1=ip_r1[ip].split("/")[0]
            ##########################
            cmd=[f"ssh -l admin {ip_r1}","admin","ena","1234","conf t","router rip","ver 2","
    redistribute static","redistribute ospf 1","default-metric 1"]
            for i in routes_r2:
                cmd.append(f"net {i}")
            cmd.append("end")
            cmd.append("exit")
            print(f"{routers[0]} manda comandos hacia {routers[1]} con configuracion= {cmd}")
            output=conectar_bridge(cisco,cmd)
        elif "3" in i:
            print(f"\nEnrutamiento OSPF {i}")
            resultado=conexiones[verifica_index(conexiones,i)]
            parser=resultado.split(":")
            routers=parser[0].split("-")
            net=parser[1]
            print(f"Conexion entre {routers[0]} y {routers[1]} con el identificador {net}")
            routes_r1=[]
            routes_r2=[]
            ip_r1=list(red[routers[0]].values())
            for i in ip_r1:
                if "/" not in i:
                    routes_r1.append(i)
            ip_r1=list(red[routers[1]].values())
            for i in ip_r1:
                if "/" not in i:
                    routes_r2.append(i)
            cmd=["conf t","int loop0","ip add 200.0.0.1 255.255.255.255",
                "no sh","exit","router ospf 1","ver 2","router ospf 1",
                "redistribute static metric 200 subnets",
                "redistribute rip metric 200 subnets"]
            for i in routes_r1:
                cmd.append(f"net {i} 0.0.0.255 area 0")
            cmd.append("end")
            print(f"{routers[0]} manda comandos hacia si mismo con configuracion= {cmd}")
            output=conectar_bridge(cisco,cmd)
            # Sale la IP R[1]
            ip_r1=list(red[routers[1]].values())
            ip=ip_r1.index(net)-1
            ip_r1=ip_r1[ip].split("/")[0]
            ##########################
            cmd=[f"ssh -l admin {ip_r1}","admin","ena","1234","conf t",
                "int loop0","ip add 200.0.0.2 255.255.255.255",
                "no sh","exit","router ospf 2","ver 2","router ospf 2",
                "redistribute static metric 200 subnets",
                "redistribute rip metric 200 subnets"]
            for i in routes_r2:
                cmd.append(f"net {i} 0.0.0.255 area 0")
            cmd.append("end")
            cmd.append("exit")
            print(f"{routers[0]} manda comandos hacia {routers[1]} con configuracion= {cmd}")
            output=conectar_bridge(cisco,cmd)
    print("\nSe han levantado todos los protocolos para comunicarnos entre routers")
```

```
#!/usr/bin/env python3

from scapy.all import *
import platform
import subprocess
from threading import Thread


"""
    @author:        Adrian González Pardo
    @date_update:   23/04/2021
    @github:        AdrianPardo99
```

```
"""

BLUE, RED, WHITE, YELLOW, MAGENTA, GREEN, END = '\33[94m', '\033[91m', '\33[97m', '\33[93m', '\033[1;35m', '\033[1;32m', '\033[0m'

"""
    @args:
            <ip> Convierte una lista de enteros a un string para presentación
"""
def arr_to_ip(ip):
    return f"{ip[0]}.{ip[1]}.{ip[2]}.{ip[3]}"

"""
    @args:
            <host> Es una dirección ip de tipo string
            <result> Es una lista en la cual se almacenan los datos de la salida
"""
def ping(host,result):
    param = '-n' if platform.system().lower()=='windows' else '-c'
    command = ['ping', param, '1', host]
    res=subprocess.Popen(command,stdout=subprocess.PIPE,stderr=subprocess.PIPE)
    output=res.stdout.read().decode("utf-8")
    r="100% packet loss" not in output
    msg=""
    res.terminate()
    if r:
        msg=f"{GREEN} with answer [   ]{END}"
    else:
        msg=f"{RED} without answer [x]{END}"
    result.append([r,f"{YELLOW} Send data to: {host.ljust(15)} {msg}",host,output.split("\n")[1]])

"""
    @args:
            <prefix> Es un valor entero que va hasta 0-32 para generar una mascara de red
"""
def create_masc_by_prefix(prefix):
    net=[]
    for i in range(4):
        if (prefix-8)>=0:
            net.append(255)
            prefix-=8
    if prefix==7:
        net.append(254)
    elif prefix==6:
        net.append(252)
    elif prefix==5:
        net.append(248)
    elif prefix==4:
        net.append(240)
    elif prefix==3:
        net.append(224)
    elif prefix==2:
        net.append(192)
    elif prefix==1:
        net.append(128)
    mis=4-len(net)
    for i in range(mis):
        net.append(0)
    return net

"""
    @args:
            <srcs> Es nuestra dirección ip en forma de string
            <host> Es la dirección ip en forma de string
            <result> Es una lista la cual almacenara los resultados de la función
"""
def is_host_up(srcs,host,result):
    p=sr1(IP(src=srcs,dst=host)/ICMP()/"hola",timeout=15,verbose=False)
    if p is None:
        result.append([False,f"{YELLOW} Send data to: {host.ljust(15)} {RED} without answer [x]{
    END}",host,None])
```

```
    else:
        result.append([True,f"{YELLOW} Send data to: {host.ljust(15)} {GREEN} with answer [   ]{END}",host,p.getlayer(IP).ttl])

"""
    @args:
            <net> Es la mascara de red en forma de lista y de tipo int
"""
def determinate_prefix(net):
    c=0
    for i in range(4):
        if net[i]==255:
            c+=8
        elif net[i]==254:
            c+=7
        elif net[i]==252:
            c+=6
        elif net[i]==248:
            c+=5
        elif net[i]==240:
            c+=4
        elif net[i]==224:
            c+=3
        elif net[i]==192:
            c+=2
        elif net[i]==128:
            c+=(1)
    return c

"""
    @args:
        <ip> Es una dirección ip la cual va a ser utilizada para generar el identificador de red en formato int
        <net> Es nuestra mascara de red en formato int
"""
def get_id_net(ip,net):
    idnet=[]
    for i in range(4):
        idnet.append((ip[i]&net[i]))
    return idnet

"""
    @args:
        <idnet> Es el identificador de la subred de tipo lista e int
        <net> Es la mascara de red de tipo lista e int
"""
def get_broadcast_ip(idnet,net):
    ran=[]
    for i in range(4):
        ran.append((idnet[i]|((~net[i])&0xFF)))
    return ran
"""
    @args:
        <ttl> Es un valor entero que va desde 0-255
"""
def check_os_by_ttl(ttl):
    if ttl<=64:
        return f"Unix-OS {64-ttl}"
    elif ttl>64 and ttl<=128:
        return f"MS-DOS_Windows-OS {128-ttl}"
    elif ttl>128:
        return f"Cisco_Router_IOS {255-ttl}"

"""
    @args:
        <ips> Es la primer dirección ip de la subred en formato de lista e int
        <broadcast> Es la dirección de Broadcast en formato de lista e int
"""
def scan_range(ips,broadcast):
    responde=[]
    threads=[]
```

```
    positivos=[]
    c=35
    i=0
    b=0
    while(True):
        if i%c==0 and i>0:
            for t in range(len(threads)):
                threads[t].join()
                #print(responde[t][1])
                if responde[t][0]:
                    ttl=responde[t][3].split("ttl=")[1]
                    ttl=int(ttl.split(" ")[0])
                    positivos.append({responde[t][2]:check_os_by_ttl(ttl)})
            threads=[]
            responde=[]
            b+=1
        threads.append(Thread(target=ping,args=(f"{ips[0]}.{ips[1]}.{ips[2]}.{ips[3]}",responde)))
        threads[-1].start()
        i+=1
        if ips[3]+1==256:
            ips[3]=0
            if ips[2]+1==256:
                ips[2]=0
                if ips[1]+1==256:
                    ips[1]=0
                else:
                    ips[1]+=1
            else:
                ips[2]+=1
        else:
            ips[3]+=1
        if ips==broadcast:
            break
    for t in range(len(threads)):
        threads[t].join()
        #print(responde[t][1])
        if responde[t][0]:
            ttl=responde[t][3].split("ttl=")[1]
            ttl=int(ttl.split(" ")[0])
            positivos.append({responde[t][2]:check_os_by_ttl(ttl)})
    return positivos

"""
    @args:
        <dict> es el diccionario de routers para ver las interconexiones que hay entre ellos
"""
def verifica_conectividad(dict):
    conexiones=[]
    for i,j in dict.items():
        for k,v in dict.items():
            if k!=i:
                for a,b in v.items():
                    if b in j.values():
                        if (f"{i}-{k}:{b}" not in conexiones) and (f"{k}-{i}:{b}" not in
    conexiones):
                            conexiones.append(f"{i}-{k}:{b}")
    return conexiones

def verifica_index(arr,patern):
    c=0
    for i in arr:
        if patern in i:
            break
        c+=1
    return c
```

```
#!/usr/bin/env python3
from netmiko import ConnectHandler

"""
    @author:        Adrian González Pardo
```

```
    @date_update:   23/04/2021
    @github:        AdrianPardo99
"""

"""
    @args:
        <cisco> Es el diccionario que contiene los datos para la conexion
        <cmd> Es la lista de comandos que va a ejecutar netmiko
"""
def conectar(cisco,cmd):
    net_connect = ConnectHandler(**cisco)
    net_connect.enable()
    output=[]
    for i in range(len(cmd)):
        output.append(net_connect.send_command(cmd[i]))
    return output

"""
    A diferencia de la función de arriba esta puede interconectarse con
    routers con routers y no equipo a router, en forma de puente la conexión
    @args:
        <cisco> Es el diccionario que contiene los datos para la conexion
        <cmd> Es la lista de comandos que va a ejecutar netmiko
"""
def conectar_bridge(cisco,cmd):
    net_connect = ConnectHandler(**cisco)
    net_connect.enable()
    output=[]
    for i in range(len(cmd)):
        output.append(net_connect.send_command_timing(cmd[i]))
    return output
```

De modo que estos scripts hacen toda la aplicación y posible la solución de la configuración de los routers

2. Parte de demostración

Para la demostración de como se desarrollo y una pequeña explicación se grabo un vídeo que se puede ver aquí destacando y pidiendo disculpa por la cantidad de sueño que se expresa directamente o indirectamente durante la grabación y presentación.
Por otro lado se anexan imagenes de las pruebas y del como se realizaron:

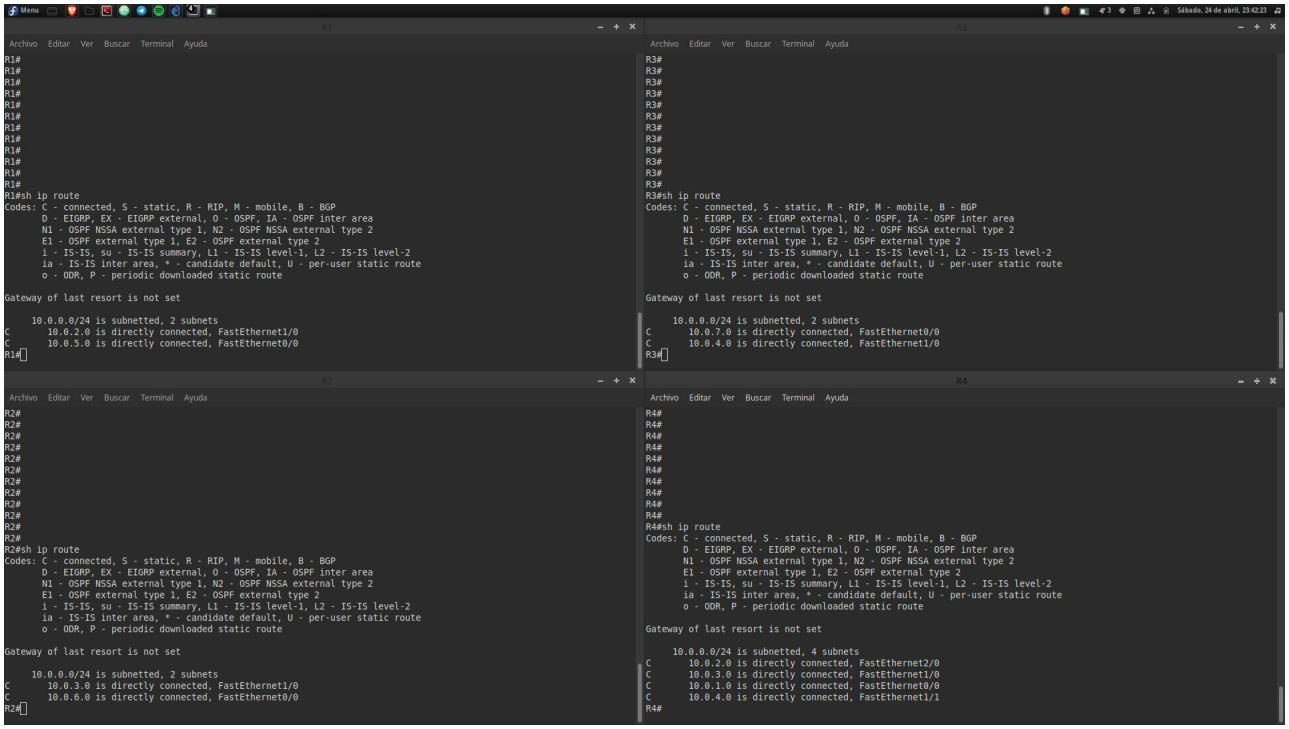

Figura 1: Tablas de enrutamiento sin configurar

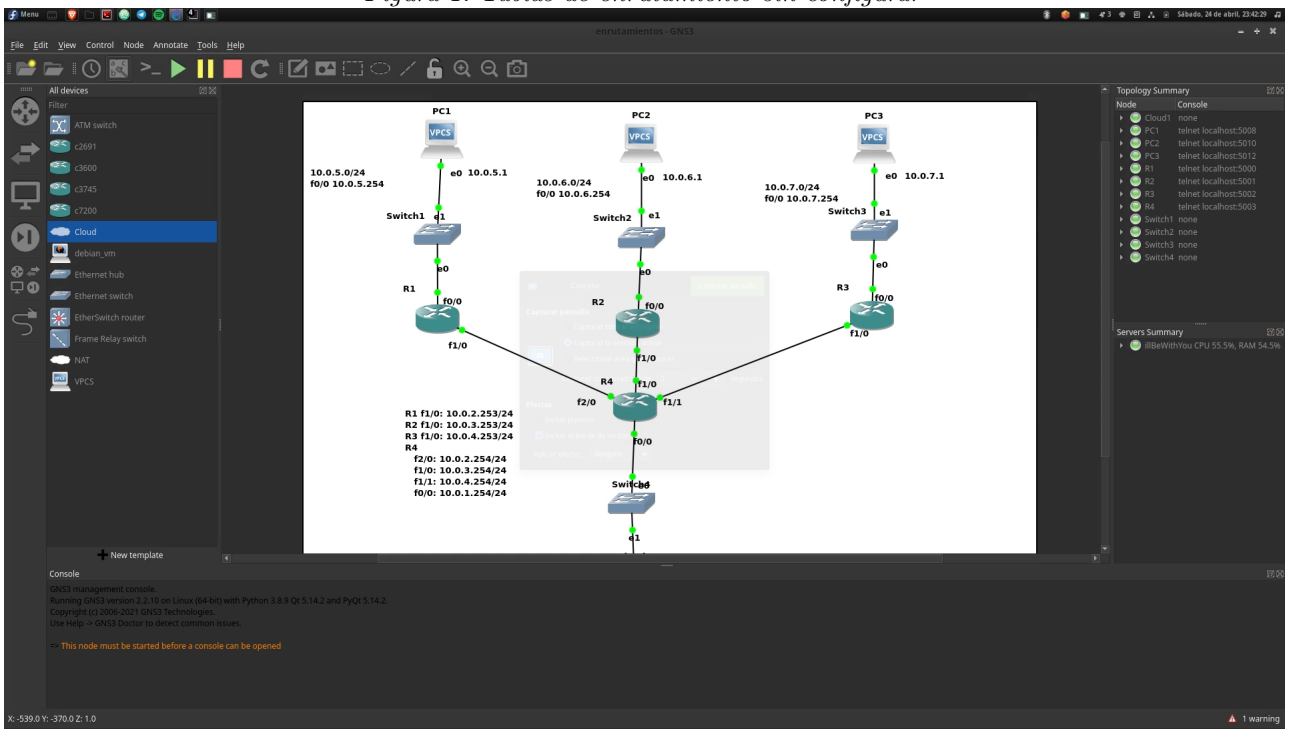

Figura 2: Topología de red

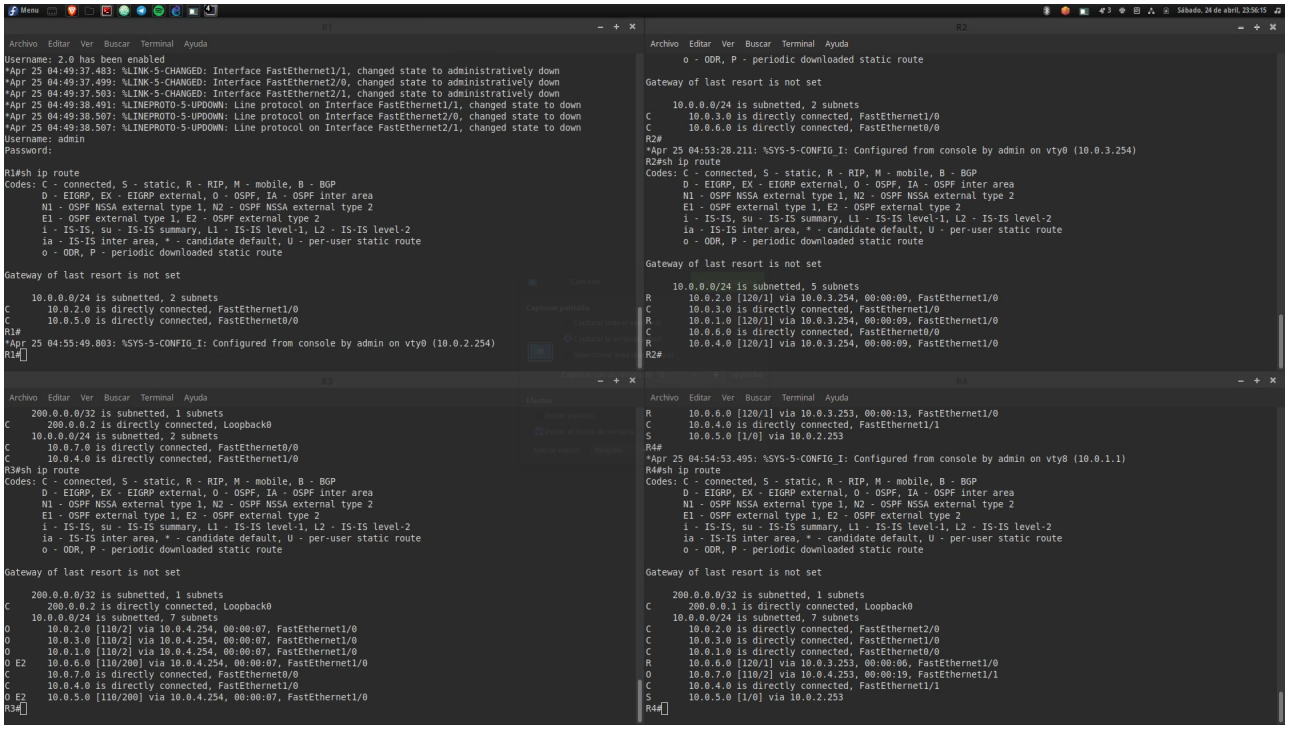

Figura 3: Tablas de enrutamiento despues de ejecución del script

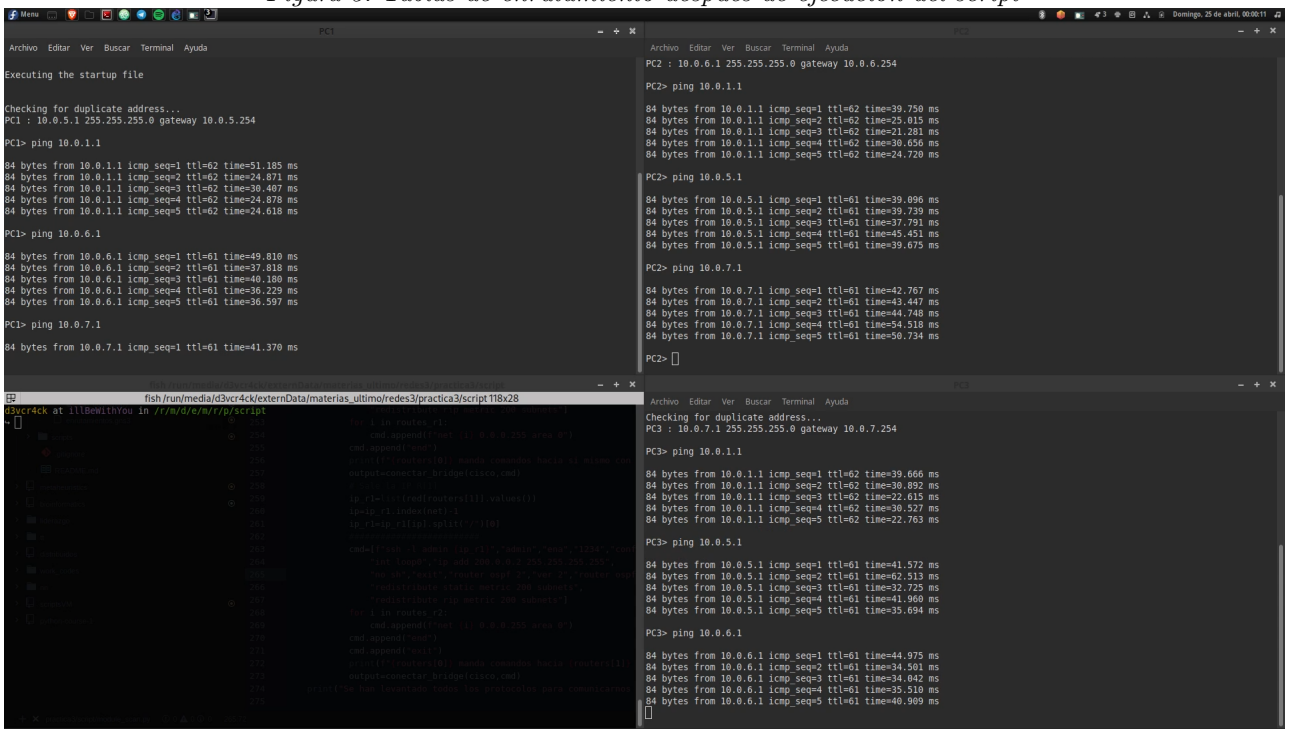

Figura 4: Pings entre subredes