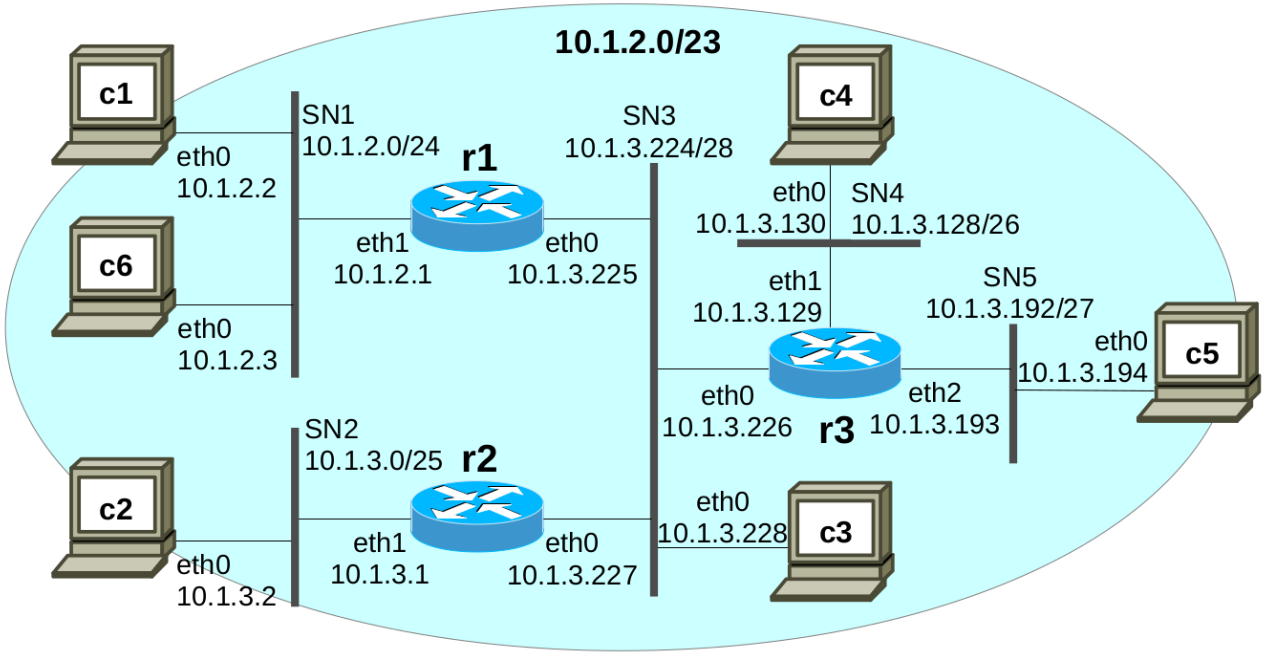
THE CONFIGURATION OF A NETWORK IN LINUX

Georgescu Marius-Daniel

Project Description

The purpose of this project is to learn the basics of setting up and configuring a network in Linux. This work is done using Netkit emulator. The network that we'll use in our project is represented in figure 1. The network's IP address is 10.1.2.0/23.



The IP addresses of every subnetwork are detailed in the following table:

Nume subrețea	Număr adrese	Prefixul blocului	Intervalul de adrese	Masca de subrețea
SN1	256	10.1.2.0/24	10.1.2.0 - 10.1.2.255	255.255.255.0
SN2	128	10.1.3.0/25	10.1.3.0 - 10.1.3.127	255.255.255.128
SN3	16	10.1.3.224/28	10.1.3.224 - 10.1.3.239	255.255.255.240
SN4	64	10.1.3.128/26	10.1.3.128 - 10.1.3.191	255.255.255.192
SN5	32	10.1.3.192/27	10.1.3.192 - 10.1.3.223	255.255.255.224

1.Interfaces configuration

Initially, the Ethernet interfaces and the routing tables of the devices are not configured. We can see this by executing the “ifconfig” and “route” commands.

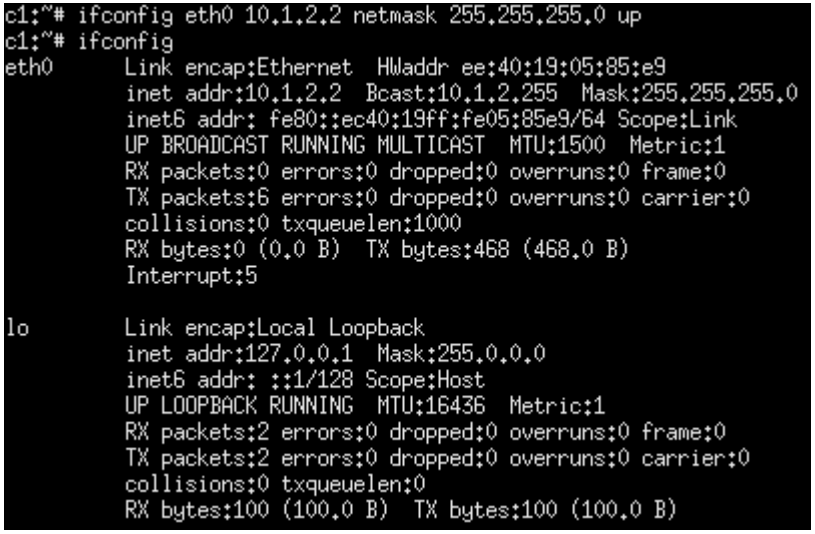
```
c1:~# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:2 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:100 (100.0 B)  TX bytes:100 (100.0 B)
```

```
c1:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
c1:~#
```

To configure the interfaces, the “ifconfig” command will be used again, more exactly:

ifconfig eth0 10.1.2.2 netmask 255.255.255.0 up.

This command activates the eth0 interface and assigns the IP address and the subnet mask.



We now need to do the same for every pc, with the information provided in the table that I have inserted earlier.

```
c2:~# ifconfig eth0  10.1.3.2 netmask 255.255.255.128 up
c2:~# ifconfig
eth0      Link encap:Ethernet  HWaddr f2:3b:18:36:f3:fd
          inet addr:10.1.3.2  Bcast:10.1.3.127  Mask:255.255.255.128
          inet6 addr: fe80::f03b:18ff:fe36:f3fd/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:238 (238.0 B)
          Interrupt:5

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:2 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:100 (100.0 B)  TX bytes:100 (100.0 B)
```

An example for c2.

We also have to do this for each router in our network. I'll only insert an example for r1, as the others are configured in an identical way.

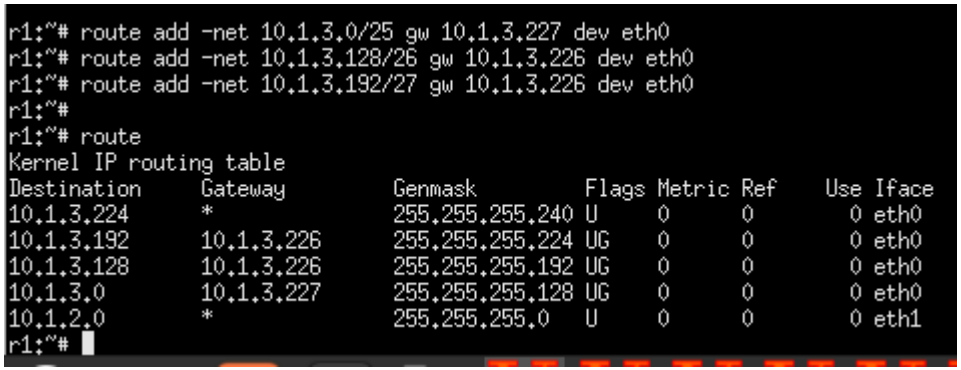
```
r1:~# ifconfig eth1 10.1.2.1 netmask 255.255.255.0 up
r1:~# ifconfig eth0 10.1.3.225 netmask 255.255.255.240 up
r1:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 0e:ab:f8:0c:10:4b
          inet addr:10.1.3.225  Bcast:10.1.3.239  Mask:255.255.255.240
          inet6 addr: fe80::cab:f8ff:fe0c:104b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:384 (384.0 B)  TX bytes:238 (238.0 B)
          Interrupt:5

eth1      Link encap:Ethernet  HWaddr fa:de:dc:30:96:57
          inet addr:10.1.2.1  Bcast:10.1.2.255  Mask:255.255.255.0
          inet6 addr: fe80::f8de:dcff:fe30:9657/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:12 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:768 (768.0 B)  TX bytes:468 (468.0 B)
          Interrupt:5
```

2. Routing tables

We'll first configure the routers routing tables. A router has to manage IP packets that come from every source in the network all the way to every destination

In our network, r1 is directly connected to SN1 and SN3, so it already knows the routes to these subnetworks. The only thing left to do with this router is to assign routes to SN2, SN4 and SN5. To do so, we use the "route add" command.



R2 is directly connected to SN2 and SN3, so it already knows the routes to these subnetworks. The only thing left to do with this router is to assign routes to SN1, SN4 and SN5. For r3, we need to assign routes to SN2 and SN1.

```
r2:~# route add -net 10.1.2.0/24 gw 10.1.3.225 dev eth0
r2:~# route add -net 10.1.3.128/26 gw 10.1.3.226 dev eth0
r2:~# route add -net 10.1.3.192/27 gw 10.1.3.226 dev eth0
r2:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.1.3.224      *               255.255.255.240 U     0      0        0 eth0
10.1.3.192      10.1.3.226      255.255.255.224 UG    0      0        0 eth0
10.1.3.128      10.1.3.226      255.255.255.192 UG    0      0        0 eth0
10.1.3.0        *               255.255.255.128 U     0      0        0 eth1
10.1.2.0        10.1.3.225      255.255.255.0   UG    0      0        0 eth0
r2:~#
```

```
r3:~# route add -net 10.1.3.0/25 gw 10.1.3.227 dev eth0
r3:~# route add -net 10.1.2.0/24 gw 10.1.3.225 dev eth0
r3:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.1.3.224      *               255.255.255.240 U     0      0        0 eth0
10.1.3.192      *               255.255.255.224 U     0      0        0 eth2
10.1.3.128      *               255.255.255.192 U     0      0        0 eth1
10.1.3.0        10.1.3.227      255.255.255.128 UG    0      0        0 eth0
10.1.2.0        10.1.3.225      255.255.255.0   UG    0      0        0 eth0
r3:~#
```

The only thing remaining is to configure every PC's routing tables. For each PC, we add a default gateway.

```
c1:~# route add default gw 10.1.2.1 dev eth0
c1:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.1.2.0        *               255.255.255.0   U     0      0        0 eth0
default         10.1.2.1        0.0.0.0         UG    0      0        0 eth0
c1:~#
```

```
c2:~# route add default gw 10.1.3.1 dev eth0
c2:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.1.3.0        *               255.255.255.128 U     0      0        0 eth0
default         10.1.3.1        0.0.0.0         UG    0      0        0 eth0
c2:~#
```

```
c3:~# route add default gw 10.1.3.228 dev eth0
c3:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.1.3.224      *               255.255.255.240 U     0      0        0 eth0
default         10.1.3.228      0.0.0.0         UG    0      0        0 eth0
c3:~#
```

///And in the same way for PC4, PC5 and PC6.

3. ARP Protocol. Direct Packet Transfer

Next up, we'll analyze in detail the communications that happen between every equipment during the IP packet transfer in our network. To deliver a packet, the IP module consults the local routing table and identifies the route that the packets need to follow, based on the destination address.

Once this information is obtained, the IP module needs to call the layer 2 service (data link) to transfer the packet to the equipment indicated by the route. But we have one more problem: the route specifies the IP address of the equipment's interface, while the layer 2 service and protocol needs a level 2 address of the interface. Here comes the MAC address into use.

In IPv4 networks, the correspondence between the IP and MAC addresses is determined by ARP (Address Resolution Protocol).

First, we start with the simplest case: a communication between equipments that are connected to the same data link (so the same IP subnetwork). In this case, the IP packet can be sent directly from source to destination using the level 2 protocol.

So, we follow the next steps:

We start the capture on r1's eth1 interface:

```
r1:~# tcpdump -s0 -ten -i eth1
```

Then, we generate traffic between c1 and c6:

```
c1:~# ping -c 2 10.1.2.3
PING 10.1.2.3 (10.1.2.3) 56(84) bytes of data.
64 bytes from 10.1.2.3: icmp_seq=1 ttl=64 time=8.66 ms
64 bytes from 10.1.2.3: icmp_seq=2 ttl=64 time=0.266 ms

--- 10.1.2.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1018ms
rtt min/avg/max/mdev = 0.266/4.463/8.660/4.197 ms
c1:~#
```

tcpdump outputs this:

```
r1:~# tcpdump -s0 -ten -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
ee:40:19:05:85:e9 > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806), length 42: arp wh
o-has 10.1.2.3 tell 10.1.2.2
1e:5e:40:ee:66:05 > ee:40:19:05:85:e9, ethertype ARP (0x0806), length 42: arp re
ply 10.1.2.3 is-at 1e:5e:40:ee:66:05
ee:40:19:05:85:e9 > 1e:5e:40:ee:66:05, ethertype IPv4 (0x0800), length 98: 10.1.
2.2 > 10.1.2.3: ICMP echo request, id 2562, seq 1, length 64
1e:5e:40:ee:66:05 > ee:40:19:05:85:e9, ethertype IPv4 (0x0800), length 98: 10.1.
2.3 > 10.1.2.2: ICMP echo reply, id 2562, seq 1, length 64
ee:40:19:05:85:e9 > 1e:5e:40:ee:66:05, ethertype IPv4 (0x0800), length 98: 10.1.
2.2 > 10.1.2.3: ICMP echo request, id 2562, seq 2, length 64
1e:5e:40:ee:66:05 > ee:40:19:05:85:e9, ethertype IPv4 (0x0800), length 98: 10.1.
2.3 > 10.1.2.2: ICMP echo reply, id 2562, seq 2, length 64
1e:5e:40:ee:66:05 > ee:40:19:05:85:e9, ethertype ARP (0x0806), length 42: arp wh
o-has 10.1.2.2 tell 10.1.2.3
ee:40:19:05:85:e9 > 1e:5e:40:ee:66:05, ethertype ARP (0x0806), length 42: arp re
ply 10.1.2.2 is-at ee:40:19:05:85:e9
^C
```

The “ping” programme requests the transmission from c1 to c6 of an ICMP Echo Request message, to which c6 responds with an ICMP Echo Reply message. This dialog is repeated 2 times. The ICMP messages are sent using IP protocols and Ethernet. The ARP Request/Reply allows to find the MAC addresses corresponding to the IP addresses.

4. Indirect Packet Transfer

We’ll now analyze the communication between equipments that are not directly connected (they are not in the same subnetwork). Therefore, the packets are sent through the routers. We start the capture on r1 eth1, r2 eth1 and r3 eth0.

```
r1:~# tcpdump -s0 -ten -i eth1
```

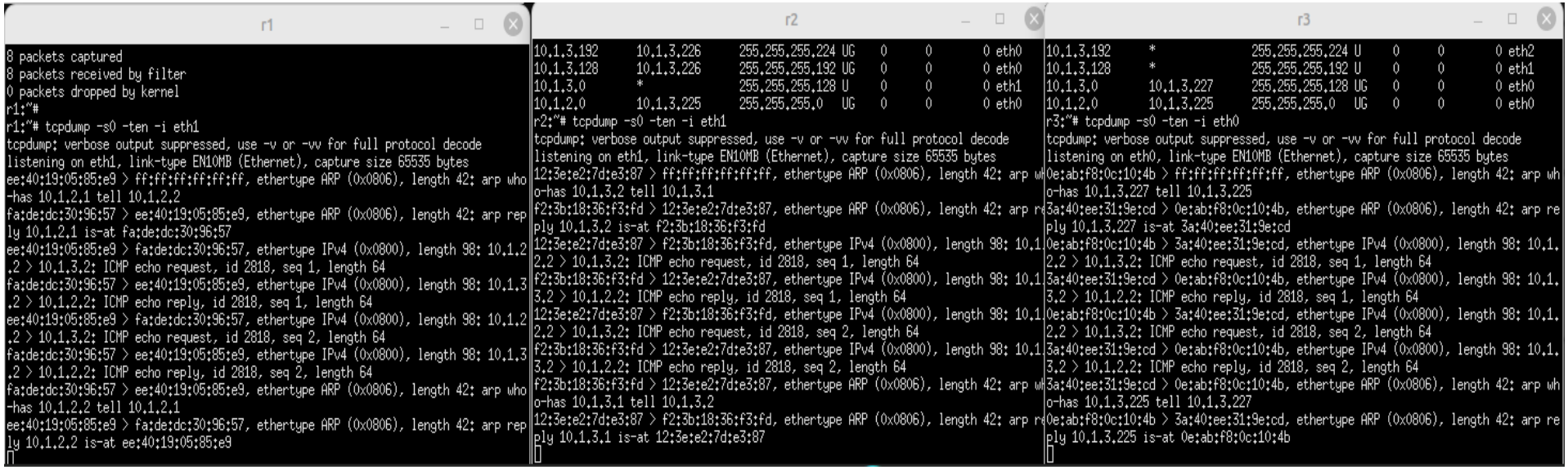
```
r2:~# tcpdump -s0 -ten -i eth1
```

```
r3:~# tcpdump -s0 -ten -i eth0
```

Then, ping c1 and c2.

```
c1:~# ping -c 2 10.1.3.2
```

The packets are sent on c1-r1-r2-c2 way and then in a reverse way.



Tcpdump outputs on r1, r2 and r3.