Otto-Friedrich-University Bamberg

Professorship for Computer Science,
Communication Services, Telecommunication,
Systems and Computer Networks

Foundation of Internet Communication

Assignment-02

Submitted by:
Group J

Reem Eslam Mohamed Mekky Khalil
Shivasharan Reddy
Azar Ghadami
Reema Miranda
Manjunath B Marigoudar

Supervisor: Prof. Dr. Udo Krieger

Bamberg, May 24, 2020
Summer Term 2020

Contents

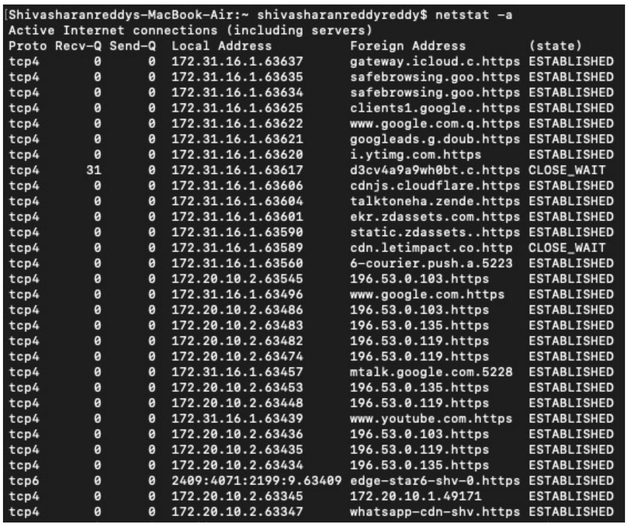
List of Figures

Chapter 1

The Netstat Command

To investigate the netstat command, you can either use any Linux machine, or you can create a Kathara node.

Summary: netstat command provides information about the network connections, the ports that are in use, and the processes using them. The -a (all) option makes netstat show all the connected and waiting sockets. As shown below: *netstat -a*



```
Shivasharanreddys-MacBook-Air:~ shivasharanreddyreddy$ netstat -a
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0      0  172.31.16.1.63637      gateway.icloud.c.https ESTABLISHED
tcp4       0      0  172.31.16.1.63635      safebrowsing.goo.https ESTABLISHED
tcp4       0      0  172.31.16.1.63634      safebrowsing.goo.https ESTABLISHED
tcp4       0      0  172.31.16.1.63625      clients1.google..https ESTABLISHED
tcp4       0      0  172.31.16.1.63622      www.google.com.q.https ESTABLISHED
tcp4       0      0  172.31.16.1.63621      googleads.g.doub.https ESTABLISHED
tcp4       0      0  172.31.16.1.63620      i.ytimg.com.https      ESTABLISHED
tcp4      31      0  172.31.16.1.63617      d3cv4a9a9wh0bt.c.https CLOSE_WAIT
tcp4       0      0  172.31.16.1.63606      cdnjs.cloudflare.https ESTABLISHED
tcp4       0      0  172.31.16.1.63604      talktoneha.zende.https ESTABLISHED
tcp4       0      0  172.31.16.1.63601      ekr.zdassets.com.https ESTABLISHED
tcp4       0      0  172.31.16.1.63590      static.zdassets..https ESTABLISHED
tcp4       0      0  172.31.16.1.63589      cdn.letimpact.co.http  CLOSE_WAIT
tcp4       0      0  172.31.16.1.63560      6-courier.push.a.5223  ESTABLISHED
tcp4       0      0  172.20.10.2.63545      196.53.0.103.https     ESTABLISHED
tcp4       0      0  172.31.16.1.63496      www.google.com.https   ESTABLISHED
tcp4       0      0  172.20.10.2.63486      196.53.0.103.https     ESTABLISHED
tcp4       0      0  172.20.10.2.63483      196.53.0.135.https     ESTABLISHED
tcp4       0      0  172.20.10.2.63482      196.53.0.119.https     ESTABLISHED
tcp4       0      0  172.20.10.2.63474      196.53.0.119.https     ESTABLISHED
tcp4       0      0  172.31.16.1.63457      mtalk.google.com.5228  ESTABLISHED
tcp4       0      0  172.20.10.2.63453      196.53.0.135.https     ESTABLISHED
tcp4       0      0  172.20.10.2.63448      196.53.0.119.https     ESTABLISHED
tcp4       0      0  172.31.16.1.63439      www.youtube.com.https  ESTABLISHED
tcp4       0      0  172.20.10.2.63436      196.53.0.103.https     ESTABLISHED
tcp4       0      0  172.20.10.2.63435      196.53.0.119.https     ESTABLISHED
tcp4       0      0  172.20.10.2.63434      196.53.0.135.https     ESTABLISHED
tcp6       0      0  2409:4071:2199:9.63409 edge-star6-shv-0.https ESTABLISHED
tcp4       0      0  172.20.10.2.63345      172.20.10.1.49171      ESTABLISHED
tcp4       0      0  172.20.10.2.63347      whatsapp-cdn-shv.https ESTABLISHED
```

Figure 1.1: netstat Command

1.1 Display the information on the TCP and UDP ports that are currently in use.

netstat -aut

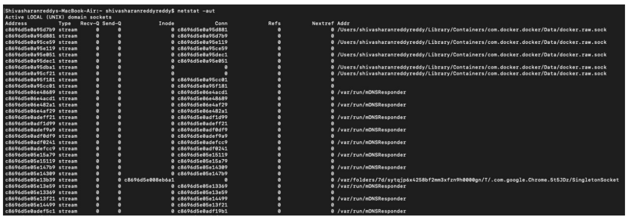


Figure 1.2: netstat Command showing TCP and IP ports in use

1.2 Display the statistics of the various networking protocols

netstat -s

```
Shivasharanreddys-MacBook-Air:~ shivasharanreddyreddy$ netstat -s
tcp:
        0 packet sent
                0 data packet (0 byte)
                0 data packet (0 byte) retransmitted
                0 resend initiated by MTU discovery
                0 ack-only packet (0 delayed)
                0 URG only packet
                0 window probe packet
                0 window update packet
                0 control packet
                0 data packet sent after flow control
                0 challenge ACK sent due to unexpected SYN
                0 challenge ACK sent due to unexpected RST
                0 checksummed in software
                        0 segment (0 byte) over IPv4
                        0 segment (0 byte) over IPv6
        0 packet received
                0 ack (for 0 byte)
                0 duplicate ack
                0 ack for unsent data
                0 packet (0 byte) received in-sequence
                0 completely duplicate packet (0 byte)
                0 old duplicate packet
                0 received packet dropped due to low memory
                0 packet with some dup. data (0 byte duped)
                0 out-of-order packet (0 byte)
                0 packet (0 byte) of data after window
                0 window probe
                0 window update packet
                0 packet recovered after loss
                0 packet received after close
                0 bad reset
                0 discarded for bad checksum
                0 checksummed in software
                        0 segment (0 byte) over IPv4
                        0 segment (0 byte) over IPv6
                0 discarded for bad header offset field
                0 discarded because packet too short
        0 connection request
        0 connection accept
        0 bad connection attempt
        0 listen queue overflow
        0 connection established (including accepts)
        0 connection closed (including 0 drop)
```

Figure 1.3: netstat Command, Showing only TCP ports

1.3 Suppose you want to write a small application that needs the process id (PID) of a given application. In order to achieve this, use a command of your choice, e.g. grep, sed or awk, to filter the output of netstat. Your application should only deliver the port number of a particular application (e.g. inetd or sshd), identified by the PID, as parameter.

Command in Linux: *netstat -anp —grep ”sshd”*
* As there is not enough network activities working with Kathara, the equivalent option windows (findstr) has been run.
The command is: *netstat -a -n -p —findstr 8452*
8452 for example, is the PID for windows host process. Here is the result:

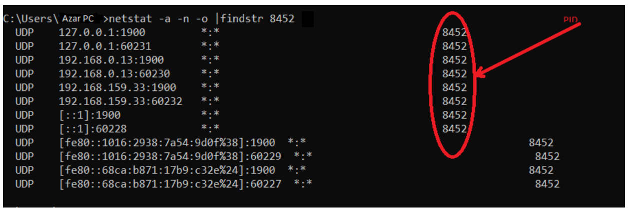


Figure 1.4: grep command to filter output of netstat

Chapter 2

Linux PC Configuration as a simple IP router

In this section we are going to set up configuration of a Linux PC as a simple IP router with two network interface

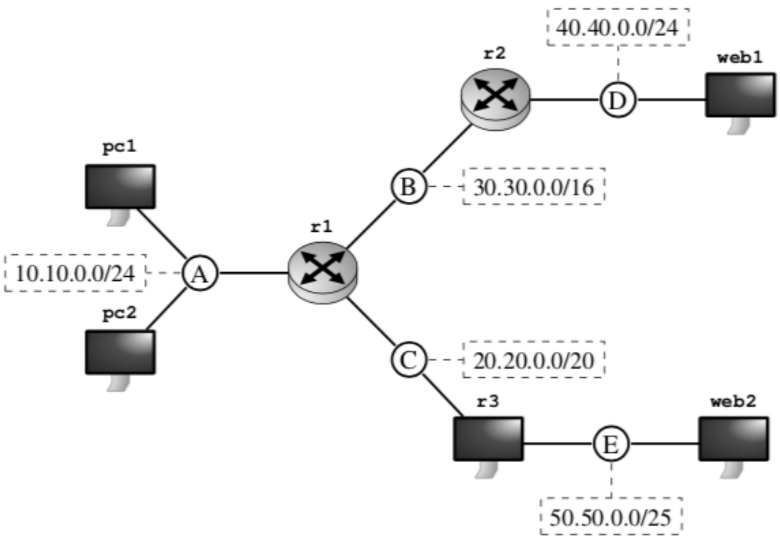


Figure 2.1: Linux PC as IP Router

2.1 Lab.conf File

```
LAB_DESCRIPTION="Assignment 02 - Static Routing Lab"
LAB_AUTHOR="Reema Miranda"

r1[0]="A"
r1[1]="B"
r1[2]="C"
r1[image]="unibaktr/vyos"

r2[0]="B"
r2[1]="D"
r2[image]="unibaktr/vyos"

r3[0]="C"
r3[1]="E"
r3[image]="kathara/base"
r3[sysctl]="net.ipv4.conf.all.forwarding=1"

pc1[0]="A"
pc1[image]="kathara/base"

pc2[0]="A"
pc2[image]="kathara/base"

web1[0]="D"
web1[image]="kathara/base"

web2[0]="E"
web2[image]="kathara/base"
```

Figure 2.2: Lab.Conf File

2.1.1 Creating the four devices

2.1.2 Creating 2 PCs ,2 web devices and 3 routers each assigned a network interface and unique collision domain

1.kathara lstart: to start the lab file where all machines are created simultaneously.

```
Reemas-MacBook-Air:Assignment GIK Network reemamiranda$ kathara lstart
======================= Starting Lab =======================
Description: Assignment 02 - Static Routing Lab
Author(s): Reema Miranda

==============================================================
Deploying links... |######                              | 1/5Deploying links... |###
Deploying links... |################################| 5/5
Deploying machines... |################################| 7/7
Reemas-MacBook-Air:Assignment GIK Network reemamiranda$ 
```

Figure 2.3: Created 2 PCs ,2 web devices,3 routers each assigned a network interface with unique collision domain

2.2 Docker Image

docker ps:To check docker images for each container

```
Assignment GIK Network — -bash — 80×24
~ — -bash                    ...top/Assignment GIK Network — -bash

CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
59b9b4f51c2d        kathara/base        "bash"              42 minutes ago
Up 42 minutes                           kathara_reemamiranda_pc2_rX_JXwnNACWUDJw
KmIzPAA
8b5d60c6aeea        kathara/base        "bash"              42 minutes ago
Up 42 minutes                           kathara_reemamiranda_web1_rX_JXwnNACWUDJ
wKmIzPAA
eeb34d9e083b        kathara/base        "bash"              42 minutes ago
Up 42 minutes                           kathara_reemamiranda_web2_rX_JXwnNACWUDJ
wKmIzPAA
dfe7e1b96fe6        kathara/base        "bash"              42 minutes ago
Up 42 minutes                           kathara_reemamiranda_pc1_rX_JXwnNACWUDJw
KmIzPAA
743f544a105d        kathara/base        "bash"              42 minutes ago
Up 42 minutes                           kathara_reemamiranda_r3_rX_JXwnNACWUDJwK
mIzPAA
5e234335de8a        unibaktr/vyos       "/sbin/init"        42 minutes ago
Up 42 minutes                           kathara_reemamiranda_r1_rX_JXwnNACWUDJwK
mIzPAA
93bb6742b8b6        unibaktr/vyos       "/sbin/init"        42 minutes ago
Up 42 minutes                           kathara_reemamiranda_r2_rX_JXwnNACWUDJwK
mIzPAA
Reemas-MacBook-Air:Assignment GIK Network reemamiranda$ 
```

Figure 2.4: Image Ports for each Machine

2.2.1 Startup Files

Assignment-02_Config and startup Files_Assignment-02_pc1.startup

```
ifconfig eth0 10.10.0.10 netmask 255.255.255.0 broadcast 10.10.0.255 up
# route add default gw 10.10.0.1
```

Figure 2.5: PC1.startup file

Assignment-02_Config and startup Files_Assignment-02_pc2.startup

```
ifconfig eth0 10.10.0.11 netmask 255.255.255.0 broadcast 10.10.0.255 up
# route add default gw 10.10.0.1
```

Figure 2.6: PC2.startup file

Assigment-02_web1.startup 106 Bytes

```
ifconfig eth0 40.40.0.100 netmask 255.255.255.0 broadcast 40.40.0.255 up
#route add default gw 40.40.0.2
```

Figure 2.7: web1.startup file

Assigment-02_web2.startup 107 Bytes

```
ifconfig eth0 50.50.0.100 netmask 255.255.255.128 broadcast 50.50.0.128 up
#route add default gw 50.50.0.3
```

Figure 2.8: web2.startup file

Assignment-02_r2.startup 215 Bytes

```
ifconfig eth0 30.30.0.2 netmask 255.255.0.0 broadcast 30.30.255.255 up
ifconfig eth1 40.40.0.2 netmask 255.255.255.0 broadcast 40.40.0.255 up
#route add -net 10.10.0.0 netmask 255.255.255.0 gw 30.30.0.1 dev eth0
```

Figure 2.9: r2.startup file

Assigment-02_r3.startup 216 Bytes

```
ifconfig eth0 20.20.0.3 netmask 255.255.240.0 broadcast 20.20.14.255 up
ifconfig eth1 50.50.0.3 netmask 255.255.255.128 broadcast 50.50.0.127 up
#route add -net 10.10.0.0 netmask 255.255.255.0 gw 20.20.0.1 dev eth0

```

Figure 2.10: r3.startup file

Assignment-02_r1.startup 345 Bytes

```
ifconfig eth0 10.10.0.1 netmask 255.255.255.0 broadcast 10.10.0.255 up
ifconfig eth1 30.30.0.1 netmask 255.255.0.0 broadcast 30.30.255.255 up
ifconfig eth2 20.20.0.1  netmask 255.255.240.0 broadcast 20.20.0.240 up
#route add -net 40.40.0.0 netmask 255.255.0.0 gw 30.30.0.2 eth1
#route add -net 50.50.0.0 netmask 255.255.240.0 gw  20.20.0.3 eth2
```

Figure 2.11: r1.startup file

2.3 Configuring the four devices

The purpose of using subnetting is to reduce the traffic congestion. Ex:pc1 and pc2 connect to same router r1 over the same interface eth0.As both send packets to router r1 there will be traffic congestion and also the speed of the network reduces .Hence we are subnetting the IP address.

2.3.1 Configuring an IP address for PC1,PC2,Web1 and Web2

```
reemamiranda — root@pc1: / — kathara ◂ kathara connect -l pc1 — 80×24
root@pc1:/# ifconfig eth0 10.10.0.10 netmask 255.255.255.0 broadcast 10.10.0.255
 up
root@pc1:/# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
3: ip6tnl0@NONE: <NOARP> mtu 1452 qdisc noop state DOWN group default qlen 1000
    link/tunnel6 :: brd ::
176: eth0@if175: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UP group default qlen 1000
    link/ether 32:71:fc:ef:2e:51 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.10.0.10/24 brd 10.10.0.255 scope global eth0
       valid_lft forever preferred_lft forever
root@pc1:/#
```

Figure 2.12: configured IP address for PC1

```
reemamiranda — root@pc2: / — kathara ◂ kathara connect -l pc2 — 80×24
root@pc2:/# ifconfig eth0 10.10.0.11 netmask 255.255.255.0 broadcast 10.10.0.255
 up
root@pc2:/#
```

Figure 2.13: configured IP address for PC2

```
reemamiranda — root@web1: / — kathara ◂ kathara connect -l web1 — 80×24
root@web1:/# ifconfig eth0 40.40.0.100 netmask 255.255.255.0 broadcast 40.40.0.2
55 up
root@web1:/#
```

Figure 2.14: configured IP address for web1

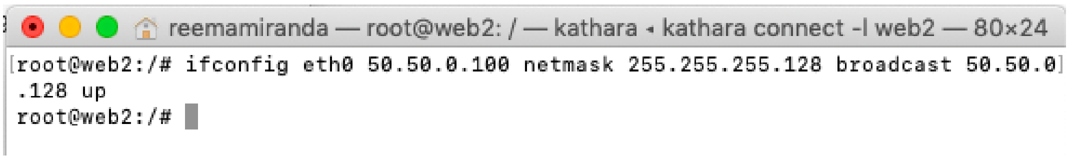


Figure 2.15: configured IP address for web2

2.3.2 Configuring an IP address for R1 ,R2 and R3 routers

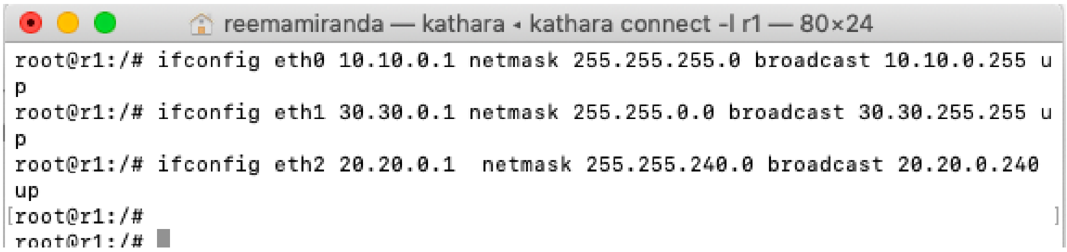


Figure 2.16: configured IP address for R1

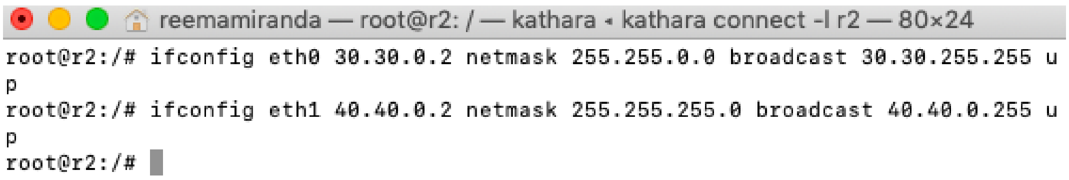


Figure 2.17: configured IP address for R2

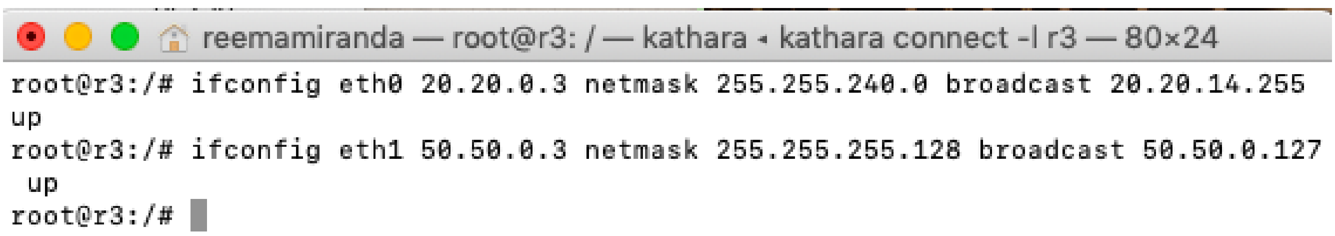


Figure 2.18: configured IP address for R3

2.3.3 Configuring default gateway to reach other networks

reemamiranda — root@pc1: / — kathara ◂ kathara connect -l pc1 — 80×24

```
root@pc1:/# # route add default gw 10.10.0.1
root@pc1:/# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.10.0.0       0.0.0.0         255.255.255.0   U     0      0        0 eth0
root@pc1:/#
```

Figure 2.19: Default gateway for PC1

In order to establish connection between host on different network we must add the default route. Ex: To ping from pc1 to web1 which are on different network interface we must add default route(gateway) from pc1 to r1.Through this gateway(ip address) we can reach other networks

reemamiranda — root@pc2: / — kathara ◂ kathara connect -l pc2 — 80×24

```
root@pc2:/# # route add default gw 10.10.0.1
root@pc2:/# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.10.0.0       0.0.0.0         255.255.255.0   U     0      0        0 eth0
root@pc2:/#
```

Figure 2.20: Default gateway for PC2

reemamiranda — root@web1: / — kathara ◂ kathara connect -l web1 — 80×24

```
root@web1:/# image[web1]="alpine"
root@web1:/# web1[0]="D"
root@web1:/# ifconfig eth0 40.40.0.100 netmask 255.255.255.0 broadcast 40.40.0.2
55 up
root@web1:/# route add default gw 40.40.0.2
root@web1:/# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         40.40.0.2       0.0.0.0         UG    0      0        0 eth0
40.40.0.0       0.0.0.0         255.255.255.0   U     0      0        0 eth0
root@web1:/#
```

Figure 2.21: Default gateway for Web1

Figure 2.22: Default gateway for Web2

2.4 Capturing the traffic on the collision domains

In mac OS we cannot find collision domains in the traffic using Wireshark. To capture the traffic, used ping command to connect to between the systems, and ran tcpdump in the background, and redirected the dump to a .pcap file. Later by importing the .pcap into Wireshark we could able to observe the traffic.
Below screenshots displays the traffic captured from different systems at collision domains,

2.4.1 Traffic from pc1 to web1

Traffic is captured when we ping web1 from pc1 using tcpdump and the .pcap file is imported into the wireshark to display the traffic flow.

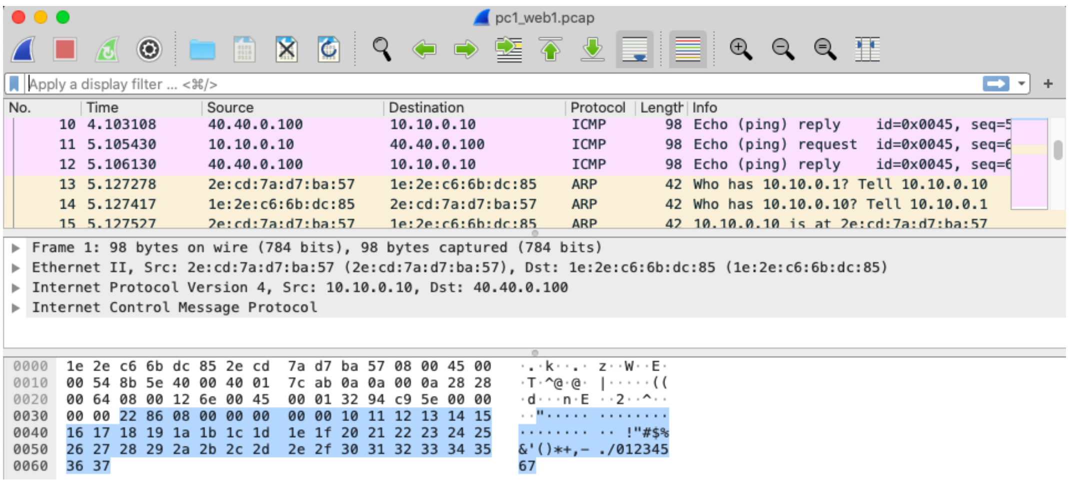


Figure 2.23: Traffic between pc1 and web1 in wireshark

2.4.2 Traffic from web1 to pc1

Traffic is captured when we ping pc1 from web1 using tcpdump and the .pcap file is imported into the wireshark to display the traffic flow.

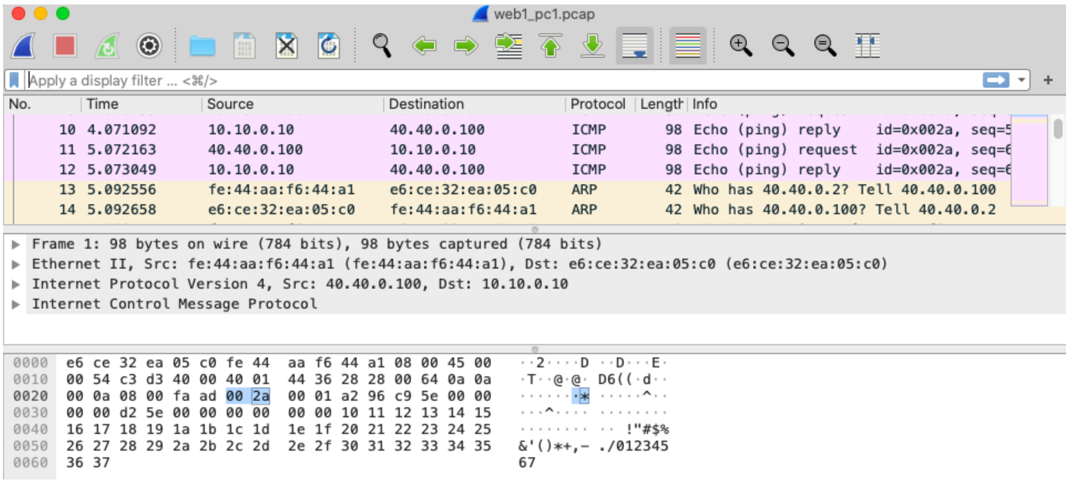


Figure 2.24: Traffic between web1 and pc1 in wireshark

2.4.3 Traffic from web2 to pc2

Traffic is captured when we ping pc2 from web2 using tcpdump and the .pcap file is imported into the wireshark to display the traffic flow.

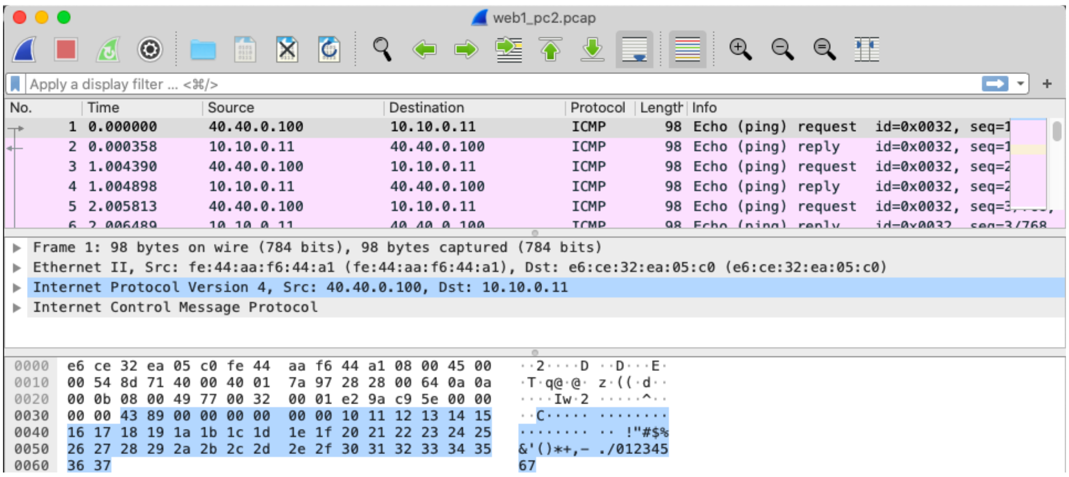


Figure 2.25: Traffic between web2 and pc2 in wireshark

2.4.4 Traffic from pc2 to web2

Traffic is captured when we ping web2 from pc2 using tcpdump and the .pcap file is imported into the wireshark to display the traffic flow.

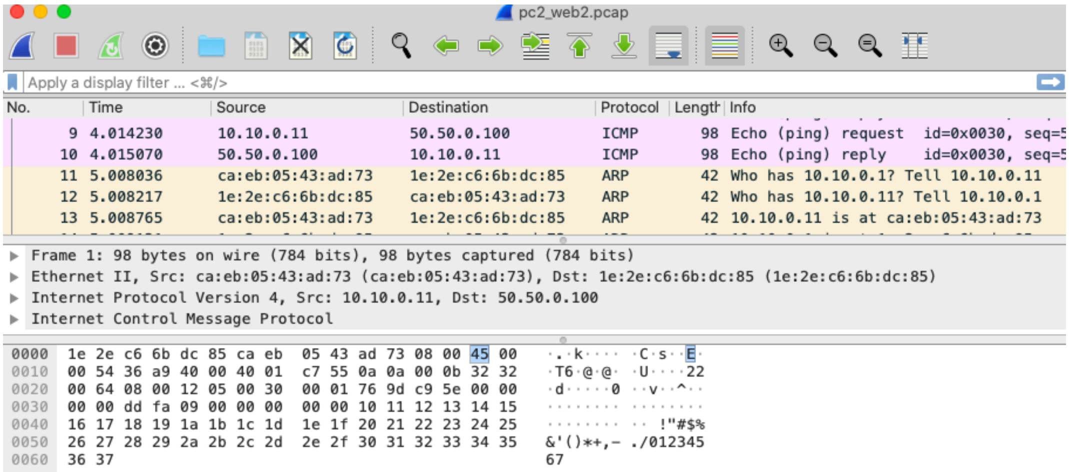


Figure 2.26: Traffic between pc2 and web2 in wireshark

2.5 Checking connectivity between all host

1.In this section we are going to check the connectivity between all the hosts.
2. we cannot reach all the host's due to lack of global connectivity so to provide global connectivity we need to add route entries to every router so that all the networks can communicate with each other.

2.5.1 Connectivity between Pc1 and Pc2 and from Pc1 to r1-eth0,r1-eth1,r1-eth2

we are able to reach from pc1,pc2 and r1 and vice versa because pc1,pc2 and R1 are in the same collision domain.

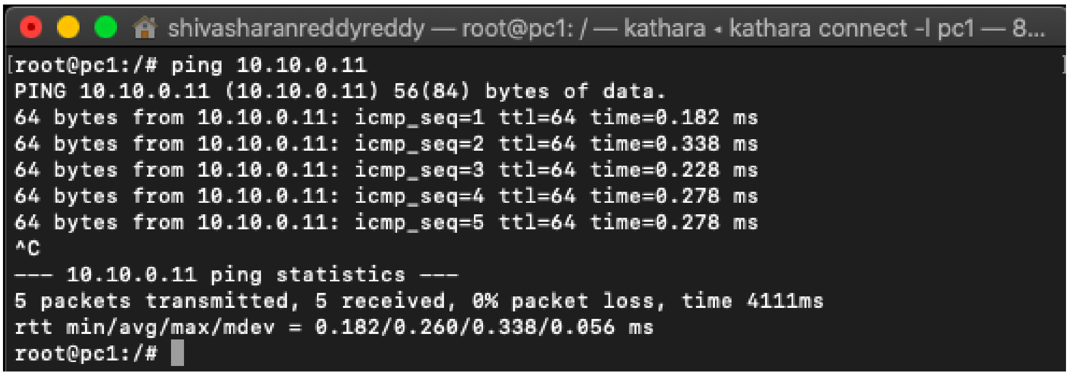


Figure 2.27: connectivity between pc1 and pc2

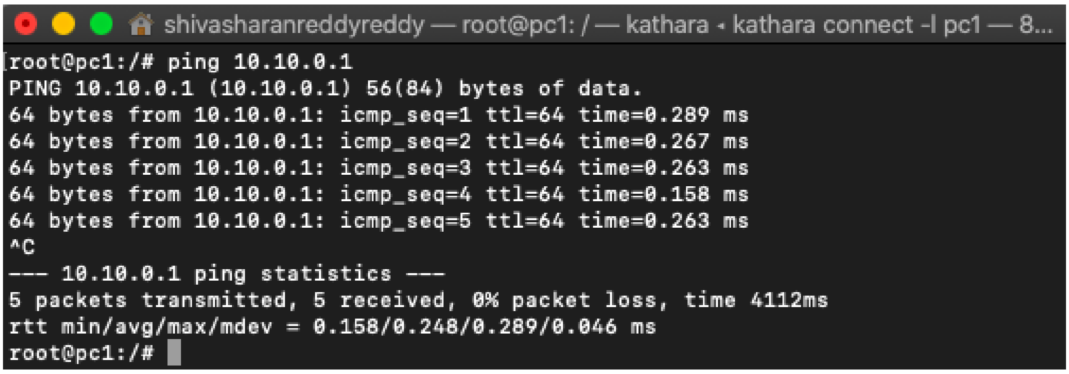


Figure 2.28: connectivity between pc1 and r1-eth0

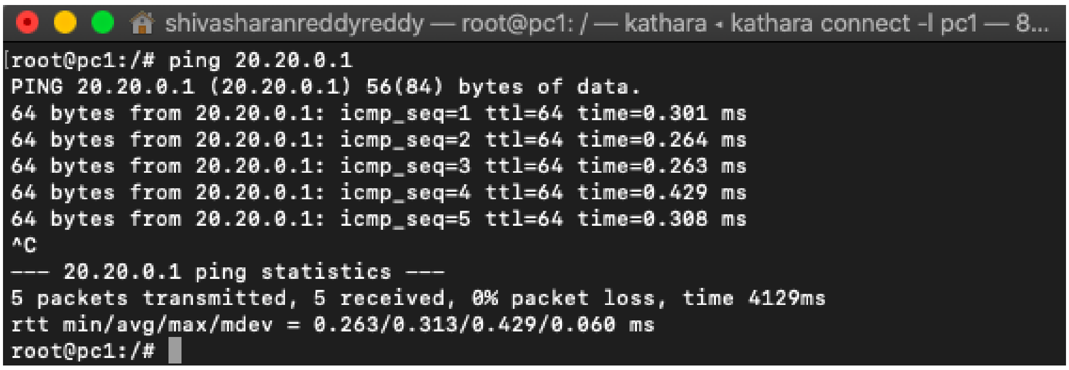


Figure 2.29: connectivity between pc1 and r1-eth1

```
shivasharanreddyreddy — root@pc1: / — kathara ◂ kathara connect -l pc1 — 8...
[root@pc1:/# ping 20.20.0.1
PING 20.20.0.1 (20.20.0.1) 56(84) bytes of data.
64 bytes from 20.20.0.1: icmp_seq=1 ttl=64 time=0.301 ms
64 bytes from 20.20.0.1: icmp_seq=2 ttl=64 time=0.264 ms
64 bytes from 20.20.0.1: icmp_seq=3 ttl=64 time=0.263 ms
64 bytes from 20.20.0.1: icmp_seq=4 ttl=64 time=0.429 ms
64 bytes from 20.20.0.1: icmp_seq=5 ttl=64 time=0.308 ms
^C
--- 20.20.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4129ms
rtt min/avg/max/mdev = 0.263/0.313/0.429/0.060 ms
root@pc1:/#
```

Figure 2.30: connectivity between pc1 and r1-eth2

[H]

2.5.2 Connectivity between Pc1 and r2 and r3

we cannot reach R2 or R3 from pc1,pc2,r1 because they are in the different network and for reaching those network we need to implement global connectivity that is we need to add routing entries in the routers
We need to add a route entry on R1,R2,R3 in order to reach between these host

```
[root@pc1:/# ping 30.30.0.2
PING 30.30.0.2 (30.30.0.2) 56(84) bytes of data.
^C
--- 30.30.0.2 ping statistics ---
12 packets transmitted, 0 received, 100% packet loss, time 11247ms
```

Figure 2.31: R2 is not reachable from PC1

[H]

```
[root@pc1:/# ping 20.20.0.3
PING 20.20.0.3 (20.20.0.3) 56(84) bytes of data.
^C
--- 20.20.0.3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2048ms
```

Figure 2.32: R3 is not reachable form pc1

[H]

2.5.3 Connectivity between web1 and r2, web2 and r3

We will be able to reach these host's because they are in the same collision domain and we have assigned their respective default gateway's.

```
root@web1:/# ping 30.30.0.2
PING 30.30.0.2 (30.30.0.2) 56(84) bytes of data.
64 bytes from 30.30.0.2: icmp_seq=1 ttl=64 time=0.193 ms
64 bytes from 30.30.0.2: icmp_seq=2 ttl=64 time=0.203 ms
64 bytes from 30.30.0.2: icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from 30.30.0.2: icmp_seq=4 ttl=64 time=0.204 ms
64 bytes from 30.30.0.2: icmp_seq=5 ttl=64 time=0.207 ms
^C
--- 30.30.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4098ms
rtt min/avg/max/mdev = 0.193/0.202/0.207/0.004 ms
```

Figure 2.33: connectivity between web1 and r2

[H]

```
root@web2:/# ping 20.20.0.3
PING 20.20.0.3 (20.20.0.3) 56(84) bytes of data.
64 bytes from 20.20.0.3: icmp_seq=1 ttl=64 time=0.293 ms
64 bytes from 20.20.0.3: icmp_seq=2 ttl=64 time=0.203 ms
64 bytes from 20.20.0.3: icmp_seq=3 ttl=64 time=0.211 ms
64 bytes from 20.20.0.3: icmp_seq=4 ttl=64 time=0.432 ms
64 bytes from 20.20.0.3: icmp_seq=5 ttl=64 time=0.206 ms
^C
--- 20.20.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4130ms
rtt min/avg/max/mdev = 0.203/0.269/0.432/0.088 ms
```

Figure 2.34: connectivity between web2 and r3

[H]

2.6 Providing global connectivity

In this section, to provide global connectivity we are going to manually add routing entries into the routing tables of r1, r2, and r3.

2.6.1 Entries into the routing table of r2

This is achieved using the following commands.
1) route add -net 10.10.0.0 net mask 255.255.255.0 gw 30.30.0.1 dev eth0
2) route add -net 50.50.0.0 net mask 255.255.255.128 gw 30.30.0.1 eth0

```
ManjunathM — root@r2: / — kathara ◂ kathara connect -l r2 — 106×24
root@r2:/# route add -net 10.10.0.0 netmask 255.255.255.0 gw 30.30.0.1 dev eth0
root@r2:/# route add -net 50.50.0.0 netmask 255.255.255.128 gw 30.30.0.1 dev eth0
root@r2:/# ip route show
10.10.0.0/24 via 30.30.0.1 dev eth0
30.30.0.0/16 dev eth0 proto kernel scope link src 30.30.0.2
40.40.0.0/24 dev eth1 proto kernel scope link src 40.40.0.2
50.50.0.0/25 via 30.30.0.1 dev eth0
root@r2:/# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.10.0.0       30.30.0.1       255.255.255.0   UG    0      0        0 eth0
30.30.0.0       0.0.0.0         255.255.0.0     U     0      0        0 eth0
40.40.0.0       0.0.0.0         255.255.255.0   U     0      0        0 eth1
50.50.0.0       30.30.0.1       255.255.255.128 UG    0      0        0 eth0
root@r2:/#
```

Figure 2.35: Routing table entries for r2

2.6.2 Entries into the routing table of r3

This is achieved using the following commands.
1) route add -net 10.10.0.0 netmask 255.255.255.0 gw 20.20.0.1 dev eth0
2) route add -net 40.40.0.0 netmask 255.255.255.0 gw 20.20.0.1 eth0

```
ManjunathM — root@r3: / — kathara ◂ kathara connect -l r3 — 80×24
root@r3:/# route add -net 10.10.0.0 netmask 255.255.255.0 gw 20.20.0.1 eth0
root@r3:/# route add -net 40.40.0.0 netmask 255.255.255.0 gw 20.20.0.1 eth0
root@r3:/# ip route show
10.10.0.0/24 via 20.20.0.1 dev eth0
20.20.0.0/20 dev eth0 proto kernel scope link src 20.20.0.3
40.40.0.0/24 via 20.20.0.1 dev eth0
50.50.0.0/25 dev eth1 proto kernel scope link src 50.50.0.3
root@r3:/# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.10.0.0       20.20.0.1       255.255.255.0   UG    0      0        0 eth0
20.20.0.0       0.0.0.0         255.255.240.0   U     0      0        0 eth0
40.40.0.0       20.20.0.1       255.255.255.0   UG    0      0        0 eth0
50.50.0.0       0.0.0.0         255.255.255.128 U     0      0        0 eth1
root@r3:/#
```

Figure 2.36: Routing table entries for r3

2.6.3 Entries into the routing table of r1

This is achieved using the following commands.
1) route add -net 40.40.0.0 netmask 255.255.0.0 gw 30.30.0.2 eth1
2) route add -net 50.50.0.0 netmask 255.255.240.0 gw 20.20.0.3 eth2

```
ManjunathM — root@r1: / — kathara ◂ kathara connect -l r1 — 80×24
root@r1:/# route add -net 50.50.0.0 netmask 255.255.255.128 gw 20.20.0.3 eth2
root@r1:/# route add -net 40.40.0.0 netmask 255.255.255.0 gw 30.30.0.2 eth1
root@r1:/# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.10.0.0       0.0.0.0         255.255.255.0   U     0      0        0 eth0
20.20.0.0       0.0.0.0         255.255.240.0   U     0      0        0 eth2
30.30.0.0       0.0.0.0         255.255.0.0     U     0      0        0 eth1
40.40.0.0       30.30.0.2       255.255.255.0   UG    0      0        0 eth1
50.50.0.0       20.20.0.3       255.255.255.128 UG    0      0        0 eth2
root@r1:/# ip route show
10.10.0.0/24 dev eth0 proto kernel scope link src 10.10.0.1
20.20.0.0/20 dev eth2 proto kernel scope link src 20.20.0.1
30.30.0.0/16 dev eth1 proto kernel scope link src 30.30.0.1
40.40.0.0/24 via 30.30.0.2 dev eth1
50.50.0.0/25 via 20.20.0.3 dev eth2
root@r1:/#
```

Figure 2.37: Routing table entries for r1

2.7 Ping from pc1 to web1

After we successfully provide global connectivity we are able to reach between all the hosts in the given network
When we ping from pc1 to web1, the packets start from pc1 and reaches r1. We have configured a route in r1 so that the R1 routes forwards packets to r2, once the packet reaches r2 it sends it to web1 because in web1 the default gateway is configures a R2 hence we will be able to ping from pc1 to web1, As shown below.

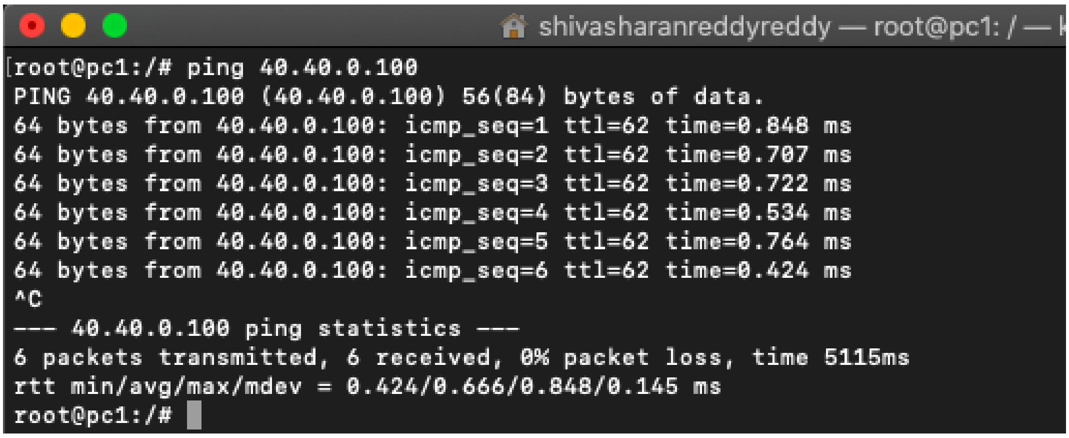


Figure 2.38: connectivity between pc1 and Web1

Chapter 3

DHCP - Dynamic Host Control Protocol

In this section we will setup an Ip address for Pc1 and Pc2 by sending Dhcp request Via r1

3.1 Configure Dhcp by setting the default router and setting the range of IPs

First we will need run the commands in Vyos mode by running the following in r1 terminal **/bin/su -s /bin/vbash - vyos**

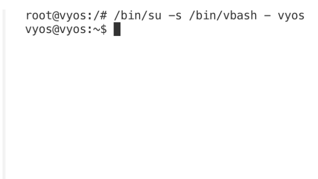


Figure 3.1: Set Terminal to be in Vyos mode

3.2 Dhcp Configuration in r1 and explaining how DHCP is working.

In order to configure Dhcp and setup ip range and everything we will need first to run **Configure** then setup everything for dhcp and after that we run Commit. After running Configure,

- We will enable dhcp-server authoritative for the machine we are running which is r1 in that case

- We will setup the default router to be r1

- We will setup the IPs range that can be provided by dhcp server to be from 10.10.0.100/24 to 10.10.0.200/24

After setting dhcp we will run commit and that‘s it

```
root@vyos:/# /bin/su -s /bin/vbash - vyos
vyos@vyos:~$ configure
[edit]
vyos@vyos# set service dhcp-server shared-network-name LAN  authoritative enable
[edit]
vyos@vyos# set service dhcp-server shared-network-name LAN subnet 10.10.0.0/24 default-router 10.10.0.1
[edit]
vyos@vyos# set service dhcp-server shared-network-name LAN subnet 10.10.0.0/24 start 10.10.0.100  stop 10.10.0.200
[edit]
vyos@vyos# commit
[edit]
vyos@vyos# 
```

Figure 3.2: Dhcp configuration in r1

3.3 Pc1 and Pc2 request ip address from Dhcp server

Lets see the steps first on Pc1 and same will be for Pc2

- pc1 will broadcast that it needs an ip address to all the devices in the same network (Dhcp-Discover).

- the default-router (r1) will send request to Dhcp-server. Dhcp gets the message from pc1 it will send Dhcp-offer with ip address **Note the offered ip will be within the range we provided for dhcp-server**

- if Pc1 accepted the offer it will send Dhcp-Request

- Dhcp will get the Request with acceptance and send back Dhcp-Pack which is the acknowledgement along with the ip address, the sub-net mask and the default gateway and the dns-server. Dhcp will keep the record that Pc1 took this ip address

```
root@pc1:/# dhclient
root@pc1:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.10.0.100  netmask 255.255.255.0  broadcast 10.10.0.255
        ether de:c6:cf:82:1d:f1  txqueuelen 1000  (Ethernet)
        RX packets 32  bytes 2828 (2.7 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5  bytes 830 (830.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 3.3: Pc1 request for ip

```
root@pc2:/# dhclient
root@pc2:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.10.0.101  netmask 255.255.255.0  broadcast 10.10.0.255
        ether 06:62:88:db:09:76  txqueuelen 1000  (Ethernet)
        RX packets 2590  bytes 246252 (240.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2523  bytes 240282 (234.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@pc2:/# 
```

Figure 3.4: Pc2 Request for ip

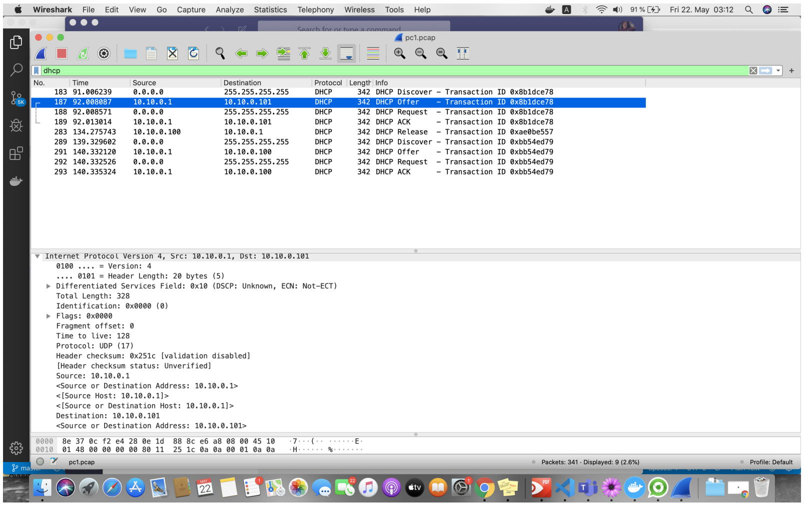


Figure 3.5: Pc1 request for ip address procedure as explained represented on wireshark

Now lets make sure of the connectivity and that r1 will be connected automatically to pc1 and pc2 by sending ping to Web2 and capture the ICMP request by wire shark

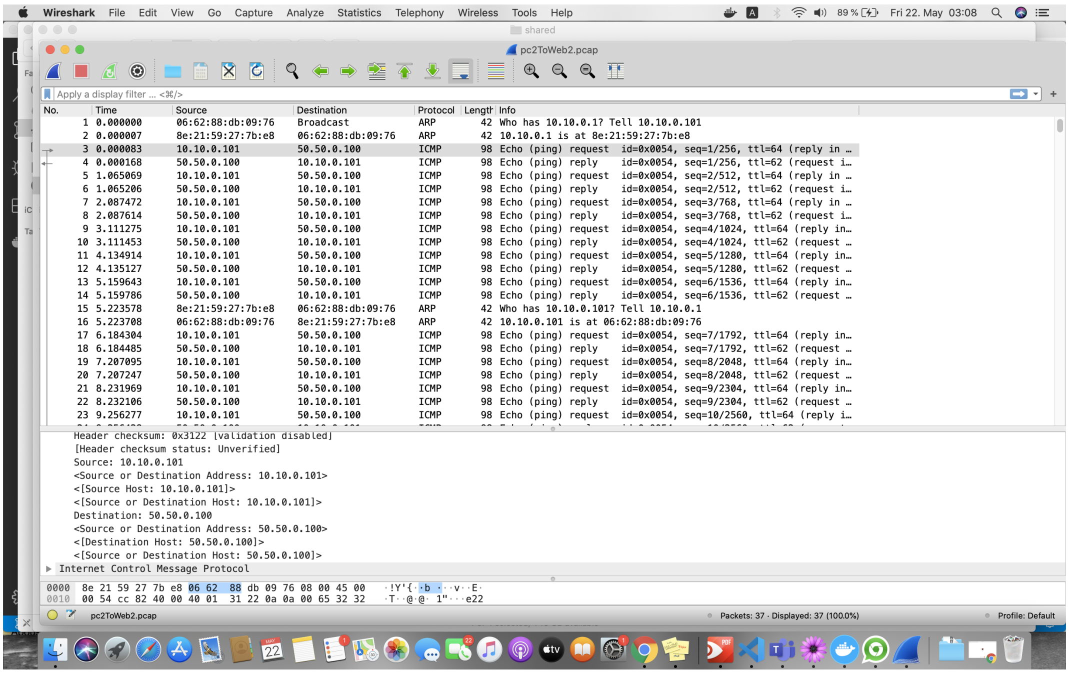


Figure 3.6: Capturing ICMP on wireshark after pc2 send ping to web2, to ensure connectivity