

Otto-Friedrich-University Bamberg
Professorship for Computer Science,
Communication Services, Telecommunication,
Systems and Computer Networks



Foundation of Internet Communication

Assignement1

Submitted by:
Group J

Reem Khalil
Shivasharan Reddy
Azar
Reema Miranda
Manjunath B Marigoudar

Supervisor: Prof. Dr. Udo Krieger

Bamberg, May 10, 2020
Summer Term 2020

Chapter 1

Wireshark Handling

Explain what we are going to do in this section here

1.1 Provide a short documentation that explained its actions

Wireshark is a network protocol analyzer. Using wireshark, we can monitor what's happening on the network.

Who use Wireshark?

- Network administrators use it to troubleshoot network problems
- Network security engineers use it to examine security problems
- QA engineers use it to verify network applications
- Developers use it to debug protocol implementations
- People use it to learn network protocol internals <https://www.overleaf.com/project/5eac>

Features

- Available for UNIX and Windows.
- it is an open source software project, and is released under the GNU General Public License (GPL)
- Capture live packet data from a network interface including Ethernet, Wireless LAN,...

- Open files containing packet data captured with tcpdump/WinDump, Wireshark, and many other packet capture programs.
- Import packets from text files containing hex dumps of packet data.
- Display packets with very detailed protocol information.
- Save packet data captured.
- Export some or all packets in a number of capture file formats.
- Filter packets on many criteria.
- Search for packets on many criteria.
- Colorize packet display based on filters.
- Create various statistics.

Wireshark is not

- it isn't an intrusion detection system. It will not warn you when someone does strange things on your network . However, if it happens, Wireshark can help you figure out what is really going on.
- Wireshark will not manipulate things on the network.. Wireshark doesn't send packets on the network.

Supported platforms

Wireshark runs on most UNIX and UNIX-like platforms including Linux and most BSD variants. It also runs on various Windows platforms

Chapter 2

Wireshark Packet Filtering

2.1 Ping tool

- Ping tool is utility used for troubleshooting, testing and diagnosing issues related to network connectivity
- When we use Wireshark for capturing live packets in the background we parallelly use terminal and ping www.google.com
- We let Wireshark to capture the packets for few seconds

```
PING www.google.com (172.217.163.132): 56 data bytes
64 bytes from 172.217.163.132: icmp_seq=0 ttl=52 time=73.983 ms
64 bytes from 172.217.163.132: icmp_seq=1 ttl=52 time=85.190 ms
64 bytes from 172.217.163.132: icmp_seq=2 ttl=52 time=87.461 ms
64 bytes from 172.217.163.132: icmp_seq=3 ttl=52 time=58.847 ms
64 bytes from 172.217.163.132: icmp_seq=4 ttl=52 time=49.609 ms
^C
--- www.google.com ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 49.609/71.018/87.461/14.741 ms
```

Figure 2.1: Pinging www.google.com

- Using wireshark packet filter `ip.addr == host/destination address`.
eg:- `ip.addr == 172.217.163.132`

ip.addr == 172.217.163.100							
No.	Time	Source	Destination	Protocol	Length	Info	
5118	43.474709	172.20.10.2	172.217.163.100	ICMP	98	Echo (ping) request	id=0xa304, seq=0/0, ttl=64 (reply in 51...
5126	43.583242	172.217.163.100	172.20.10.2	ICMP	98	Echo (ping) reply	id=0xa304, seq=0/0, ttl=52 (request in ...
5189	44.479044	172.20.10.2	172.217.163.100	ICMP	98	Echo (ping) request	id=0xa304, seq=1/256, ttl=64 (reply in ...
5196	44.559059	172.217.163.100	172.20.10.2	ICMP	98	Echo (ping) reply	id=0xa304, seq=1/256, ttl=52 (request in ...
5243	45.481205	172.20.10.2	172.217.163.100	ICMP	98	Echo (ping) request	id=0xa304, seq=2/512, ttl=64 (reply in ...
5247	45.559178	172.217.163.100	172.20.10.2	ICMP	98	Echo (ping) reply	id=0xa304, seq=2/512, ttl=52 (request in ...
5399	46.486006	172.20.10.2	172.217.163.100	ICMP	98	Echo (ping) request	id=0xa304, seq=3/768, ttl=64 (reply in ...
▶ Frame 5118: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface en0, id 0 ▶ Ethernet II, Src: Apple_9a:6a:10 (98:46:0a:9a:6a:10), Dst: 46:00:10:16:40:64 (46:00:10:16:40:64) ▶ Internet Protocol Version 4, Src: 172.20.10.2, Dst: 172.217.163.100 ▶ Internet Control Message Protocol							
0000	46 00 10 16 40 64 98 46	0a 9a 6a 10 08 00 45 00	F...@.F...j...E.				
0010	00 54 b6 81 00 00 40 01	bd d3 ac 14 0a 02 ac d9	.T...@.				
0020	a3 64 00 00 6b 18 a3 04	00 00 5c b0 0f 9f 00 01	.d...k...:A.....				
0030	90 8f 00 00 0a 0b 0c 0d	0e 0f 10 11 12 13 14 15				
0040	15 17 18 19 1a 1b 1c 1d	1e 1f 20 21 22 23 24 25 !"#%&				
0050	26 27 28 29 2a 2b 2c 2d	2e 2f 30 31 32 33 34 35	&'()*+,-./012345				
0060	36 37		67				

Figure 2.2: Display filter

2.2 Packets sent by the host IP Address

- The Packets sent by the host when we visit the URL <http://www.caida.org/tools/visualization/mapnet> can be displayed with the help of the display filter `ip.addr == destination address`.
eg:- `ip.addr == 192.168.43.251`.

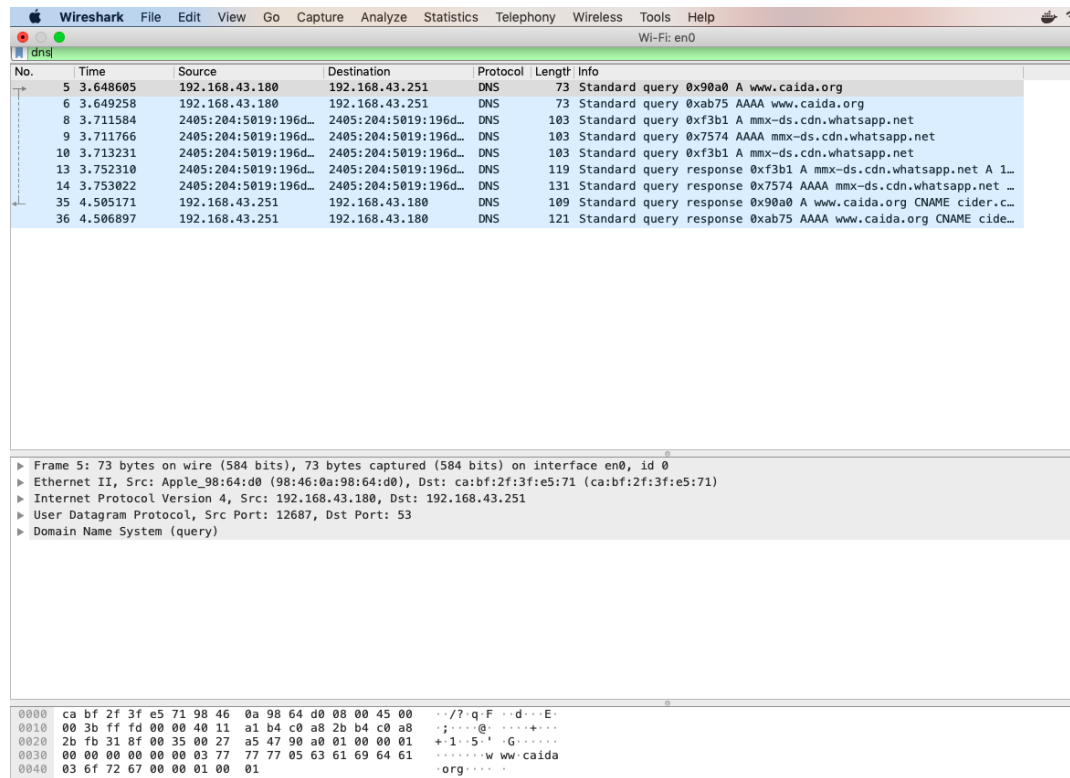


Figure 2.3: Destination ip Address with DNS filter

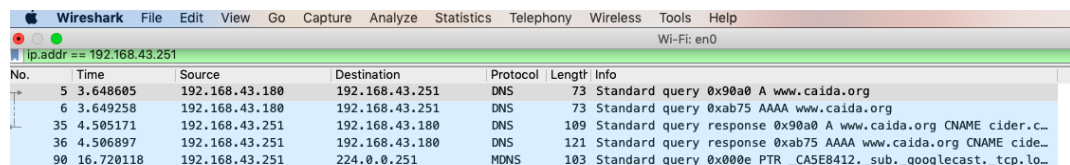


Figure 2.4: IP address display filter

2.3 Packets sent by the host MAC Address

- The Mac address of my system is 98:46:0a:98:64:d0.
- The packets sent by the host MAC address, if we visit the URL <http://www.caid.a.org/tools/visualization/mapnet> can be displayed with the filter `eth.addr==mac address`
eg:- `eth.addr== 98:46:0a:98:64:d0`

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.43.180	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1
2	1.000813	192.168.43.180	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1
3	2.001128	192.168.43.180	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1
4	3.001251	192.168.43.180	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1
5	4.681125	192.168.43.180	17.134.127.250	TCP	78	49365 → 443 [SYN, ECN, CWR] Seq=0 Win=65535 Len=0 MSS=1460 W...
6	5.041818	17.134.127.250	192.168.43.180	TCP	74	443 → 49365 [SYN, ACK, ECN] Seq=0 Ack=1 Win=43440 Len=0 MSS=...
7	5.041951	192.168.43.180	17.134.127.250	TCP	66	49365 → 443 [ACK] Seq=1 Ack=1 Win=131712 Len=0 TSval=6470375...
8	5.044981	192.168.43.180	17.134.127.250	TLSv1	583	Client Hello
9	5.210257	17.134.127.250	192.168.43.180	TCP	66	443 → 49365 [ACK] Seq=1 Ack=518 Win=45056 Len=0 TSval=337701...
10	5.867053	17.134.127.250	192.168.43.180	TLSv1	1396	Server Hello
11	5.867060	17.134.127.250	192.168.43.180	TCP	1424	443 → 49365 [ACK] Seq=1331 Ack=518 Win=45056 Len=1358 TSval=...
12	5.867150	192.168.43.180	17.134.127.250	TCP	66	49365 → 443 [ACK] Seq=518 Ack=1331 Win=130368 Len=0 TSval=64...
13	5.872963	17.134.127.250	192.168.43.180	TLSv1	305	Certificate, Server Key Exchange, Server Hello Done
14	5.873057	192.168.43.180	17.134.127.250	TCP	66	49365 → 443 [ACK] Seq=518 Ack=2928 Win=130816 Len=0 TSval=64...
15	5.878791	17.134.127.250	192.168.43.180	TCP	305	[TCP Spurious Retransmission] 443 → 49365 [PSH, ACK] Seq=268...
16	5.878873	192.168.43.180	17.134.127.250	TCP	78	[TCP Window Update] 49365 → 443 [ACK] Seq=518 Ack=2928 Win=1...
17	5.885500	192.168.43.180	17.134.127.250	TLSv1	159	Client Key Exchange, Change Cipher Spec, Encrypted Handshake...
18	6.176722	17.134.127.250	192.168.43.180	TCP	66	443 → 49365 [ACK] Seq=2928 Ack=611 Win=45056 Len=0 TSval=337...
19	6.346651	17.134.127.250	192.168.43.180	TLSv1	117	Change Cipher Spec, Encrypted Handshake Message
20	6.353008	192.168.43.180	17.134.127.250	TCP	66	49365 → 443 [ACK] Seq=611 Ack=2979 Win=131008 Len=0 TSval=64...
21	6.353009	192.168.43.180	17.134.127.250	TLSv1	384	Application Data
22	6.353010	192.168.43.180	17.134.127.250	TLSv1	198	Application Data
23	6.639609	17.134.127.250	192.168.43.180	TCP	66	443 → 49365 [ACK] Seq=2979 Ack=929 Win=49152 Len=0 TSval=337...
24	6.639614	17.134.127.250	192.168.43.180	TCP	66	443 → 49365 [ACK] Seq=2979 Ack=1061 Win=49152 Len=0 TSval=33...
25	7.146570	17.134.127.250	192.168.43.180	TLSv1	306	Application Data
26	7.146780	192.168.43.180	17.134.127.250	TCP	66	49365 → 443 [ACK] Seq=1061 Ack=3210 Win=130816 Len=0 TSval=6...

▶ Frame 1: 217 bytes on wire (1736 bits), 217 bytes captured (1736 bits) on interface en0, id 0
 ▼ Ethernet II, Src: Apple_98:46:d0 (98:46:0a:98:64:d0), Dst: IPv4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)
 ▶ Destination: IPv4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)
 ▶ Source: Apple_98:46:d0 (98:46:0a:98:64:d0)
 Type: IPv4 (0x0800)
 ▶ Internet Protocol Version 4, Src: 192.168.43.180, Dst: 239.255.255.250
 ▶ User Datagram Protocol, Src Port: 55646, Dst Port: 1900
 ▼ Simple Service Discovery Protocol
 ▶ M-SEARCH * HTTP/1.1\r\n
 HOST: 239.255.255.250:1900\r\n
 MAN: "ssdp:discover"\r\n
 MX: 1\r\n
 ST: urn:dial-multiscreen-org:service:dial:1\r\n
 USER-AGENT: Google Chrome/81.0.4044.129 Mac OS X\r\n
 \r\n
 [Full request URI: http://239.255.255.250:1900*]
 [HTTP request 1/4]

Figure 2.5: Mac address display filter

2.4 tcpdump filter

- tcpdump expression that captures packets containing IP datagrams with a source or destination IP address equal to your IP address.

1) host 192.168.43.180.

2.5 TCP Packet Capture

- Tcpdump expression that captures TCP packets using port number 22.

1) `tcp dst port 22`

2) `tcp src port 22`

2.6 Display Packets with defined frame size

- Syntax for a display filter that shows only IP datagrams with a destination IP address equal to 192.168.178.1 and frame size greater than 350 bytes

1) `ip.dst == 192.168.178.1 and frame.len > 350`

Chapter 3

Basic Linux network administration

In this section we are going to set up a small emulated LAN of the topology in figure 3.1 using basic Linux network functionality and Kathará.

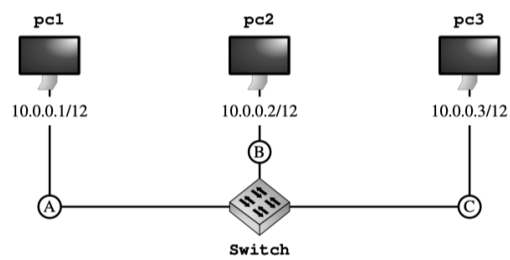


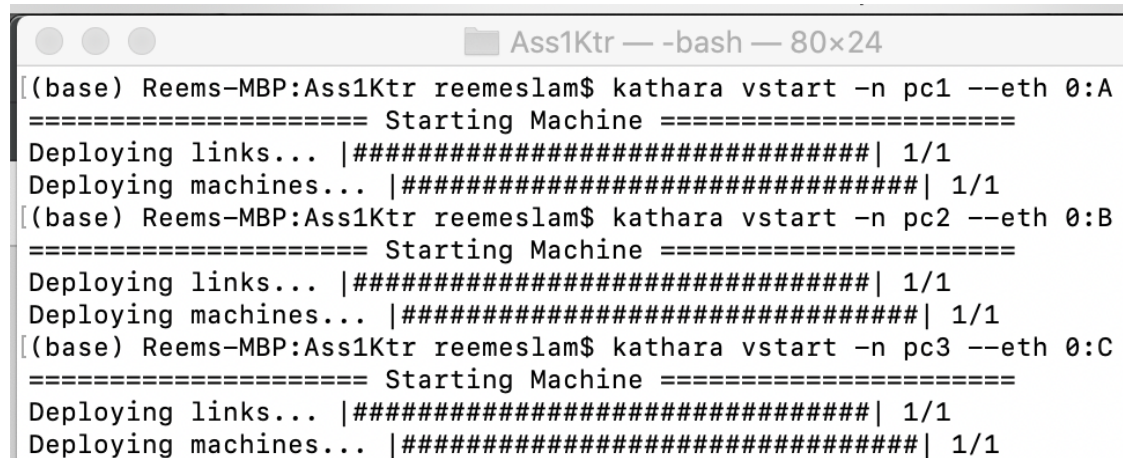
Figure 1: LAN setup

Linux PC	IP Address	Interface	CD
pc1	10.0.0.1/12	eth0	A
pc2	10.0.0.2/12	eth0	B
pc3	10.0.0.3/12	eth0	C
switch		eth0	A
		eth1	B
		eth2	C

Figure 3.1: LAN switching configuration

3.1 Creating the four devices

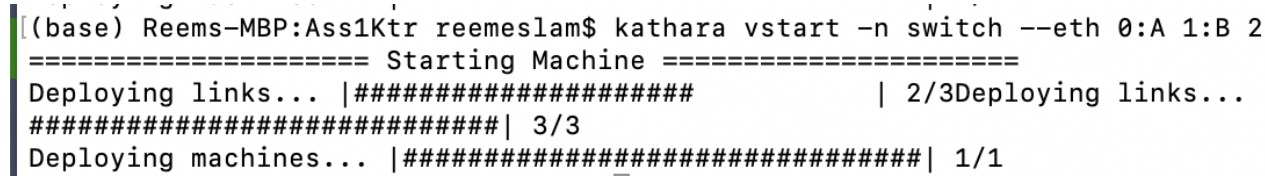
3.1.1 Creating Three PCs each assigned a network interface and unique collision domain



```
Ass1Ktr — -bash — 80x24
[(base) Reems-MBP:Ass1Ktr reemeslam$ kathara vstart -n pc1 --eth 0:A
===== Starting Machine =====
Deploying links... |#####| 1/1
Deploying machines... |#####| 1/1
[(base) Reems-MBP:Ass1Ktr reemeslam$ kathara vstart -n pc2 --eth 0:B
===== Starting Machine =====
Deploying links... |#####| 1/1
Deploying machines... |#####| 1/1
[(base) Reems-MBP:Ass1Ktr reemeslam$ kathara vstart -n pc3 --eth 0:C
===== Starting Machine =====
Deploying links... |#####| 1/1
Deploying machines... |#####| 1/1
```

Figure 3.2: Created three PCs each assigned a network interface with unique collision domain

3.1.2 Creating a switch that has three network interfaces attached, where each interface is connected to a different collision domain



```
[(base) Reems-MBP:Ass1Ktr reemeslam$ kathara vstart -n switch --eth 0:A 1:B 2:C
===== Starting Machine =====
Deploying links... |#####| 2/3Deploying links...
#####| 3/3
Deploying machines... |#####| 1/1
```

Figure 3.3: Created a Switch with three network interfaces, where each interface is connected to a different collision domain

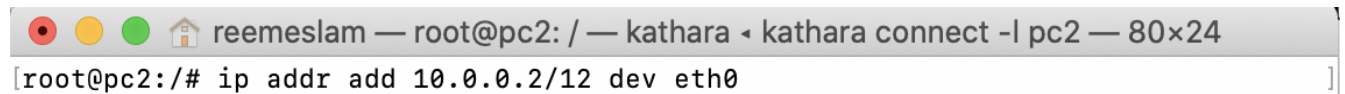
3.2 Configuring the four devices

3.2.1 Configuring an IP address for PC1, PC2 and PC2

A terminal window titled 'reemeslam — root@pc1: / — kathara • kathara connect -l pc1 — 80x24'. The command prompt is '[root@pc1:/# ip addr add 10.0.0.1/12 dev eth0]'. The IP address '10.0.0.1' is highlighted in blue.

```
reemeslam — root@pc1: / — kathara • kathara connect -l pc1 — 80x24
[root@pc1:/# ip addr add 10.0.0.1/12 dev eth0
```

Figure 3.4: configured IP address for PC1

A terminal window titled 'reemeslam — root@pc2: / — kathara • kathara connect -l pc2 — 80x24'. The command prompt is '[root@pc2:/# ip addr add 10.0.0.2/12 dev eth0]'.

```
reemeslam — root@pc2: / — kathara • kathara connect -l pc2 — 80x24
[root@pc2:/# ip addr add 10.0.0.2/12 dev eth0
```

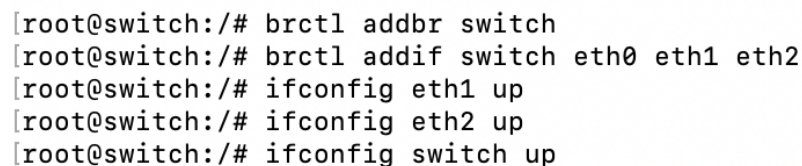
Figure 3.5: configured IP address for PC2

A terminal window titled 'reemeslam — root@pc3: / — kathara • kathara connect -l pc3 — 80x24'. The command prompt is '[root@pc3:/# ip addr add 10.0.0.3/12 dev eth0]'.

```
reemeslam — root@pc3: / — kathara • kathara connect -l pc3 — 80x24
[root@pc3:/# ip addr add 10.0.0.3/12 dev eth0
```

Figure 3.6: configured IP address for PC3

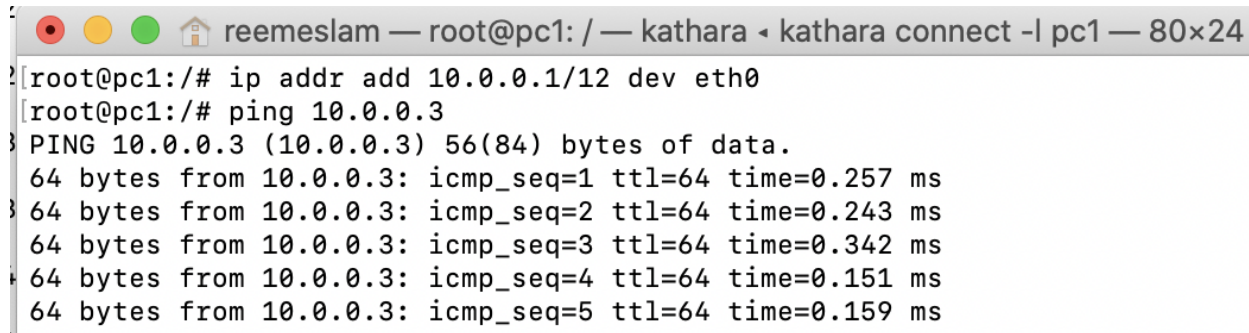
3.2.2 Configuring a bridge for the switch, set-up the environment and turning on the bridge

A terminal window showing the configuration of a bridge named 'switch'. The commands are: 'brctl addbr switch', 'brctl addif switch eth0 eth1 eth2', 'ifconfig eth1 up', 'ifconfig eth2 up', and 'ifconfig switch up'.

```
[root@switch:/# brctl addbr switch
[root@switch:/# brctl addif switch eth0 eth1 eth2
[root@switch:/# ifconfig eth1 up
[root@switch:/# ifconfig eth2 up
[root@switch:/# ifconfig switch up
```

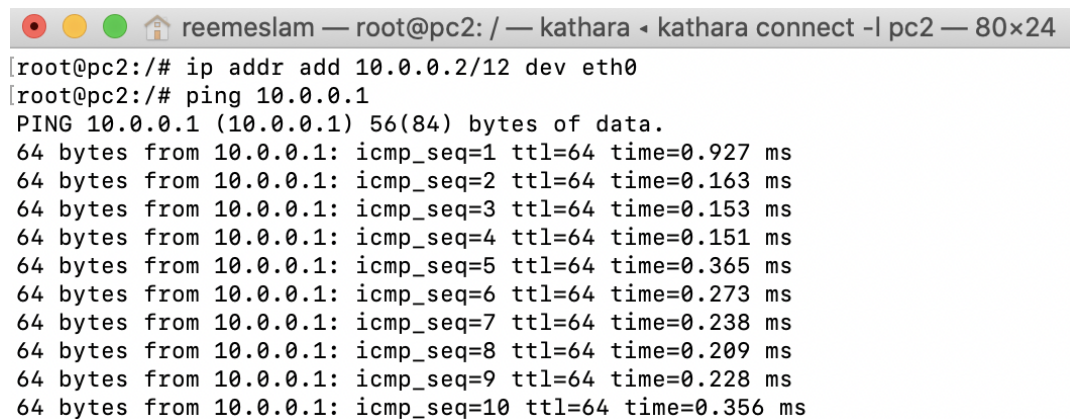
Figure 3.7: Created a bridge with name switch, Added the three interfaces to the switch, turned on eth1 & eth2 and Turned the bridge on

3.2.3 Testing the network by sending pings between PCs



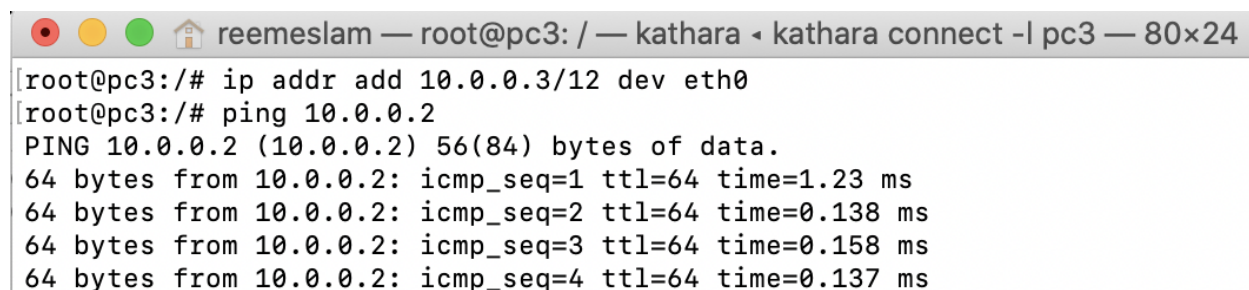
```
reemeslam — root@pc1: / — kathara ◀ kathara connect -l pc1 — 80x24
[root@pc1:/# ip addr add 10.0.0.1/12 dev eth0
[root@pc1:/# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
 64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.257 ms
 64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.243 ms
 64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.342 ms
 64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.151 ms
 64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.159 ms
...
```

Figure 3.8: Sending ping from PC-1 to PC-3 to check connectivity



```
reemeslam — root@pc2: / — kathara ◀ kathara connect -l pc2 — 80x24
[root@pc2:/# ip addr add 10.0.0.2/12 dev eth0
[root@pc2:/# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
 64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.927 ms
 64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.163 ms
 64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.153 ms
 64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.151 ms
 64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.365 ms
 64 bytes from 10.0.0.1: icmp_seq=6 ttl=64 time=0.273 ms
 64 bytes from 10.0.0.1: icmp_seq=7 ttl=64 time=0.238 ms
 64 bytes from 10.0.0.1: icmp_seq=8 ttl=64 time=0.209 ms
 64 bytes from 10.0.0.1: icmp_seq=9 ttl=64 time=0.228 ms
 64 bytes from 10.0.0.1: icmp_seq=10 ttl=64 time=0.356 ms
```

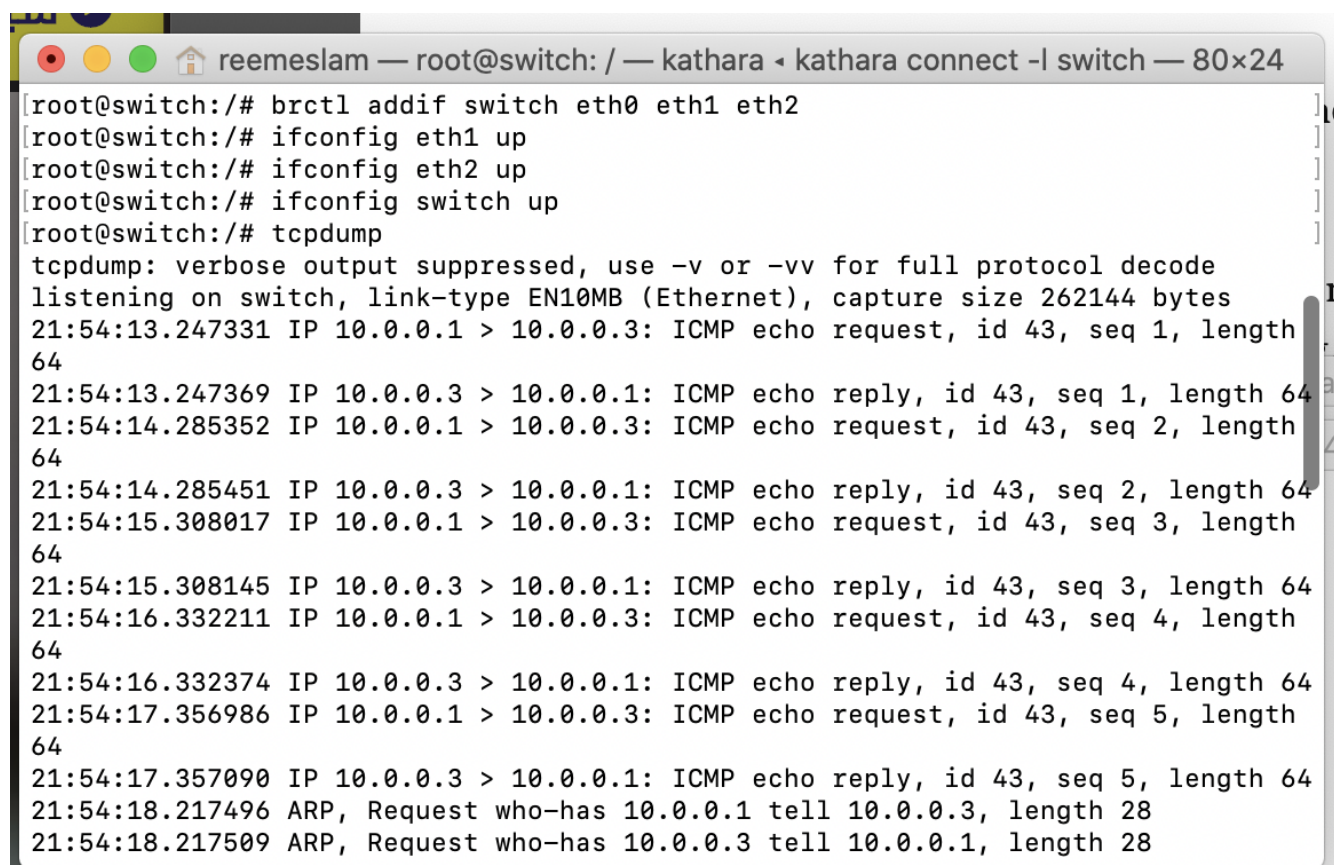
Figure 3.9: Sending ping from PC-2 to PC-1 to check connectivity



```
reemeslam — root@pc3: / — kathara ◀ kathara connect -l pc3 — 80x24
[root@pc3:/# ip addr add 10.0.0.3/12 dev eth0
[root@pc3:/# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
 64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.23 ms
 64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.138 ms
 64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.158 ms
 64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.137 ms
```

Figure 3.10: Sending ping from PC-3 to PC-2 to check connectivity

3.3 Checking connectivity by Capturing the ICMP messages exchanged by devices on sending ping from Pc-1 to PC-3.



```
reemeslam — root@switch: / — kathara • kathara connect -l switch — 80x24
[root@switch:/# brctl addif switch eth0 eth1 eth2
[root@switch:/# ifconfig eth1 up
[root@switch:/# ifconfig eth2 up
[root@switch:/# ifconfig switch up
[root@switch:/# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on switch, link-type EN10MB (Ethernet), capture size 262144 bytes
21:54:13.247331 IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 43, seq 1, length 64
21:54:13.247369 IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 43, seq 1, length 64
21:54:14.285352 IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 43, seq 2, length 64
21:54:14.285451 IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 43, seq 2, length 64
21:54:15.308017 IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 43, seq 3, length 64
21:54:15.308145 IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 43, seq 3, length 64
21:54:16.332211 IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 43, seq 4, length 64
21:54:16.332374 IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 43, seq 4, length 64
21:54:17.356986 IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 43, seq 5, length 64
21:54:17.357090 IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 43, seq 5, length 64
21:54:18.217496 ARP, Request who-has 10.0.0.1 tell 10.0.0.3, length 28
21:54:18.217509 ARP, Request who-has 10.0.0.3 tell 10.0.0.1, length 28
```

Figure 3.11: Capturing the ICMP messages between PC-1 and PC-3