Kathará

kathara lab

basic IPv4 configuration, ping, traceroute and arp

Version	1.0
Author(s)	L. Ariemma, T. Caiazzi, G. Di Battista, M. Patrignani, M. Pizzonia, F. Ricci, M. Rimondini
E-mail	contact@kathara.org
Web	http://www.kathara.org/
Description	basic IPv4 configuration commands, usage of ping and traceroute, arp behaviour

copyright notice

- All the pages/slides in this presentation, including but not limited to, images, photos, animations, videos, sounds, music, and text (hereby referred to as "material") are protected by copyright.
- This material, with the exception of some multimedia elements licensed by other organizations, is property of the authors and/or organizations appearing in the first slide.
- This material, or its parts, can be reproduced and used for didactical purposes within universities and schools, provided that this happens for non-profit purposes.
- Information contained in this material cannot be used within network design projects or other products of any kind.
- Any other use is prohibited, unless explicitly authorized by the authors on the basis of an explicit agreement.
- The authors assume no responsibility about this material and provide this material "as is", with no implicit or explicit warranty about the correctness and completeness of its contents, which may be subject to changes.
- This copyright notice must always be redistributed together with the material, or its portions.

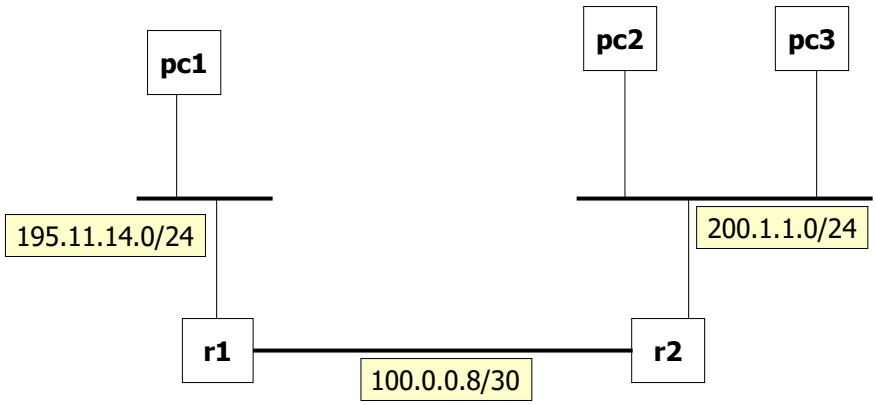
content of the lab

- there are two routers, called r1 and r2, and three hosts, called pc1, pc2, and pc3
  - they are connected via three LANs
  - we force their MAC addresses to be easily readable
- we will learn how to:
  - assign an IPv4 address and a netmask to the interface of a host
  - assign a default gateway to the interface of a host
  - set the routing table of a router
- we will use the ping and traceroute commands
- we will observe the behavior of ARP

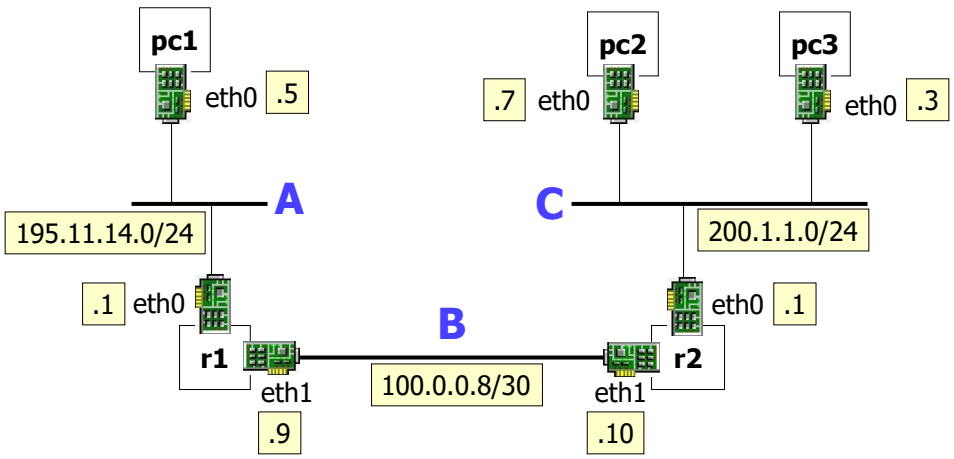
Kathará

lab configuration

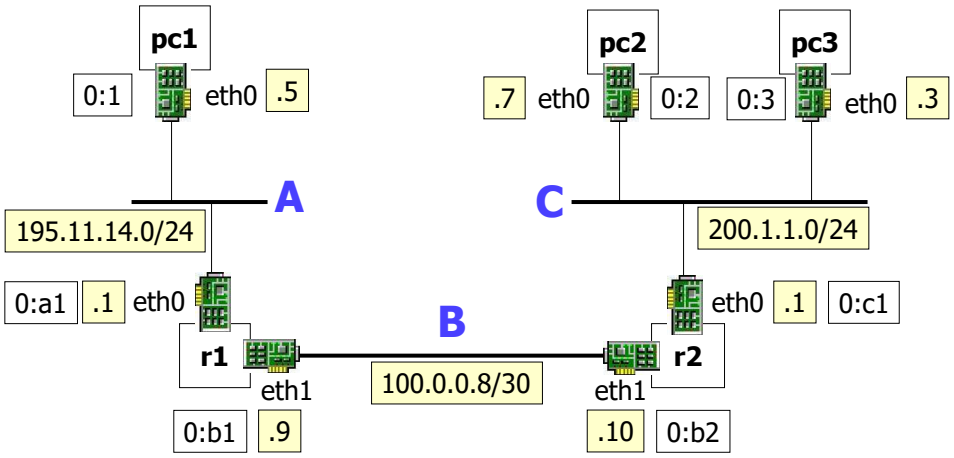
network topology – high level view



network topology – IPv4 address plan



network topology – MAC addresses



a quick look at the lab

lab.conf

```
r1[0]="A"
r1[1]="B"
r1[image]="kathara/base"
r1[ipv6]="false"

r2[0]="C"
r2[1]="B"
r2[image]="kathara/base"
r2[ipv6]="false"

pc1[0]="A"
pc1[image]="kathara/base"
pc1[ipv6]="false"
```

lab.conf

```
pc2[0]="C"
pc2[image]="kathara/base"
pc2[ipv6]="false"

pc3[0]="C"
pc3[image]="kathara/base"
pc3[ipv6]="false"

wireshark[bridged]=true
wireshark[port]="3000:3000"
wireshark[image]="lscr.io/linuxserver/wireshark"
wireshark[num_terms]=0
```

a quick look at the lab

pc1.startup

```
ip link set dev eth0 address 00:00:00:00:00:01
ip address add 195.11.14.5/24 dev eth0
ip route add default via 195.11.14.1
```

pc2.startup

```
ip link set dev eth0 address 00:00:00:00:00:02
ip address add 200.1.1.7/24 dev eth0
ip route add default via 200.1.1.1 dev eth0
```

pc3.startup

```
ip link set dev eth0 address 00:00:00:00:00:03
ip address add 200.1.1.3/24 dev eth0
ip route add default via 200.1.1.1 dev eth0
```


a quick look at the lab

an IPv4 address is assigned to the eth0 interfaces of hosts

pc1.startup

```
ip link set dev eth0 address 00:00:00:00:00:01
ip address add 195.11.14.5/24 dev eth0
ip route add default via 195.11.14.1
```

pc2.startup

```
ip link set dev eth0 address 00:00:00:00:00:02
ip address add 200.1.1.7/24 dev eth0
ip route add default via 200.1.1.1 dev eth0
```

pc3.startup

```
ip link set dev eth0 address 00:00:00:00:00:03
ip address add 200.1.1.3/24 dev eth0
ip route add default via 200.1.1.1 dev eth0
```

a quick look at the lab

an IPv4 address is added to the eth0 interfaces of hosts

a default gateway is set for all hosts

pc1.startup

```
ip link set dev eth0 address 00:00:00:00:00:01
ip address add 195.11.14.5/24 dev eth0
ip route add default via 195.11.14.1
```

pc2.startup

```
ip link set dev eth0 address 00:00:00:00:00:02
ip address add 200.1.1.7/24 dev eth0
ip route add default via 200.1.1.1 dev eth0
```

pc3.startup

```
ip link set dev eth0 address 00:00:00:00:00:03
ip address add 200.1.1.3/24 dev eth0
ip route add default via 200.1.1.1 dev eth0
```


a quick look at the lab

r1.startup

```
ip link set dev eth0 address 00:00:00:00:00:a1
ip link set dev eth1 address 00:00:00:00:00:b1
ip address add 195.11.14.1/24 dev eth0
ip address add 100.0.0.9/30 dev eth1
ip route add 200.1.1.0/24 via 100.0.0.10 dev eth1
```

a quick look at the lab

r1.startup

```
ip link set dev eth0 address 00:00:00:00:00:a1
ip link set dev eth1 address 00:00:00:00:00:b1
ip address add 195.11.14.1/24 dev eth0
ip address add 100.0.0.9/30 dev eth1
ip route add 200.1.1.0/24 via 100.0.0.10 dev eth1
```

an IPv4 address is assigned to interfaces eth0 and eth1 of router r1

consequently, the corresponding LANs are considered *directly connected*

a quick look at the lab

r1.startup

```
ip link set dev eth0 address 00:00:00:00:00:a1
ip link set dev eth1 address 00:00:00:00:00:b1
ip address add 195.11.14.1/24 dev eth0
ip address add 100.0.0.9/30 dev eth1
ip route add 200.1.1.0/24 via 100.0.0.10 dev eth1
```

an IPv4 address is given to interfaces eth0 and eth1 of router r1

consequently, the corresponding LANs are considered *directly connected*

a row is added to the routing table on how to reach a LAN that is not directly connected

a quick look at the lab

r1.startup

```
ip link set dev eth0 address 00:00:00:00:00:a1
ip link set dev eth1 address 00:00:00:00:00:b1
ip address add 195.11.14.1/24 dev eth0
ip address add 100.0.0.9/30 dev eth1
ip route add 200.1.1.0/24 via 100.0.0.10 dev eth1
```

an IPv4 address is given to interfaces eth0 and eth1 of router r1

consequently, the corresponding LANs are considered *directly connected*

a row is added to the routing table on how to reach a LAN that is not directly connected

similar configuration for router r2

start the lab

- start the lab

```
user@localhost:~$ cd kathara-lab_basic-ipv4
user@localhost:~/kathara-lab_basic-ipv4$ lstart
```

Katharà

useful commands

check the IPv4 addresses

- on pc1, pc2, pc3, r1, and r2
  - perform the `ip address` command, to check the IPv4 addresses assigned to the interfaces
    - look at eth and loopback interfaces

loopback interface
127.0.0.1/8

eth0
195.11.14.5/24

pc1

```
root@pc1:/# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
group default 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
7: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
state UP group default qlen 1000
    link/ether 00:00:00:00:00:01 brd ff:ff:ff:ff:ff:ff
    inet 195.11.14.5/24 scope global eth0
       valid_lft forever preferred_lft forever
```

check the default route

- on pc1, pc2, and pc3
  - perform the `routel` command, to check the presence of a default route

default route by r1

loopback prefix

useful information: for each d.c. network, my address and broadcast

pc1

```
root@pc1:/# routel
Dst             Gateway         Prefsrc         Protocol Scope   Dev     Table
default         195.11.14.1                                      eth0
195.11.14.0/24                  195.11.14.5     kernel   link    eth0
127.0.0.0/8                     127.0.0.1       kernel   host    lo      local
127.0.0.1                       127.0.0.1       kernel   host    lo      local
127.255.255.255                 127.0.0.1       kernel   link    lo      local
195.11.14.5                     195.11.14.5     kernel   host    eth0    local
195.11.14.255                   195.11.14.5     kernel   link    eth0    local
```


check the router routing tables

- on r1, and r2
  - perform the `routel` command, to check the routing table

Lan B

Lan A

Lan C

useful information: for each d.c. network, my address and broadcast

r1

```
root@r1:/# routel
Dst             Gateway         Prefsrc         Protocol Scope   Dev     Table
100.0.0.8/30                    100.0.0.9       kernel   link    eth1
195.11.14.0/24                  195.11.14.1     kernel   link    eth0
200.1.1.0/24    100.0.0.10                                       eth1
100.0.0.9                       100.0.0.9       kernel   host    eth1    local
100.0.0.11                      100.0.0.9       kernel   link    eth1    local
127.0.0.0/8                     127.0.0.1       kernel   host    lo      local
127.0.0.1                       127.0.0.1       kernel   host    lo      local
127.255.255.255                 127.0.0.1       kernel   link    lo      local
195.11.14.1                     195.11.14.1     kernel   host    eth0    local
195.11.14.255                   195.11.14.1     kernel   link    eth0    local
```

r2

snif the traffic

- connect the wireshark device to collision domain C

```
user@localhost:~/kathara-lab_basic-ipv4$ kathara lconfig -n wireshark --add C
```

- open any browser on the host machine
  - on **localhost:3000**
  - snif eth1

ping from pc3 to pc2 and related arp behavior

on pc3

1. inspect the ARP cache
2. execute a ping command towards pc2
3. inspect again the ARP cache
4. give a look at the packets captured by Wireshark

inspecting the arp cache of pc3

```
ARP(8)            Linux System Administrator's Manual
NAME
       arp - manipulate the system ARP cache

SYNOPSIS
    arp [-vn] [-H type] [-i if] [-ae] [hostname]
    arp [-v] [-i if] -d hostname [pub]
    arp [-v] [-H type] [-i if] -s hostname hw_addr [temp]
    arp [-v] [-H type] [-i if] -s hostname hw_addr [netmask nm] pub
    arp [-v] [-H type] [-i if] -Ds hostname ifname [netmask nm] pub
    arp [-vnD] [-H type] [-i if] -f [filename]

DESCRIPTION
Arp manipulates or displays the kernel's IPv4 network neighbour cache.
It can add entries to the table, delete one or display the currennt content.

ARP stands for Address Resolution Protocol, which is used to find the media
access control address of  a  network neighbour for a given IPv4 Address

.......
```

inspecting the arp cache

the arp cache is initially empty

sending packets to 200.1.1.7 requires address resolution

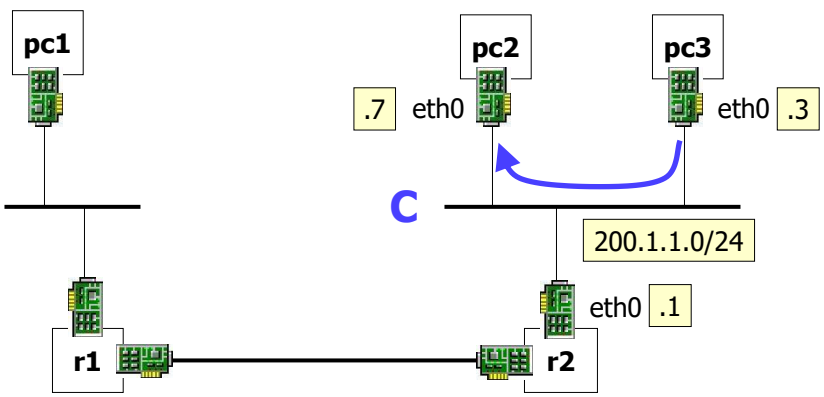
pc3

```
root@pc3:/# arp
root@pc3:/# ping 200.1.1.7
PING 200.1.1.7 (200.1.1.7) 56(84) bytes of data.
64 bytes from 200.1.1.7: icmp_seq=1 ttl=64 time=1.93 ms
64 bytes from 200.1.1.7: icmp_seq=2 ttl=64 time=0.638 ms
--- 200.1.1.7 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.638/1.283/1.929/0.645 ms
root@pc3:/# arp
Address                  HWtype  HWaddress           Flags Mask    Iface
200.1.1.7                ether   00:00:00:00:00:02   C             eth0
```

address resolution results are stored in the arp cache

inspecting the arp cache

- traffic within the same network does not traverse routers



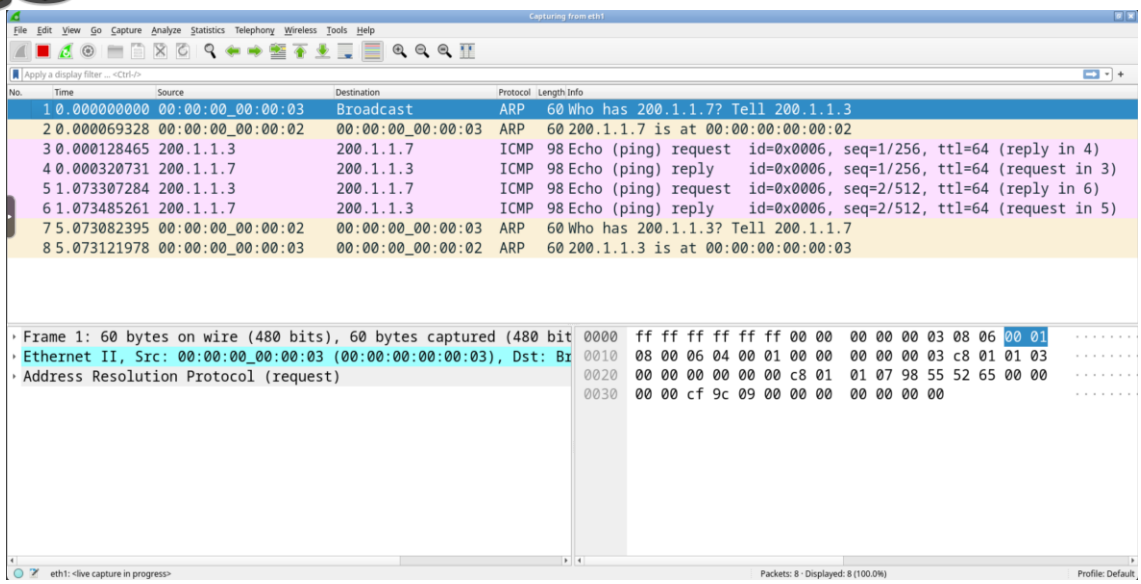
inspecting the arp cache

- communications are usually bi-directional
- the receiver of the arp request learns the mac address of the other party, to avoid a new arp in opposite direction (standard behavior, see rfc 826)

pc2

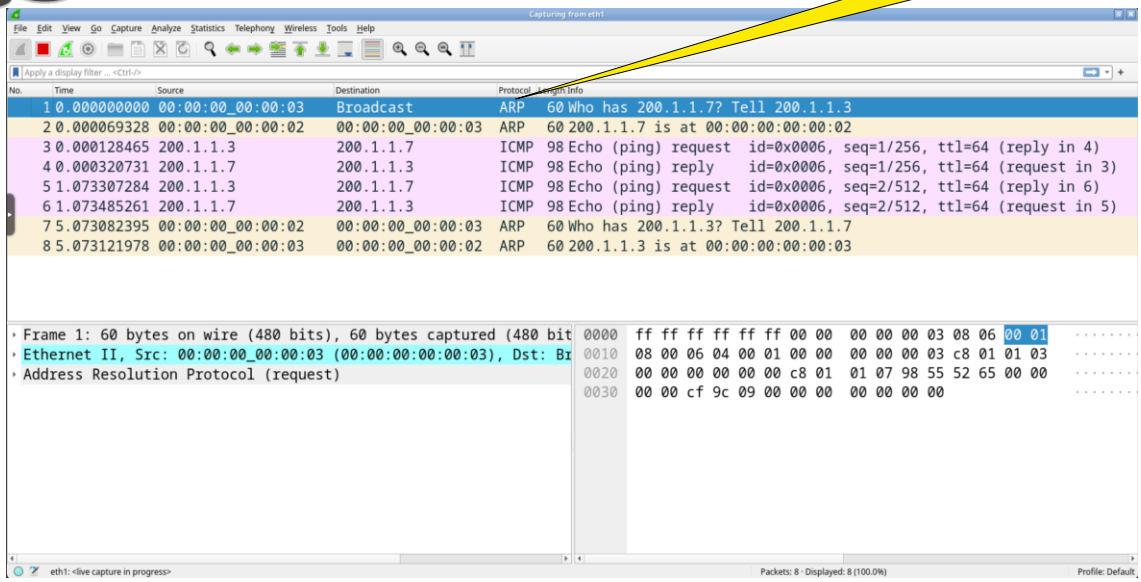
```
root@pc2:/# arp -n
Address                  HWtype  HWaddress           Flags Mask  Iface
200.1.1.3                ether   00:00:00:00:00:03   C           eth0
```


wireshark



wireshark

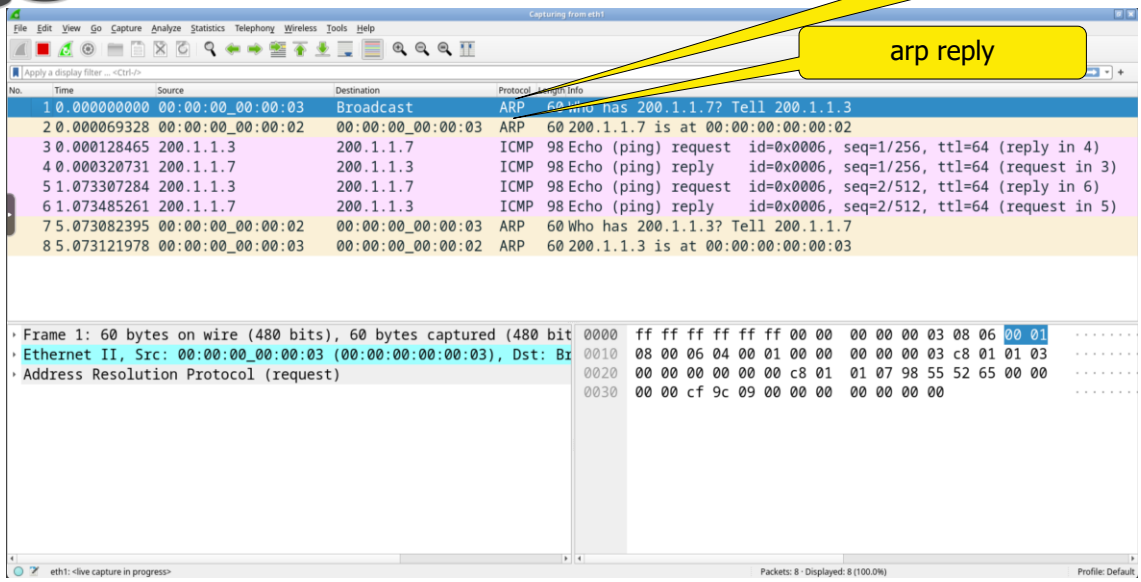
arp request



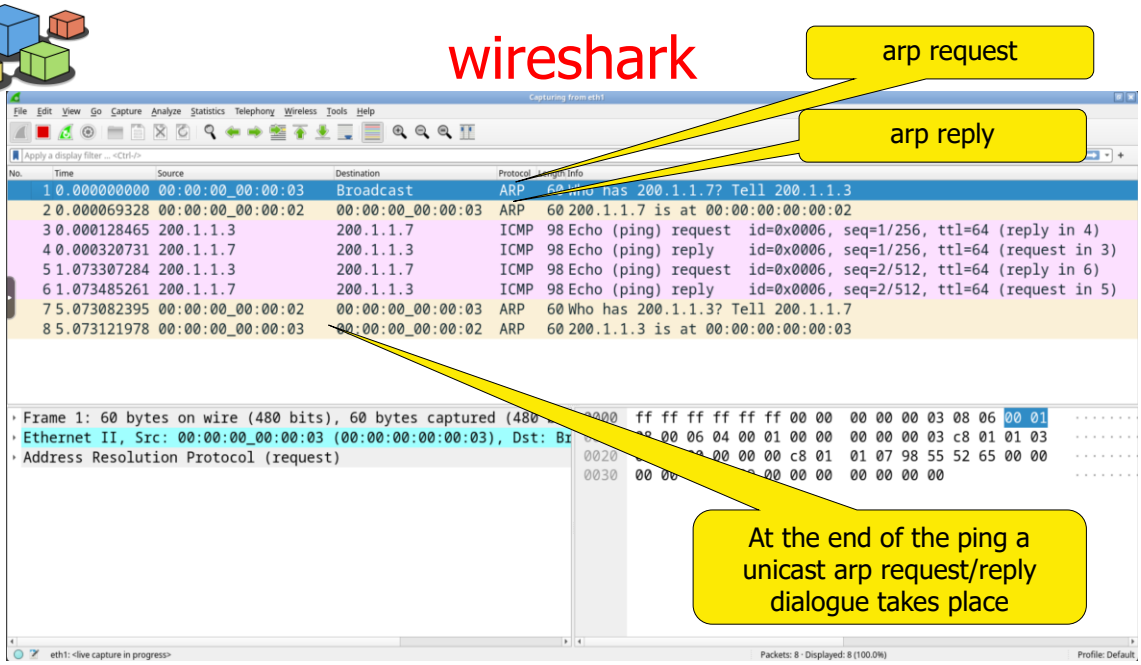
wireshark

arp request

arp reply



wireshark



Kathará

ping from pc2 to pc1 and related arp behavior

snif the traffic

- connect the wireshark host to collision domain B

```
user@localhost:~/kathara-lab_basic-ipv4$ kathara lconfig -n wireshark
--add B
```

- open any browser on the host machine
  - on **localhost:3000**
  - snif eth2

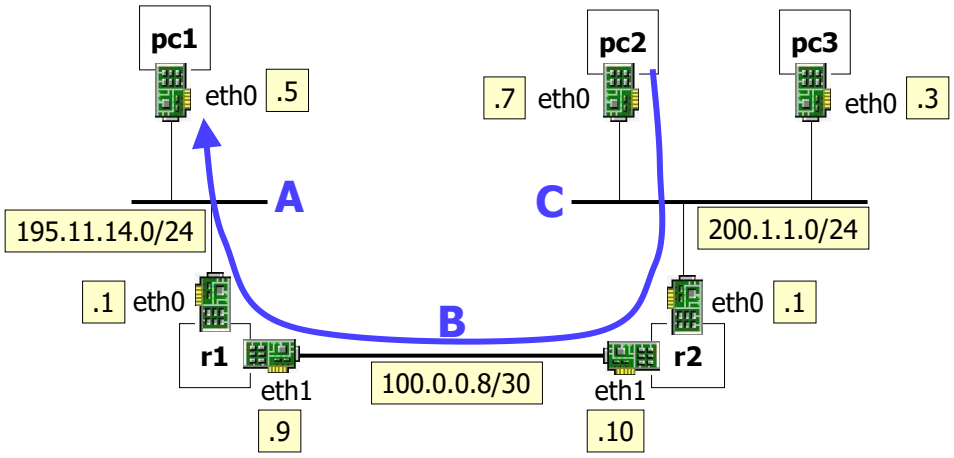
on pc2

- execute a ping command towards pc1

pc2

```
root@pc2:/# ping 195.11.14.5
PING 195.11.14.5 (195.11.14.5) 56(84) bytes of data.
64 bytes from 195.11.14.5: icmp_seq=1 ttl=62 time=5.86 ms
64 bytes from 195.11.14.5: icmp_seq=2 ttl=62 time=1.69 ms
--- 195.11.14.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 1.688/3.771/5.855/2.083 ms
```

inspecting the arp cache (non local traffic)



inspecting the arp cache (non local traffic)

- when ip traffic is addressed outside the local network, the sender needs the mac address of the router
- arp requests can get replies only within the local network

pc2

```
root@pc2:/# arp -n
Address                  HWtype  HWaddress           Flags Mask   Iface
200.1.1.1                ether   00:00:00:00:00:c1   C            eth0
200.1.1.3                ether   00:00:00:00:00:03   C            eth0
root@pc2:/#
```

mac address of eth0 on r2

inspecting the arp cache
(non local traffic)

- what about routers?
- routers perform arp too (hence have arp caches) anytime they have to send ip packets on an ethernet lan

r1

```
root@r1:/# arp -n
Address                  HWtype  HWaddress           Flags Mask   Iface
195.11.14.5              ether   00:00:00:00:00:01   C            eth0
100.0.0.10               ether   00:00:00:00:00:b2   C            eth1
root@r1:/#
```

pc1

r2 (eth1)

r2

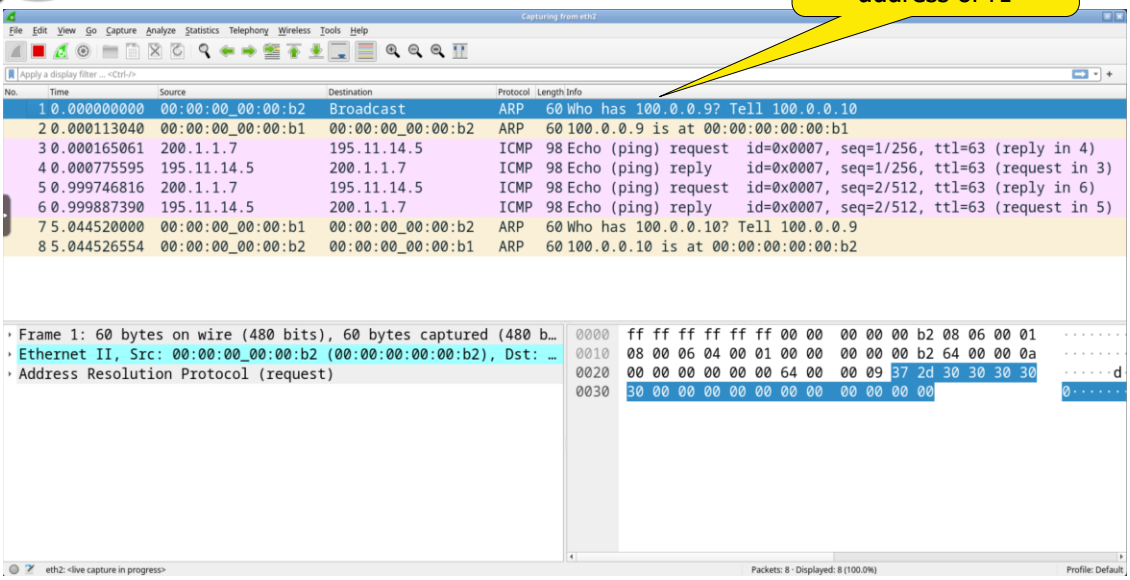
```
root@r2:/# arp -n
Address                  HWtype  HWaddress           Flags Mask   Iface
100.0.0.9                ether   00:00:00:00:00:b1   C            eth1
200.1.1.7                ether   00:00:00:00:00:02   C            eth0
root@r2:/#
```

r1 (eth1)

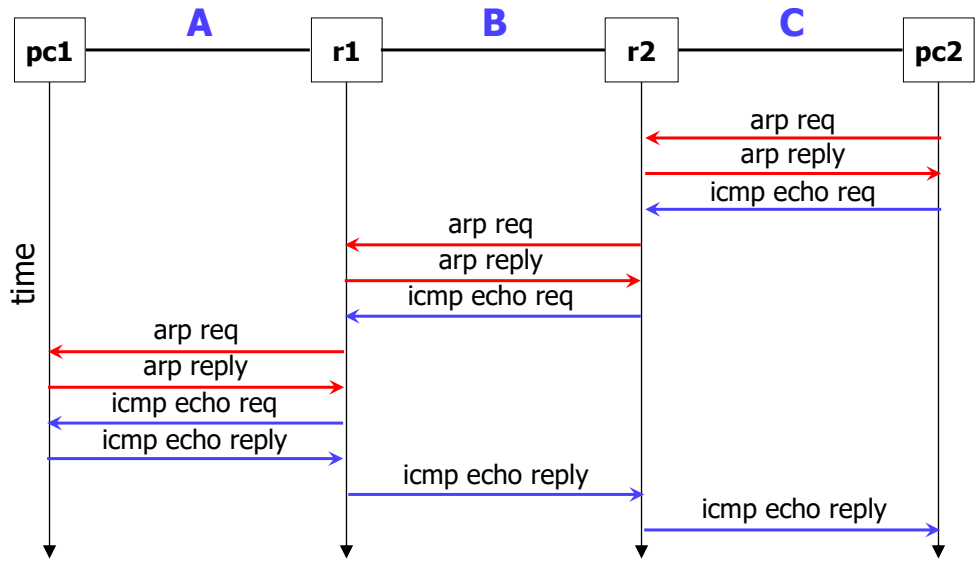
pc2

wireshark

which is the mac address of r1



understanding the whole picture



Kathará

traceroute from pc2 to pc1 and related arp behavior

snif the traffic

- the wireshark host is already connected to collision domain C
- open any browser on the host machine
  - on **localhost:3000**
  - snif eth1

on pc2

- execute a traceroute command towards pc1

eth0 of r2

eth1 of r1

eth0 of pc1

Minimal time (sec. if ≤10, ms if >10) interval between probes (default 0)

pc2

```
root@pc2:/# traceroute 195.11.14.5 -z 1
traceroute to 195.11.14.5 (195.11.14.5), 30 hops max, 60 byte packets
 1  200.1.1.1 (200.1.1.1)  0.882 ms  0.662 ms  0.456 ms
 2  100.0.0.9 (100.0.0.9)  0.903 ms  0.877 ms  1.218 ms
 3  195.11.14.5 (195.11.14.5)  0.987 ms  1.354 ms  1.015 ms
root@pc2:/#
```

udp packet and corresponding ICMP Time-to-live exceeded

wireshark

Time to live: 1

udp packet and corresponding ICMP Time-to-live exceeded

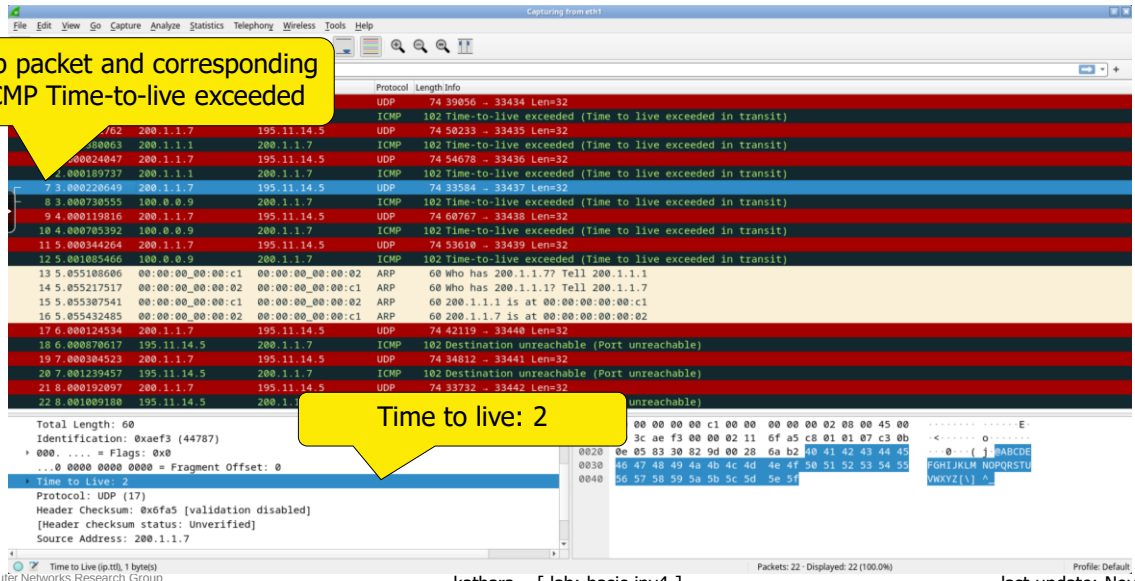
wireshark

3 probes for each *ttl*

wireshark

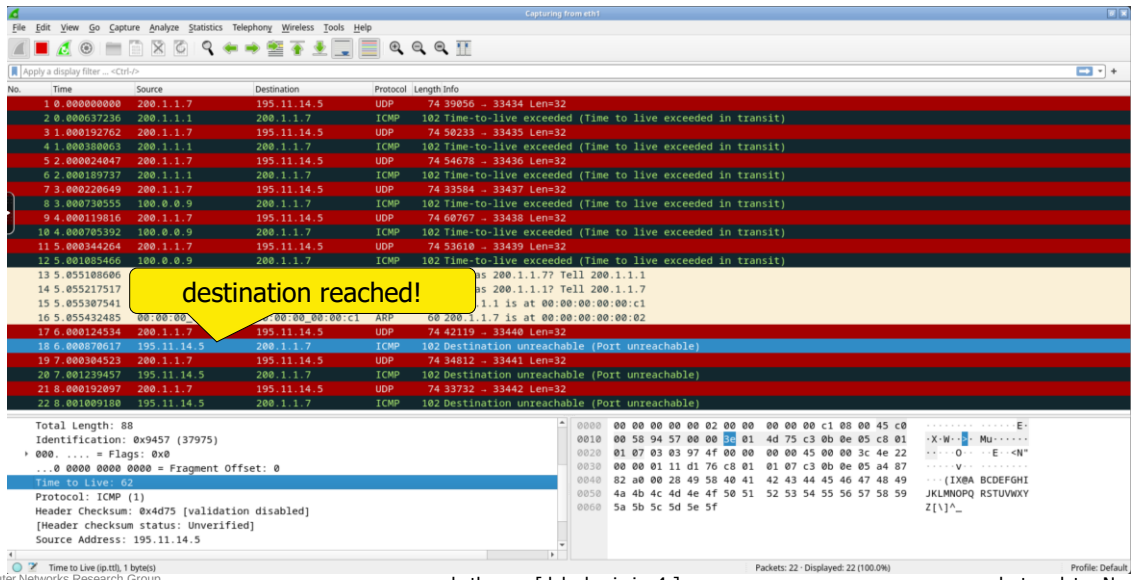
udp packet and corresponding ICMP Time-to-live exceeded

Time to live: 2

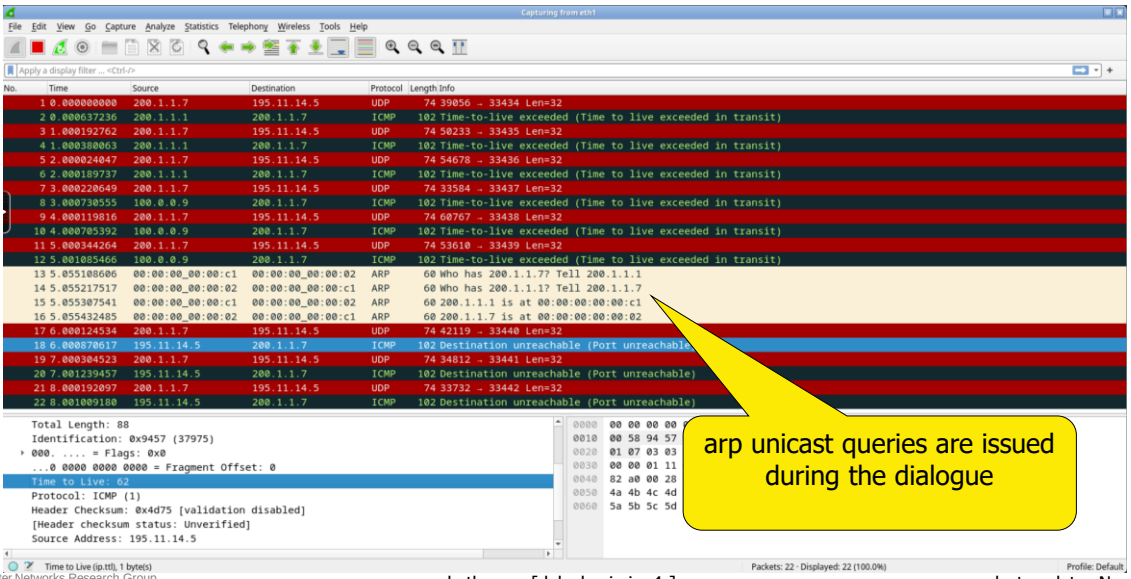


wireshark

destination reached!



wireshark



proposed exercises

- check the different error messages obtained by trying to ping an unreachable destination in the case of
  - local destination
  - non-local destination
- which packets are exchanged in the local collision domain in the two cases?