Università degli Studi Roma Tre
Dipartimento di Ingegneria
Computer Networks Research Group

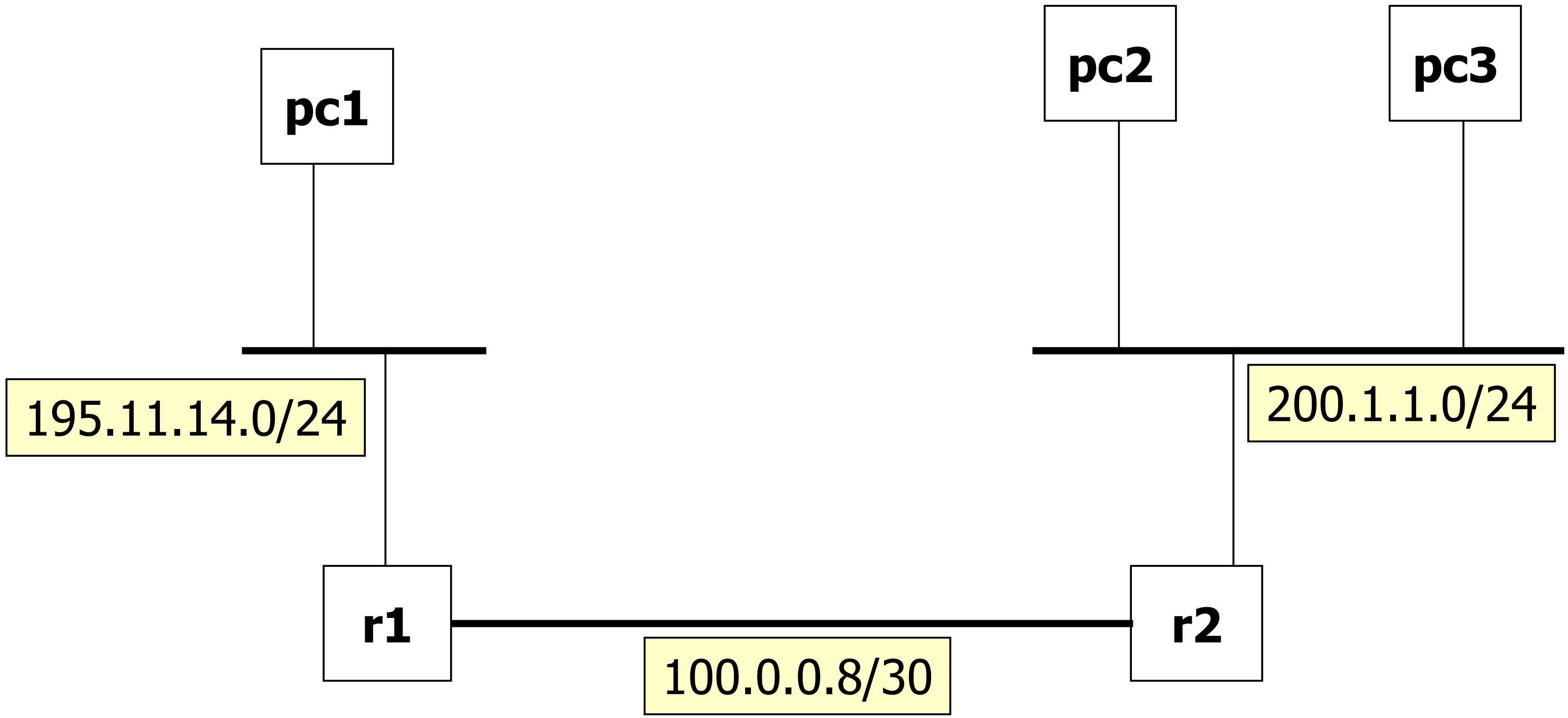
kathara lab

arp

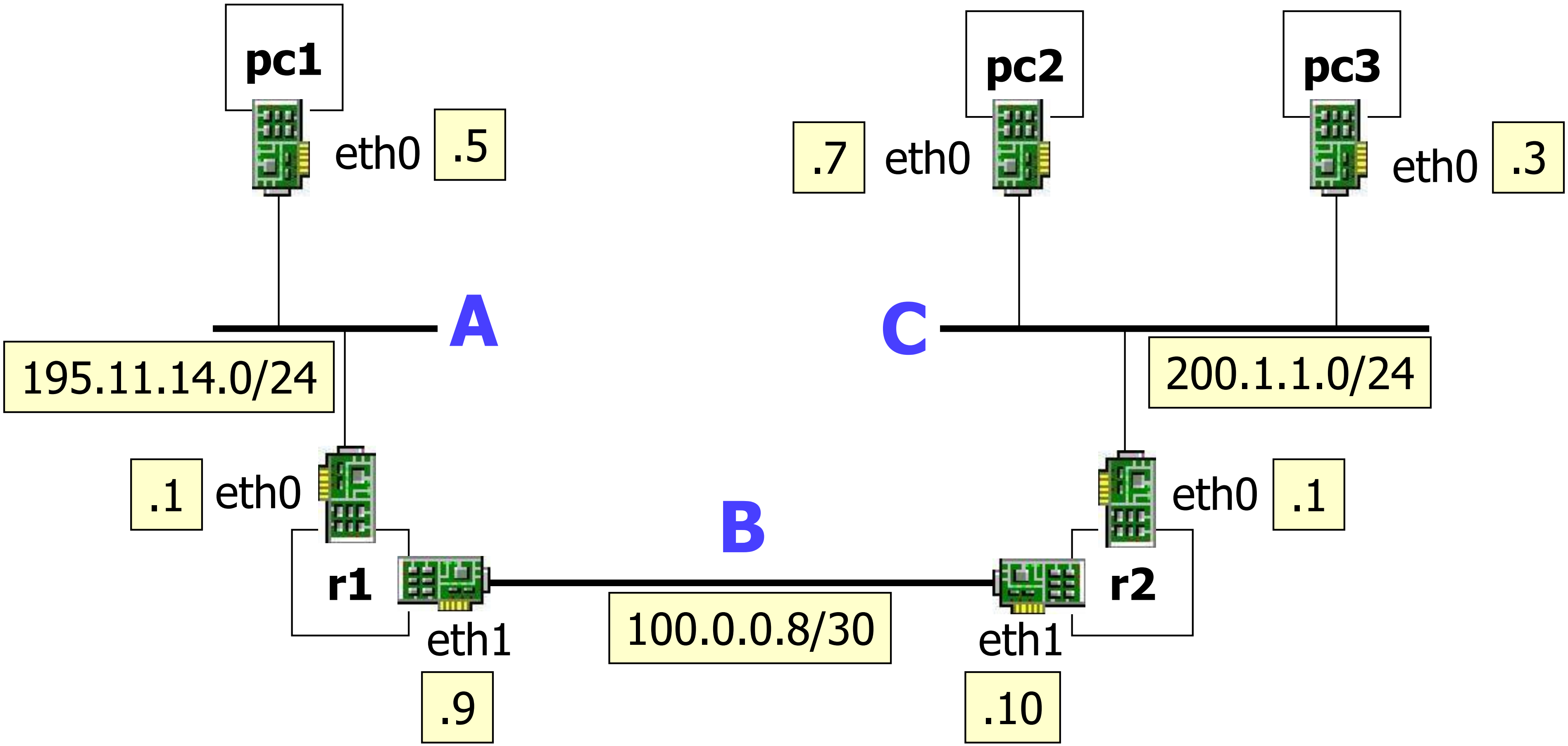
Version	1.0
Author(s)	G. Di Battista, M. Patrignani, M. Pizzonia, F. Ricci, M. Rimondini
E-mail	contact@kathara.org
Web	http://www.kathara.org/
Description	using the address resolution protocol for local and non local traffic – kathara version of the netkit lab on arp version 2.2

copyright notice

step 1 – network topology high level view



step 1 – network topology configuration details



step 2 – a quick look at the lab

lab.conf

```
r1[0]="A"
r1[1]="B"

r2[0]="C"
r2[1]="B"

pc1[0]="A"

pc2[0]="C"

pc3[0]="C"
```

pc1.startup

```
ifconfig eth0 195.11.14.5 up
route add default gw 195.11.14.1
```

pc2.startup

```
ifconfig eth0 200.1.1.7 up
route add default gw 200.1.1.1
```

pc3.startup

```
ifconfig eth0 200.1.1.3 up
route add default gw 200.1.1.1
```

step 2 – a quick look at the lab

r1.startup

```
ifconfig eth0 195.11.14.1 up
ifconfig eth1 100.0.0.9 netmask 255.255.255.252 broadcast 100.0.0.11 up
route add -net 200.1.1.0 netmask 255.255.255.0 gw 100.0.0.10 dev eth1
```

r2.startup

```
ifconfig eth0 200.1.1.1 up
ifconfig eth1 100.0.0.10 netmask 255.255.255.252 broadcast 100.0.0.11 up
route add -net 195.11.14.0 netmask 255.255.255.0 gw 100.0.0.9 dev eth1
```

- start the lab

host machine

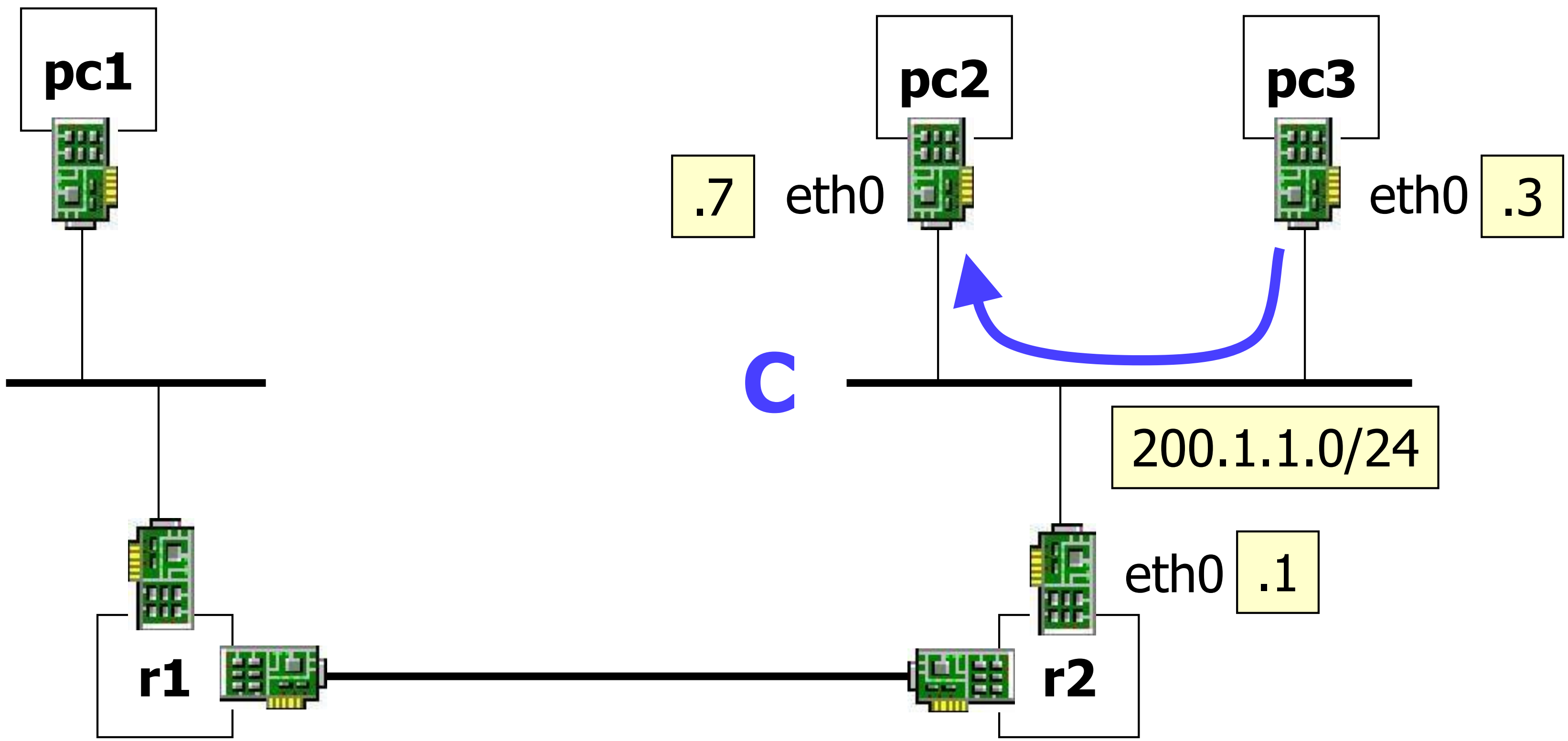
```
user@localhost:~$ cd kathara-lab_arp
user@localhost:~/kathara-lab_arp$ lstart
```

step 3 – inspecting the arp cache

```
ARP(8)                    Linux Programmer's Manual                    ARP(8)

NAME
       arp - manipulate the system ARP cache

SYNOPSIS
       arp [-vn] [-H type] [-i if] -a [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 ifa [netmask nm] pub
       arp [-vnD] [-H type] [-i if] -f [filename]

DESCRIPTION
       Arp  manipulates  the  kernel's ARP cache in various ways.  The primary
       options are clearing an address mapping entry and manually  setting  up
       one.   For  debugging  purposes, the arp program also allows a complete
       dump of the ARP cache.
......
```

step 3 – inspecting the arp cache (local traffic)

- traffic within the same network does not traverse routers



step 3 – inspecting the arp cache (local traffic)

pc3

```
pc3:~# arp
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=1.39 ms
64 bytes from 200.1.1.7: icmp_seq=2 ttl=64 time=0.542 ms

--- 200.1.1.7 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1022ms
rtt min/avg/max/mdev = 0.542/0.969/1.396/0.427 ms
pc3:~# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
200.1.1.7                ether   FE:FD:C8:01:01:07   C                      eth0
pc3:~# █
```

the arp cache is initially empty

sending packets to 200.1.1.7 requires address resolution

address resolution results are stored in the arp cache

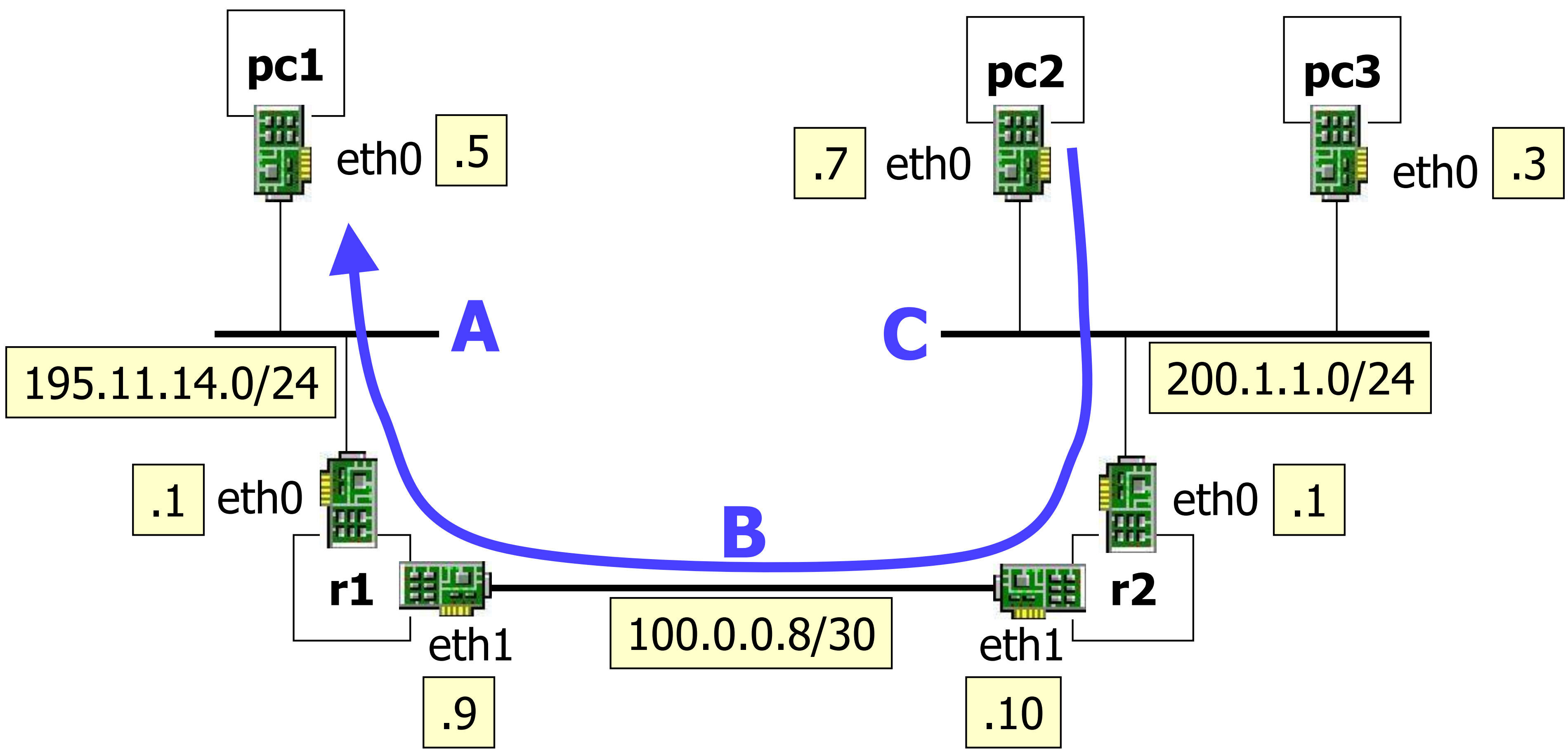
step 3 – inspecting the arp cache (local traffic)

- 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

```
pc2:~# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
200.1.1.3                ether   FE:FD:C8:01:01:03   C                     eth0
pc2:~#
```

step 4 – inspecting the arp cache (non local traffic)



step 4 – 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

```
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=30.4 ms
64 bytes from 195.11.14.5: icmp_seq=2 ttl=62 time=1.02 ms

--- 195.11.14.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1013ms
rtt min/avg/max/mdev = 1.024/15.731/30.438/14.707 ms
pc2:~# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
200.1.1.1                ether   FE:FD:C8:01:01:01   C                     eth0
200.1.1.3                ether   FE:FD:C8:01:01:03   C                     eth0
pc2:~#
```

step 4 – 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

```
r1:~# arp -n
Address          HWtype  HWaddress           Flags Mask      Iface
100.0.0.10       ether   FE:FD:64:00:00:0A   C               eth1
195.11.14.5      ether   FE:FD:C3:0B:0E:05   C               eth0
r1:~# █
```

r2 (eth1)

pc1

r2

```
r2:~# arp -n
Address          HWtype  HWaddress           Flags Mask      Iface
200.1.1.7        ether   FE:FD:C8:01:01:07   C               eth0
100.0.0.9        ether   FE:FD:64:00:00:09   C               eth1
r2:~# █
```

pc2

r1 (eth1)

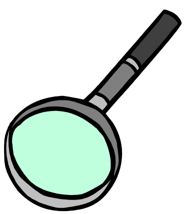
step5 – sniffing arp traffic

- restart the lab in order to clear arp caches

host machine

```
user@localhost:~/kathara-lab_arp$ lclean
user@localhost:~/kathara-lab_arp$ lstart
```

- get ready to sniff

 =

virtual machine

```
vm:~# tcpdump -tenni eth0
```

sniff on this interface

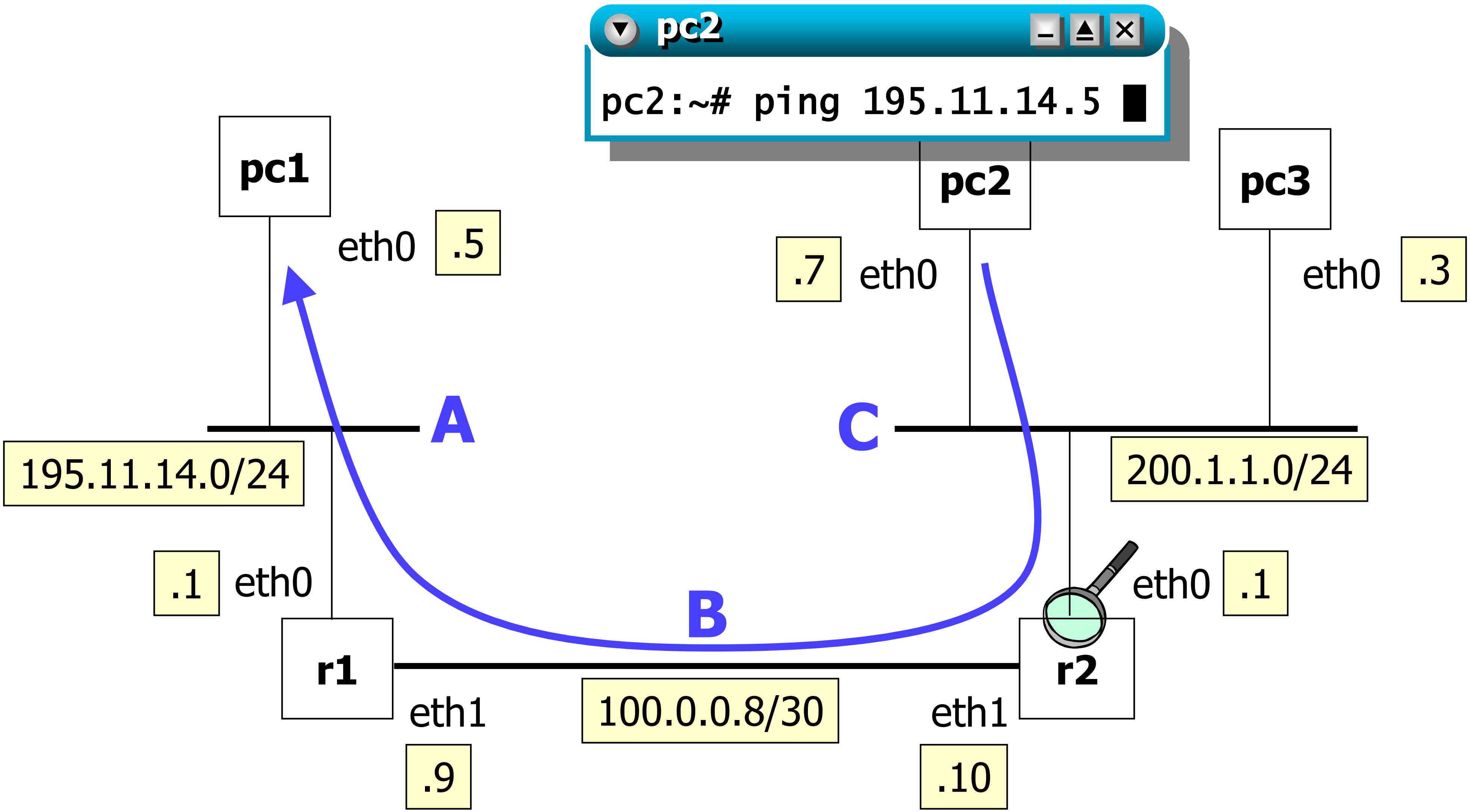
No DNS look up

No protocol name

suppress timestamps (kathara is not a simulation system, hence timestamps are not meaningful)

show link-level headers (mac addresses)

step 5 – sniffing arp traffic



step 5 – sniffing arp traffic

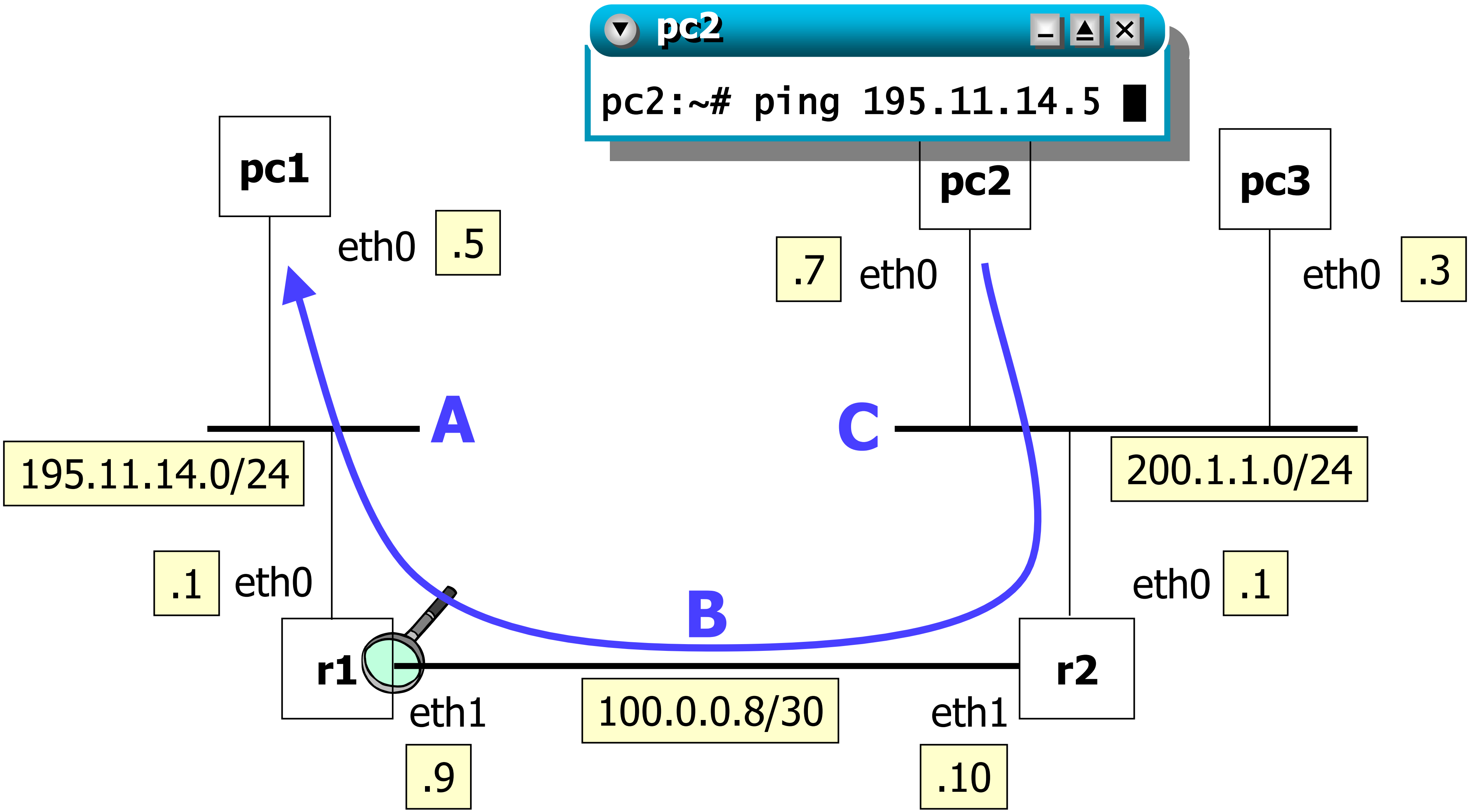
- on collision domain C

r2

```
r2:~# tcpdump -tenni eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
fe:fd:c8:01:01:07 > Broadcast, ethertype ARP (0x0806), length 42: arp who-has
200.1.1.1 tell 200.1.1.7
fe:fd:c8:01:01:01 > fe:fd:c8:01:01:07, ethertype ARP (0x0806), length 42: arp reply
200.1.1.1 is-at fe:fd:c8:01:01:01
fe:fd:c8:01:01:07 > fe:fd:c8:01:01:01, ethertype IPv4 (0x0800), length 98: IP
200.1.1.7 > 195.11.14.5: icmp 64: echo request seq 1
fe:fd:c8:01:01:01 > fe:fd:c8:01:01:07, ethertype IPv4 (0x0800), length 98: IP
195.11.14.5 > 200.1.1.7: icmp 64: echo reply seq 1
█
```

1. **pc2** asks all the stations on collision domain C: "who has 200.1.1.1?" (200.1.1.1 is **pc2**'s default gateway)
2. **r2** replies ⇒ both **pc2** and **r2** update their arp cache
3. **pc2** sends to **r2** the ip packet (icmp echo request) for **pc1**
4. **r2** sends to **pc2** the corresponding echo reply (generated by **pc1**)

step 5 – sniffing arp traffic



step 6 – sniffing arp traffic

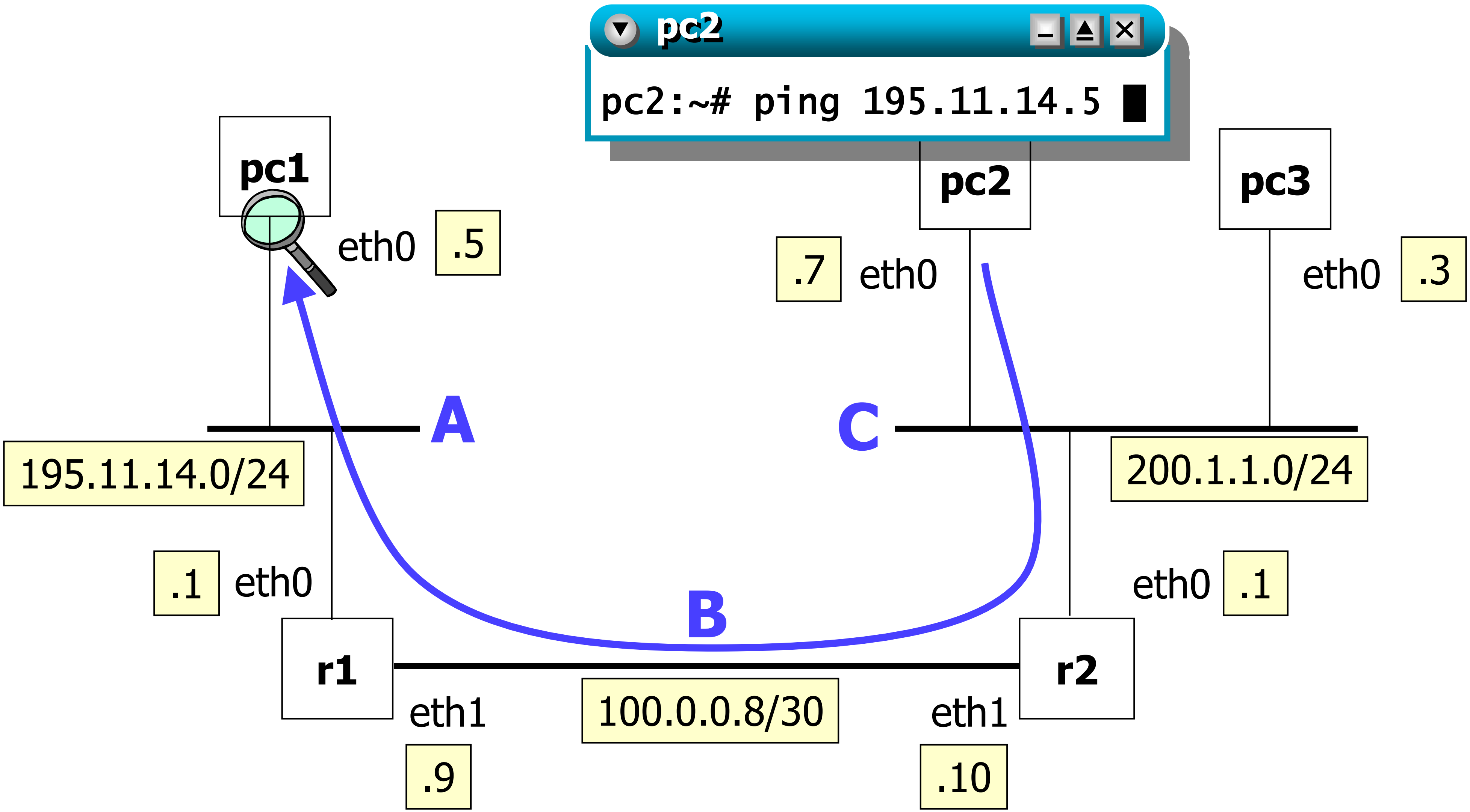
- on collision domain B

r1

```
r1:~# tcpdump -tenni eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 96 bytes
fe:fd:64:00:00:0a > Broadcast, ethertype ARP (0x0806), length 42: arp who-has
100.0.0.9 tell 100.0.0.10
fe:fd:64:00:00:09 > fe:fd:64:00:00:0a, ethertype ARP (0x0806), length 42: arp reply
100.0.0.9 is-at fe:fd:64:00:00:09
fe:fd:64:00:00:0a > fe:fd:64:00:00:09, ethertype IPv4 (0x0800), length 98: IP
200.1.1.7 > 195.11.14.5: icmp 64: echo request seq 1
fe:fd:64:00:00:09 > fe:fd:64:00:00:0a, ethertype IPv4 (0x0800), length 98: IP
195.11.14.5 > 200.1.1.7: icmp 64: echo reply seq 1
█
```

1. **r2** asks all the stations on collision domain B: "who has 100.0.0.9?" (100.0.0.9 is the next hop obtained from the routing table)
2. **r1** replies ⇒ both **r1** and **r2** update their arp cache
3. **r2** sends to **r1** the echo request generated by **pc2** for **pc1**
4. **r1** sends to **r2** the echo reply generated by **pc1** for **pc2**

step 5 – sniffing arp traffic



step 5 – sniffing arp traffic

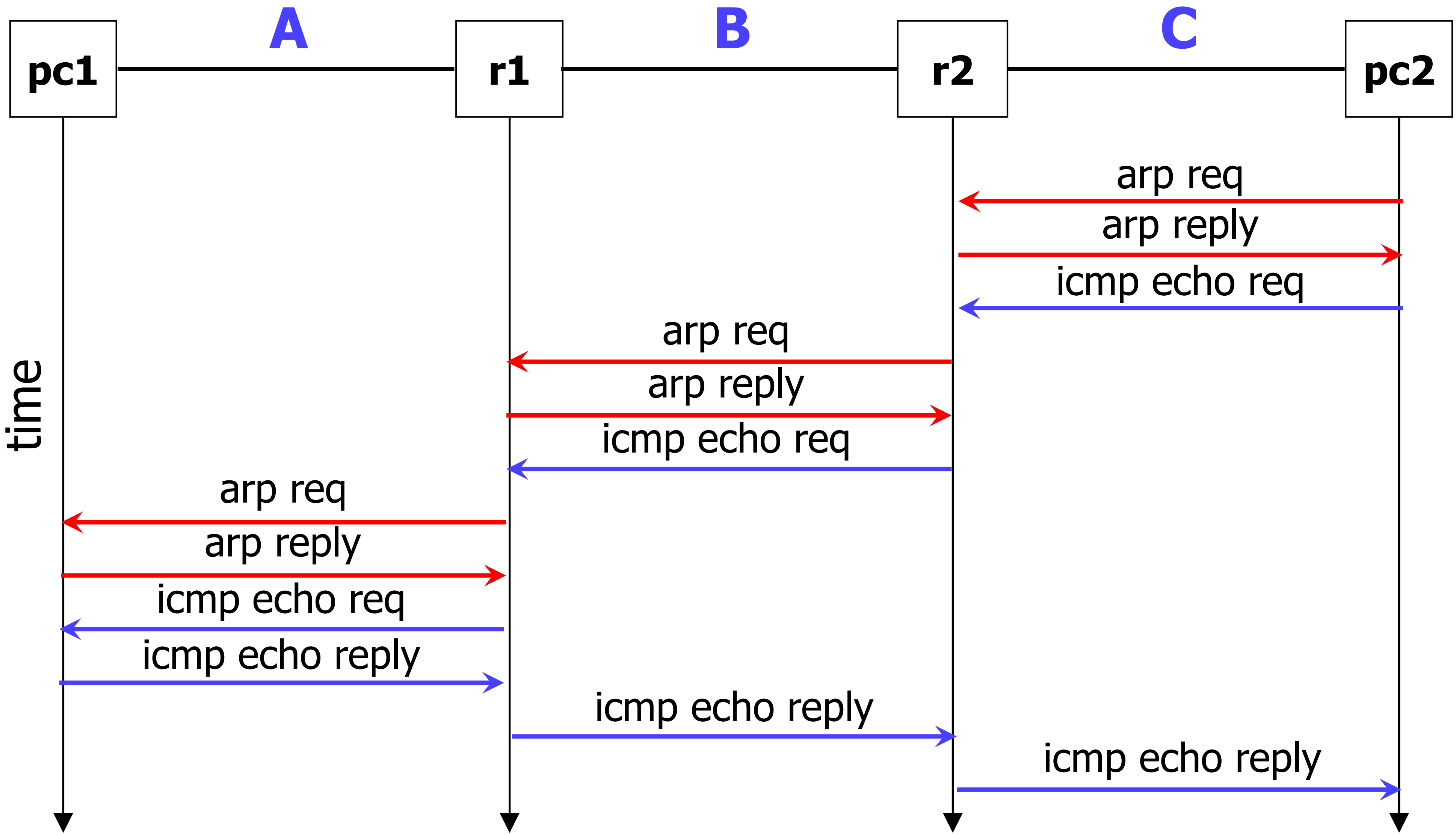
- on collision domain A

pc1

```
pc1:~# tcpdump -tenni eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
fe:fd:c3:0b:0e:01 > Broadcast, ethertype ARP (0x0806), length 42: arp who-has
195.11.14.5 tell 195.11.14.1
fe:fd:c3:0b:0e:05 > fe:fd:c3:0b:0e:01, ethertype ARP (0x0806), length 42: arp reply
195.11.14.5 is-at fe:fd:c3:0b:0e:05
fe:fd:c3:0b:0e:01 > fe:fd:c3:0b:0e:05, ethertype IPv4 (0x0800), length 98: IP
200.1.1.7 > 195.11.14.5: icmp 64: echo request seq 1
fe:fd:c3:0b:0e:05 > fe:fd:c3:0b:0e:01, ethertype IPv4 (0x0800), length 98: IP
195.11.14.5 > 200.1.1.7: icmp 64: echo reply seq 1 █
```

1. **r1** asks all the stations on collision domain A: "who has 195.11.14.5?" (195.11.14.5 is the destination address of the icmp request obtained from the ip header)
2. **pc1** replies ⇒ both **pc1** and **r1** update their arp cache
3. **r1** sends the ip packet (echo request) to **pc1**
4. **pc1** generates the corresponding echo reply for **pc2** and sends it to **r1**

step 6 – understanding the whole picture



step 7 – arp implementation details

r2

```
r2:~# tcpdump -tenni eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes

......

fe:fd:c8:01:01:01 > fe:fd:c8:01:01:07, ethertype ARP (0x0806), length 42: arp who-has
200.1.1.7 tell 200.1.1.1
fe:fd:c8:01:01:07 > fe:fd:c8:01:01:01, ethertype ARP (0x0806), length 42: arp reply
200.1.1.7 is-at fe:fd:c8:01:01:07
```

- arp requests are usually in broadcast
- it may also happen that a station (router/pc) sends a unicast arp request to check if an entry of the arp cache is still valid (suggested by the standard, see rfc 826)
- unicast arp requests may be performed periodically on each entry of the arp cache, depending on the implementation

proposed exercises

- what packets can we observe over all the three collision domains as the ping from **`pc2`** to **`pc1`** continues (ignore any implementation dependent arp behavior)?

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?