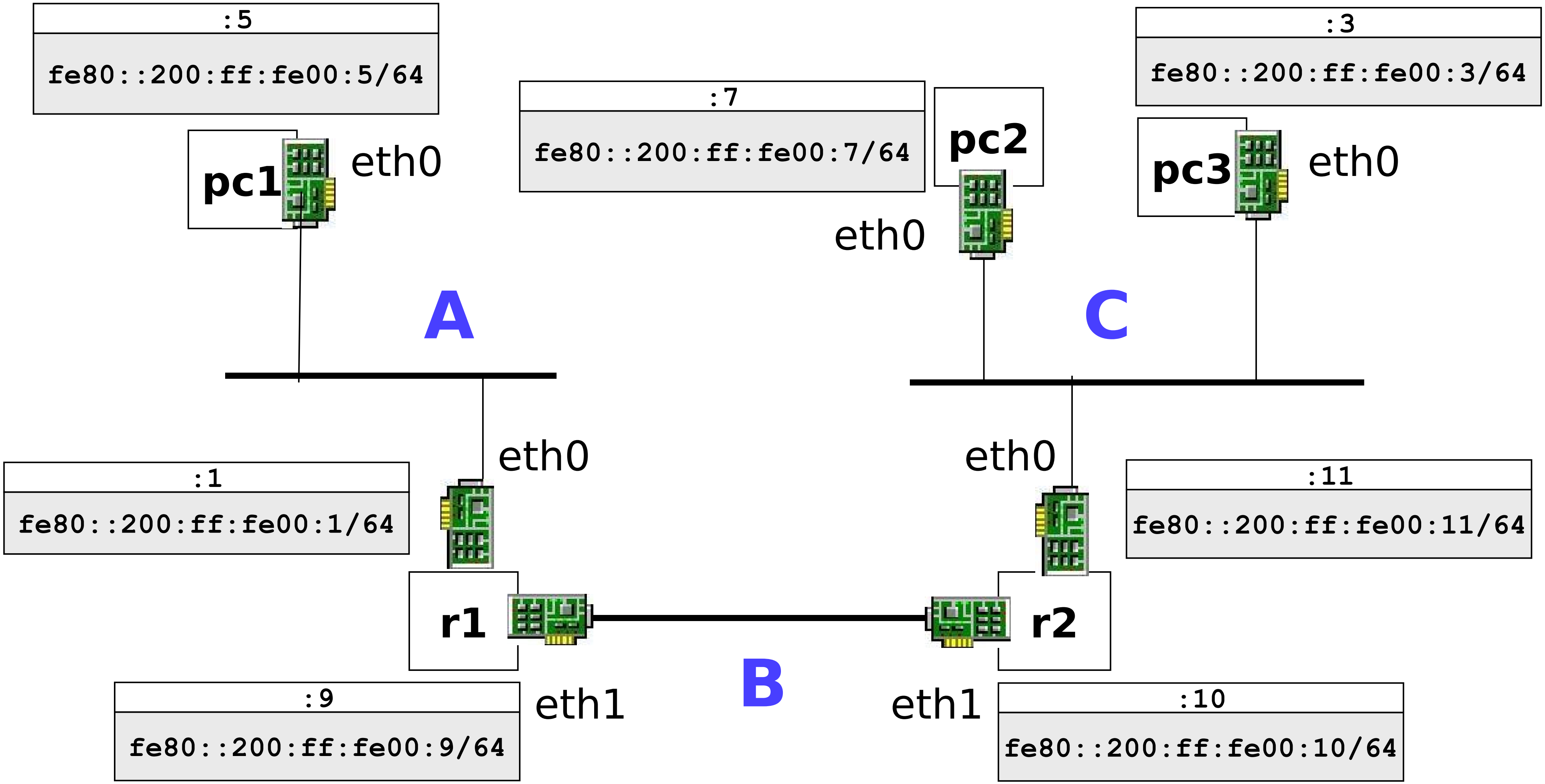
ISTITUTO TECNICO INDUSTRIALE STATALE
C. ZUCCANTE

netkit lab

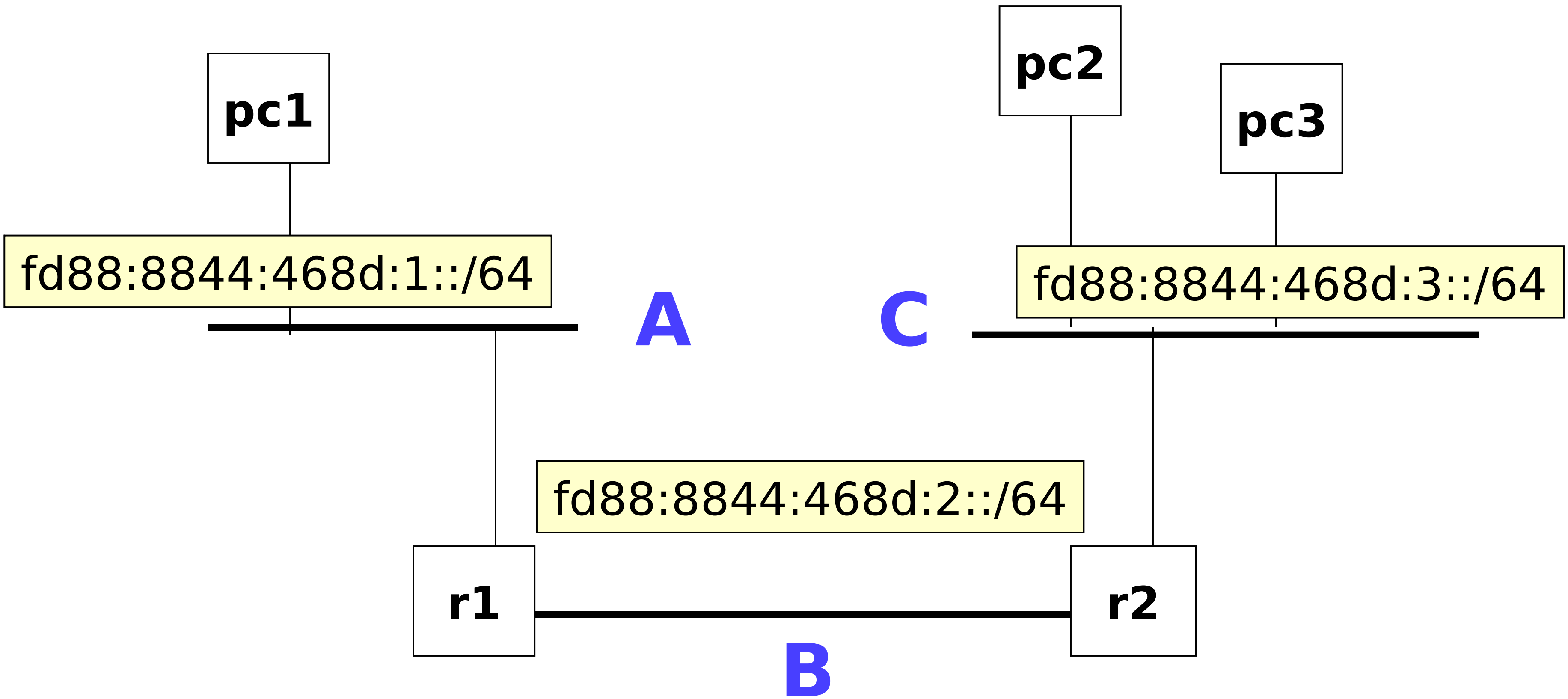
IPv6 Neighbor Discovery (NDP)

Version	1.0
Author(s)	S. Doro based on work “ARP” by G. Di Battista, M. Patrignani, M. Pizzonia, F. Ricci, M. Rimondini
E-mail	sandro.doro@gmail.com
Web	http://netkit.zuccante.it/
Description	using the ICMPv6 Neighbor Discovery and ICMPv6 Neighbor Solicitation

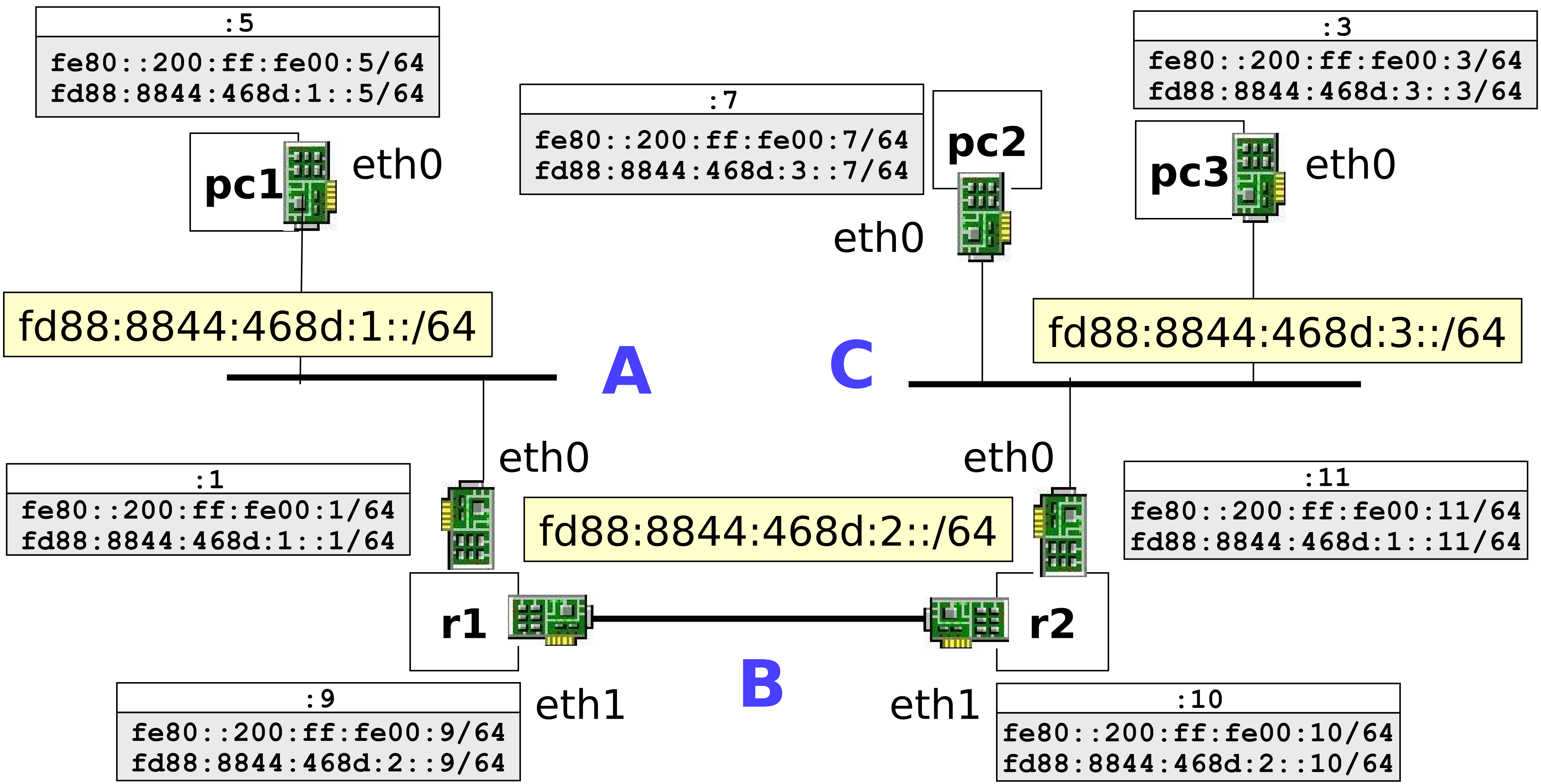
step 1 – network topology scope link details



step 1 – network topology
site link details (rfc 4193)



step 1 – network topology
all link 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

```
ip link set eth0 address 00:00:00:00:00:05
ip link set eth0 up
ip -6 addr add fd88:8844:468d:1::5/64 dev eth0
ip -6 route add default via fd88:8844:468d:1::1
```

pc2.startup

```
ip link set eth0 address 00:00:00:00:00:07
ip link set eth0 up
ip -6 addr add fd88:8844:468d:1::7/64 dev eth0
ip -6 route add default via fd88:8844:468d:3::11
```

pc3.startup

```
ip link set eth0 address 00:00:00:00:00:03
ip link set eth0 up
ip -6 addr add fd88:8844:468d:1::3/64 dev eth0
ip -6 route add default via fd88:8844:468d:3::11
```

step 2 – a quick look at the lab

r1.startup

```
# ---------------------------------------------- setup eth0
ip link set eth0 address 00:00:00:00:00:01
ip link set eth0 up
ip -6 addr add fd88:8844:468d:1::1/64 dev eth0
# ---------------------------------------------- setup eth1
ip link set eth1 address 00:00:00:00:00:09
ip link set eth1 up
ip -6 addr add fd88:8844:468d:2::9/64 dev eth1
# ---------------------------------------------- add route
ip -6 route add default via fd88:8844:468d:2::10
# ---------------------------------------------- enable IPv6 forwarding
echo "1" > /proc/sys/net/ipv6/conf/all/forwarding
```

step 2 – a quick look at the lab

r2.startup

```
ip link set eth0 address 00:00:00:00:00:11
ip link set eth0 up
ip -6 addr add fd88:8844:468d:3::11/64 dev eth0
ip link set eth1 address 00:00:00:00:00:10
ip link set eth1 up
ip -6 addr add fd88:8844:468d:2::10/64 dev eth1
ip -6 route add default via fd00:2::9
echo "1" > /proc/sys/net/ipv6/conf/all/forwarding
```

- start the lab

host machine

```
user@localhost:~$ cd netkit-lab_ndp
user@localhost:~/netkit-lab_ndp$ lstart
user@localhost:~/netkit-lab_ndp$ devilspie ds & # optional
```

step 3 – inspecting neighbors

With following command you can display the learnt or configured IPv6 neighbors

```
# ip -6 neigh show [dev <device>]
```

The following example shows one neighbor, which is a reachable router

```
# ip -6 neigh show
fe80::23ff:6789 dev eth0 lladdr 00:01:23:45:67:89 router nud reachable
```

With following command you are able to manually add an entry

```
# ip -6 neigh add <IPv6 address> lladdr <link-layer address> dev <device>
```

Example:

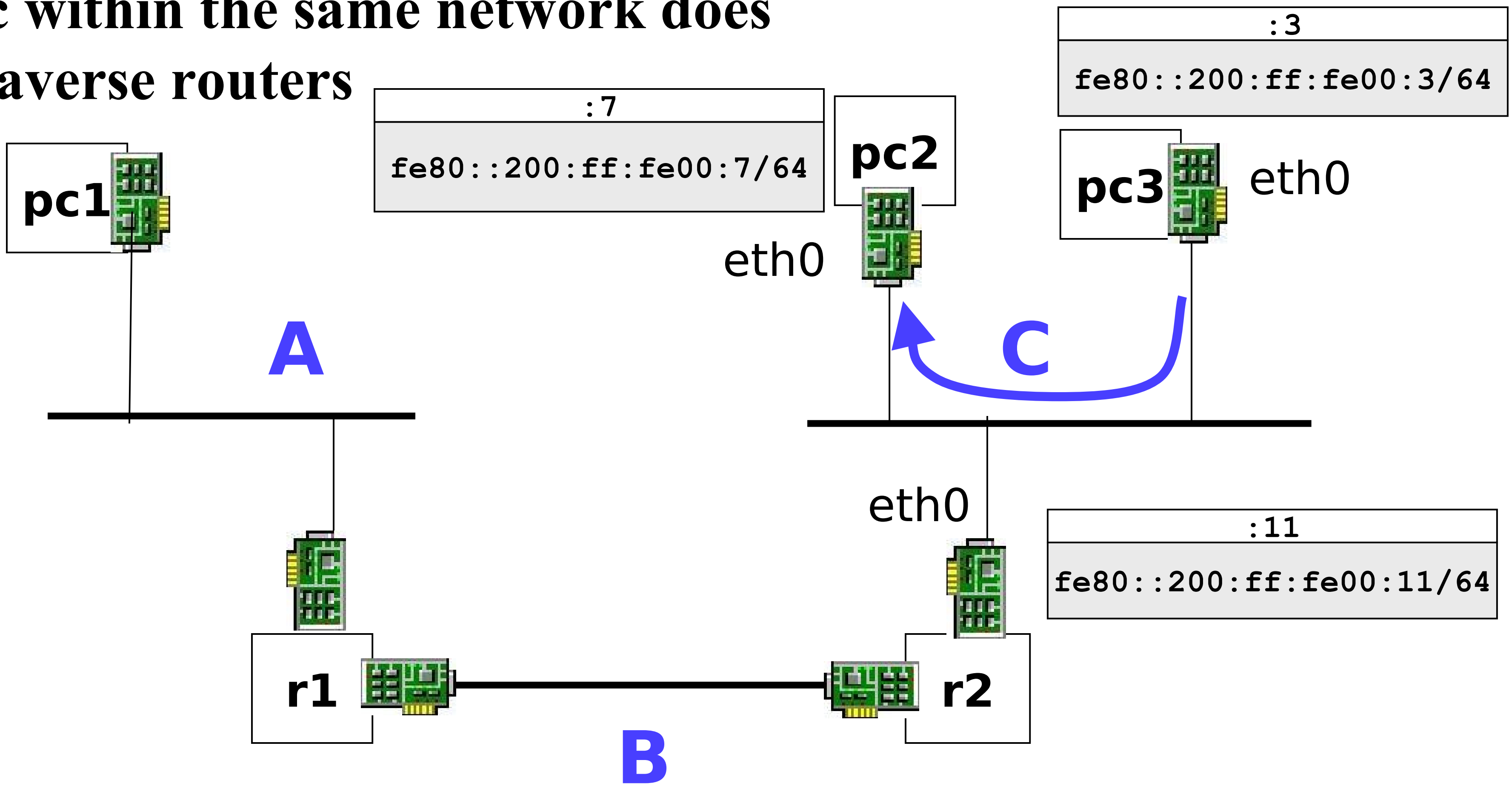
```
# ip -6 neigh add fec0::1 lladdr 02:01:02:03:04:05 dev eth0
```

step 3 – inspecting neighbors (continue)

```
Like adding also an entry can be deleted:

# ip -6 neigh del <IPv6 address> lladdr <link-layer address> dev <device>

Example:

# ip -6 neigh del fec0::1 lladdr 02:01:02:03:04:05 dev eth0

The tool “ip” is less documentated, but very strong. See online “help” for
more:

# ip -6 neigh help
Usage: ip neigh { add | del | change | replace } { ADDR [ lladdr LLADDR ]
          [ nud { permanent | noarp | stale | reachable } ]
          | proxy ADDR } [ dev DEV ]
       ip neigh {show|flush} [ to PREFIX ] [ dev DEV ] [ nud STATE ]
```

step 3 – inspecting neighbors (local traffic)

traffic within the same network does not traverse routers



step 3 – inspecting neighbors (local traffic)

the neighbor cache is initially empty

sending packets to **fe80::200:ff:fe00:7**r equires address resolution

pc3

```
pc3:~# ip -6 neigh show
pc3:~# ping6 -c 1 fe80::200:ff:fe00:7%eth0
PING fe80::200:ff:fe00:7%eth0(fe80::200:ff:fe00:7) 56 data bytes
64 bytes from fe80::200:ff:fe00:7: icmp_seq=1 ttl=64 time=4.04 ms

--- fe80::200:ff:fe00:7%eth0 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 4.041/4.041/4.041/0.000 ms
pc3:~# ip -6 neigh show
fe80::200:ff:fe00:7 dev eth0 lladdr 00:00:00:00:00:07 STALE
pc3:~# █
```

address resolution results are stored in the neighbor cache

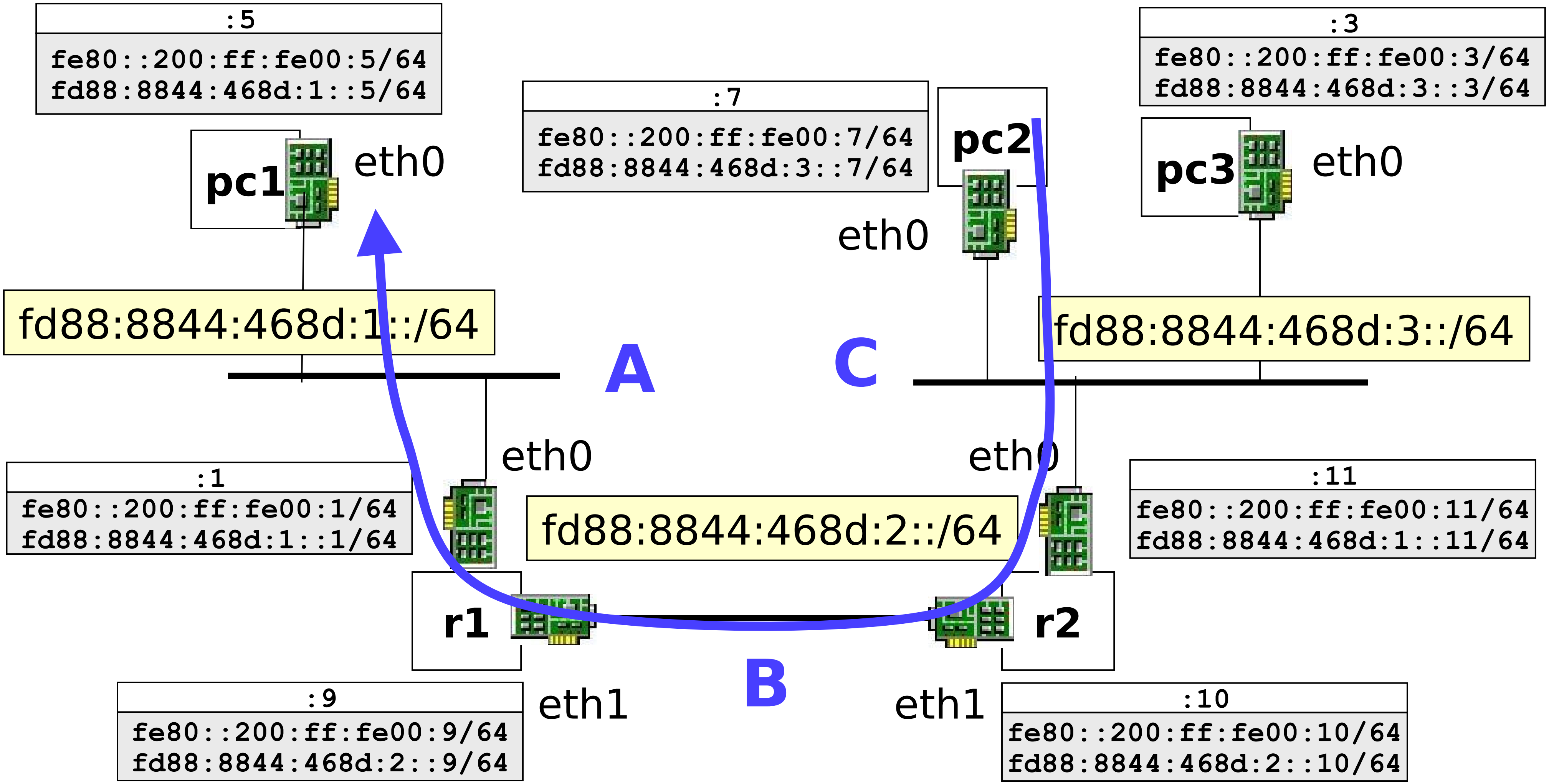
step 3 – inspecting the neighbors cache (local traffic)

- communications are usually bi-directional
- pc3 send multicast to ff02::1:ff00:7 neighbor solicitation "**who has fe80::200:ff:fe00:7 ?**"
- pc2 (the receiver) sends to fe80::200:ff:fe00:3 ICMP6, neighbor advertisement "**is fe80::200:ff:fe00:7**". It learns the mac address of the other party, to avoid a new request in opposite direction (standard behavior, see rfc 5942)

pc2

```
pc2:~# ip -6 neigh show
fe80::200:ff:fe00:3 dev eth0 lladdr 00:00:00:00:00:03 REACHABLE
pc2:~#
```

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



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

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

pc2

```
pc2:~# ping6 -c 1 fd88:8844:468d:1::5
PING fd88:8844:468d:1::5(fd88:8844:468d:1::5) 56 data bytes
64 bytes from fd88:8844:468d:1::5: icmp_seq=1 ttl=62 time=45.9 ms

--- fd88:8844:468d:1::5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 45.906/45.906/45.906/0.000 ms
pc2:~# ip -6 neig show
fd88:8844:468d:3::11 dev eth0 lladdr 00:00:00:00:00:11 router REACHABLE
fe80::200:ff:fe00:11 dev eth0 lladdr 00:00:00:00:00:11 router REACHABLE
pc2:~#
```

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

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

r1

```
r1:~# ip -6 neigh show
fe80::200:ff:fe00:10 dev eth1 lladdr 00:00:00:00:00:10 router REACHABLE
fd88:8844:468d:2::10 dev eth1 lladdr 00:00:00:00:00:10 router REACHABLE
fd88:8844:468d:1::5 dev eth0 lladdr 00:00:00:00:00:05 REACHABLE
fe80::200:ff:fe00:5 dev eth0 lladdr 00:00:00:00:00:05 REACHABLE
r1:~#
```

r2 (eth1)

pc1

r2

```
r2:~# ip -6 neigh show
fd88:8844:468d:2::9 dev eth1 lladdr 00:00:00:00:00:09 router REACHABLE
fe80::200:ff:fe00:9 dev eth1 lladdr 00:00:00:00:00:09 router REACHABLE
fd88:8844:468d:3::7 dev eth0 lladdr 00:00:00:00:00:07 REACHABLE
fe80::200:ff:fe00:7 dev eth0 lladdr 00:00:00:00:00:07 REACHABLE
r2:~#
```

pc2

r1 (eth1)

step5 – sniffing arp traffic

- restart the lab in order to clear neighbor caches

host machine

```
user@localhost:~/netkit-lab_ndp$ lcrash
user@localhost:~/netkit-lab_ndp$ lstart
```

- get ready to sniff

 =

virtual machine

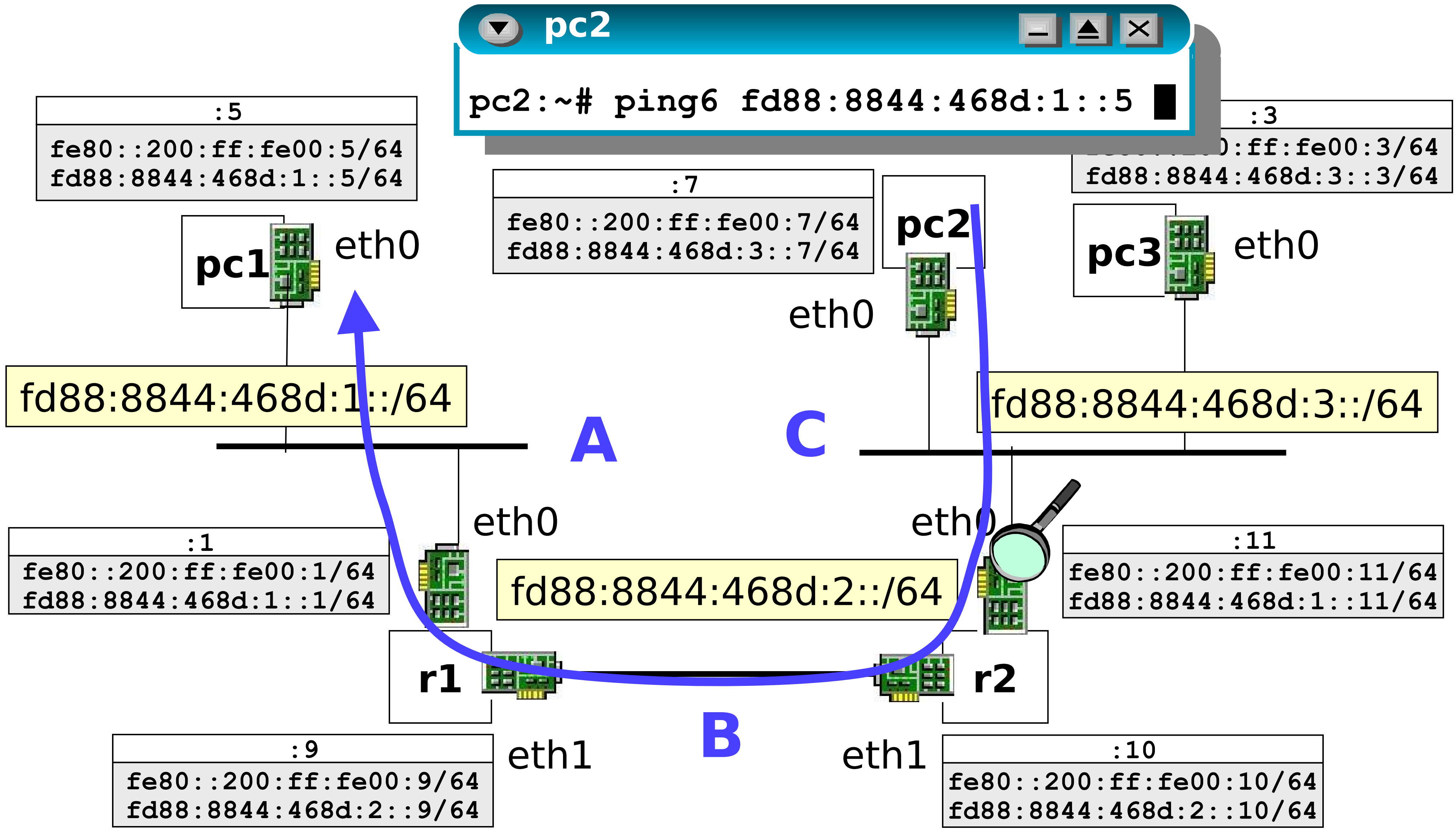
```
vm:~# tcpdump -e -t -i eth0
```

show link-level headers (mac addresses)

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

sniff on this interface

step 5 – sniffing neighbor traffic



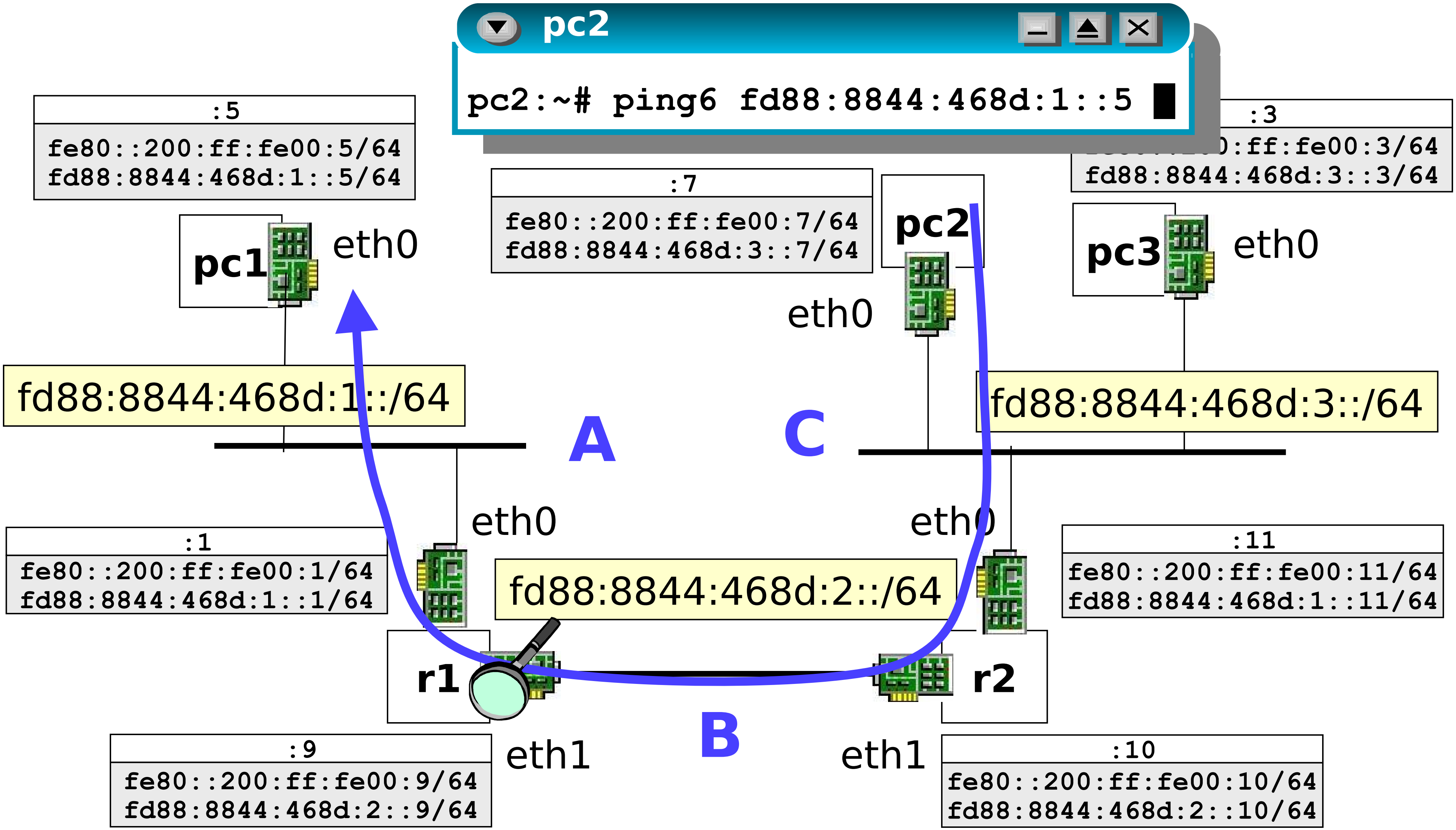
step 5 – sniffing ND traffic

r2

```
r2:~# tcpdump -t -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
IP6 fd88:8844:468d:3::7 > ff02::1:ff00:11: ICMP6, neighbor solicitation, who has
fd88:8844:468d:3::11, length 32
IP6 fd88:8844:468d:3::11 > fd88:8844:468d:3::7: ICMP6, neighbor advertisement, tgt is
fd88:8844:468d:3::11, length 32
IP6 fd88:8844:468d:3::7 > fd88:8844:468d:1::5: ICMP6, echo request, seq 1, length 64
IP6 fd88:8844:468d:1::5 > fd88:8844:468d:3::7: ICMP6, echo reply, seq 1, length 64
█
```

1. **pc2** asks all the stations on collision domain C: “who has fd88:8844:468d:3::11?” (fd88:8844:468d:3::11 is **pc2**’s default gateway)
2. **r2** replies ⇒ both **pc2** and **r2** update their neighbor 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 neighbor traffic



step 6 – sniffing ND traffic

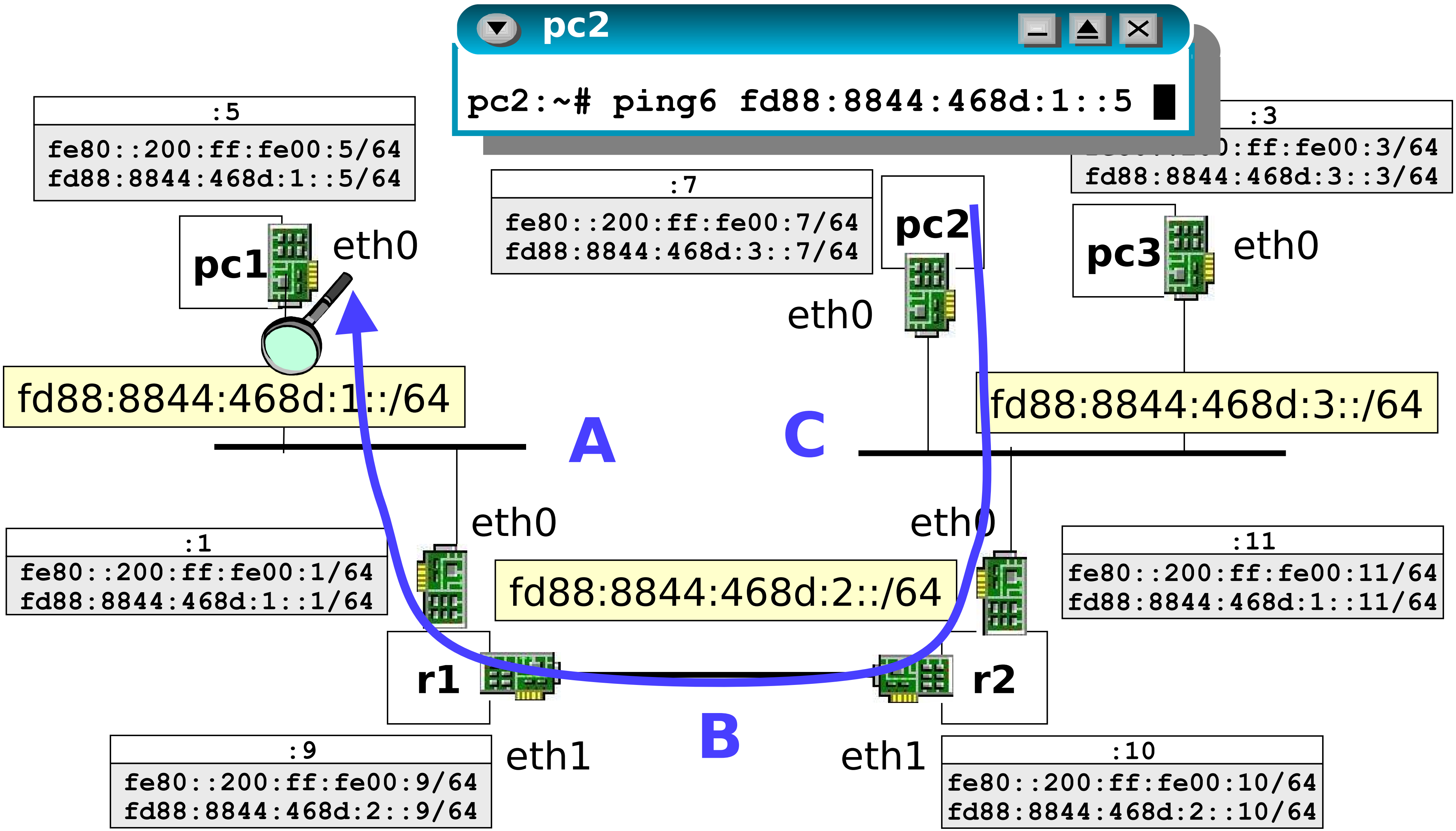
- on collision domain B

r1

```
r1:~# tcpdump -t -i eth1
IP6 fe80::200:ff:fe00:10 > ff02::1:ff00:9: ICMP6, neighbor solicitation, who has
fd88:8844:468d:2::9, length 32
IP6 fd88:8844:468d:2::9 > fe80::200:ff:fe00:10: ICMP6, neighbor advertisement, tgt is
fd88:8844:468d:2::9, length 32
IP6 fd88:8844:468d:3::7 > fd88:8844:468d:1::5: ICMP6, echo request, seq 1, length 64
IP6 fe80::200:ff:fe00:9 > ff02::1:ff00:10: ICMP6, neighbor solicitation, who has
fd88:8844:468d:2::10, length 32
IP6 fd88:8844:468d:2::10 > fe80::200:ff:fe00:9: ICMP6, neighbor advertisement, tgt is
fd88:8844:468d:2::10, length 32
IP6 fd88:8844:468d:1::5 > fd88:8844:468d:3::7: ICMP6, echo reply, seq 1, length 64
```

1. **r2** asks all the stations on collision domain B: “who has fd88:8844:468d:2::9?” (fd88:8844:468d:2::9 is the next hop obtained from the routing table)
2. **r1** replies ⇒ both **r1** and **r2** update their neighbor 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 neighbor traffic



step 5 – sniffing ND traffic

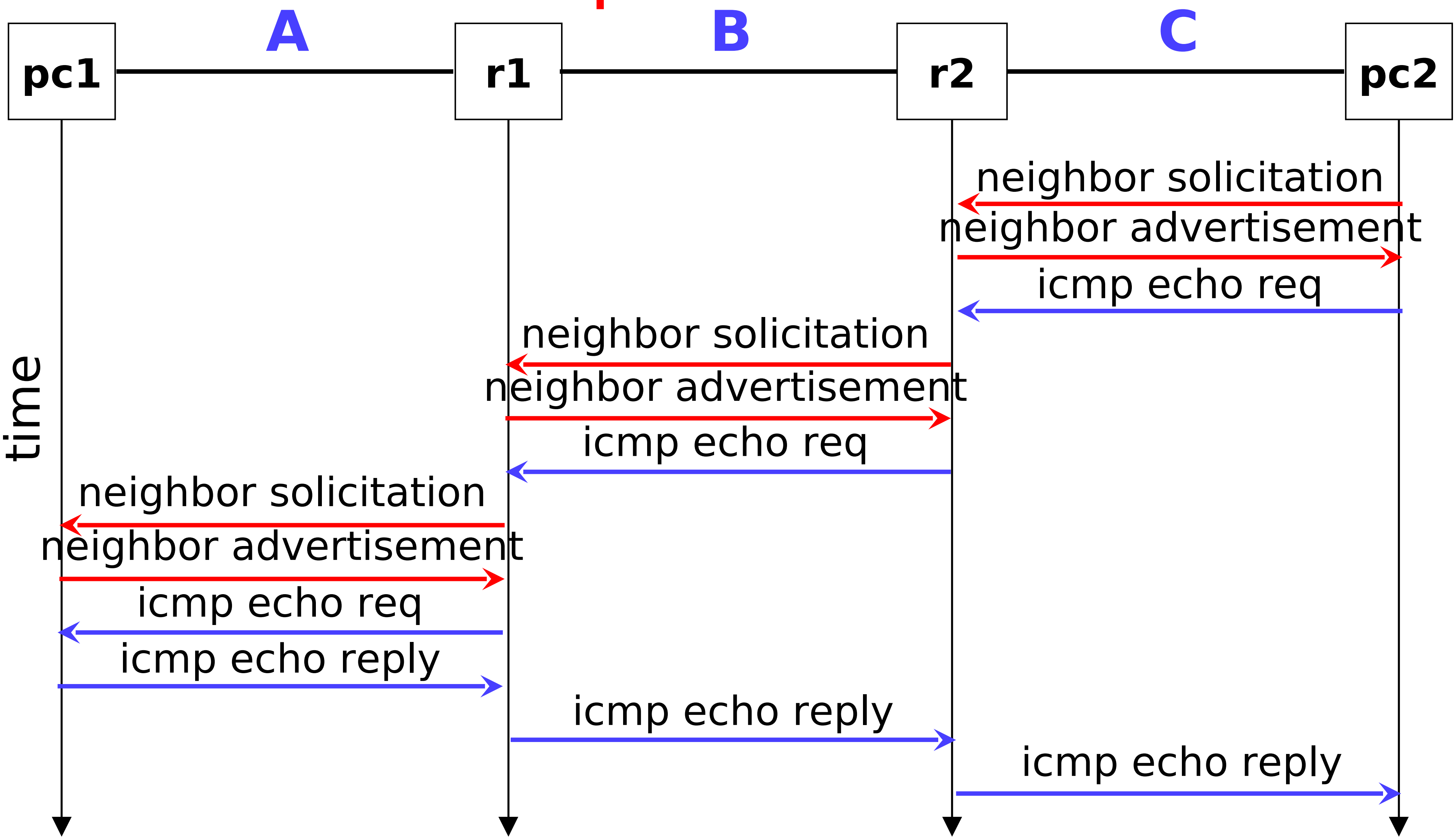
- on collision domain A

pc1

```
pc1:~# tcpdump -t -i eth0
IP6 fe80::200:ff:fe00:1 > ff02::1:ff00:5: ICMP6, neighbor solicitation, who has
fd88:8844:468d:1::5, length 32
IP6 fd88:8844:468d:1::5 > fe80::200:ff:fe00:1: ICMP6, neighbor advertisement, tgt is
fd88:8844:468d:1::5, length 32
IP6 fd88:8844:468d:3::7 > fd88:8844:468d:1::5: ICMP6, echo request, seq 1, length 64
IP6 fd88:8844:468d:1::5 > ff02::1:ff00:1: ICMP6, neighbor solicitation, who has
fd88:8844:468d:1::1, length 32
IP6 fd88:8844:468d:1::1 > fd88:8844:468d:1::5: ICMP6, neighbor advertisement, tgt is
fd88:8844:468d:1::1, length 32
IP6 fd88:8844:468d:1::5 > fd88:8844:468d:3::7: ICMP6, echo reply, seq 1, length 64
```

1. **r1** asks all the stations on collision domain A: “who has fd88:8844:468d:1::5?” (fd88:8844:468d:1::5 the destination address of the icmp request obtained from the ip header)
2. **pc1** replies ⇒ both **pc1** and **r1** update their neighbor 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 – neighbor implementation details

r2

```
r2:~# tcpdump -t -i eth0
IP6 fd88:8844:468d:3::7 > ff02::1:ff00:11: ICMP6, neighbor solicitation, who has
fd88:8844:468d:3::11, length 32
IP6 fd88:8844:468d:3::11 > fd88:8844:468d:3::7: ICMP6, neighbor advertisement, tgt is
fd88:8844:468d:3::11, length 32
IP6 fd88:8844:468d:3::7 > fd88:8844:468d:1::5: ICMP6, echo request, seq 1, length 64
IP6 fd88:8844:468d:1::5 > fd88:8844:468d:3::7: ICMP6, echo reply, seq 1, length 64
```

- neighbor solicitation (type 135) requests use a multicast address with the prefix ff02:0:0:0:0:1:ff00:0000/104 concatenated with the 24 low-order bits of a corresponding IPv6 unicast address (ff02::1:ff00:11)
- A node may also send unsolicited Neighbor Advertisements to announce a link-layer address change
- 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
- 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 in case of Duplicate Address Detection (DAD) ?

proposed exercises

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