IPv6 Neighbor Discovery (NDP)

<table>
<thead>
<tr>
<th>Version</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>S. Doro based on work “ARP” by G. Di Battista, M. Patrignani, M. Pizzonia, F. Ricci, M. Rimondini</td>
</tr>
<tr>
<td>E-mail</td>
<td><a href="mailto:sandro.doro@gmail.com">sandro.doro@gmail.com</a></td>
</tr>
<tr>
<td>Web</td>
<td><a href="http://netkit.zuccante.it/">http://netkit.zuccante.it/</a></td>
</tr>
<tr>
<td>Description</td>
<td>using the ICMPv6 Neighbor Discovery and ICMPv6 Neighbor Solicitation</td>
</tr>
</tbody>
</table>
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.
step 1 - network topology
scope link details

A

pc1
eth0

r1
eth0

r2
eth1

B

pc2
eth0

pc3
eth0

C

fe80::200:ff:fe00:1/64

fe80::200:ff:fe00:5/64

fe80::200:ff:fe00:7/64

fe80::200:ff:fe00:9/64

fe80::200:ff:fe00:11/64

fe80::200:ff:fe00:3/64

fe80::200:ff:fe00:10/64
step 1 – network topology
site link details (rfc 4193)
step 1 – network topology

all link details

<table>
<thead>
<tr>
<th>Link Details</th>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>pc1-eth0</td>
<td>:5</td>
<td>:1</td>
</tr>
<tr>
<td>pc2-eth0</td>
<td>:7</td>
<td>:9</td>
</tr>
<tr>
<td>pc3-eth0</td>
<td>:11</td>
<td>:10</td>
</tr>
<tr>
<td>r1-eth1</td>
<td>:9</td>
<td>:1</td>
</tr>
<tr>
<td>r2-eth1</td>
<td>:7</td>
<td>:10</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>fd88:8844:468d:1::/64</td>
<td>fd88:8844:468d:2::/64</td>
<td>fd88:8844:468d:3::/64</td>
</tr>
<tr>
<td>fd88:8844:468d:1::1/64</td>
<td>fd88:8844:468d:2::9/64</td>
<td>fd88:8844:468d:3::3/64</td>
</tr>
<tr>
<td>fd88:8844:468d:1::1/64</td>
<td>fd88:8844:468d:2::9/64</td>
<td>fd88:8844:468d:3::3/64</td>
</tr>
<tr>
<td>:5</td>
<td>:7</td>
<td>:11</td>
</tr>
<tr>
<td>fe80::200:ff:fe00:5/64</td>
<td>fe80::200:ff:fe00:7/64</td>
<td>fe80::200:ff:fe00:11/64</td>
</tr>
<tr>
<td>fd88:8844:468d:1::/64</td>
<td>fd88:8844:468d:2::/64</td>
<td>fd88:8844:468d:3::/64</td>
</tr>
</tbody>
</table>
step 2 – a quick look at the lab

lab.conf

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

```bash
# ----------------------------------------------- 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

```
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
```
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 documented, 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

```
c3:~# 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
```

sending packets to fe80::200:ff:fe00:7 requires address resolution

address resolution results are stored in the neighbor cache
step 3 – inspecting the neighbors cache (local traffic)

- Communications are usually bi-directional.
- PC3 sends 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)

```
```
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:~# 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
```
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
step5 – sniffing arp traffic

- restart the lab in order to clear neighbor caches

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

- get ready to sniff

```
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

A: pc1
B: r1
C: pc2

pc2:~# ping6 fd88:8844:468d:1::5
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**)

```
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
```
step 5 – sniffing neighbor traffic

pc2:~# ping6 fd88:8844:468d:1::5
step 6 – sniffing ND traffic

- on collision domain B

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

pc2:~# ping6 fd88:8844:468d:1::5

pc1

eth0

fd88:8844:468d:1::/64

r1

eth1

fd88:8844:468d:1::1/64

fe80::200:ff:fe00:1/64

fe80::200:ff:fe00:9/64

fe80::200:ff:fe00:10/64

pc2

eth0

fd88:8844:468d:1::5/64

fd88:8844:468d:3::5/64

fd88:8844:468d:2::5/64

pc3

eth0

fd88:8844:468d:3::/64

fe80::200:ff:fe00:7/64

fe80::200:ff:fe00:11/64

fe80::200:ff:fe00:10/64

r2

eth1

fd88:8844:468d:2::/64

fe80::200:ff:fe00:1/64

fe80::200:ff:fe00:3/64

fe80::200:ff:fe00:12/64

A

B

C
step 5 – sniffing ND traffic

on collision domain A

1. \textbf{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. \textbf{pc1} replies ⇒ both \textbf{pc1} and \textbf{r1} update their neighbor cache

3. \textbf{r1} sends the ip packet (echo request) to \textbf{pc1}

4. \textbf{pc1} generates the corresponding echo reply for \textbf{pc2} and sends it to \textbf{r1}
step 6 – understanding the whole picture

pc1 ---- A ---- r1 ---- B ---- r2 ---- C ---- pc2

neighbor solicitation
neighbor advertisement
icmp echo req
icmp echo reply

neighbor solicitation
neighbor advertisement
icmp echo req
icmp echo reply

icmp echo reply
icmp echo reply
step 7 – neighbor implementation details

- neighbor solicitation (type 135) requests use a multicast address with the prefix ff02::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

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
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?