Facultade de Informática
UNIVERSIDADE DA CORUÑA

DISEÑO DE REDES

Práctica 1 - DiffServ en INET

Estudante: Tomé Maseda Dorado
Estudante: Jorge Álvarez Cabado

A Coruña, outubro de 2021.

Índice Xeral

Índice de Figuras

Introducción

En esta práctica se estudiará en profundidad el funcionamiento de distintos protocolos IPv6, en concreto, nos centraremos en el protocolo Neighbour Discovery. Para aclarar algunas referencias a los dispositivos de la red dejamos un diccionario con las correspondencias entre las palabras usadas en las explicaciones y los dispositivos de red a los que se refiere:

- Cliente: *client*
- Servidor Telnet: *srvTelnet*
- Servidor HTTP: *srvHTTP*
- Router 0 y Router 1: *r0* y *r1*

Capítulo 1

Ficheros utilizados en la simulación

1.1 Fichero omnetpp.ini

```
#
# This ini file runs Telnet sessions on the NClients network, using
# TelnetApp+TcpGenericServerApp.
#
# See also fileTransfer.ini and basicHTTP.ini for different kinds of
# network traffic.
#
[General]
network = P2
sim-time-limit = 1800s
**.interfaceTable.displayAddresses = true

#Telnet
**.client.numApps = 2
**.client.app[0].typename = "TelnetApp"
**.client.app[0].localPort = -1
**.client.app[0].connectPort = 23
**.client.app[0].connectAddress = "srvTelnet"
**.client.app[0].startTime = 6s

#comandos en una sesion 148,41
**.client.app[0].numCommands = int(exponential(5))
#longitud de los comandos 15B de media
**.client.app[0].commandLength = intWithUnit(truncnormal(15B, 5B))
#salida que producen los comandos 100B de media
**.client.app[0].commandOutputLength =
    intWithUnit(truncnormal(100B, 20B))
**.client.app[0].thinkTime = 5s
**.client.app[0].idleInterval = 3s

```

```
**.srvTelnet.numApps = 1
**.srvTelnet.app[*].typename = "TcpGenericServerApp"
#**.srvTelnet.app[0].localAddress = ""
**.srvTelnet.app[0].localPort = 23
#**.srvTelnet.app[0].replyDelay = 0s

#Http
**.client.app[1].typename = "TcpBasicClientApp"
**.client.app[1].localPort = -1
**.client.app[1].connectPort = 80
**.client.app[1].connectAddress = "srvHTTP"
**.client.app[1].startTime = 10s
**.client.app[1].thinkTime = 5s
**.client.app[1].idleInterval = 3s

#longitud de la peticion 300B de media
**.client.app[1].requestLength = intWithUnit(truncnormal(300B, 20B))
#longitud de la respuesta 20KiB de media
**.client.app[1].replyLength = intWithUnit(truncnormal(20KiB, 2KiB))

**.srvHTTP.numApps = 1
**.srvHTTP.app[*].typename = "TcpGenericServerApp"
#**.srvHttp.app[0].localAddress = ""
**.srvHTTP.app[0].localPort = 80
#**.srvHttp.app[0].replyDelay = 0s
```

1.2 Fichero P2.ned

```
import
    inet.networklayer.configurator.ipv6.Ipv6FlatNetworkConfigurator;
import inet.node.ipv6.Router6;
import inet.node.ipv6.StandardHost6;
import inet.node.ethernet.EtherSwitch;
import ned.DatarateChannel;


network P2
{
    parameters:

    types:
        channel fiberline extends DatarateChannel
        {
            delay = 1us;
```

```
16              datarate = 512Mbps;
17          }
18          channel ethernetline extends DatarateChannel
19          {
20              delay = 0.1us;
21              datarate = 10Mbps;
22          }
23      submodules:
24          configurator: Ipv6FlatNetworkConfigurator {
25              @display("p=722,43");
26          }
27          r0: Router6 {
28              @display("p=407,136");
29          }
30          r1: Router6 {
31              @display("p=569,141");
32          }
33          internet: Router6 {
34              @display("p=495,65;i=misc/cloud");
35          }
36          sw0: EtherSwitch {
37              @display("p=327,190");
38          }
39          sw1: EtherSwitch {
40              @display("p=621,217");
41          }
42          client: StandardHost6 {
43              @display("p=226,295");
44          }
45          srvTelnet: StandardHost6 {
46              @display("p=362,306;i=device/server");
47          }
48          srvHTTP: StandardHost6 {
49              @display("p=687,302;i=device/server");
50          }
51      connections:
52          client.ethg++ <--> ethernetline <--> sw0.ethg++;
53          srvTelnet.ethg++ <--> ethernetline <--> sw0.ethg++;
54          sw0.ethg++ <--> ethernetline <--> r0.ethg++;
55          r0.ethg++ <--> ethernetline <--> internet.ethg++;
56
57          srvHTTP.ethg++ <--> ethernetline <--> sw1.ethg++;
58          sw1.ethg++ <--> ethernetline <--> r1.ethg++;
59          r1.ethg++ <--> ethernetline <--> internet.ethg++;
60
61  }
```

Capítulo 2

Preguntas acerca de la simulación

2.1 Pregunta 1

Durante los 2 primeros segundos se envían paquetes NS. ¿Cuál es su objetivo? Muestre una captura del log que muestren el motivo del envío, filtrando por nivel ipv6.neighbourDiscovery en uno de los nodos.

Los paquetes NS (Neighbor Solicitation) son paquetes del protocolo Neighbor Discovery, este protocolo tiene múltiples funciones como localizar a los routers en un segmento de red, descubrimiento de prefijos de red, descubrimiento de parámetros de red como el MTU…

En este caso los paquetes NS se utilizan para comprobar si una dirección IPv6 está ocupada.
¿Cómo lo comprueban? Cuando un nodo envía un paquete NS en un segmento de red, si alguno de sus vecinos en el segmento de red está usando la dirección IPv6 indicada en el paquete NS, responderá a este paquete, para descubrir si una dirección IPv6 está ocupada, el nodo simplemente enviará paquetes NS en su segmento de red y si nadie responde, significa que esa dirección no está ocupada, este protocolo es el que se conoce como Duplicate Address Detection (DAD) y se utiliza en la autoconfiguración de direcciones de IPv6.

En concreto, aquí se filtraron los paquetes para ver solo los del cliente y entender mejor el proceso, este envía un paquete NS sin dirección de origen, porque aún no tiene, para descubrir si la dirección que quiere tener asignada está ocupada (DAD), una vez recibe un *dadTimeout* (nadie responde) y sabe que esa dirección no se está usando, se asigna en la interfaz correspondiente (eth0) la dirección deseada.

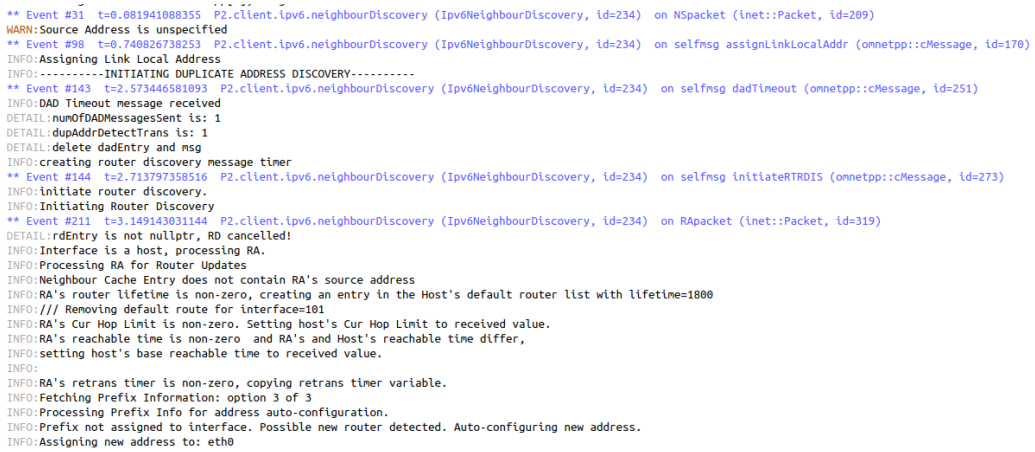


Figura 2.1: Log filtrando por nivel ipv6.neighbourDiscovery en el cliente

2.2 Pregunta 2

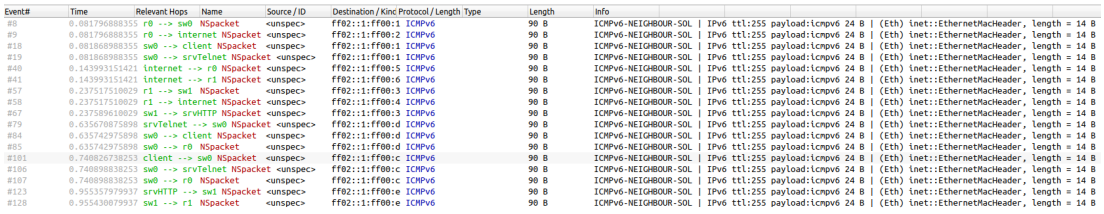
Elija uno de esos paquetes NS. Muestre una captura del tráfico de paquetes en la que se vean las IPs origen y destino del paquete. ¿Cómo se construye la IP destino? Explique el motivo de cada una de ellas..

Se ha escogido el paquete NS del cliente como en la pregunta 1, aunque se muestran también los paquetes NS del resto de nodos. El paquete tiene como dirección de origen *<unspec>*, esto significa que no se especifica una dirección de origen, debido a que el cliente aún no tiene una dirección asignada, de hecho, precisamente para eso está mandando un paquete NS, para saber si una dirección concreta está libre y solicitar a un router que se la asigne.

El paquete tiene como dirección de destino ff02::1:ff00:c, esta dirección es una dirección multicast de nodo solicitado y se construye con los siguientes parámetros:

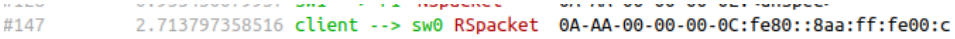
- Últimos 24 bits de la dirección link-local unicast de la interfaz correspondiente del nodo
- Prefijo multicast de nodo solicitado ff02::1:ff00::/104

Dirección de destino = Prefijo multicast de nodo solicitado + Últimos 24 bits de la dirección link-local de la interfaz eth0 del cliente (fe80::8aa:ff:fe**00:c**) = ff02::1:ff00::/104 + 00:000c = ff02::1:ff**00:c** = Dirección multicast de nodo solicitado de client en eth0



Event#	Time	Relevant Hops	Name	Source / ID	Destination / Kinc	Protocol / Length	Type	Length	Info
#8	0.081796888355	r0 --> sw0	NSpacket	<unspec>	ff02::1:ff00:1	ICMPv6		90 B	ICMPv6-NEIGHBOUR-SOL | IPv6 ttl:255 payload:icmpv6 24 B | (Eth) inet::EthernetMacHeader, length = 14 B
#9	0.081796888355	r0 --> internet	NSpacket	<unspec>	ff02::1:ff00:2	ICMPv6		90 B	ICMPv6-NEIGHBOUR-SOL | IPv6 ttl:255 payload:icmpv6 24 B | (Eth) inet::EthernetMacHeader, length = 14 B
#18	0.081868988355	sw0 --> client	NSpacket	<unspec>	ff02::1:ff00:1	ICMPv6		90 B	ICMPv6-NEIGHBOUR-SOL | IPv6 ttl:255 payload:icmpv6 24 B | (Eth) inet::EthernetMacHeader, length = 14 B
#19	0.081868988355	sw0 --> srvTelnet	NSpacket	<unspec>	ff02::1:ff00:1	ICMPv6		90 B	ICMPv6-NEIGHBOUR-SOL | IPv6 ttl:255 payload:icmpv6 24 B | (Eth) inet::EthernetMacHeader, length = 14 B
#40	0.143993151421	internet --> r0	NSpacket	<unspec>	ff02::1:ff00:5	ICMPv6		90 B	ICMPv6-NEIGHBOUR-SOL | IPv6 ttl:255 payload:icmpv6 24 B | (Eth) inet::EthernetMacHeader, length = 14 B
#41	0.143993151421	internet --> r1	NSpacket	<unspec>	ff02::1:ff00:6	ICMPv6		90 B	ICMPv6-NEIGHBOUR-SOL | IPv6 ttl:255 payload:icmpv6 24 B | (Eth) inet::EthernetMacHeader, length = 14 B
#57	0.237517510029	r1 --> sw1	NSpacket	<unspec>	ff02::1:ff00:3	ICMPv6		90 B	ICMPv6-NEIGHBOUR-SOL | IPv6 ttl:255 payload:icmpv6 24 B | (Eth) inet::EthernetMacHeader, length = 14 B
#58	0.237517510029	r1 --> internet	NSpacket	<unspec>	ff02::1:ff00:4	ICMPv6		90 B	ICMPv6-NEIGHBOUR-SOL | IPv6 ttl:255 payload:icmpv6 24 B | (Eth) inet::EthernetMacHeader, length = 14 B
#67	0.237589610029	sw1 --> srvHTTP	NSpacket	<unspec>	ff02::1:ff00:3	ICMPv6		90 B	ICMPv6-NEIGHBOUR-SOL | IPv6 ttl:255 payload:icmpv6 24 B | (Eth) inet::EthernetMacHeader, length = 14 B
#79	0.635670875898	srvTelnet --> sw0	NSpacket	<unspec>	ff02::1:ff00:d	ICMPv6		90 B	ICMPv6-NEIGHBOUR-SOL | IPv6 ttl:255 payload:icmpv6 24 B | (Eth) inet::EthernetMacHeader, length = 14 B
#84	0.635742975898	sw0 --> client	NSpacket	<unspec>	ff02::1:ff00:d	ICMPv6		90 B	ICMPv6-NEIGHBOUR-SOL | IPv6 ttl:255 payload:icmpv6 24 B | (Eth) inet::EthernetMacHeader, length = 14 B
#85	0.635742975898	sw0 --> r0	NSpacket	<unspec>	ff02::1:ff00:d	ICMPv6		90 B	ICMPv6-NEIGHBOUR-SOL | IPv6 ttl:255 payload:icmpv6 24 B | (Eth) inet::EthernetMacHeader, length = 14 B
#101	0.740826738253	client --> sw0	NSpacket	<unspec>	ff02::1:ff00:c	ICMPv6		90 B	ICMPv6-NEIGHBOUR-SOL | IPv6 ttl:255 payload:icmpv6 24 B | (Eth) inet::EthernetMacHeader, length = 14 B
#106	0.740898838253	sw0 --> srvTelnet	NSpacket	<unspec>	ff02::1:ff00:c	ICMPv6		90 B	ICMPv6-NEIGHBOUR-SOL | IPv6 ttl:255 payload:icmpv6 24 B | (Eth) inet::EthernetMacHeader, length = 14 B
#107	0.740898838253	sw0 --> r0	NSpacket	<unspec>	ff02::1:ff00:c	ICMPv6		90 B	ICMPv6-NEIGHBOUR-SOL | IPv6 ttl:255 payload:icmpv6 24 B | (Eth) inet::EthernetMacHeader, length = 14 B
#123	0.955357979937	srvHTTP --> sw1	NSpacket	<unspec>	ff02::1:ff00:e	ICMPv6		90 B	ICMPv6-NEIGHBOUR-SOL | IPv6 ttl:255 payload:icmpv6 24 B | (Eth) inet::EthernetMacHeader, length = 14 B
#128	0.955430079937	sw1 --> r1	NSpacket	<unspec>	ff02::1:ff00:e	ICMPv6		90 B	ICMPv6-NEIGHBOUR-SOL | IPv6 ttl:255 payload:icmpv6 24 B | (Eth) inet::EthernetMacHeader, length = 14 B

Figura 2.2: Paquetes NS enviados en la red



```
#147          2.713797358516 client --> sw0 RSpacket  0A-AA-00-00-00-0C:fe80::8aa:ff:fe00:c
```

Figura 2.3: Dirección link-local unicast del cliente

2.3 Pregunta 3

¿Qué dirección MAC destino tiene el paquete elegido? ¿Qué dirección debería tener (calcule los 6 bytes) según lo visto en teoría? (Nota: para ver direcciones MAC en la ventana de tráfico de paquetes marque "Show all PDU destination fields" y "Show all PDU source fields" en las opciones de la ventana).

Tanto en el paquete escogido como en el resto de paquetes vemos que se usa la dirección MAC de destino FF-FF-FF-FF-FF-FF que es la dirección MAC de broadcast. Según lo visto en teoría, la dirección de destino del paquete se debería usar una dirección Ethernet multicast, que se construye de la siguiente forma:

Dirección Ethernet multicast = Prefijo dirección Ethernet multicast + Últimos 32 bit de la dirección multicast de nodo solicitado del destino (ff02::1:textbfff00:c) = 33-33 + FF-00-00-0C = **33-33-FF-00-00-0C**

Event#	Time	Relevant Hops	Name	Source / ID	Destination / Kind
#8	0.081796888355	r0 --> sw0	NSpacket	0A-AA-00-00-00-01:<unspec>	FF-FF-FF-FF-FF-FF:ff02::1:ff00:1
#9	0.081796888355	r0 --> internet	NSpacket	0A-AA-00-00-00-02:<unspec>	FF-FF-FF-FF-FF-FF:ff02::1:ff00:2
#18	0.081868988355	sw0 --> client	NSpacket	0A-AA-00-00-00-01:<unspec>	FF-FF-FF-FF-FF-FF:ff02::1:ff00:1
#19	0.081868988355	sw0 --> srvTelnet	NSpacket	0A-AA-00-00-00-01:<unspec>	FF-FF-FF-FF-FF-FF:ff02::1:ff00:1
#40	0.143993151421	internet --> r0	NSpacket	0A-AA-00-00-00-05:<unspec>	FF-FF-FF-FF-FF-FF:ff02::1:ff00:5
#41	0.143993151421	internet --> r1	NSpacket	0A-AA-00-00-00-06:<unspec>	FF-FF-FF-FF-FF-FF:ff02::1:ff00:6
#57	0.237517510029	r1 --> sw1	NSpacket	0A-AA-00-00-00-03:<unspec>	FF-FF-FF-FF-FF-FF:ff02::1:ff00:3
#58	0.237517510029	r1 --> internet	NSpacket	0A-AA-00-00-00-04:<unspec>	FF-FF-FF-FF-FF-FF:ff02::1:ff00:4
#67	0.237589610029	sw1 --> srvHTTP	NSpacket	0A-AA-00-00-00-03:<unspec>	FF-FF-FF-FF-FF-FF:ff02::1:ff00:3
#79	0.635670875898	srvTelnet --> sw0	NSpacket	0A-AA-00-00-00-0D:<unspec>	FF-FF-FF-FF-FF-FF:ff02::1:ff00:d
#84	0.635742975898	sw0 --> client	NSpacket	0A-AA-00-00-00-0D:<unspec>	FF-FF-FF-FF-FF-FF:ff02::1:ff00:d
#85	0.635742975898	sw0 --> r0	NSpacket	0A-AA-00-00-00-0D:<unspec>	FF-FF-FF-FF-FF-FF:ff02::1:ff00:d
#101	0.740826738253	client --> sw0	NSpacket	0A-AA-00-00-00-0C:<unspec>	FF-FF-FF-FF-FF-FF:ff02::1:ff00:c
#106	0.740898838253	sw0 --> srvTelnet	NSpacket	0A-AA-00-00-00-0C:<unspec>	FF-FF-FF-FF-FF-FF:ff02::1:ff00:c
#107	0.740898838253	sw0 --> r0	NSpacket	0A-AA-00-00-00-0C:<unspec>	FF-FF-FF-FF-FF-FF:ff02::1:ff00:c
#123	0.955357979937	srvHTTP --> sw1	NSpacket	0A-AA-00-00-00-0E:<unspec>	FF-FF-FF-FF-FF-FF:ff02::1:ff00:e
#128	0.955430079937	sw1 --> r1	NSpacket	0A-AA-00-00-00-0E:<unspec>	FF-FF-FF-FF-FF-FF:ff02::1:ff00:e
#147	2.713797358516	client --> sw0	RSpacket	0A-AA-00-00-00-0C:fe80::8aa:ff:fe00:c	FF-FF-FF-FF-FF-FF:ff02::2

Figura 2.4: Direcciones MAC de origen y destino de los paquetes NS

2.4 Pregunta 4

¿Por qué no recibe respuesta ninguno de esos paquete NS? Observe la tabla de interfaces del equipo cliente. Muestre el contenido de la tabla antes y después del timeout de espera de respuesta y explique qué cambia. (Nota: para ver correctamente toda la información de cada interfaz de red copie el contenido con botón derecho → Copy Value y péguelo en la memoria, en lugar de usar capturas de pantalla)

No se recibe respuesta a ninguno de los paquetes NS porque se usan para detectar direcciones duplicadas, el equipo envía el paquete con dirección de destino la que el mismo quiere asignarse, si la dirección no se está usando (este mismo caso), el equipo no recibirá ninguna respuesta, simplemente esperará a que pase el timeout de espera de respuesta y si no ha recibido ningún paquete se asignará la dirección deseada.

Tabla de interfaces del cliente (P2.client.interfaceTable.idToInterface.elements[])

Antes del *dadTimeout*:

```
lo0 ID:100 MTU:4470 UP LOOPBACK CARRIER macAddr:n/a Ipv6:{
  Addrs:::1(loopback)  expiryTime: inf prefExpiryTime: inf
  Node: dupAddrDetectTrans=1 reachableTime=33.706910891924
   }

eth0 ID:101 MTU:1500 UP BROADCAST CARRIER MULTICAST
    macAddr:0A-AA-00-00-00-0C Ipv6:{
  Addrs:fe80::8aa:ff:fe00:c(link tent)  expiryTime: inf
    prefExpiryTime: inf
  mcastgrps:ff02::1   Node: dupAddrDetectTrans=1
    reachableTime=34.376823452767
   }
```

Después del *dadTimeout*:

```
lo0 ID:100 MTU:4470 UP LOOPBACK CARRIER macAddr:n/a Ipv6:{
  Addrs:::1(loopback)  expiryTime: inf prefExpiryTime: inf
  Node: dupAddrDetectTrans=1 reachableTime=33.706910891924
   }

eth0 ID:101 MTU:1500 UP BROADCAST CARRIER MULTICAST
    macAddr:0A-AA-00-00-00-0C Ipv6:{
  Addrs:fe80::8aa:ff:fe00:c(link)  expiryTime: inf prefExpiryTime:
    inf
  mcastgrps:ff02::1   Node: dupAddrDetectTrans=1
    reachableTime=34.376823452767
```

```
9     }
```

Después del timeout cambia la etiqueta asignada a dirección link-local de la interfaz eth0 del cliente, esta pasa de *link tent* a *link*, *link tent* quiere decir que la dirección aún no ha sido asignada a la interfaz pero que se va a intentar, una vez pasado el timeout, el cliente ya sabe que esa dirección no está siendo usada (porque nadie respondió) y la dirección pasa a etiquetarse como *link*, lo que quiere decir que ya ha sido asignada.

2.5 Pregunta 5

¿Cuándo se envía el primer paquete RA? ¿Se envía como respuesta a algún paquete? Muestre capturas del contenido del paquete RA y explique los campos que considere más relevantes. (Nota: para ver las diferentes partes del paquete expanda encapsulatedPacket ⇒ content ⇒ chunks. El contenido relevante en este caso estará en Ipv6RouterAdvertisement. Expanda todas las options para ver el contenido completo).

El primer paquete RA se envía en t=3.149143031144s desde el router r0 al cliente y se envía como respuesta al paquete RS envíado previamente por el cliente (t=2.713928758516s). El paquete RS (Router Solicitation) es un paquete enviado por un equipo al router para solicitar la asignación de una dirección IPv6, por otra parte, el paquete RA (Router Advertisement) es la respuesta del router hacia dicho equipo cuando le llega el paquete RS, este paquete RA contiene el prefijo global o prefijo de red de la dirección IP que se le va a asignar al equipo (si los flags son los apropiados).

En la siguiente imagen se aprecia perfectamente esa comunicación entre r0 y el cliente (partes marcadas en rojo), inicialmente r0 recibe un paquete RS con la MAC del cliente 0A-AA-00-00-00-0C (Event #164), a continuación, en el log se ve un mensaje interno del router *sendSolicitedRA* que está preparando su paquete RA de respuesta (Event #191), por último, el cliente recibe el paquete RA de respuesta con el prefijo de red aaaa:0:65::/64 (Event #211).

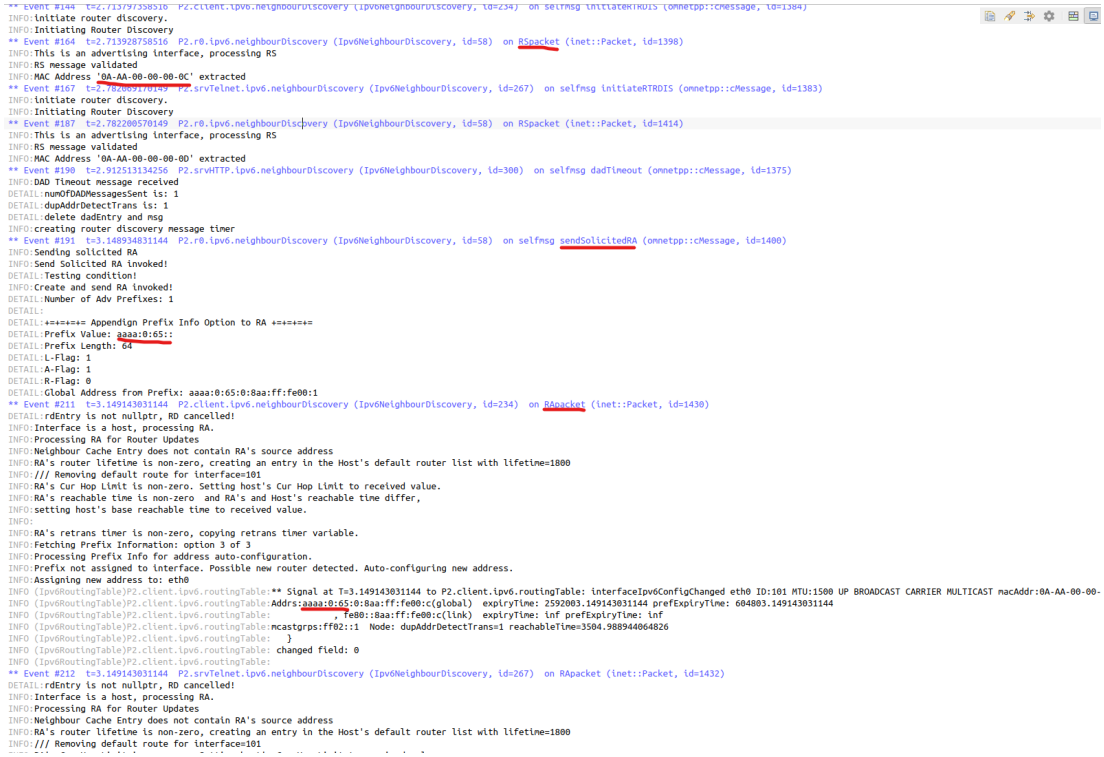


Figura 2.5: Comunicación entre *client* y *r0* (paquetes RS y RA)

Ahora veamos los campos más relevantes del contenido del paquete RA.

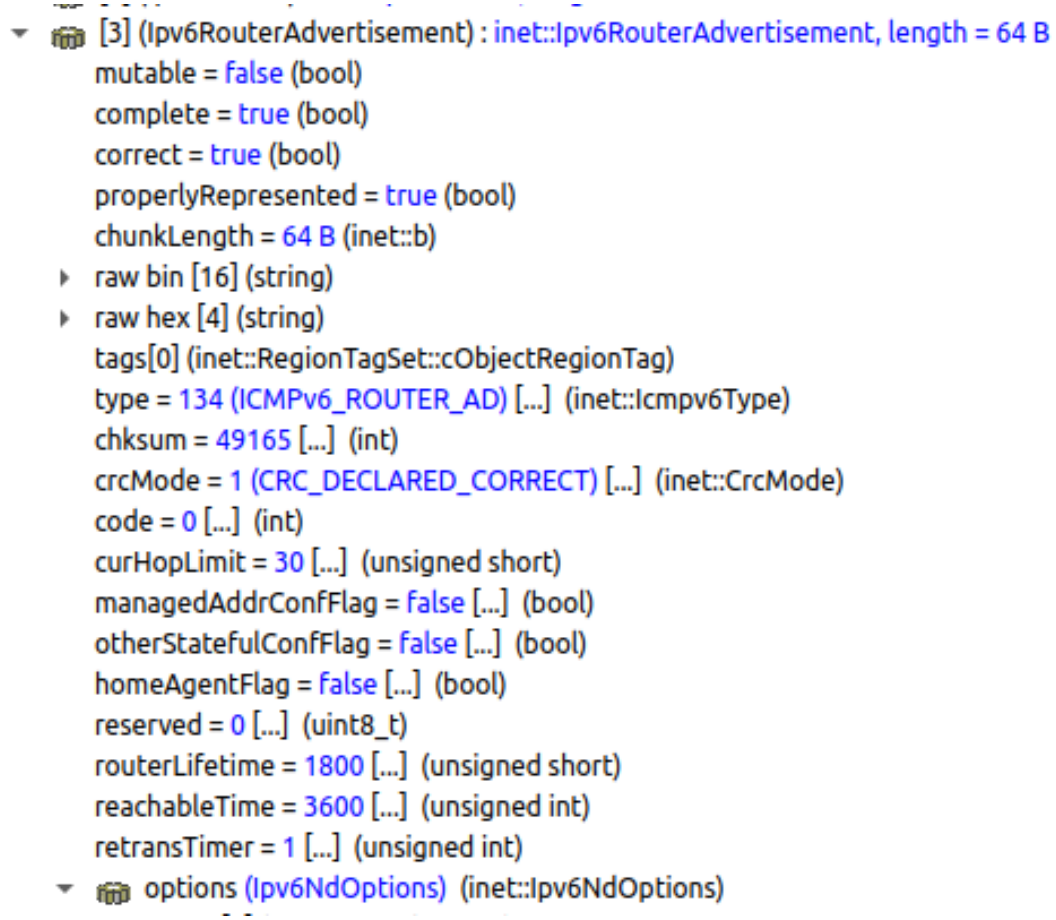


Figura 2.6: Contenido del paquete RA enviado de *r0* a *client*

En esta captura hay dos campos bastante relevantes a comentar, en primer lugar, el campo *managedAddrConfFlag*, este flag indica al cliente que debe solicitar su IP a un servidor DHCPv6, por otro lado, está el campo *otherStatefulConfFlag*, este flag le indica al cliente que debe solicitar a un servidor DHCPv6 las direcciones DNS necesarias. Los dos flags están a 0 (false), por tanto, el router le está diciendo al cliente a través del paquete RA que utilizará SLAAC (Stateless Address Auto Configuration) para obtener una dirección IP y no utilizará un servidor DHCPv6 para obtener ningún otro parámetro de configuración (configuración por defecto en routers Cisco). Otro campo no tan relevante es el *type*, pero está bien mencionarlo para recordar que los paquetes RA y los paquetes RS van encapsulados en ICMPv6.

Figura 2.7: Contenido del paquete RA enviado de *r0* a *client* (2)

En esta captura podemos ver el campo *autoAddressConFlag*, este flag le indica al cliente que puede crearse una dirección a sí mismo a partir del prefijo de red enviado en el paquete y un identificador de interfaz, prefijo de red el cuál está indicado más abajo en el campo *prefix* así como su longitud *prefixLength*. Otros campos que pueden resultar de interés son *linkLayerAddress* que es la dirección MAC del router (0A-AA-00-00-00-01) o *mtu* que es la unidad máxima de transferencia del enlace (1280B).

2.6 Pregunta 6

¿Qué ocurre en un nodo que recibe un RA? Muestre la tabla de interfaces del nodo cliente justo antes y justo después de recibir el primer RA y explique los cambios (de nuevo, copie y pegue el texto de la tabla de interfaces en lugar de usar capturas).

Cuando un nodo recibe un paquete RA, este configurará su dirección IPv6 en función de los parámetros contenidos en el paquete, hay 3 opciones en cuanto a la configuración y están relacionadas con los campos mencionados en el ejercicio anterior, *managedAddrConf-Flag* (M) y *otherStatefulConfFlag* (O), dependiendo de estos dos flags, el equipo que recibe el paquete RA configurará su equipo de una de estas 3 formas:

- SLAAC (M=false O=false)

- SLAAC y DHCPv6 para las direcciones DNS (M=false O=true)
- Todo el direccionamiento excepto la puerta de enlace se sonfigura por DHCPv6 (M=true O=false)

En este caso, estamos en la primera opción (configuración por SLAAC) y como el flag *autoAddressConFlag* (A) = true, se usará el prefijo de red enviado en el paquete RA para realizar la configuración por SLAAC.

Tabla de interfaces del cliente (P2.client.interfaceTable.idToInterface.elements[])

Antes de recibir el paquete RA:

```
lo0 ID:100 MTU:4470 UP LOOPBACK CARRIER macAddr:n/a Ipv6:{
  Addrs:::1(loopback)  expiryTime: inf prefExpiryTime: inf
  Node: dupAddrDetectTrans=1 reachableTime=33.706910891924
   }
eth0 ID:101 MTU:1500 UP BROADCAST CARRIER MULTICAST
    macAddr:0A-AA-00-00-00-0C Ipv6:{
  Addrs:fe80::8aa:ff:fe00:c(link)  expiryTime: inf prefExpiryTime:
    inf
  mcastgrps:ff02::1   Node: dupAddrDetectTrans=1
    reachableTime=34.376823452767
   }
```

Después de recibir el paquete RA:

```
lo0 ID:100 MTU:4470 UP LOOPBACK CARRIER macAddr:n/a Ipv6:{
  Addrs:::1(loopback)  expiryTime: inf prefExpiryTime: inf
  Node: dupAddrDetectTrans=1 reachableTime=33.706910891924
   }
eth0 ID:101 MTU:1500 UP BROADCAST CARRIER MULTICAST
    macAddr:0A-AA-00-00-00-0C Ipv6:{
  Addrs:aaaa:0:65:0:8aa:ff:fe00:c(global)  expiryTime:
    2592003.149143031144 prefExpiryTime: 604803.149143031144
              , fe80::8aa:ff:fe00:c(link)  expiryTime: inf
    prefExpiryTime: inf
  mcastgrps:ff02::1   Node: dupAddrDetectTrans=1
    reachableTime=4683.278708998114
   }
```

Después de recibir el paquete RA el cliente configura una Global Address Unicast (GUA) (dirección con la etiqueta *global*). Además, como se indicó previamente, según el valor de los flags enviados en el paquete, el equipo se autoconfigura su propia dirección IPv6 (SLAAC) a partir del prefijo de red y un identificador de interfaz, en este caso, los últimos 64 bits de la dirección link-local unicast de esa interfaz.

GUA = Prefijo de red + Últimos 64 bits link-local unicast = aaaa:0:65:: + ...:8aa:ff:fe00:c = **aaaa:0:65:0:8aa:ff:fe00:c**

2.7 Pregunta 7

En torno al instante t = 6s vuelve a enviarse un paquete NS. ¿Cuál es su objetivo? Muestre capturas del log (de nuevo filtrando por nivel ipv6.neighbourDiscovery) que expliquen el motivo del envío.

En el instante t=6 el cliente intenta enviar un paquete SYN al servidor Telnet, el cliente tiene la dirección IPv6 del servidor (aaaa:0:65:0:8aa:ff:fe00:d), sin embargo, no tiene una correspondencia entre su IP y su MAC en la Neighbor Cache, por eso envía un paquete NS a la dirección multicast de nodo solicitado del servidor y crea una entrada vacía con la IP del servidor en su Neighbor Cache.

El servidor Telnet recibe el paquete NS y devuelve un paquete NA (Neighbor Advertisement) al cliente con su MAC, además, el servidor Telnet aprovecha para crear una entrada en su Neighbor Cache con la correspondencia IP/MAC del cliente, que viene incluida en el paquete NS.

Una vez el cliente recibe el paquete NA enviado por el servidor, comprueba que el equipo que le envió su dirección MAC es el servidor (dirección IP de origen = aaaa:0:65:0:8aa:ff:fe00:d) y actualiza la entrada que había creado en la Neighbor Cache con la dirección MAC del servidor.

ACLARACIÓN: En la captura se aprecia también un caso similar entre el cliente y r0, aunque este paquete NS se origina porque r0 sabe la dirección IPv6 de siguiente salto (el cliente) a la que tiene que mandar el paquete que le llegó pero no sabe su dirección MAC.

```
** Event #276  t=6  P2.client.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=234)  on SYN (inet::Packet, id=1871)
INFO:Packet (inet::Packet)SYN (64 B) [[inet::Ipv6Header, length = 40 B | inet::tcp::TcpHeader, 1025->23 [Syn] Seq=1500000 Win=7504, length = 24 B]] arrived from Ipv6 module.
INFO:Next Hop Address is: aaaa:0:65:0:8aa:ff:fe00:d on interface: 101
INFO:No Entry exists in the Neighbour Cache.
DETAIL:Creating an INCOMPLETE entry in the neighbour cache.
DETAIL:Initiating Address Resolution for:aaaa:0:65:0:8aa:ff:fe00:d on Interface:101
INFO:Preparing to send NS to solicited-node multicast group
INFO:on the next hop interface
INFO:Reachability State is INCOMPLETE. Address Resolution already initiated.
INFO:Add packet to entry's queue until Address Resolution is complete.
** Event #296  t=6.000157  P2.srvTelnet.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=267)  on NSpacket (inet::Packet, id=1885)
INFO:Process NS for Non-Tentative target address.
INFO:Neighbour Entry not found. Create a Neighbour Cache Entry.
** Event #318  t=6.000314  P2.client.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=234)  on NApacket (inet::Packet, id=1897)
INFO:NA received. Target Address found in Neighbour Cache
INFO:ND is updating Neighbour Cache Entry.
INFO:Reachability confirmed through successful Addr Resolution.
INFO:Sending queued packet (inet::Packet)SYN
** Event #319  t=6.000314  P2.r0.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=58)  on NApacket (inet::Packet, id=1898)
INFO:Packet (inet::Packet)NApacket (72 B) [[inet::Ipv6Header, length = 40 B | inet::Ipv6NeighbourAdvertisement, length = 32 B]] arrived from Ipv6 module.
INFO:Next Hop Address is: aaaa:0:65:0:8aa:ff:fe00:c on interface: 101
INFO:No Entry exists in the Neighbour Cache.
DETAIL:Creating an INCOMPLETE entry in the neighbour cache.
DETAIL:Initiating Address Resolution for:aaaa:0:65:0:8aa:ff:fe00:c on Interface:101
INFO:Preparing to send NS to solicited-node multicast group
INFO:on the next hop interface
INFO:Reachability State is INCOMPLETE. Address Resolution already initiated.
INFO:Add packet to entry's queue until Address Resolution is complete.
** Event #357  t=6.0004742  P2.client.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=234)  on NSpacket (inet::Packet, id=1933)
INFO:Process NS for Non-Tentative target address.
INFO:Neighbour Entry not found. Create a Neighbour Cache Entry.
** Event #394  t=6.0006312  P2.r0.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=58)  on NApacket (inet::Packet, id=1946)
INFO:NA received. Target Address found in Neighbour Cache
INFO:ND is updating Neighbour Cache Entry.
INFO:Reachability confirmed through successful Addr Resolution.
INFO:Sending queued packet (inet::Packet)NApacket
```

Figura 2.8: Log filtrando por nivel ipv6.neighbourDiscovery en el instante t=6

2.8 Pregunta 8

Explique las IPs y MACs origen y destino de dicho NS.

Dirección MAC de origen: 0A-AA-00-00-00-0C
Dirección IP de origen: aaaa:0:65:0:8aa:ff:fe00:c

Las direcciones MAC e IP de origen no tienen nada de especial, simplemente son las direcciones MAC e IP (unicast) del cliente que es el equipo que envía el paquete NS.

Dirección MAC de destino: FF-FF-FF-FF-FF-FF
Dirección IP de destino: ff02::1:ff00:d

Como dirección MAC de destino se utiliza la dirección de broadcast, ya que se desconoce la dirección MAC de destino. Como dirección IP de destino se utiliza la dirección multicast de nodo solicitado del destino, es decir, del servidor, de esta forma no molestamos al resto de dispositivos enviando una dirección IP de broadcast como en IPv4 (en IPv6 no existen), con las direcciones multicast de nodo solicitado, el resto de dispositivos podrán descartar el paquete sin tener que analizar su contenido.

2.9 Pregunta 9

¿Recibe respuesta este NS? Muestre capturas del tráfico de paquetes y el contenido del paquete de respuesta.

Sí, como se mencionó previamente en el ejercicio 7 (Sección 2.7 página 18), el cliente recibe como respuesta un paquete NA (Neighbor Advertisement) con la dirección MAC del servidor en su interior, que es a quién el cliente había solicitado la dirección MAC.

#279	6	client --> sw0 NSpacket	aaaa:0:65:0:8aa:ff:fe00:c	ff02::1:ff00:d	ICMPv6	98 B
#284	6.0000785	sw0 --> srvTelnet NSpacket	aaaa:0:65:0:8aa:ff:fe00:c	ff02::1:ff00:d	ICMPv6	98 B
#285	6.0000785	sw0 --> r0 NSpacket	aaaa:0:65:0:8aa:ff:fe00:c	ff02::1:ff00:d	ICMPv6	98 B
#299	6.000157	srvTelnet --> sw0 NApacket	aaaa:0:65:0:8aa:ff:fe00:d	aaaa:0:65:0:8aa:ff:fe00:c	ICMPv6	98 B
#306	6.0002355	sw0 --> client NApacket	aaaa:0:65:0:8aa:ff:fe00:d	aaaa:0:65:0:8aa:ff:fe00:c	ICMPv6	98 B
#307	6.0002355	sw0 --> r0 NApacket	aaaa:0:65:0:8aa:ff:fe00:d	aaaa:0:65:0:8aa:ff:fe00:c	ICMPv6	98 B
#324	6.000314	client --> sw0 SYN	1025	aaaa:0:65:0:8aa:ff:fe00:d:23	TCP	90 B
#325	6.000314	r0 --> sw0 NSpacket	aaaa:0:65:0:8aa:ff:fe00:1	ff02::1:ff00:c	ICMPv6	98 B
#333	6.0003861	sw0 --> srvTelnet SYN	1025	aaaa:0:65:0:8aa:ff:fe00:d:23	TCP	90 B
#340	6.0003957	sw0 --> client NSpacket	aaaa:0:65:0:8aa:ff:fe00:1	ff02::1:ff00:c	ICMPv6	98 B
#350	6.0004582	srvTelnet --> sw0 SYN+ACK	23	aaaa:0:65:0:8aa:ff:fe00:c:1025	TCP	90 B
#352	6.0004677	sw0 --> srvTelnet NSpacket	aaaa:0:65:0:8aa:ff:fe00:1	ff02::1:ff00:c	ICMPv6	98 B
#360	6.0004742	client --> sw0 NApacket	aaaa:0:65:0:8aa:ff:fe00:c	aaaa:0:65:0:8aa:ff:fe00:1	ICMPv6	98 B
#366	6.0005303	sw0 --> client SYN+ACK	23	aaaa:0:65:0:8aa:ff:fe00:c:1025	TCP	90 B
#376	6.0005527	sw0 --> r0 NApacket	aaaa:0:65:0:8aa:ff:fe00:c	aaaa:0:65:0:8aa:ff:fe00:1	ICMPv6	98 B
#377	6.0005557	sw0 --> srvTelnet NApacket	aaaa:0:65:0:8aa:ff:fe00:c	aaaa:0:65:0:8aa:ff:fe00:1	ICMPv6	98 B

Figura 2.9: Tráfico de paquetes NS y paquetes NA.

En t=6s (#279) el cliente envía el paquete NS al servidor Telnet (a través de *sw0*), en t=6.000157s (#299) el servidor Telnet responde al cliente con un paquete NA (a través de *sw0*).

ACLARACIÓN: Si nos fijamos en el tráfico de paquetes, los paquetes NS y los paquetes NA llegan al resto de nodos de ese segmento de red, en este caso, llegan a *r0*, esto ocurre porque los paquetes NS y NA se envían a la dirección multicast de nodo solicitado, entonces llegarán a todos los nodos del segmento de red. La ventaja de usar estas direcciones multicast de nodo solicitado en vez de un broadcast como en IPv4, es que los nodos a los que le llegan estos paquetes aunque no sean para ellos, pueden descartarlos sin tener que leer el contenido del paquete como en un broadcast, simplemente, mapearan la dirección multicast de nodo solicitado a una dirección MAC y la tarjeta de red (NIC) de ese nodo aceptará o descartará el paquete en función de la MAC mapeada.

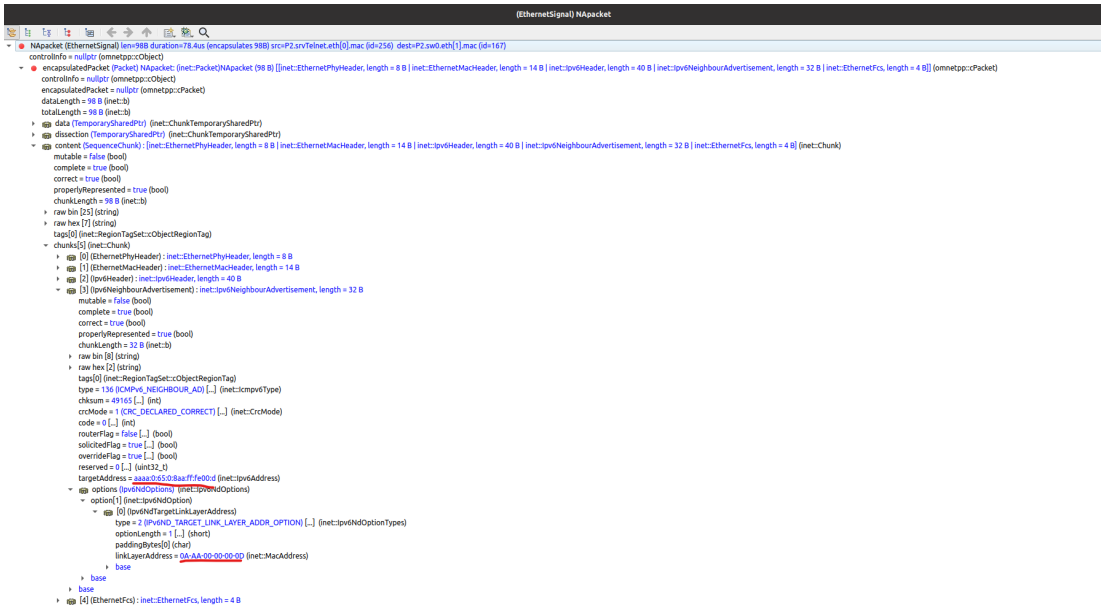


Figura 2.10: Contenido del paquete NA enviado de *srvTelnet* a *client*.

El paquete NA contiene un campo *targetAddress* donde el servidor indica su dirección IP para que el cliente pueda comprobar que el paquete NA recibido proviene de la dirección que estaba buscando.

Por otra parte, el paquete contiene otro campo *linkLayerAddress* contenido en *IPv6NdTargetLinkLayerAddress* que contiene la dirección MAC del servidor, es decir, la dirección MAC que el cliente había solicitado en su paquete NS.

2.10 Pregunta 10

Muestre capturas de la neighbor cache del nodo que generó el NS justo antes y justo después de recibir el paquete de respuesta y explíquelas.

Neighbor Cache del cliente (P2.client.ipv6.neighbourDiscovery.neighbourMap)

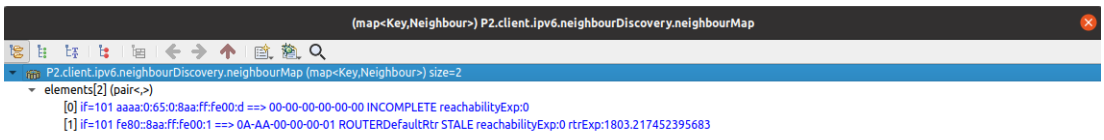


Figura 2.11: Neighbor Cache de *client* antes de recibir un paquete NA.

Antes de recibir el paquete de respuesta, el cliente solo tiene en su Neighbor Cache una

entrada con la correspondencia IP-MAC del router y una entrada en estado INCOMPLETE, con la IP del servidor Telnet y una MAC por defecto para completar la entrada mientras no se tenga la MAC del servidor (00-00-00-00-00-00).

ACLARACIÓN: Esto ocurre en mi caso, en el que paré la simulación justo en el instante en el que el cliente ya había enviado el paquete NS pero el servidor aún no había respondido con su paquete NA. Si hubiese parado la simulación un poco antes, antes de que el cliente hubiese enviado su paquete NS, en la Neighbor Cache no existiría la entrada incompleta con la IP del servidor (la crea al enviar el paquete NS), simplemente solo estaría la entrada correspondiente a *r0*.

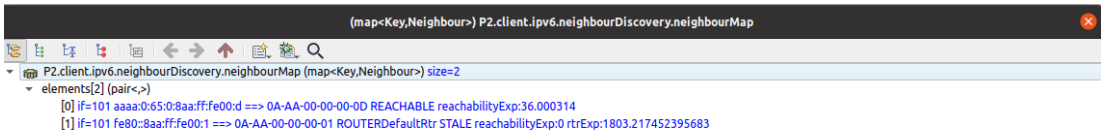


Figura 2.12: Neighbor Cache de *client* después de recibir un paquete NA.

Una vez el cliente recibe el paquete RA con la MAC del servidor, actualiza la entrada de la Neighbor Cache creada previamente, introduce la MAC del servidor en ella y cambia su estado a REACHABLE (el servidor ya es un vecino alcanzable).

2.11 Pregunta 11

¿Se generan más NS a lo largo de la simulación? ¿Cuál es su objetivo? Muestre capturas del log en las que se muestre el motivo del envío.

Hasta t=10.0015909s se siguen generando paquetes NS por el mismo motivo que antes, se están estableciendo conexiones TCP entre el cliente y los servidores, lo que provoca que todos los nodos intermedios en el camino, además de estos tres equipos, tengan que actualizar su Neighbor Cache para tener la información necesaria para poder enviarse paquetes TCP entre el cliente y los servidores.

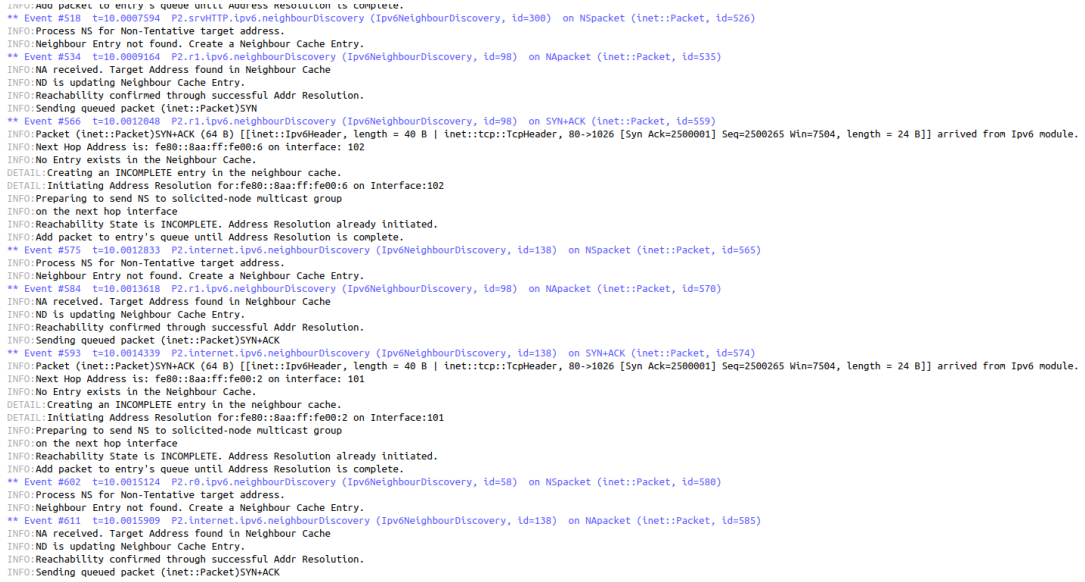


Figura 2.13: Log del envío de paquetes NS y NA debido a conexiones TCP.

A partir de este momento, se envían NS a medida que es necesario actualizar las entradas de la Neighbor Cache. Las entradas de la Neighbor Cache tienen dos estados normalmente, STALE y REACHABLE, y otros tres estados transitorios, DELAY, PROBE e INCOMPLETE. Cuando lleva un tiempo sin recibirse un paquete de la IP/MAC asociada a una entrada o se recibe un NA no solicitado de esa IP/MAC, esa entrada pasa de REACHABLE a STALE, en este estado STALE se suele enviar un paquete TCP para comprobar si el equipo responde, si responde en menos de 5 segundos vuelve al estado REACHABLE y si no responde pasa al estado PROBE. Es este estado (PROBE) en el que se envía de nuevo un paquete NS con una *targetAddress* igual a la IP de la entrada correspondiente. Si este equipo no responde con un paquete NA tras haber enviado tres paquetes NS seguidos, la entrada se elimina de la Neighbor Cache y, si responde, la entrada vuelve al estado REACHABLE.

Hay algún detalle más en la transición de estados, pero ya la hemos visto previamente (transición de INCOMPLETE a REACHABLE) y no resulta de interés para el caso actual.

```
** Event #3922  t=11.0004582  P2.srvTelnet.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=267)  on selfmsg NUDTimeout (omnetpp::cMessage, id=425)
INFO:NUD Timeout message received
INFO:NUD has timed out
DETAIL:Neighbour Entry is still in DELAY state.
DETAIL:Entering PROBE state. Sending NS probe.
** Event #3946  t=11.0006024  P2.client.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=234)  on NSpacket (inet::Packet, id=2155)
INFO:Process NS for Non-Tentative target address.
** Event #3989  t=11.0007538  P2.client.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=234)  on NSpacket (inet::Packet, id=2180)
WARN:Hop limit is not 255! NS validation failed!
** Event #4011  t=11.0008386  P2.srvTelnet.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=267)  on NApacket (inet::Packet, id=2194)
INFO:NA received. Target Address found in Neighbour Cache
INFO:NA override flag is TRUE. or Advertised MAC is same as NCE's. or NA MAC is not specified.
INFO:Solicited Flag is TRUE. Set NCE state to REACHABLE.
INFO:NUD in progress. Cancelling NUD Timer
INFO:Updating NCE's router flag to 0
** Event #4012  t=11.0008386  P2.r0.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=58)  on NApacket (inet::Packet, id=2195)
INFO:Packet (inet::Packet)NApacket (72 B) [[inet::Ipv6Header, length = 40 B | inet::Ipv6NeighbourAdvertisement, length = 32 B]] arrived from Ipv6 module.
INFO:Next Hop Address is: aaaa:0:65:0:8aa:ff:fe00:d on interface: 101
INFO:No Entry exists in the Neighbour Cache.
DETAIL:Creating an INCOMPLETE entry in the neighbour cache.
DETAIL:Initiating Address Resolution for:aaaa:0:65:0:8aa:ff:fe00:d on Interface:101
INFO:Preparing to send NS to solicited-node multicast group
INFO:on the next hop interface
INFO:Reachability State is INCOMPLETE. Address Resolution already initiated.
INFO:Add packet to entry's queue until Address Resolution is complete.
** Event #4053  t=11.0009956  P2.srvTelnet.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=267)  on NSpacket (inet::Packet, id=2220)
INFO:Process NS for Non-Tentative target address.
INFO:Neighbour Entry not found. Create a Neighbour Cache Entry.
** Event #4089  t=11.0011526  P2.r0.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=58)  on NApacket (inet::Packet, id=2239)
INFO:NA received. Target Address found in Neighbour Cache
INFO:ND is updating Neighbour Cache Entry.
INFO:Reachability confirmed through successful Addr Resolution.
INFO:Sending queued packet (inet::Packet)NApacket
** Event #4106  t=11.0013096  P2.srvTelnet.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=267)  on NApacket (inet::Packet, id=2246)
WARN:Hop Limit is not 255! NA validation failed!
** Event #8276  t=15  P2.client.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=234)  on selfmsg NUDTimeout (omnetpp::cMessage, id=480)
INFO:NUD Timeout message received
INFO:NUD has timed out
DETAIL:Neighbour Entry is still in DELAY state.
DETAIL:Entering PROBE state. Sending NS probe.
** Event #8296  t=15.0001442  P2.r0.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=58)  on NSpacket (inet::Packet, id=4285)
INFO:Process NS for Non-Tentative target address.
** Event #8318  t=15.0003012  P2.client.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=234)  on NApacket (inet::Packet, id=4297)
INFO:NA received. Target Address found in Neighbour Cache
INFO:NA override flag is TRUE. or Advertised MAC is same as NCE's. or NA MAC is not specified.
INFO:Solicited Flag is TRUE. Set NCE state to REACHABLE.
INFO:NUD in progress. Cancelling NUD Timer
INFO:Updating NCE's router flag to 1
** Event #8321  t=15.0010606  P2.srvHTTP.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=300)  on selfmsg NUDTimeout (omnetpp::cMessage, id=553)
```

Figura 2.14: Log del envío de paquetes NS y NA para actualizar la Neighbor Cache.

Nos centraremos en la comunicación entre el cliente y el servidor Telnet, ya que la comunicación entre el servidor Telnet y *r0* es un caso similar a los ya vistos anteriormente (se crea una nueva entrada en la Neighbor Cache y se actualiza - transición de INCOMPLETE a REACHABLE).

En t=11.0004582s (Event #3922) se ve en el log un mensaje interno del servidor Telnet, *NUDTimeout*, ¿Qué significa esto? Neighbor Unreachability Detection Timeout, esto significa que el servidor Telnet tiene una entrada en su Neighbor Cache con la correspondencia IP/MAC del cliente en estado DELAY, el servidor Telnet envió un paquete TCP para comprobar que el cliente sigue siendo alcanzable y este no respondió antes de que se pasase el *NUDTimeout* (por defecto, 5s). Cuando el servidor Telnet recibe este timeout, cambia el estado de la entrada de DELAY a PROBE y envía un paquete NS al cliente para comprobar que sigue ahí.

En t=11.0006024s (Event #3946) el cliente recibe el paquete NS y envía un paquete NA de respuesta al servidor Telnet.

En t=11.0008386s (Event #4011) el servidor Telnet recibe el paquete NA, comprueba que la *targetAddress* coincide con la del cliente, actualiza la entrada de la Neighbor Cache cambiando su estado de PROBE a REACHABLE de nuevo y cancela el NUD Timer, temporizador para llevar el control de los timeouts de cada estado.

Este sería el proceso que se repetirá continuamente cada vez que finalice un timeout a lo largo de la simulación, de hecho, en t=15s (Event #8276), vuelve a pasar lo mismo entre el cliente y *r0*, el cliente recibe el mensaje de timeout y vuelve a repetirse el mismo proceso entre estos dos nodos.

A continuación, se muestra la transición de estados de la entrada correspondiente al cliente en la Neighbor Cache del servidor Telnet.

Justo antes de recibir el mensaje de *NUDTimeout*:

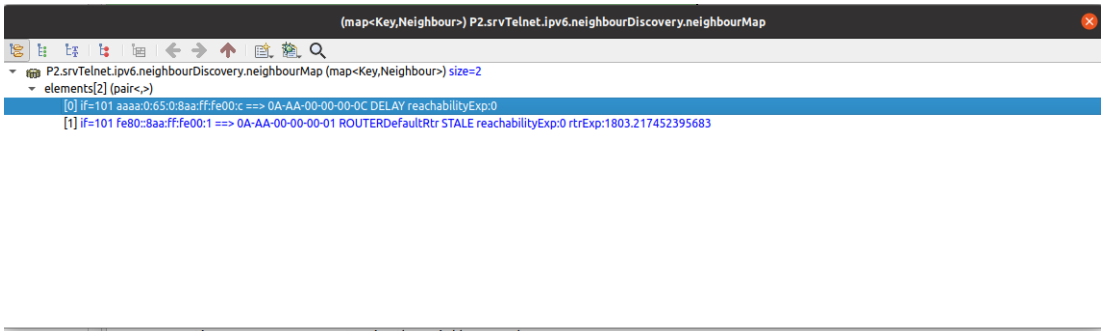


Figura 2.15: Neighbor Cache de *srvTelnet* (Entrada correspondiente a *client* en estado DELAY).

Tras recibir el mensaje de *NUDTimeout* y enviar un paquete NS al cliente:

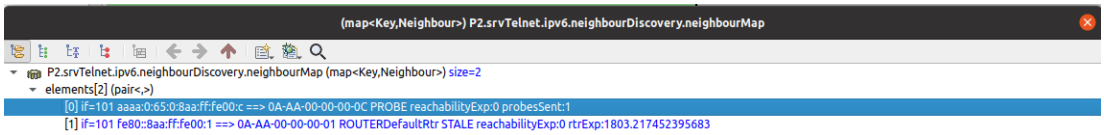


Figura 2.16: Neighbor Cache de *srvTelnet* (Entrada correspondiente a *client* en estado PROBE).

Tras recibir el paquete NA de respuesta del cliente:

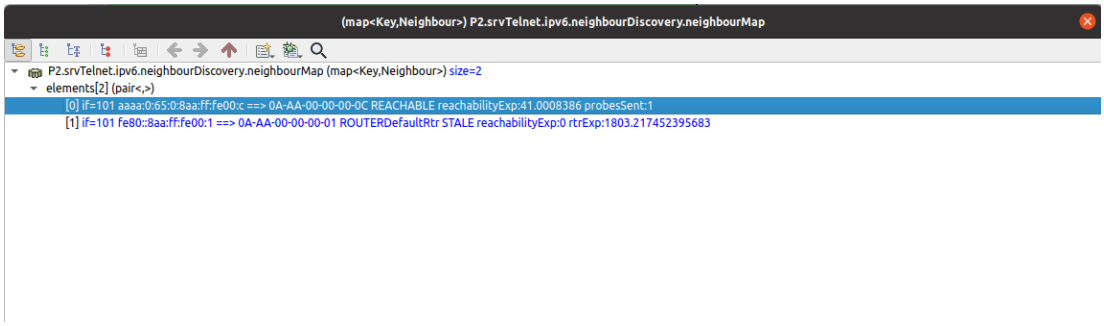


Figura 2.17: Neighbor Cache de *srvTelnet* (Entrada correspondiente a *client* en estado REACHABLE).

2.12 Pregunta 12

Elija uno y explique IPs y MACs origen y destino. ¿Son del mismo tipo que en los casos anteriores? ¿Por qué?

Dirección MAC de origen: 0A-AA-00-00-00-0D
Dirección IP de origen: aaaa:0:65:0:8aa:ff:fe00:d

Las direcciones MAC e IP de origen no tienen nada de especial, al igual que en el ejercicio 8 (Sección 2.8 página 19), simplemente son las direcciones MAC e IP (unicast) del servidor Telnet que es el equipo que envía el paquete NS.

Dirección MAC de destino: FF-FF-FF-FF-FF-FF
Dirección IP de destino: aaaa:0:65:0:8aa:ff:fe00:c

La dirección IP de destino si que cambia en este caso, en el ejercicio 8 (Sección 2.8 página 19), la dirección que se usaba era la dirección multicast de nodo solicitado del servidor Telnet que era el destino en ese caso, ahora el destino es el cliente pero no se utiliza su dirección multicast de nodo solicitado, si no su dirección IPv6 global (GUA), esto se debe a que antes no existía una entrada en la Neighbor Cache con la IP de destino, se estaba creando en ese momento, ahora la entrada ya está creada y al llegar un timeout lo que hará el origen será coger la dirección IP de la entrada que provocó el timeout y enviar un paquete NS a esa dirección IP y MAC FF-FF-FF-FF-FF-FF para comprobar su disponibilidad.

2.13 Pregunta 13

¿Se generan más RA a lo largo de la simulación? ¿Cuál es su objetivo? ¿Se generan como respuesta a algún paquete?

Una vez todos los hosts han configurado sus interfaces de red, el router no recibirá ningún paquete RS más, suponiendo que no se une un nuevo host a la red, que es el caso de la simulación. Sin embargo, SÍ que se enviarán más paquetes RA porque todos los routers enviarán paquetes RA a todos los hosts cada 200s, con el objetivo de anunciar a los hosts que siguen activos y que actualicen sus parámetros de red si alguno ha cambiado.

Entonces la respuesta es, SÍ, se seguiran enviando paquetes RA y NO, no se generan como respuesta a algún paquete, simplemente son mensajes periódicos para informar a los hosts de que siguen ahí y cuál es su estado.

A continuación se muestra como el cliente y el servidor Telnet reciben un paquete RA de *r0* y procesan la información del paquete para actualizar sus parámetros de red.

```
** Event #280739  t=220.798824441664  P2.client.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=234)  on RApacket (inet::Packet, id=279452)
DETAIL:rdEntry is nullptr, not cancelling RD!
INFO:Interface is a host, processing RA.
INFO:Processing RA for Router Updates
INFO:RA's Cur Hop Limit is non-zero. Setting host's Cur Hop Limit to received value.
INFO:RA's reachable time is non-zero  and RA's and Host's reachable time differ,
INFO:setting host's base reachable time to received value.
INFO:
INFO:RA's retrans timer is non-zero, copying retrans timer variable.
INFO:Fetching Prefix Information: option 3 of 3
INFO:Processing Prefix Info for address auto-configuration.
INFO:The received Prefix is already assigned to the interface
** Event #280740  t=220.798824441664  P2.srvTelnet.ipv6.neighbourDiscovery (Ipv6NeighbourDiscovery, id=267)  on RApacket (inet::Packet, id=279454)
DETAIL:rdEntry is nullptr, not cancelling RD!
INFO:Interface is a host, processing RA.
INFO:Processing RA for Router Updates
INFO:RA's Cur Hop Limit is non-zero. Setting host's Cur Hop Limit to received value.
INFO:RA's reachable time is non-zero  and RA's and Host's reachable time differ,
INFO:setting host's base reachable time to received value.
INFO:
INFO:RA's retrans timer is non-zero, copying retrans timer variable.
INFO:Fetching Prefix Information: option 3 of 3
INFO:Processing Prefix Info for address auto-configuration.
INFO:The received Prefix is already assigned to the interface
```

Figura 2.18: Log de los paquetes RA enviados desde *r0* a *client* y *srvTelnet* en t=220s.

¿Se puede diferenciar si un paquete RA es una respuesta a un paquete RS o es un mensaje periódico? Sí, los paquetes RA de respuesta a un paquete RS tendrán como dirección de destino la del host que envió el paquete RS, si es un mensaje periódico tendrá como dirección de destino la dirección multicast que incluye a todos los equipos IPv6 del segmento de red (ff02::1).

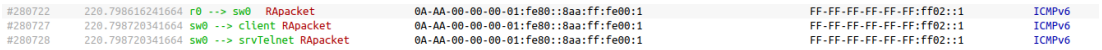


Figura 2.19: Paquetes RA enviados desde *r0* a *client* y *srvTelnet* en t=220s.

Como se ve en el tráfico de paquetes, la dirección de destino es la dirección multicast IPv6 all-nodes (ff02::1).