Facultade de Informática
UNIVERSIDADE DA CORUÑA

DISEÑO DE REDES

Práctica 3 - Mobile IP en INET

Estudante: Tomé Maseda Dorado
Estudante: Jorge Álvarez Cabado

A Coruña, November de 2021.

Índice Xeral

1 Ficheros utilizados na simulación **5**

1.1 Ficheros .ini 5

1.2 Ficheros .ned 8

2 Preguntas acerca da simulación **13**

2.1 Pregunta 1 13

2.2 Pregunta 2 14

2.3 Pregunta 3 15

2.4 Pregunta 4 18

2.5 Pregunta 5 19

2.6 Pregunta 6 21

2.7 Pregunta 7 22

2.8 Pregunta 8 25

2.9 Pregunta 9 26

2.10 Pregunta 10 28

2.11 Pregunta 11 30

2.12 Pregunta 12 34

2.13 Pregunta 13 36

2.14 Cambios en el omnetpp.ini debido a un bug 38

2.15 Pregunta 14 39

Índice de Figuras

2.1 Paquetes DHCP intercambiados entre *mobile* y *dhcp1* 13
2.2 Paquete DHCPOFFER enviado de *dhcp1* a *mobile* 16
2.3 Paquete DHCPOFFER enviado de *dhcp1* a *mobile* (opciones) 17
2.4 Ventana de error OMNET++ (*Wrong source address*) 18
2.5 Paquetes TCP enviados entre *rC* y *static* 20
2.6 Protocolo ARQ de parada y espera (*Stop and Wait ARQ*) 21
2.7 Paquetes UDP enviados por *mobile* a *static* (t=40s) 22
2.8 Paquetes UDP recibidos en *static* provenientes de *mobile* (t=40s) 23
2.9 Paquetes UDP enviados por *mobile* a *static* (t=115s) 23
2.10 Paquetes UDP recibidos en *static* provenientes de *mobile* (t=115s) 23
2.11 Cabecera *Ieee80211DataHeader* de un paquete UDP enviado por *mobile* al nodo *static* 24
2.12 Tabla de interfaces del router *r1* 24
2.13 Paquetes UDP enviados por *mobile* a *static* (t=130s) 24
2.14 Paquetes UDP recibidos en *static* provenientes de *mobile* (t=130s) 25
2.15 Paquetes UDP enviados por *mobile* a *static* (t=1800s) 25
2.16 Paquetes UDP recibidos en *static* provenientes de *mobile* (t=1800s) 25
2.17 Paquetes UDP enviados por *static* a *mobile* (t=1800s) 26
2.18 Paquetes UDP recibidos en *mobile* provenientes de *static* (t=1800s) 26
2.19 Paquetes BU y BAck enviados entre *mobile* (MN) y *r1* (HA) 27
2.20 Log de *r1* (HA) filtrando a nivel *mipv6support* 27
2.21 Binding Cache de *r1* (HA) tras la creación del túnel 28
2.22 Log de *r1* (HA) filtrando por nivel *ipv6* e *ipv6tunneling* 29
2.23 Anatomía (a nivel IP) del segmento TCP encapsulado. Paquete externo (gris) y paquete interno (negro). 29
2.24 Log de *r1* (HA) filtrando por nivel *ipv6* e *ipv6tunneling* 29

2.25 Anatomía (a nivel IP) del segmento TCP encapsulado. Paquete externo (negro) y paquete interno (gris). 30
2.26 Cabecera externa de un segmento TCP enviado de *mobile* (MN) a *static* (CN). . 31
2.27 Cabecera interna de un segmento TCP enviado de *mobile* (MN) a *static* (CN). . 32
2.28 Cabecera externa de un segmento TCP enviado de *static* (CN) a *mobile* (MN). . 33
2.29 Cabecera interna de un segmento TCP enviado de *static* (CN) a *mobile* (MN). . 34
2.30 Rutas seguidas por los paquetes TCP (Azul) cuando *mobile* (MN) está en la cobertura de *ap1* 35
2.31 Rutas seguidas por los paquetes TCP (Azul) cuando *mobile* (MN) está en la cobertura de *ap2* 36
2.32 Rutas seguidas por los paquetes TCP (Azul) cuando *mobile* (MN) está en la cobertura de *ap1* 37
2.33 Rutas seguidas por los paquetes TCP (Azul) cuando *mobile* (MN) está en la cobertura de *ap2* 37
2.34 Ventana de error (*Implicit Chunk Serialization*) 38
2.35 Modificación de la configuración del fichero omnetpp.ini 39
2.36 Paquetes UDP enviados por *mobile* a *static* (t=1800s) 39
2.37 Paquetes UDP recibidos en *static* provenientes de *mobile* (t=1800s) 39
2.38 Paquetes UDP enviados por *mobile* a *static* (t=1800s) 40
2.39 Paquetes UDP recibidos en *static* provenientes de *mobile* (t=1800s) 40

Introducción

En esta práctica estudiaremos el comportamiento de una red IPv4 y una red IPv6 (con Mobile IPv6) en las que un nodo móvil se ma va moviendo entre las coberturas de dos puntos de acceso distintos.

NOTA: Por simplicidad, en las preguntas relacionadas con *Mobile IPv6* se usa la siguiente terminología:

- *Nodo móvil* (MN): Nodo que va cambiando su punto de conexión, entre *ap1* y *ap2* durante la simulación, en este caso, *mobile*.
- *Home Address* (HoA): Dirección IP original de MN.
- *Care-of Address* (CoA): Dirección IP actual de MN.
- *Home Agent* (HA): Router de la red original de MN, en este caso, *r1*.
- *Correspondent Node* (CN): Nodo con el que se comunica MN.

Capítulo 1

Ficheros utilizados en la simulación

1.1 Ficheros .ini

```
[General]
sim-time-limit = 1800s

**.visualizer.mobilityVisualizer.displayMovementTrails = true
**.visualizer.*.displayRoutes = true
**.visualizer.*.displayLinks = true
**.visualizer.*.displayCommunicationRanges = true
**.radio.transmitter.power = 2.0mW
**.interfaceTable.displayAddresses = true

**.mobile.mobility.constraintAreaMinX = 0m
**.mobile.mobility.constraintAreaMinY = 0m
**.mobile.mobility.constraintAreaMinZ = 0m
**.mobile.mobility.constraintAreaMaxX = 500m
**.mobile.mobility.constraintAreaMaxY = 500m
#**.mobile.mobility.constraintAreaMaxX = 700m
#**.mobile.mobility.constraintAreaMaxY = 300m
**.mobile.mobility.constraintAreaMaxZ = 0m

**.mobile.mobility.typename = "RectangleMobility"
**.mobile.mobility.startPos = 0
#**.mobile.mobility.speed = 10mps
**.mobile.mobility.speed = 16.66mps
**.mobile*.mobility.updateInterval = 100ms

**.static.numApps = 1

*.ap*.wlan*.mgmt.beaconInterval = 0.5s
**.neighbourDiscovery.minIntervalBetweenRAs = 0.5s

```

```
31
32 [Config _IPv4]
33 description = Hosts configured via DHCP
34 network = IPv4
35
36 *.configurator.config = xml("<config> \
37                                <interface hosts='r1' names='eth0'
    address='8.0.0.1' netmask='255.255.0.0'/> \
38                                <interface hosts='r2' names='eth0'
    address='8.1.0.1' netmask='255.255.0.0'/> \
39                                <interface hosts='dhcp1'
    names='eth0' address='8.0.0.100' netmask='255.255.0.0'/> \
40                <interface hosts='dhcp2' names='eth0'
    address='8.1.0.100' netmask='255.255.0.0'/> \
41                <interface names='eth*' address='8.x.x.x'
    netmask='255.255.0.0'/> \
42                             </config>")
43
44 **.mobile.numApps = 2
45 **.mobile.app[1].typename = "DhcpClient"
46
47 **.dhcp1.numApps = 1
48 **.dhcp1.app[0].typename = "DhcpServer"
49 **.dhcp1.app[0].numReservedAddresses = 10 # ip to start to lease
50 **.dhcp1.app[0].maxNumClients = 100
51 **.dhcp1.app[0].leaseTime = 100s
52
53 **.dhcp2.numApps = 1
54 **.dhcp2.app[0].typename = "DhcpServer"
55 **.dhcp2.app[0].numReservedAddresses = 10 # ip to start to lease
56 **.dhcp2.app[0].maxNumClients = 100
57 **.dhcp2.app[0].leaseTime = 100s
58
59
60 [Config _MIPv6]
61 network = MobileIPv6
62 **.mobile.numApps = 1
63
64 [Config _UDP]
65 **.messageLength = 100B
66
67 [Config _TCP]
68 **.thinkTime = 2s
69 **.numRequestsPerSession = 100000
70 **.requestLength = 100B
71 **.replyLength = 100B
```

```
72
73 [Config _UDP_mobile-static]
74 extends = _UDP
75 **.mobile.app[0].typename = "UdpBasicApp"
76 **.mobile.app[0].startTime = 10s
77 **.mobile.app[0].sendInterval = 2s
78 **.mobile.app[0].destAddresses = "static"
79 **.mobile.app[0].destPort = 5001
80
81 **.static.app[0].typename = "UdpSink"
82 **.static.app[0].localPort = 5001
83
84 [Config _UDP_static-mobile]
85 extends = _UDP
86 **.static.app[0].typename = "UdpBasicApp"
87 **.static.app[0].startTime = 10s
88 **.static.app[0].sendInterval = 2s
89 **.static.app[0].destAddresses = "mobile"
90 **.static.app[0].destPort = 5001
91
92 **.mobile.app[0].typename = "UdpSink"
93 **.mobile.app[0].localPort = 5001
94
95 [Config _TCP_mobile-static]
96 extends = _TCP
97 **.mobile.app[0].typename = "TcpBasicClientApp"
98 **.mobile.app[0].startTime = 10s
99 **.mobile.app[0].localPort = -1
100 **.mobile.app[0].connectAddress = "static"
101 **.mobile.app[0].connectPort = 1000
102 **.mobile.app[0].idleInterval = truncnormal(3600s,1200s)
103
104
105 **.static.app[0].typename = "TcpGenericServerApp"
106 **.static.app[0].localAddress = ""
107 **.static.app[0].localPort = 1000
108 **.static.app[0].replyDelay = 0s
109
110 [Config _TCP_static-mobile]
111 extends = _TCP
112 **.static.app[0].typename = "TcpBasicClientApp"
113 **.static.app[0].startTime = 10s
114 **.static.app[0].localPort = -1
115 **.static.app[0].connectAddress = "mobile"
116 **.static.app[0].connectPort = 1000
117 **.static.app[0].idleInterval = truncnormal(3600s,1200s)
```

```
118
119
120
121 **.mobile.app[0].typename = "TcpGenericServerApp"
122 **.mobile.app[0].localAddress = ""
123 **.mobile.app[0].localPort = 1000
124 **.mobile.app[0].replyDelay = 0s
125
126 [Config IPv4_TCP_mobile-static]
127 extends = _IPv4, _TCP_mobile-static
128
129 [Config IPv4_TCP_static-mobile]
130 extends = _IPv4, _TCP_static-mobile
131
132 [Config IPv4_UDP_mobile-static]
133 extends = _IPv4, _UDP_mobile-static
134
135 [Config IPv4_UDP_static-mobile]
136 extends = _IPv4, _UDP_static-mobile
137
138 [Config MobileIPv6_TCP_mobile-static]
139 extends = _MIPv6, _TCP_mobile-static
140
141 [Config MobileIPv6_TCP_static-mobile]
142 extends = _MIPv6, _TCP_static-mobile
143
144 [Config MobileIPv6_UDP_mobile-static]
145 extends = _MIPv6, _UDP_mobile-static
146
147 [Config MobileIPv6_UDP_static-mobile]
148 extends = _MIPv6, _UDP_static-mobile
```

1.2 Ficheros .ned

Fichero IPv4.ned

```
1 import inet.networklayer.configurator.ipv4.Ipv4NetworkConfigurator;
2 import inet.node.ethernet.Eth100M;
3 import inet.node.inet.StandardHost;
4 import inet.node.inet.WirelessHost;
5 import inet.node.wireless.AccessPoint;
6 import inet.node.inet.Router;
7 import
      inet.physicallayer.ieee80211.packetlevel.Ieee80211ScalarRadioMedium;
8 import inet.visualizer.contract.IIntegratedVisualizer;
```

```
9  import ned.DatarateChannel;
10
11 network IPv4
12 {
13     parameters:
14
15         @display("bgb=900,400");
16     types:
17         channel ethernetline extends DatarateChannel
18         {
19             delay = 0.1us;
20             datarate = 10Mbps;
21         }
22     submodules:
23         visualizer: <default("IntegratedCanvasVisualizer")> like
       IIntegratedVisualizer if hasVisualizer() {
24             @display("p=854.4375,23.625;is=s");
25         }
26         radioMedium: Ieee80211ScalarRadioMedium {
27             @display("p=631.3125,26.25;is=s");
28         }
29         configurator: Ipv4NetworkConfigurator {
30             @display("p=752.0625,22.3125;is=s");
31         }
32         mobile: WirelessHost {
33             @display("p=102.33375,56.46;i=device/pocketpc");
34         }
35         r1: Router {
36             @display("p=286.125,199.5");
37         }
38         ap1: AccessPoint {
39             @display("p=248.0625,82.6875");
40         }
41         dhcp1: StandardHost {
42             @display("p=115.5,198.1875;i=device/server2");
43         }
44         r2: Router {
45             @display("p=433.125,185.0625");
46         }
47         ap2: AccessPoint {
48             @display("p=490.875,77.4375");
49         }
50         dhcp2: StandardHost {
51             @display("p=665.4375,128.625;i=device/server2");
52         }
53         internet: Router {
```

```
54              @display("p=355.6875,273;i=misc/cloud");
55          }
56          rC: Router {
57              @display("p=404.25,342.5625");
58          }
59          static: StandardHost {
60              @display("p=622.125,341.25;i=device/pc");
61          }
62      connections:
63          ap1.ethg++ <--> ethernetline <--> r1.ethg++;
64          ap2.ethg++ <--> ethernetline <--> r2.ethg++;
65          dhcp1.ethg++ <--> ethernetline <--> ap1.ethg++;
66
67          ap2.ethg++ <--> ethernetline <--> dhcp2.ethg++;
68
69          r1.ethg++ <--> ethernetline <--> internet.ethg++;
70          r2.ethg++ <--> ethernetline <--> internet.ethg++;
71          internet.ethg++ <--> ethernetline <--> rC.ethg++;
72          static.ethg++ <--> ethernetline <--> rC.ethg++;
73
74  }
```

Fichero MobileIPv6.ned

```
1   import
        inet.networklayer.configurator.ipv6.Ipv6FlatNetworkConfigurator;
2   import inet.node.xmipv6.WirelessHost6;
3   import inet.node.xmipv6.CorrespondentNode6;
4   import inet.node.xmipv6.HomeAgent6;
5   import inet.node.wireless.AccessPoint;
6   import
        inet.physicallayer.ieee80211.packetlevel.Ieee80211ScalarRadioMedium;
7   import inet.visualizer.contract.IIntegratedVisualizer;
8   import inet.node.ipv6.Router6;
9   import ned.DatarateChannel;
10
11
12  network MobileIPv6
13  {
14      parameters:
15
16          @display("bgb=978.64,435.2125");
17      types:
18          channel ethernetline extends DatarateChannel
19          {
20              delay = 0.1us;
21              datarate = 10Mbps;
```

```
22          }
23      submodules:
24          visualizer: <default("IntegratedCanvasVisualizer")> like
        IIntegratedVisualizer if hasVisualizer() {
25              @display("p=824.5513,35.2875");
26          }
27          radioMedium: Ieee80211ScalarRadioMedium {
28              @display("p=602.24,32.935");
29          }
30          configurator: Ipv6FlatNetworkConfigurator {
31              @display("p=718.6887,30.5825");
32          }
33          mobile: WirelessHost6 {
34              @display("p=203,38;i=device/pocketpc");
35          }
36          r1: HomeAgent6 {
37              @display("p=290,208");
38          }
39          ap1: AccessPoint {
40              @display("p=252,111");
41          }
42          r2: Router6 {
43              @display("p=444,207");
44          }
45          ap2: AccessPoint {
46              @display("p=469,104");
47          }
48          internet: Router6 {
49              @display("p=374,277;i=misc/cloud");
50          }
51          rC: Router6 {
52              @display("p=487,327");
53          }
54          static: CorrespondentNode6 {
55              @display("p=606,325;i=device/pc");
56          }
57      connections:
58          ap1.ethg++ <--> ethernetline <--> r1.ethg++;
59          ap2.ethg++ <--> ethernetline <--> r2.ethg++;
60
61          r1.ethg++ <--> ethernetline <--> internet.ethg++;
62          r2.ethg++ <--> ethernetline <--> internet.ethg++;
63          internet.ethg++ <--> ethernetline <--> rC.ethg++;
64          static.ethg++ <--> ethernetline <--> rC.ethg++;
65
66  }
```

Capítulo 2

Preguntas acerca de la simulación

2.1 Pregunta 1

Muestre una captura de los 4 paquetes DHCP intercambiados entre el nodo móvil y el servidor DHCP durante el proceso de obtención de IP en el escenario 1, filtrando para que solo aparezcan el nodo móvil, el punto de acceso y el servidor DHCP. Explique la función de cada paquete.

#465	1.965825815057	mobile --> ap1 DHCPDISCOVER	<unspec>:68	255.255.255.255:67	UDP	DATA	343 B
#475	1.965964625321	ap1 --> dhcp1 DHCPDISCOVER	<unspec>:68	255.255.255.255:67	UDP		313 B
#479	1.965974625321	ap1 --> mobile WlanAck		0A-AA-00-00-00-01	IEEE 802.11 MACACK		31 B
#502	1.966215125321	dhcp1 --> ap1 DHCPOFFER	8.0.0.100:67	255.255.255.255:68	UDP		340 B
#515	1.966658625321	ap1 --> mobile DHCPDISCOVER	<unspec>:68	255.255.255.255:67	UDP	DATA	343 B
#532	1.967146625321	ap1 --> mobile DHCPOFFER	8.0.0.100:67	255.255.255.255:68	UDP	DATA	367 B
#552	1.967343435543	mobile --> ap1 DHCPREQUEST	<unspec>:68	255.255.255.255:67	UDP	DATA	355 B
#562	1.967486245759	ap1 --> dhcp1 DHCPREQUEST	<unspec>:68	255.255.255.255:67	UDP		325 B
#566	1.967496245759	ap1 --> mobile WlanAck		0A-AA-00-00-00-01	IEEE 802.11 MACACK		31 B
#589	1.967746345759	dhcp1 --> ap1 DHCPACK	8.0.0.100:67	255.255.255.255:68	UDP		340 B
#595	1.967940245759	ap1 --> mobile DHCPREQUEST	<unspec>:68	255.255.255.255:67	UDP	DATA	355 B
#614	1.968192245759	ap1 --> mobile DHCPACK	8.0.0.100:67	255.255.255.255:68	UDP	DATA	367 B

Figura 2.1: Paquetes DHCP intercambiados entre *mobile* y *dhcp1*

Los paquetes enviados son:

DHCPDISCOVER

El nodo móvil envia un paquete DHCPDISCOVER para buscar qué servidores DHCP hay disponibles en la red.

DHCPOFFER

El servidor DHCP recibe el paquete DHCPDISCOVER y responde con un DHCPOFFER para informar al nodo móvil de que él es un servidor DHCP y está disponible, además, en este paquete el servidor DHCP ya incluye la dirección IP que se le va a asignar al nodo móvil.

DHCPREQUEST

El nodo móvil envía una DHCPREQUEST con la dirección IP indicada en el DHCPOFFER recibido previamente. El nodo móvil envía este paquete para solicitar y validar la dirección

IP indicada en el paquete DHCPOFFER.

DHCPACK

El servidor responde con un DHCPACK, con la dirección IP y los parámetros de red necesarios, confirmando así la dirección IP. Una vez enviado este paquete, el nodo móvil ya tiene una dirección IP asignada (la indicada en el paquete DHCPACK).

ACLARACIÓN: Hay dos paquetes enviados por el punto de acceso al nodo móvil que no realizan ninguna función, DHCPDISCOVERY (Event #515) y DHCPREQUEST (Event #595), estos paquetes son el DHCPDISCOVERY (Event #465) y DHCPREQUEST (Event #552) que el propio nodo móvil había enviado previamente. El nodo móvil recibe estos paquetes porque el punto de acceso recibió ese mismo paquete proveniente del nodo móvil y con dirección de broadcast como destino, por tanto, el punto de acceso reenviará este paquete a todos los equipos de la red, incluído el nodo móvil, por eso recibe sus propios paquetes (DHCPDISCOVERY y DHCPREQUEST).

2.2 Pregunta 2

¿Qué direcciones IP origen y destino utiliza cada paquete DHCP? Explique su motivo.

DHCPDISCOVER

IP origen: unspecified **IP destino:** 255.255.255.255

Se usa como dirección de origen *unspecified* porque el nodo móvil aún no tiene ninguna IP asignada.

Se usa como dirección de destino la de broadcast porque el nodo móvil no sabe donde está el servidor DHCP (no conoce su dirección IP).

NOTA: **DHCPOFFER**

IP origen: 8.0.0.100 **IP destino:** 255.255.255.255

Se usa como dirección de origen la del servidor DHCP (comportamiento habitual).

Se usa como dirección de destino la de broadcast porque el servidor DHCP aún no conoce la dirección IP del nodo móvil, basicamente porque aún no existe, es el propio servidor DHCP quién se la está asignando.

DHCPREQUEST

IP origen: unspecified **IP destino:** 255.255.255.255

Se usa como dirección de origen *unspecified* porque el nodo móvil sigue sin tener una dirección IP asignada.

Como dirección de destino se sigue usando la dirección de broadcast, cuando se podrían enviar los paquetes a la dirección IP del servidor DHCP (el nodo móvil ahora sí que la conoce),

pero se siguen usando direcciones de broadcast para que así, si existen otros servidores DHCP en la red, sepan que se está realizando dicha asignación y así no intenten realizar la misma asignación o asignar la misma dirección IP a otro equipo.

DHCPACK

IP origen: 8.0.0.100 **IP destino:** 255.255.255.255

Se usa como dirección de origen la del servidor DHCP (comportamiento habitual).

Se usa como dirección de destino la de broadcast porque el servidor DHCP aún sigue sin conocer la dirección IP del nodo móvil, realmente sí que la conoce, él mismo se la está asignando en este paquete, pero por esta misma razón no puede enviar este paquete a la dirección IP que se le va a asignar, porque hasta que el nodo móvil reciba el DHCPACK, no tendrá ”oficialmente” configurada esa dirección IP, es decir, no recibirá paquetes enviados a esa dirección IP.

2.3 Pregunta 3

Muestre el contenido del paquete DHCPOFFER, explicando los campos más relevantes.

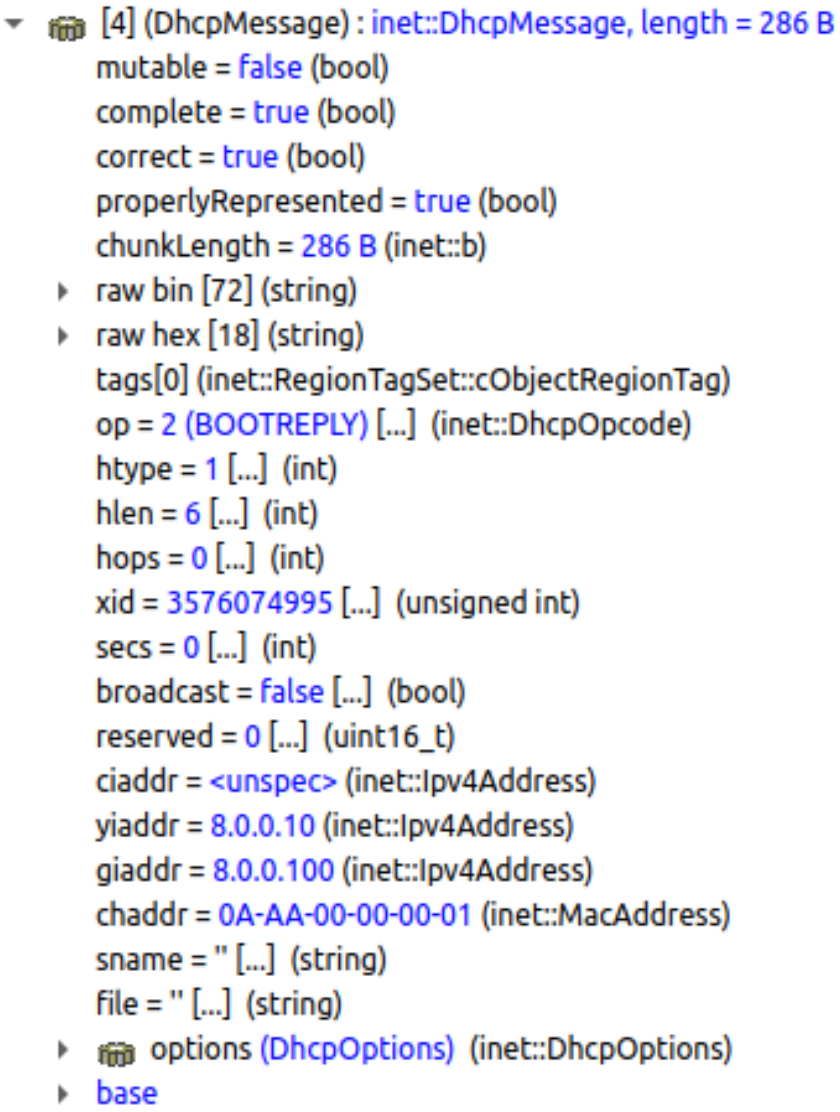

Figura 2.2: Paquete DHCPOFFER enviado de *dhcp1* a *mobile*

En esta captura vemos los siguientes campos relevantes:

- *ciaddr*: Contiene la dirección IP del cliente DHCP (el nodo móvil) que, como aún no tiene una IP asignada su valor es *unspecified* (sin especificar).
- *yiaddr*: Contiene la dirección IP que el servidor DHCP le ofrece al nodo móvil, es decir, la futura dirección IP del nodo móvil si todo sucede con normalidad (8.0.0.10).
- *giaddr*: Contiene la dirección IP del dispositivo que actúa como puerta de enlace, típicamente si las peticiones DHCP pasan por el router, este fija el campo con su dirección IP, como estos paquetes no están siendo enrutados (van del punto de acceso directamente

al servidor DHCP), este campo contiene la dirección IP del servidor DHCP (8.0.0.100).

- *chaddr*: Contiene la dirección MAC del cliente (0A-AA-00-00-00-01).

Ahora observemos el apartado de opciones DHCP.

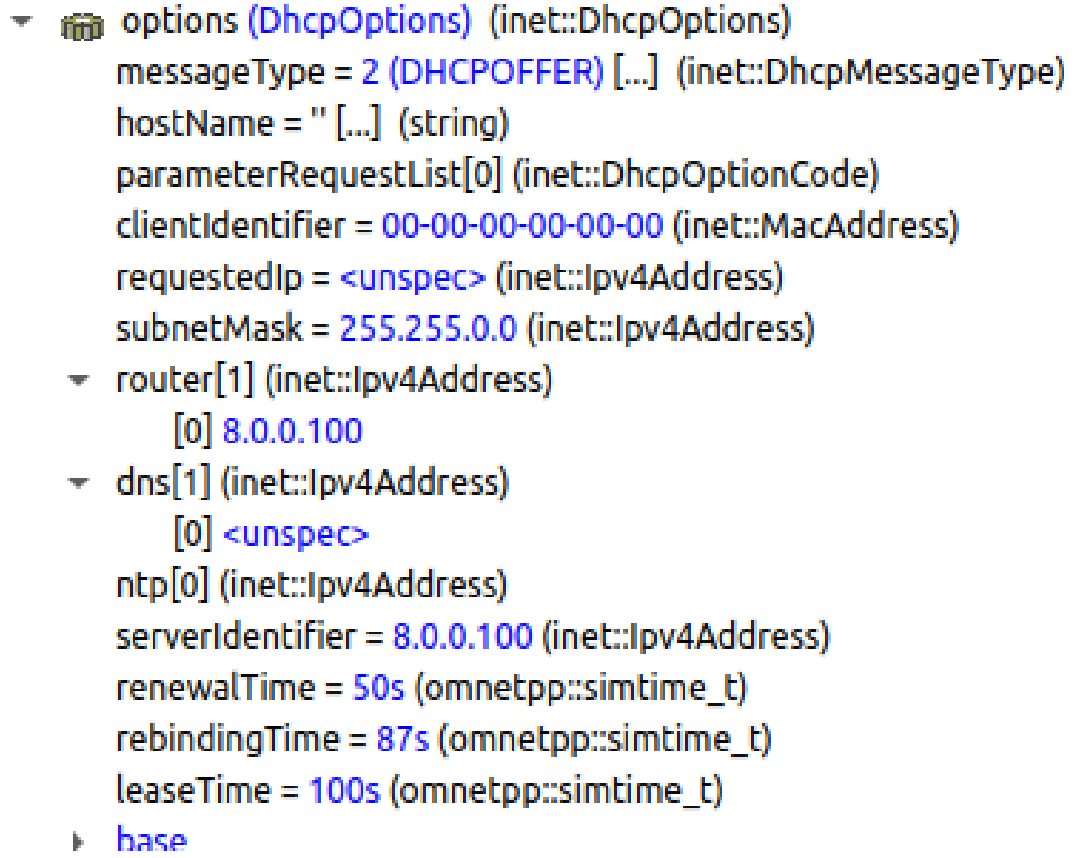

Figura 2.3: Paquete DHCPOFFER enviado de *dhcp1* a *mobile* (opciones)

- *messageType*: Indica el tipo de paquete DHCP que se está enviando, en este caso, DHCPOFFER.

- *clientIdentifier*: Identificador del cliente, de momento ninguno, este campo lo fijará *mobile* con el valor del campo *chaddr* (su dirección MAC) cuando envíe la DHCPREQUEST.

- *requestedIp*: Indica la dirección IP solicitada, de momento ninguna, este campo lo fijará *mobile* con el valor del campo *yiaddr* (que es la IP que le propone el servidor) cuando envíe la DHCPREQUEST.

- *subnetMask*: Máscara de red (255.255.0.0).

- *router*: Lista de direcciones de los routers de la subred del cliente (*mobile*), como de momento no se conocen, la dirección IP del servidor DHCP (8.0.0.100).
- *dns*: Lista de direcciones de los servidores disponibles en la subred del cliente (*mobile*), en este caso, ninguno.
- *serverIdentifier*: Dirección IP del servidor DHCP (8.0.0.100).
- *leaseTime*: Tiempo de concesión de la IP, en este caso, 100s (lo indicado en el omnetpp.ini).
- *renewalTime*: Tiempo que tiene que pasar (50s) para que el ciente renueve el *leaseTime*, es decir, cada 50s *mobile* pasará al estado RENEWING y tendrá que negociar de nuevo un *leaseTime* con el servidor DHCP.
- *rebindingTime*: Tiempo que tiene que pasar para que el cliente considere que el servidor DHCP ya no está disponible y busque otro servidor DHCP (87s), es decir, si pasa el *renewalTime*, *mobile* intentará negociar de nuevo un *leaseTime* y, si no lo consigue, al haber pasado 87s desde la última negociación, *mobile* considerará que el servidor DHCP ya no está disponible, por tanto, pasará a estado REBINDING y enviará un DHCPDISCOVERY para buscar nuevos servidores DHCP disponibles en la red.

2.4 Pregunta 4

En el escenario (1), ¿qué ocurre cuando el nodo móvil abandona la cobertura de AP1 y entra en la de AP2? Muestre una captura de la ventana de error que aparece y explique el motivo. ¿Qué ocurriría en un escenario real en el que un nodo con una conexión TCP abierta cambia de IP?

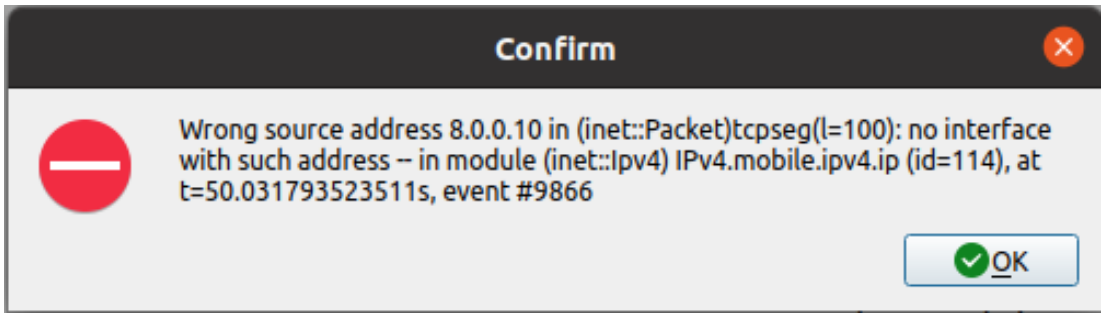

Figura 2.4: Ventana de error OMNET++ (*Wrong source address*)

Cuando el nodo móvil abandona la cobertura de AP1 y entra en la de AP2 vuelve a solicitar al servidor DHCP (*dhcp2* en este caso) una dirección IP y se le asigna, por tanto, a partir de ese momento la dirección IP del nodo móvil es 8.1.0.10.

Hasta aquí parece que todo debería funcionar, sin embargo, el nodo móvil tiene una conexión TCP abierta con el nodo *static*, las conexiones TCP se definen como [IP origen, Puerto origen, IP Destino, Puerto destino].

Cuando el nodo móvil cambia de cobertura y se asigna una nueva dirección IP, los parámetros de la conexión TCP siguen siendo los mismos, por tanto, cuando el nodo móvil intenta enviar un paquete TCP a *static*, lo intenta enviar con la dirección IP de origen 8.0.0.10, su dirección IP antigua, que es la que tiene definida en los parámetros de la conexión TCP.

Al procesar la capa IP del paquete que se intenta enviar, OMNET detecta que la dirección que se quiere poner como origen (8.0.0.10) no está asignada a ninguna de las interfaces del nodo móvil y devuelve este error.

En un escenario real, se aplicaría la solución que conocemos como *Two-Tier IP addressing* en la que el nodo móvil mantendría su dirección IP original 8.0.0.10 (Home Address), pero usaría una dirección IP temporal 8.1.0.10 (Care-of Address), evitando así este problema ya que puede enviar paquetes TCP con origen 8.0.0.10 encapsulados en un paquete con origen 8.1.0.10 (su IP actual), esto es lo que se conoce como *tunneling (IP-over-IP)*.

La dirección de origen del paquete no daría problemas porque es 8.1.0.10, además, cuándo este paquete llegue al router *r1* (*Home Agent*) de su red original (*Home Network*), este mismo desencapsularía el paquete y enviaría al nodo *static* el paquete interno con la dirección IP original 8.0.0.10.

2.5 Pregunta 5

¿Qué ocurre en el escenario (2) durante el paso de AP1 a AP2? ¿Se pierde algún segmento TCP durante la transmisión? Muestre una captura del tráfico de paquetes filtrando para que solo se muestren los nodos static y su router RC en la que se vea el número de secuencia de los segmentos. Explique lo que se observa

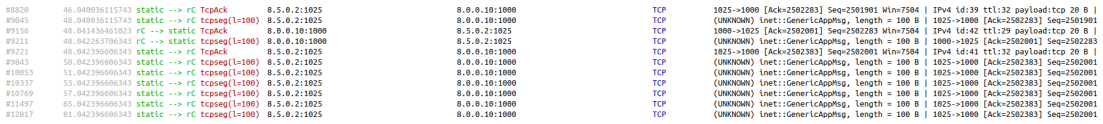

Figura 2.5: Paquetes TCP enviados entre *rC* y *static*

En la captura se muestran también paquetes previos al paso de *ap1* a *ap2* para apreciar las diferencias entre una comunicación normal y la comunicación durante el cambio de cobertura.

En una comunicación normal, *static* recibe un ACK de *mobile* (Event #9156) con el número de secuencia del próximo segmento que vaya a enviar (**2502001**) además de un segmento con número de secuencia **2502283**.

Ahora *static* responde a *mobile* con un ACK=**2502383**, que es el resultado de coger el número de secuencia anterior (**2502283**) y sumarle 100. Además envía un segmento con el número de secuencia que le indicó *mobile* en el ACK mencionado previamente (**2502001**).

La comunicación habitual sería así constantemente, ahora *mobile* recibiría estos dos paquetes y respondería con un ACK=2502101 (2502001+100) y un segmento con número de secuencia 2502383.

Pero, **¿Qué ocurre en este caso?**

En este caso, como el nodo *mobile* cambia de cobertura, su dirección IP ya no es 8.0.0.10, *static* envía segmentos con número de secuencia **2502001** esperando recibir el ACK correspondiente (**2502101**), pero no lo recibe en ningún momento porque los segmentos que está enviando no llegan en ningún momento a *mobile* y, por tanto, este no responde.

Los paquetes que se ven en la captura, son intentos consecutivos de enviar el mismo paquete con el mismo número de secuencia (**2502001**) esperando obtener una respuesta.

Entonces, **¿Se pierde algún segmento?**

El segmento que no se ha podido enviar se pierde, pero si la conexión TCP no finaliza y *static* vuelve a tener a su alcance a *mobile*, es decir, cuándo *mobile* vuelva a conectarse a *ap1* y se le asigne su antigua dirección IP (8.0.0.10) por DHCP, *static* enviará el segmento de nuevo y *mobile* lo recibirá correctamente.

NOTA: En este caso no ocurre lo mencionado previamente (al menos en mi simulación) porque, entre cada intento de reenvío del segmento, *static* espera cada vez más tiempo (porque así lo define TCP), entonces, durante el tiempo que pasa *mobile* conectado a *ap1*, *static* no intenta reenviar el segmento de nuevo y *mobile* no recibe nada, podría coincidir que sí, pero en este caso se da la casualidad de que no.

2.6 Pregunta 6

En este escenario, ¿cada cuánto tiempo se produce un intento de reenvío de paquete por parte del nodo static sin recibir ACK? ¿Qué progresión sigue?

Si observamos la captura anterior (Figura 2.5), vemos que el primer segmento (Event #9843) se envía en t=50,04s, el siguiente en t=51,04s, el siguiente en t=53,04s, el siguiente en t=57,04s... y así sucesivamente. Se ve claramente que la progresión que sigue es que cada tiempo de espera es el doble que el anterior, sigue la progresion 1s-2s-4s-8s-16s...

De esta información podemos deducir 2 aspectos de la implementación del cliente TCP:

1. El cliente TCP utiliza el protocolo ARQ de parada y espera (Figura 2.6), es decir, el cliente no envía datos nuevos (los siguientes segmentos) hasta confirmar que el receptor ha recibido correctamente los datos anteriores (espera a que llegue un ACK), a no ser que el timeout de retransmisión (RTO) finalice, en ese caso retransmite los datos.

2. El cliente esteblece un *Retransmission Timeout (RTO)* inicial igual a 1s y, para cada retransmisión, duplica el valor del RTO (RTO=RTO*2).

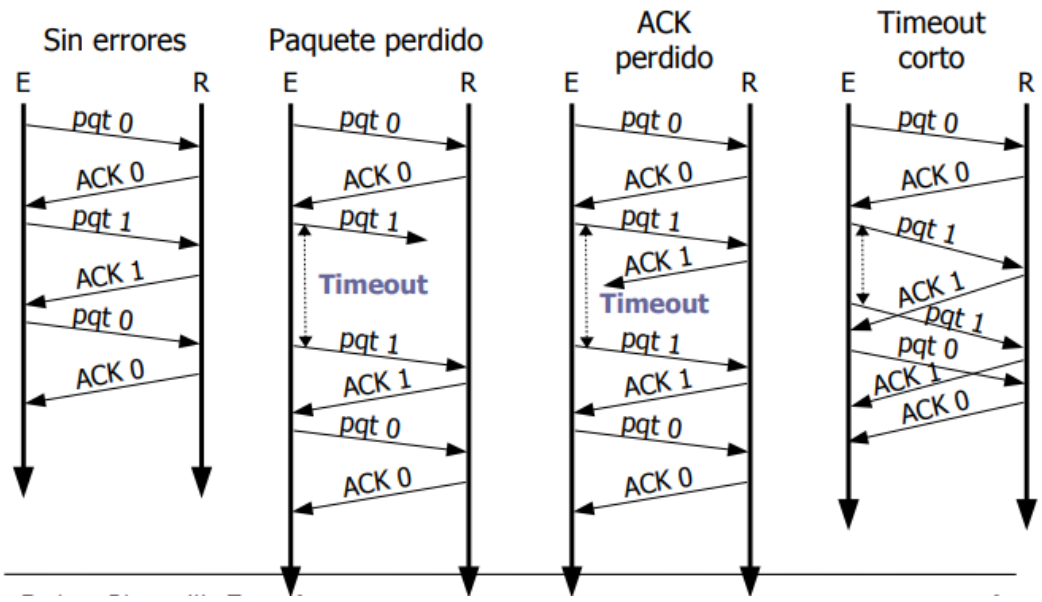

Figura 2.6: Protocolo ARQ de parada y espera (*Stop and Wait ARQ*)

2.7 Pregunta 7

En el escenario (3), ¿qué ocurre cuando el nodo sale de la cobertura de AP1 y entra en la de AP2? ¿Y cuando vuelve a la cobertura de AP1? Muestre capturas del nivel de aplicación del nodo mobile y del nodo static que muestren el número de paquetes enviados y recibidos (doble click en cada uno de los nodos). ¿Cuántos datagramas UDP se pierden durante toda la transmisión?

Para este apartado se ajusto la velocidad a la que se mueve el nodo móvil a 16.66 mps(m/s), de forma que tarde 120s en dar una vuelta completa, lo que significa que pasará unos 60s en cada punto de acceso aproximadamente, como se indica en el enunciado. Los cálculos realizados para obtener esta velocidad fueron:

Distancia eje horizontal = constraintAreaMaxX - constraintAreaMinX = 500 - 0 = 500m
Distancia eje vertical = constraintAreaMaxY - constraintAreaMinY = 500 - 0 = 500m

Como se mueve recorriendo un rectángulo:

Distancia a recorrer = (Distancia eje horizontal + Distancia eje vertical)*2 = (500 + 500)*2 = 2000m

Si queremos que tarde 120s en recorrer 2000m:

speed = 2000m / 120s = **16.66 m/s**

ACLARACIÓN: La velocidad no se ajustó antes porque para los apartados anteriores era irrelevante, se produciría el mismo resultado en otros instantes de tiempo, la velocidad usada en los apartados anteriores era 10m/s.

Simulamos hasta t=40s, instante (aproximado) en el que *mobile* cambia de cobertura, cambia en el segundo 40 por la posición en la que comienza, es decir, una vuelta completa teniendo en cuenta la posición inicial sería algo como: 40s en la cobertura de *ap1* - 60s en la cobertura de *ap2* - 20s en la cobertura de *ap1*.

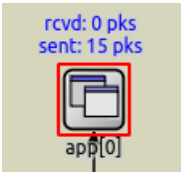

Figura 2.7: Paquetes UDP enviados por *mobile* a *static* (t=40s)

Figura 2.8: Paquetes UDP recibidos en *static* provenientes de *mobile* (t=40s)

Como vemos en la capturas, se han perdido 5 datagramas UDP (paquetes enviados - paquetes recibidos = 15 - 10 = 5).

Ahora simulamos hasta t=115s, momento en el que *mobile* vuelve a estar conectado a *ap1*, a partir del segundo 100 como se menciona previamente *mobile* ya está en la cobertura de ap1, pero hasta t=115s no tiene una IP asignada en la red de *ap1*.

Figura 2.9: Paquetes UDP enviados por *mobile* a *static* (t=115s)

Figura 2.10: Paquetes UDP recibidos en *static* provenientes de *mobile* (t=115s)

Ahora vemos que *mobile* ha enviado 53 paquetes, sin embargo, el nodo *static* no recibió ningun paquete desde t=40s (sigue teniendo paquetes recibidos = 10), en este caso, se han perdido 43 datagramas UDP.

Entonces, ¿Qué está pasando exactamente?

Aquí el problema viene dado por la actualización de la MAC de la puerta de enlace. En Mobile IP existen dos parámetros importantes que se especifican en la cabecera *Ieee80211DataHeader* de capa 2:

- *receiverAddress:* Dirección MAC del punto de acceso, en este caso, debería ser la dirección MAC de *ap2*.
- *address3*: Dirección MAC del router o puerta de enlace, en este caso, debería ser la dirección MAC de *r2*.

Ahora veamos esta cabecera en un paquete UDP enviado por *mobile* cuando tenía asignada una IP en la red de *ap2*.

Cuando *mobile* cambia de cobertura y se le asigna una IP en la red de AP2 no actualiza correctamen

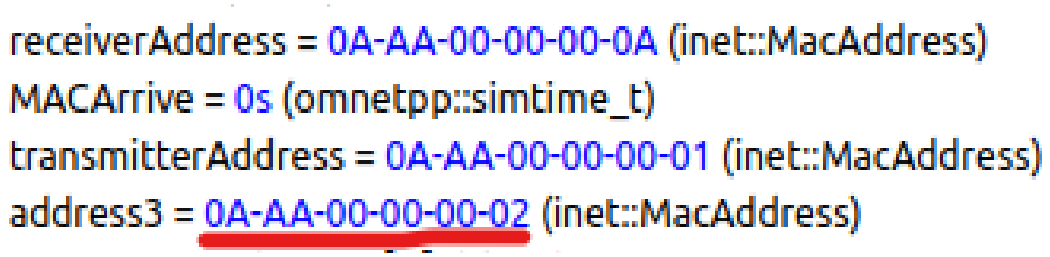

Figura 2.11: Cabecera *Ieee80211DataHeader* de un paquete UDP enviado por *mobile* al nodo *static*

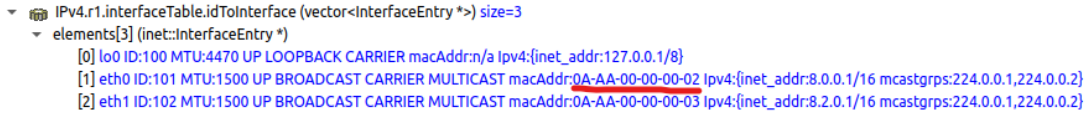

Figura 2.12: Tabla de interfaces del router *r1*

Como vemos en las capturas, el campo *address3* contiene la dirección MAC de *r1*, cuando tendría que contener la dirección MAC de *r2*, entonces lo que está pasando es que *mobile* envía datagramas UDP, y cuando estos llegan a *r2*, este comprueba si el campo *address3* coincide con su MAC, como no coincide, descarta los paquetes.

Por esta razón, *static* no recibe ningún datagrama UDP durante todo el tiempo que *mobile* está en la cobertura de *ap2*. Realmente, este problema se solucionaría cuando la entrada de la ARP Caché de *mobile* correspondiente a *r1* expirase y *mobile* realizase la resolución ARP correspondiente para saber la MAC de *r2*. Sin embargo, en este caso el tiempo que *mobile* pasa en la cobertura de *ap2* no es suficiente para que la entrada de la ARP Caché expire, por tanto, no se recibe ni un solo datagrama UDP en todo ese intervalo de tiempo (entre t=40s y t=115s).

Cuando *mobile* vuelve a AP1, como sigue teniendo la MAC de *r1* registrada en su ARP Caché, los paquetes volveran a llegar correctamente, si avanzamos un poco más la simulación hasta t=130s, vemos que los siguientes datagramas UDP enviados llegan correctamente.

Figura 2.13: Paquetes UDP enviados por *mobile* a *static* (t=130s)

Figura 2.14: Paquetes UDP recibidos en *static* provenientes de *mobile* (t=130s)

Como vemos ahora se perdieron 45 paquetes, es decir, se perdieron 2 paquetes más desde el instante anterior y después los paquetes ya comenzaron a llegar correctamente.

Por último, **¿Cuántos paquetes se pierden durante toda la transmisión?**

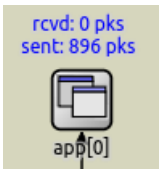

Figura 2.15: Paquetes UDP enviados por *mobile* a *static* (t=1800s)

Figura 2.16: Paquetes UDP recibidos en *static* provenientes de *mobile* (t=1800s)

Durante toda la transmisión se pierden 896-211 = **685 paquetes**, lo que supone un 685/896= 76,45% del tráfico UDP (inviable en un escenario real).

2.8 Pregunta 8

En el escenario (4), ¿qué ocurre al intercambiar los roles en UDP? ¿Cuántos paquetes se pierden ahora? Muestre de nuevo capturas del nivel de aplicación del nodo mobile y del nodo static.

Al intercambiar los roles UDP, el problema está en que el nodo *static*, que ahora es el cliente UDP, no sabe cuando el nodo *mobile* (servidor UDP) cambia de dirección, entonces siempre enviará los paquetes UDP a la misma dirección, 8.0.0.10, que es la dirección que toma inicialmente *mobile*.

Por tanto, cuando *mobile* esté conectado a *ap1* recibirá los paquetes UDP y cuando esté conectado a *ap2* no.

Figura 2.17: Paquetes UDP enviados por *static* a *mobile* (t=1800s)

Figura 2.18: Paquetes UDP recibidos en *mobile* provenientes de *static* (t=1800s)

En este caso se pierden 896-222 = **674 paquetes**. ¿Porqué se pierden menos paquetes?

Bueno, esto depende del tiempo que el servidor UDP esté recibiendo paquetes, en el caso anterior si se cumpliese lo explicado previamente (2.7), deberían perderse exactamente los mismos paquetes porque aunque se debiese a una situación distinta, estaría pasando exactamente lo mismo, el servidor UDP solo recibe paquetes cuando el nodo *mobile* está conectado a *ap1* y el nodo *mobile* pasa exactamente el mismo tiempo conectado a *ap1* en las dos simulaciones.

Lo que ocurre en este caso es que, en algún/os instante/s de tiempo en que el nodo *mobile* estaba conectado a *ap2* en la configuración anterior, la entrada correspondiente a *r1* de la ARP Caché de *mobile* expiró, por tanto, creó una entrada para *r2* y empezó a recibir datagramas UDP estando conectado a *ap2*.

Pero, ¿Esto no provoca que aún lleguen correctamente más paquetes en el caso anterior?

Depende, en este caso no, esto pasa porque si el nodo móvil cambia la dirección MAC del router justo cuando le queda poco para cambiar de cobertura, sí, comienza a recibir paquetes en la cobertura en la que está, pero casi no recibe ningún paquete porque queda poco tiempo hasta que cambie de cobertura y, cuando cambie, va a tener el mismo problema que tenía en la cobertura de *ap2* en la cobertura de *ap1* (ahora tiene registrada en la ARP Caché la dirección MAC de *r2* estando conectado a *ap1*), provocando que al final sea menor la cantidad de tiempo en que los paquetes se reciben correctamente.

2.9 Pregunta 9

En el escenario (5), explique el proceso de creación del túnel con capturas de: 1) los paquetes intercambiados entre mobile y R1 (home agent); 2) el log de R1 filtrando a nivel mipv6support; y 3) la bindingCache del home agent tras la creación

del túnel.

NOTA: Revisar la terminología especificada en la introducción (Sección página 3), porque es la que se usará de ahora en adelante.

El túnel se crea cuando MN cambia de cobertura, es decir, se mueve de su red original (*Home Network*) a la nueva red (*Foreign Network*). Una vez MN, recibe un paquete RA y detecta que se ha movido en capa 3 (ha cambiado de red), empieza a comunicarse con HA para establecer su CoA.

En primer lugar, *mobile* (MN) envía un paquete BU (*BindingUpdate*) a *r1* (HA) para indicarle que se ha movido de red y enviarle su nueva dirección IP.

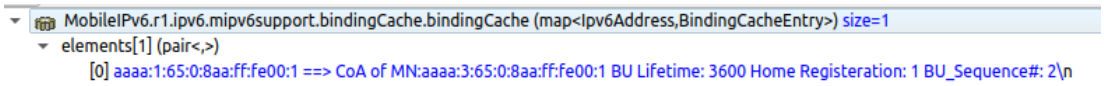

Figura 2.19: Paquetes BU y BAck enviados entre *mobile* (MN) y *r1* (HA)

Ahora *r1* (HA) confirma que ha recibido el paquete respondiendo con un paquete BAck (*Binding Acknowledgement*), además, *r1* (HA) actualiza su Binding Cache con la correspondencia entre la Hoa y la Coa del nodo *mobile* (MN), para interceptar todos los paquetes que vayan destinados a este nodo y reenviárselos a través del túnel.

Una vez *r1* (HA) envía el BAck el túnel ya ha sido creado y todos los paquetes enviados a la HoA de *mobile* (MN), los interceptará *r1* (HA) y se los reenviará a *mobile* (MN) a través del túnel.

```
** Event #12158  t=32.445790062984  MobileIPv6.r1.ipv6.mipv6support.xMobileIPv6 (xMIPv6, id=156)  on Binding Update (inet::Packet, id=106176)
INFO: Received MIPv6 related message
INFO:Processing of MIPv6 related mobility message
INFO:Message recognised as BINDING UPDATE (BU)
INFO:Entered BU processing method
INFO:
INFO:<<<<<<<<<ROUTINE WHERE BU GETS VALIDATED>>>>>>>>>>>>>>>>>
INFO:BU validation passed
INFO:
INFO:++++++++++++++++++++Binding Cache Being Updated in Routing Table6 ++++++++++++++
INFO:Scheduled BC expiry for time 3632.445790062984s
INFO:Appending ControlInfo to mobility message
INFO:ControlInfo appended successfully. Sending mobility message to Ipv6 module
DETAIL:controlInfo: DestAddr=aaaa:3:65:0:8aa:ff:fe00:1 SrcAddr=aaaa:1:65:0:8aa:ff:fe00:2 InterfaceId=102
INFO:Tunneling: Created tunnel with entry=aaaa:1:65:0:8aa:ff:fe00:2, exit=aaaa:3:65:0:8aa:ff:fe00:1 and trigger=aaaa:1:65:0:8aa:ff:fe00:1
** Event #12564  t=33.445790014647  MobileIPv6.r1.ipv6.mipv6support.xMobileIPv6 (xMIPv6, id=156)  on Binding Update (inet::Packet, id=106812)
INFO: Received MIPv6 related message
INFO:Processing of MIPv6 related mobility message
INFO:Message recognised as BINDING UPDATE (BU)
INFO:Entered BU processing method
INFO:
INFO:<<<<<<<<<ROUTINE WHERE BU GETS VALIDATED>>>>>>>>>>>>>>>>>
INFO:BU validation passed
INFO:
INFO:++++++++++++++++++++Binding Cache Being Updated in Routing Table6 ++++++++++++++
INFO:Scheduled BC expiry for time 3633.445790014647s
INFO:Appending ControlInfo to mobility message
INFO:ControlInfo appended successfully. Sending mobility message to Ipv6 module
DETAIL:controlInfo: DestAddr=aaaa:3:65:0:8aa:ff:fe00:1 SrcAddr=aaaa:1:65:0:8aa:ff:fe00:2 InterfaceId=102
INFO:Destroy tunnel entry =aaaa:1:65:0:8aa:ff:fe00:2, exit = aaaa:3:65:0:8aa:ff:fe00:1, destTrigger = aaaa:1:65:0:8aa:ff:fe00:1
INFO:Tunneling: Created tunnel with entry=aaaa:1:65:0:8aa:ff:fe00:2, exit=aaaa:3:65:0:8aa:ff:fe00:1 and trigger=aaaa:1:65:0:8aa:ff:fe00:1
```

Figura 2.20: Log de *r1* (HA) filtrando a nivel *mipv6support*

En el log podemos ver más en detalle lo que ocurre. Cuando *r1* (HA) recibe el paquete BU, valida que el paquete sea correcto y lee la nueva dirección IP de *mobile* (MN), la cuál este mismo nodo le indicó en el paquete BU. Una vez hecho esto, *r1* (HA) crea el túnel con su dirección como entrada (entry=aaaa:1:65:0:8aa:ff:fe00:2), la dirección actual de *mobile* (CoA) como salida (exit=aaaa:3:65:0:8aa:ff:fe00:1) y la dirección original de *mobile* (HoA) como trigger (trigger=aaaa:1:65:0:8aa:ff:fe00:1).

El trigger del túnel es la dirección de destino que deben tener los paquetes que le llegan a *r1* (HA), para que este los reenvíe a *mobile* (MN) a través del túnel.

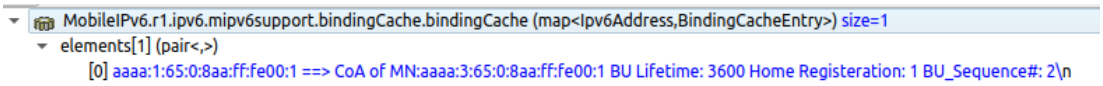

Figura 2.21: Binding Cache de *r1* (HA) tras la creación del túnel

Como se mencionó previamente, cuando *r1* (HA) recibe el paquete BU actualiza su Binding Cache con la correspondencia entre la HoA y la CoA de *mobile* (MN).

HoA = **aaaa:1:65:0:8aa:ff:fe00:1**
CoA = **aaaa:3:65:0:8aa:ff:fe00:1**

2.10 Pregunta 10

Explique el proceso de encapsulado y reenvío (paquetes del nodo static a mobile) y de desencapsulado y reenvío (paquetes de mobile a static) en el home agent en el escenario (5) con capturas del log (filtrando por los niveles ipv6/ipv6 e ipv6/ipv6tunneling en el nodo R1).

Proceso de encapsulado y reenvío

Cuando el nodo *static* (CN) envía un paquete a MN, lo enviará a la dirección original (HoA) de *mobile* (MN), por tanto, este paquete llegará a *r1* (HA), veamos en detalle lo que pasa.

```
** Event #13015  t=34.021050425983  MobileIPv6.r1.ipv6.ipv6 (Ipv6, id=149)  on tcpseg(l=100) (inet::Packet, id=107469)
INFO:Routing datagram `' with dest=aaaa:1:65:0:8aa:ff:fe00:1, requested nexthop is <unspec> on unspec interface:
INFO:Looking up tunnels...
DETAIL:found vIf=2147483647
INFO:tunneling: src addr=aaaa:6:66:0:8aa:ff:fe00:f, dest addr=aaaa:1:65:0:8aa:ff:fe00:1
** Event #13016  t=34.021050425983  MobileIPv6.r1.ipv6.iptunneling (Ipv6Tunneling, id=150)  on tcpseg(l=100) (inet::Packet, id=107469)
INFO:Looking up tunnels...
DETAIL:found vIf=2147483647
** Event #13017  t=34.021050425983  MobileIPv6.r1.ipv6.ipv6 (Ipv6, id=149)  on tcpseg(l=100) (inet::Packet, id=107469)
INFO:Routing datagram `' with dest=aaaa:3:65:0:8aa:ff:fe00:1, requested nexthop is <unspec> on unspec interface:
INFO:do longest prefix match in routing table
INFO:finished longest prefix match in routing table
INFO:next hop for aaaa:3:65:0:8aa:ff:fe00:1 is fe80::8aa:ff:fe00:a, interface eth1
DETAIL:link-layer address: 0A-AA-00-00-00-0A
```

Figura 2.22: Log de *r1* (HA) filtrando por nivel *ipv6* e *ipv6tunneling*

r1 (HA) recibe el paquete con la HoA de *mobile* (MN) como destino, entonces, comprueba si tiene algún túnel que tenga esa dirección como trigger y encuentra el túnel creado previamente (vIf=2147483647).

Entonces lo que hará será encapsular el paquete recibido para enviarlo a través del túnel, el paquete interno será el paquete recibido con origen src = aaaa:6:66:0:8aa:ff:fe00:f (Dirección de *static* (CN)) y destino dest = aaaa:1:65:0:8aa:ff:fe00:1 (HoA de *mobile* (MN)) y el paquete externo tendrá como origen la entrada del túnel, es decir, la dirección de *r1* (HA), src = entry = aaaa:1:65:0:8aa:ff:fe00:2 y como destino la salida, es decir, la CoA de *mobile* (MN), dest = exit = aaaa:3:65:0:8aa:ff:fe00:1, el paquete resultante tendría la siguiente forma.

Figura 2.23: Anatomía (a nivel IP) del segmento TCP encapsulado. Paquete externo (gris) y paquete interno (negro).

Proceso de desencapsulado y reenvío

Ahora, cuando *mobile* (MN) recibe el paquete a través del túnel responde con un *TcpAck* (el paquete encapsulado era un segmento TCP), veamos que pasa cuando *r1* (HA) recibe la respuesta.

```
** Event #13119  t=34.022530149007  MobileIPv6.r1.ipv6.ipv6 (Ipv6, id=149)  on TcpAck (inet::Packet, id=107611)
INFO:Routing datagram `' with dest=aaaa:1:65:0:8aa:ff:fe00:2, requested nexthop is <unspec> on unspec interface:
INFO:local delivery
INFO:0 extension header(s) for processing...
INFO:Tunnelled IP datagram
** Event #13120  t=34.022530149007  MobileIPv6.r1.ipv6.iptunneling (Ipv6Tunneling, id=150)  on TcpAck (inet::Packet, id=107611)
** Event #13121  t=34.022530149007  MobileIPv6.r1.ipv6.ipv6 (Ipv6, id=149)  on TcpAck (inet::Packet, id=107611)
INFO:Routing datagram `' with dest=aaaa:6:66:0:8aa:ff:fe00:f, requested nexthop is <unspec> on unspec interface:
INFO:Looking up tunnels...
DETAIL:found vIf=-1
INFO:next hop for aaaa:6:66:0:8aa:ff:fe00:f is fe80::8aa:ff:fe00:a, interface eth1
DETAIL:link-layer address: 0A-AA-00-00-00-0A
```

Figura 2.24: Log de *r1* (HA) filtrando por nivel *ipv6* e *ipv6tunneling*

r1 (HA) recibe el *TcpAck* y ve como dirección de destino su propia IP, entonces detecta que es un paquete que ha sido tunelizado previamente, por tanto, lo desencapsula para ver las IPs

origen y destino del paquete interno, el paquete que recibe *r1* es de la siguiente forma.

Figura 2.25: Anatomía (a nivel IP) del segmento TCP encapsulado. Paquete externo (negro) y paquete interno (gris).

Lo que hará *r1* (HA) exactamente, será leer las direcciones origen y destino marcadas en negro, una vez detecta que el paquete está tunelizado, lo desencapsula y lee las direcciones origen y destino del paquete gris, que son las que usará ahora para enviale el paquete al destino, en este caso, *static* (CN), este recibirá el paquete con origen la HoA de *mobile* (MN).

ACLARACIÓN: Nótese que en la Figura 2.23 los colores negro y gris representan lo contrario que en la Figura 2.25.

2.11 Pregunta 11

Muestre el contenido de un paquete encapsulado IPv6 en el escenario (5) a nivel IPv6Header. ¿Qué IP origen y destino utilizan los paquetes interno y externo de los segmentos TCP enviados de mobile a static? ¿Y los de static a mobile que ya han pasado por el home agent?

Segmento TCP enviado de *mobile* (MN) a *static* (CN)

Figura 2.26: Cabecera externa de un segmento TCP enviado de *mobile* (MN) a *static* (CN).

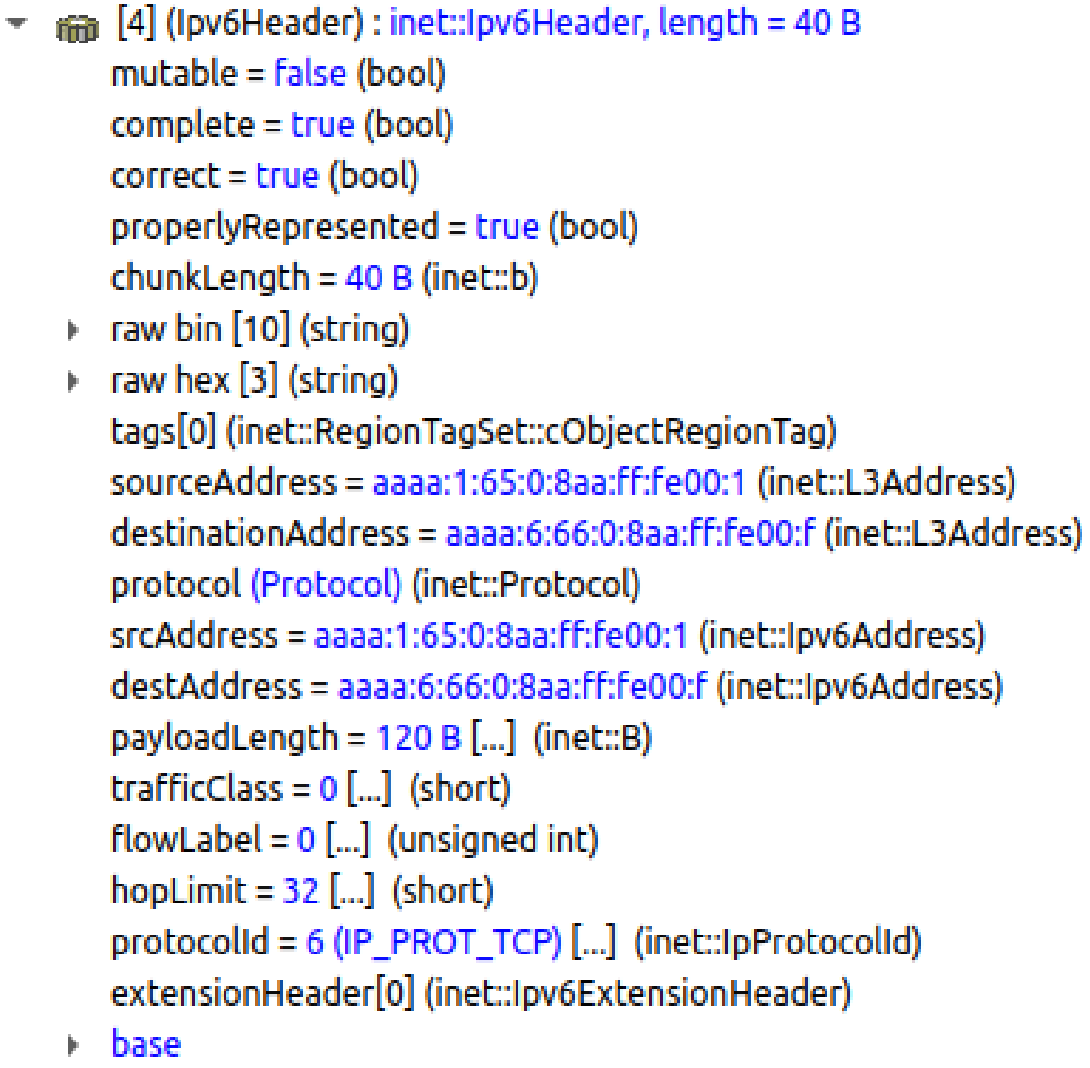

Figura 2.27: Cabecera interna de un segmento TCP enviado de *mobile* (MN) a *static* (CN).

Como el paquete está tunelizado tiene dos cabeceras IP (*IP over IP*). La cabecera interna contiene como dirección origen la HoA de *mobile* (MN) y como dirección destino la IP de *static* (CN). La cabecera externa contiene como dirección origen la CoA de *mobile* (MN) y como dirección destino la IP de *r1* (HA). Esta es la estructura mencionada previamente en la Figura 2.25. Si este mismo paquete lo observásemos una vez *r1* (HA) lo ha reeenviado, solo tendría la cabecera IP interna.

Segmento TCP enviado de *static* (CN) a *mobile* (MN)

Figura 2.28: Cabecera externa de un segmento TCP enviado de *static* (CN) a *mobile* (MN).

Figura 2.29: Cabecera interna de un segmento TCP enviado de *static* (CN) a *mobile* (MN).

De nuevo, como el paquete ya ha pasado por *r1* (HA), está tunelizado y tiene dos cabeceras IP (*IP over IP*). La cabecera interna contiene como dirección origen la IP de *static* (CN) y como dirección destino la HoA de *mobile* (MN). La cabecera externa contiene como dirección origen la IP de *r1* (HA) y como dirección destino la CoA de *mobile* (MN). Esta es la estructura mencionada previamente en la Figura 2.23. Si este mismo paquete lo observásemos antes de que llegase a *r1* (HA) solo tendría la cabecera IP interna.

2.12 Pregunta 12

En comparación con el escenario (1), ¿qué ocurre en el escenario (5) al pasar el nodo móvil al segundo punto de acceso? Muestre capturas del escenario en las que

se visualicen las rutas seguidas por los paquetes en cada punto de acceso.

NOTA: Para este apartado se comentó en el fichero omnetpp.ini la línea ***.interfaceTable.displayAddresses = true* para que no se muestren las interfaces de cada dispositivo porque si no no se veían con claridad las rutas de los paquetes.

Ahora, cuando *mobile* (MN) cambia de cobertura, INET no devuelve ningún error, porque el problema que se daba en el escenario (1), se soluciona con el tunneling de *Mobile IPv6* explicado previamente.

Antes el problema venía dado porque *mobile* (MN) intentaba enviar un paquete con su dirección IP original como origen, la cuál ya no existía en ninguna de sus interfaces al cambiar de cobertura, ahora ese problema se soluciona tunelizando ese paquete, el paquete con la dirección IP original va encapsulado en otro paquete con la dirección IP actual como origen.

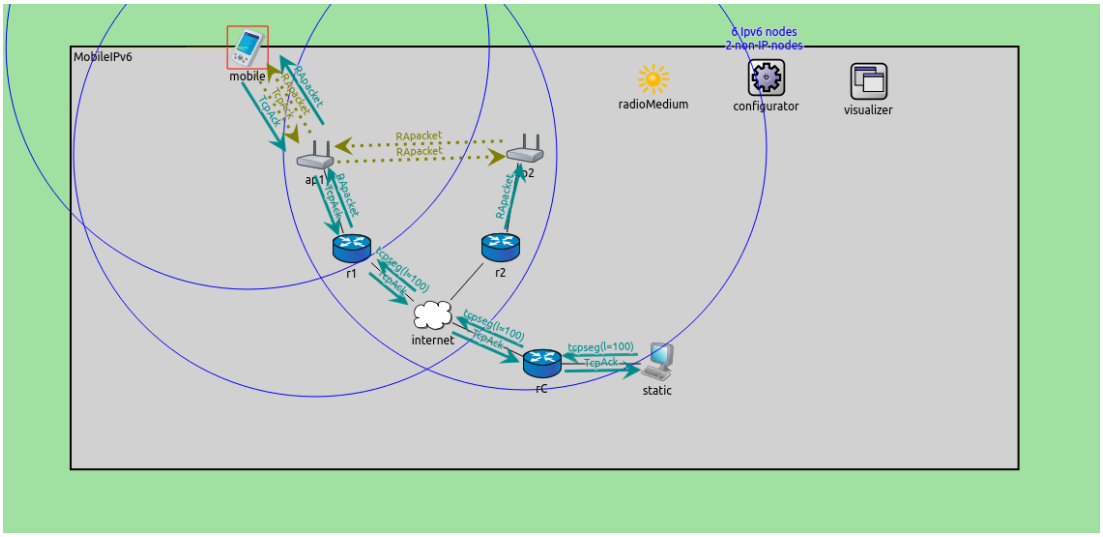

Figura 2.30: Rutas seguidas por los paquetes TCP (Azul) cuando *mobile* (MN) está en la cobertura de *ap1*

En la cobertura de *ap1* la comunicación entre *mobile* (MN) y *static* (CN) sucede con normalidad, los paquetes atraviesan los nodos intermedios hasta llegar al destino correspondiente.

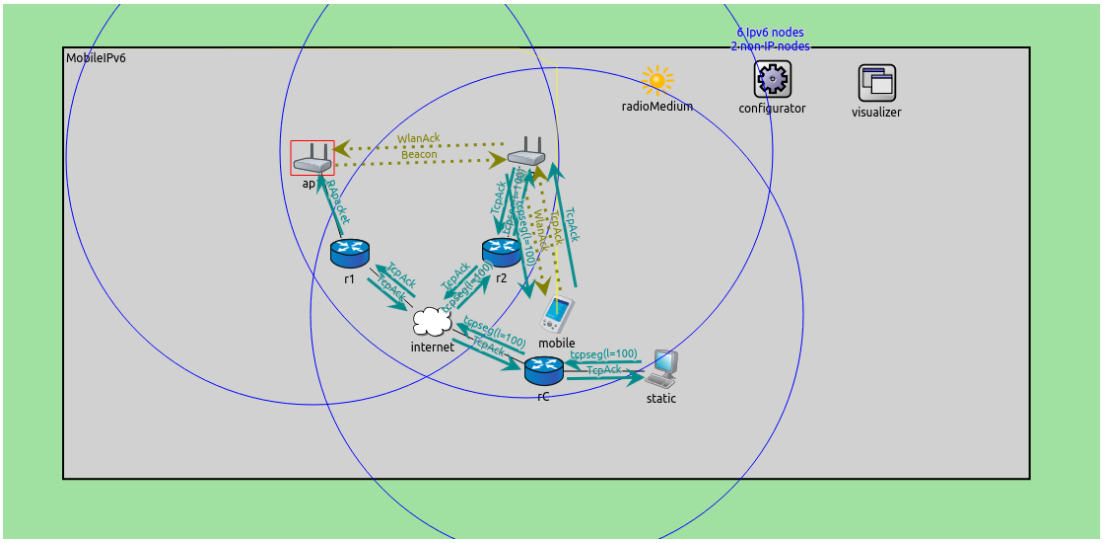

Figura 2.31: Rutas seguidas por los paquetes TCP (Azul) cuando *mobile* (MN) está en la cobertura de *ap2*

En la cobertura de *ap2* la comunicación entre *mobile* (MN) y *static* (CN) se dirige a *r1* (HA) en primer lugar y este se encarga de reenviar el paquete al destino correspondiente.

2.13 Pregunta 13

En comparación con el escenario (2), ¿qué ocurre en el escenario (6) al pasar el nodo móvil al segundo punto de acceso? Muestre capturas del escenario con el nodo móvil en cada punto de acceso.

NOTA: Para este apartado se comentó en el fichero omnetpp.ini la línea ***.interfaceTable.displayAddresses = true* para que no se muestren las interfaces de cada dispositivo porque si no no se veían con claridad las rutas de los paquetes.

Ahora, cuando *mobile* (MN) cambia de cobertura, ya no se pierden paquetes como en el escenario (2), al crear un túnel entre *mobile* (MN) y *r1* (HA), aunque *static* (CN) siga enviando paquetes a la dirección IP original de *mobile* (MN), *r1* (HA) los recibirá y reenviará encapsulados a *mobile* (MN) a través del túnel.

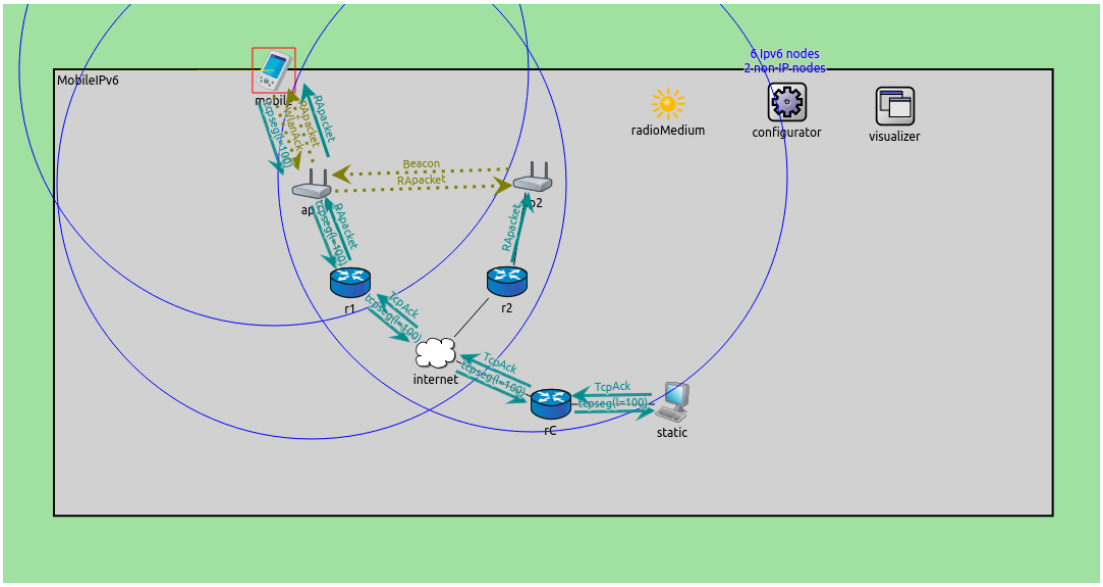

Figura 2.32: Rutas seguidas por los paquetes TCP (Azul) cuando *mobile* (MN) está en la cobertura de *ap1*

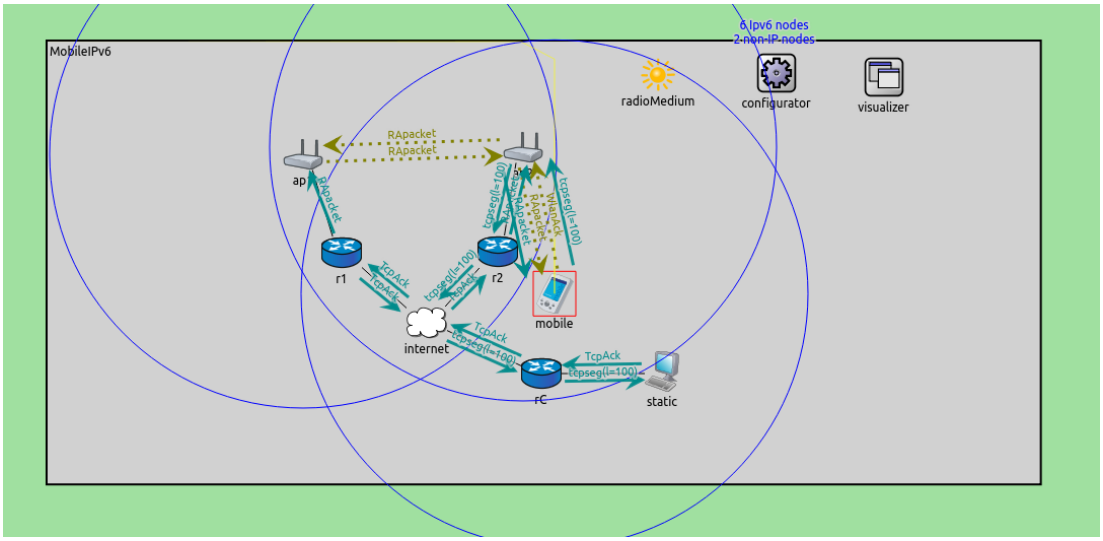

Figura 2.33: Rutas seguidas por los paquetes TCP (Azul) cuando *mobile* (MN) está en la cobertura de *ap2*

No se explica el tráfico de estas capturas porque sucede exactamente lo mismo que en el escenario (5), con la diferencia de que ahora *static* (CN) es el cliente TCP y *mobile* (MN) el servidor TCP (explicado en Sección 2.12 página 34).

2.14 Cambios en el omnetpp.ini debido a un bug

Con la configuración utilizada hasta el momento, al ejecutar los escenarios con *Mobile IPv6* cuando el nodo móvil pasa por cierta zona INET nos devuelve un error.

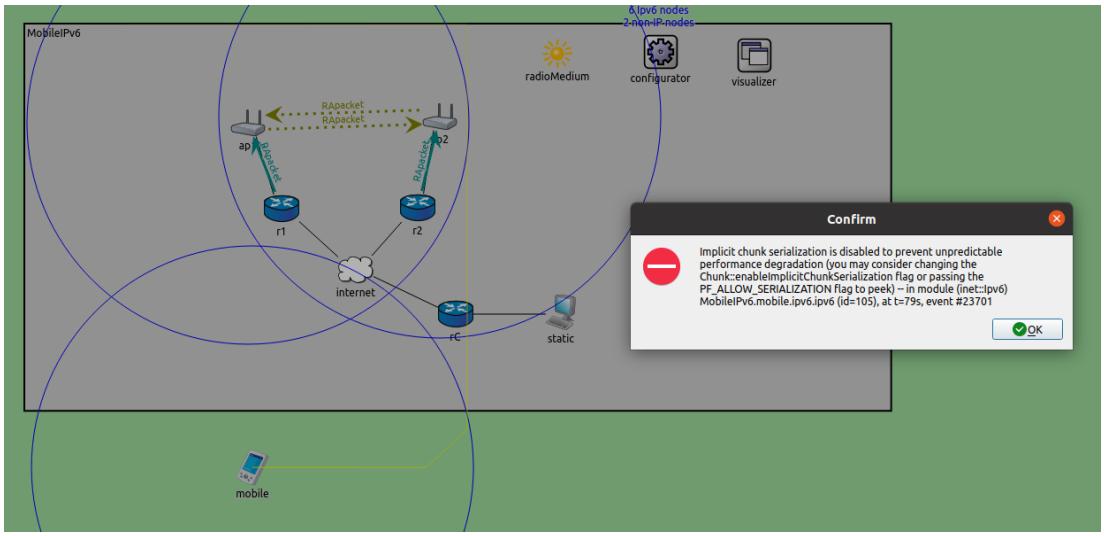

Figura 2.34: Ventana de error (*Implicit Chunk Serialization*)

Estuvimos informándonos sobre el error, al parecer se debe a que se intenta utilizar un chunk de un tipo como si fuese de otro tipo que no tiene ninguna relación con el primero, entonces, no hay forma de hacer una conversión entre estos dos tipos. Sin embargo, aparecen ciertas explicaciones de porqué aparece el error, pero ninguna solución, al parecer es un bug de INET.

En nuestro caso, nos fijamos que nos devuelve el error siempre que el nodo móvil pasa por el mismo punto, entonces probamos a cambiar el rango de movimiento del nodo móvil y ya no nos devuelve el error, aunque seguimos sin saber de qué depende, porque hicimos pruebas variando la velocidad del nodo móvil y con algunas velocidades devuelve el error y con otras no.

Para poder ejecutar la simulación completa y responder a la última pregunta realizamos la siguiente modificación en el fichero omnetpp.ini.

```
**.mobile.mobility.constraintAreaMinX = 0m
**.mobile.mobility.constraintAreaMinY = 0m
**.mobile.mobility.constraintAreaMinZ = 0m
#**.mobile.mobility.constraintAreaMaxX = 500m
#**.mobile.mobility.constraintAreaMaxY = 500m
**.mobile.mobility.constraintAreaMaxX = 700m
**.mobile.mobility.constraintAreaMaxY = 300m
**.mobile.mobility.constraintAreaMaxZ = 0m
```

Figura 2.35: Modificación de la configuración del fichero omnetpp.ini

2.15 Pregunta 14

. ¿Cuántos datagramas se pierden en los escenarios (7) y (8) en comparación con los escenarios (3) y (4)? Muestre capturas de los nodos en las que se vean los paquetes enviados y recibidos y explique la diferencia observada.

IMPORTANTE: Ir a la sección 2.14 página 38 antes de leer esta pregunta, se explica en detalle porque se ha modificado la configuración del fichero omnetpp.ini para responder a esta pregunta.

Figura 2.36: Paquetes UDP enviados por *mobile* a *static* (t=1800s)

Figura 2.37: Paquetes UDP recibidos en *static* provenientes de *mobile* (t=1800s)

En el escenario (7) se pierden 896 - 585 = 311 paquetes.

Figura 2.38: Paquetes UDP enviados por *mobile* a *static* (t=1800s)

Figura 2.39: Paquetes UDP recibidos en *static* provenientes de *mobile* (t=1800s)

En el escenario (8) se pierden 896 - 573 = 323 paquetes

escenario (7) = 311 paquetes « 685 paquetes = escenario (3) escenario (8) = 323 paquetes « 674 paquetes = escenario (4)

Como podemos ver se pierden muchos menos paquetes en el escenario IPv6 que en el escenario IPv4, esto se debe a la gestión de las IPs explicada previamente con el protocolo *Mobile IPv6*, antes, al cambiar de cobertura, el nodo móvil dejaba de enviar o recibir paquetes y ahora puede seguir enviando o recibiendo paquetes independientemente de la cobertura en la que esté.

De todas formas, se siguen perdiendo paquetes, esto se debe a lo que se conoce como *Handover Latency*, que es el tiempo que tarda el nodo móvil desde que sale de su red original hasta que crea el túnel con su HA y ya puede seguir recibiendo paquetes. Durante este tiempo, todos los paquetes enviados se pierden. Además, se pueden perder paquetes debido a otros aspectos de la red como retardos o congestión en los routers.