Facultade de Informática
UNIVERSIDADE DA CORUÑA

DISEÑO DE REDES

Práctica 5 - MANETs en INET

Estudante: Tomé Maseda Dorado
Estudante: Jorge Álvarez Cabado

A Coruña, December de 2021.

Índice Xeral

Índice de Figuras

Introducción

En esta práctica se estudiará en profundidad el funcionamiento de las redes MANET, en concreto, compararemos el funcionamiento de un protocolo reactivo, AODV, frente a un protocolo proactivo, DSDV. Para cada uno, observaremos su funcionamiento en distintas situaciones y, finalmente, compararemos las diferencias de rendimiento entre uno y otro para un escenario concreto.

Capítulo 1

Ficheros utilizados en la simulación

1.1 Ficheros .ini

```
[General]
sim-time-limit = 500s
seed-1-mt = 8321

**.n = 15
#**.radio.transmitter.power = 1.5mW
**.radio.transmitter.power = 2mW

num-rngs = 2
**.mobility.rng-0 = 1

# random mobility model - "Optimized Smooth Handoffs in Mobile IP"
    by Perkins &Wang
**.mobile[*].mobility.typename = "MassMobility"
**.mobile[*].mobility.initFromDisplayString   =   false
**.mobile[*].mobility.angleDelta = normal(0deg, 30deg)
**.mobile[*].mobility.changeInterval = normal(5s, 0.1s)
#**.mobile[*].mobility.speed = 1mps
**.mobile[*].mobility.speed = 3mps

**.mobile[*].mobility.constraintAreaMinX = 300m
**.mobile[*].mobility.constraintAreaMinY = 100m
**.mobile[*].mobility.constraintAreaMinZ = 0m
**.mobile[*].mobility.constraintAreaMaxX = 900m
**.mobile[*].mobility.constraintAreaMaxY = 400m
**.mobile[*].mobility.constraintAreaMaxZ = 0m

**.static*.numApps = 1

**.static1.app[0].typename = "UdpBasicApp"
```

```
30 **.static1.app[0].destAddresses = "static2"
31 **.static1.app[0].destPort = 5001
32 **.static1.app[0].startTime = 10s
33 **.static1.app[0].sendInterval = 200ms
34 **.static1.app[0].messageLength = 100B
35
36 **.static2.app[0].typename = "UdpEchoApp"
37 **.static2.app[0].localPort = 5001
38
39 **.visualizer.*.displayInterfaceTables = true
40 **.visualizer.*.interfaceFilter = "wlan*"
41 **.visualizer.*.displayRoutes = true
42 **.visualizer.*.packetFilter = "UdpBasicAppData*"
43
44 *.configurator.config = xml("<config> \
45                 <interface hosts='static1' address='8.1.1.1'
    netmask='255.0.0.0'/> \
46                 <interface hosts='static2' address='8.2.2.2'
    netmask='255.0.0.0'/> \
47                                 <interface hosts='mobile[0]'
    address='8.3.3.0' netmask='255.0.0.0'/> \
48                 <interface hosts='mobile[*]' address='8.3.3.x'
    netmask='255.0.0.0'/> \
49                             </config>")
50
51 #**.hasStatus = true
52 #*.scenarioManager.script = xml("<scenario>\
53 #<at t='13'><shutdown module='mobile[1]'/></at></scenario>")
54
55 [Config Aodv]
56 network = AodvNetwork
57
58 [Config Dsdv]
59 network = DsdvNetwork
60 **.routing.typename = "Dsdv"
61 #**.routing.helloInterval=10s
```

Listing 1.1: Fichero omnetpp.ini

1.2 Ficheros .ned

```
1 import inet.networklayer.configurator.ipv4.Ipv4NetworkConfigurator;
2 import
    inet.physicallayer.ieee80211.packetlevel.Ieee80211ScalarRadioMedium;
3 import inet.visualizer.contract.IIntegratedVisualizer;
```

```
import inet.common.scenario.ScenarioManager;
import inet.node.aodv.AodvRouter;

network AodvNetwork
{
    parameters:
        int n;
        @display("bgb=1067.04,552.96");
    submodules:
        configurator: Ipv4NetworkConfigurator {
            @display("p=69.12,80.64");
        }
        radioMedium: Ieee80211ScalarRadioMedium {
            @display("p=70.560005,180");
        }
        visualizer: <default("IntegratedCanvasVisualizer")> like
    IIntegratedVisualizer if hasVisualizer() {
            @display("p=67.68,279.36002");
        }
        scenarioManager: ScenarioManager {
            @display("p=63.36,385.92");
        }
        mobile[n]: AodvRouter{
            @display("p=607.36,237.6;i=device/pocketpc");
        }
        static1: AodvRouter {
            @display("p=207.36,237.6;i=device/laptop");
        }
        static2: AodvRouter {
            @display("p=1007.36,237.6;i=device/laptop");
        }
}
```

Listing 1.2: Fichero AodvNetwork.ned

```
import inet.networklayer.configurator.ipv4.Ipv4NetworkConfigurator;
import
    inet.physicallayer.ieee80211.packetlevel.Ieee80211ScalarRadioMedium;
import inet.visualizer.contract.IIntegratedVisualizer;
import inet.common.scenario.ScenarioManager;
import inet.node.inet.ManetRouter;

network DsdvNetwork
{
    parameters:
        int n;
```

```
11          @display("bgb=1067.04,552.96");
12      submodules:
13          configurator: Ipv4NetworkConfigurator {
14              @display("p=69.12,80.64");
15          }
16          radioMedium: Ieee80211ScalarRadioMedium {
17              @display("p=70.560005,180");
18          }
19          visualizer: <default("IntegratedCanvasVisualizer")> like
    IIntegratedVisualizer if hasVisualizer() {
20              @display("p=67.68,279.36002");
21          }
22          scenarioManager: ScenarioManager {
23
24              @display("p=63.36,385.92");
25          }
26          mobile[n]: ManetRouter {
27              @display("p=607.36,237.6;i=device/pocketpc");
28          }
29          static1: ManetRouter {
30              @display("p=207.36,237.6;i=device/laptop");
31          }
32          static2: ManetRouter {
33              @display("p=1007.36,237.6;i=device/laptop");
34          }
35  }
```

Listing 1.3: Fichero DsdvNetwork.ned

Capítulo 2

Preguntas acerca de la simulación

2.1 Pregunta 1

Simule el escenario con AODV con entre 15 y 20 nodos móviles y una potencia de 1.5mW. ¿En qué momento se realiza la primera transmisión (de cualquier tipo de paquete)? Muestre el contenido de ese paquete explicando los campos más importantes.

El primer paquete se envía en t=10.002964223082s, este paquete es un RREQ (*Route Request*) del protocolo AODV, para entender un poco mejor que pasa, antes de ver el contenido del paquete, veamos el log en t=10s.

```
Event #1711  t=10  AodvNetwork.static1.udp (Udp, id=958)  on UdpBasicAppData-0 (inet::Packet, id=28778)
O:Sending app packet UdpBasicAppData-0 over ipv4.
Event #1712  t=10  AodvNetwork.static1.ipv4.ip (Ipv4, id=1005)  on UdpBasicAppData-0 (inet::Packet, id=28778)
O:Received (inet::Packet)UdpBasicAppData-0 (108 B) [[inet::UdpHeader, port:1025->5001, payloadLength:100 B, length = 8 B | inet::ApplicationPacket, length = 100 B]] from upper layer.
O (Aodv)AodvNetwork.static1.aodv:Finding route for source <unspec> with destination 8.2.2.2
O (Aodv)AodvNetwork.static1.aodv:Missing route for destination 8.2.2.2
AIL (Aodv)AodvNetwork.static1.aodv:Queuing datagram, source <unspec>, destination 8.2.2.2
O (Aodv)AodvNetwork.static1.aodv:Starting route discovery with originator 8.1.1.1 and destination 8.2.2.2
O (Aodv)AodvNetwork.static1.aodv:Sending a Route Request with target 8.2.2.2 and TTL= 2
Event #1713  t=10.002789399409  AodvNetwork.mobile[3].mobility (MassMobility, id=231)  on selfmsg move (omnetpp::cMessage, id=28259)
Event #1714  t=10.002888383903  AodvNetwork.mobile[12].mobility (MassMobility, id=771)  on selfmsg move (omnetpp::cMessage, id=28556)
Event #1715  t=10.002964223082  AodvNetwork.static1.aodv (Aodv, id=963)  on selfmsg aodv-send-jitter (inet::aodv::PacketHolderMessage, id=28784)
Event #1716  t=10.002964223082  AodvNetwork.static1.udp (Udp, id=958)  on aodv::Rreq (inet::Packet, id=28783)
O:Sending app packet aodv::Rreq over ipv4.
```

Figura 2.1: Log de la simulación en t=10s

Viendo la figura, podemos ver el motivo de este RREQ, en t=10s, el nodo *static1* intenta enviar un paquete UDP al nodo *static2*, cuando busca en sus tablas de enrutamiento una ruta con destino 8.2.2.2 (dirección IP de *static2*), no encuentra ninguna ruta, por tanto, introduce el datagrama UDP en la cola y envía un paquete RREQ con la dirección 8.2.2.2 como objetivo, para encontrar una ruta hacia este nodo.

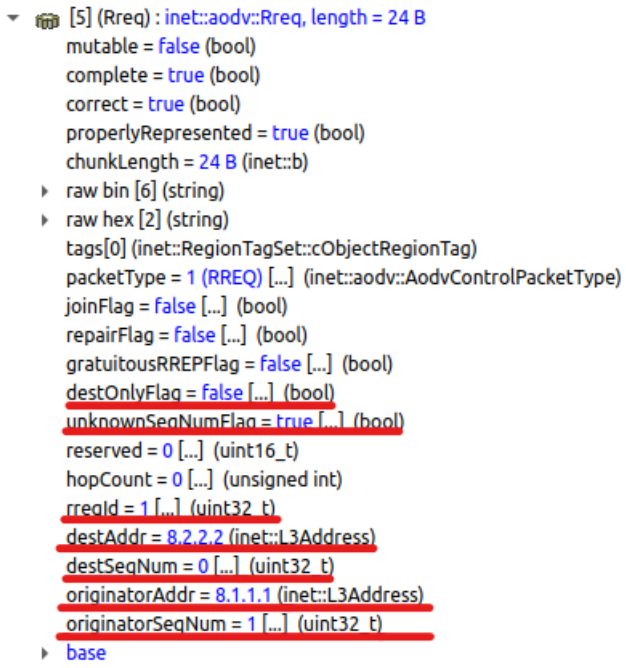

Figura 2.2: Contenido del primer paquete RREQ (t=10.002964223082s)

Hay varios campos relevantes a comentar:

- ***rreqId***: Número de secuencia que identifica, junto con el campo **originatorAddr**, únicamente a ese paquete RREQ, en este caso, al ser el primero que se envía desde ese origen (*static1*), tiene un *rreqId*=1.

- ***destAddr***: Dirección IP del destino para el que se quiere encontrar una ruta, en este caso, el nodo *static2*, con dirección IP 8.2.2.2.

- ***destSeqNum***: Es el último número de secuencia recibido previamente por el origen (*static1*), sobre una ruta hacia el destino *static2*, al ser el primer paquete RREQ, no existen rutas previamente recibidas y el valor de este campo se establece a 0.

- ***unknownSeqNumFlag***: Indica que el valor del campo *destSeqNum* es desconocido, en este caso, su valor es *true*, ya que como se mencionó previamente, no se conoce ningún número de secuencia recibido previamente porque este es el primer paquete RREQ.

- ***originatorAddr***: Dirección IP del nodo que originó el *Route Request*, en este caso, el nodo *static1* con dirección IP 8.1.1.1.

- ***originatorSeqNum***: Es el número de secuencia que van a utilizar los nodos intermedios que reciban el RREQ, para registrar la ruta que apunta hacia el nodo origen *static1*,

en este caso, como es el primer paquete RREQ tiene un valor igual a 1, este valor irá incrementando cada vez que se envíe un RREQ.

- ***destOnlyFlag***: Indica que sólo el destino (*static2*) puede responder a ese RREQ, en este caso su valor es *false*, de forma que si un nodo intermedio ya tiene almacenada una ruta hacia el destino pueda responder sin reenviar el RREQ, aunque no va a ser el caso porque es el primer RREQ enviado en la red.

- ***gratuitousRREPFlag***: Indica que si un nodo intermedio tiene una ruta hacia el destino, además de responder al origen con un RREP con la ruta hacia el destino, también envíe un RREP al destino con una ruta hacia el origen para establecer una comunicación bidireccional, en este caso, se marca como *false* (comportamiento por defecto).

ACLARACIÓN: Tanto ***destSeqNum*** como ***originatorSeqNum***, son campos que actúan como timestamps, marcas de tiempo que indican lo reciente que es una ruta, de esta forma, cuando un nodo recibe una ruta, puede saber si es más o menos reciente que la que tenga en su tabla de enrutamiento y, en caso de ser más reciente, actualizar su tabla de enrutamiento.

Comentar también que a nivel IP, el campo se envía a la dirección de broadcast y a nivel UDP, se envía por el puerto 654, que es el puerto estblecido en el registro de protocolos de la IANA para todos los mensajes del protocolo AODV.

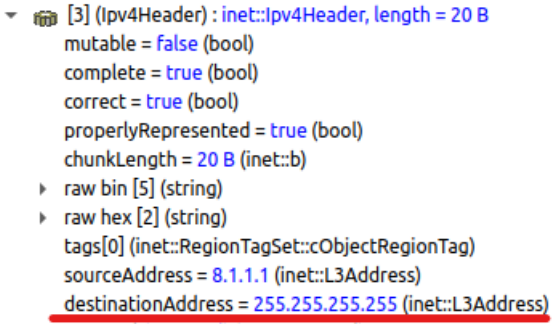

Figura 2.3: Contenido del primer paquete RREQ (t=10.002964223082s) en la capa IP

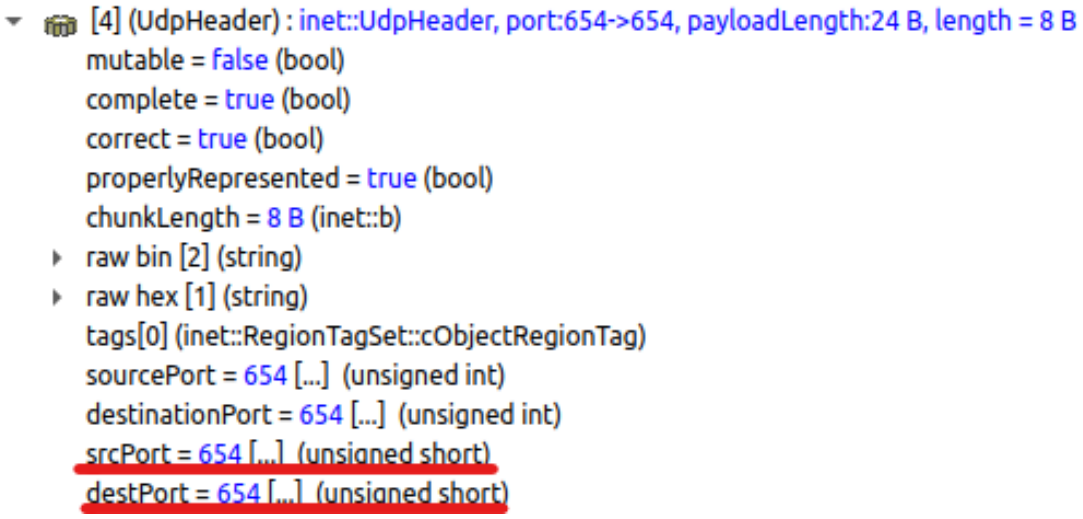

Figura 2.4: Contenido del primer paquete RREQ (t=10.002964223082s) en la capa UDP

2.2 Pregunta 2

¿Llega el primer RREQ enviado por static1 hasta static2 a través de alguna de las rutas existentes? ¿Por qué? ¿En qué instante recibe static2 su primer RREQ?

No, el primer paquete RREQ enviado por *static1* no llega a *static2*, sin embargo, sí que existen rutas en ese instante para alcanzar a *static2*.

```
** Event #2365  t=10.32  AodvNetwork.static1.aodv (Aodv, id=963)  on selfmsg  (inet::aodv::WaitForRrep, id=623)
INFO:We didn't get any Route Reply within RREP timeout
INFO:Sending a Route Request with target 8.2.2.2 and TTL= 4
```

Figura 2.5: Log de la finalización del *RREP_TIMEOUT* en t=10.32s

En este log, vemos que finaliza el RREP Timeout, entonces, el nodo *static1* envía un paquete RREQ de nuevo, pero ahora utiliza un TTL=4, si nos fijamos en el paquete anterior, el paquete tenía un TTL=2, esa es la razón de que no llegase a *static2*, porque son necesarios más de 2 saltos (nodos intermedios) para alcanzar a *static2*.

Lo que hará *static1*, tal como define el protocolo AODV, es intentar encontrar rutas lo más cortas posibles, comenzará con un TTL=*TTL_START*=2 y si finaliza el *RREP_TIMEOUT* sin recibir ninguna respuesta, enviará de nuevo un paquete RREQ, incrementando el TTL según el parámetro *TTL_INCREMENT*=2, *static1* seguirá enviando paquetes RREQ incrementando el TTL hasta recibir un RREP de respuesta o alcanzar un TTL=*TTL_THRESHOLD*=7, a partir de ahí utilizará un TTL=*NET_DIAMETER*=35 para seguir enviando paquetes RREQ, cada uno de estos paquetes tendrá un *rreqId* nuevo.

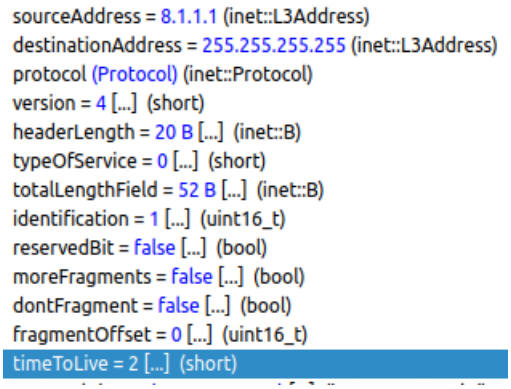

Figura 2.6: TTL del primer paquete RREQ enviado en t=10.002964223082s

Parameter Name	Value
ACTIVE_ROUTE_TIMEOUT	3,000 Milliseconds
ALLOWED_HELLO_LOSS	2
BLACKLIST_TIMEOUT	RREQ_RETRIES * NET_TRAVERSAL_TIME
DELETE_PERIOD	see note below
HELLO_INTERVAL	1,000 Milliseconds
LOCAL_ADD_TTL	2
MAX_REPAIR_TTL	0.3 * NET_DIAMETER
MIN_REPAIR_TTL	see note below
MY_ROUTE_TIMEOUT	2 * ACTIVE_ROUTE_TIMEOUT
NET_DIAMETER	35
NET_TRAVERSAL_TIME	2 * NODE_TRAVERSAL_TIME * NET_DIAMETER
NEXT_HOP_WAIT	NODE_TRAVERSAL_TIME + 10
NODE_TRAVERSAL_TIME	40 milliseconds
PATH_DISCOVERY_TIME	2 * NET_TRAVERSAL_TIME
RERR_RATELIMIT	10
RING_TRAVERSAL_TIME	2 * NODE_TRAVERSAL_TIME * (TTL_VALUE + TIMEOUT_BUFFER)
RREQ_RETRIES	2
RREQ_RATELIMIT	10
TIMEOUT_BUFFER	2
TTL_START	1
TTL_INCREMENT	2
TTL_THRESHOLD	7
TTL_VALUE	see note below

Figura 2.7: Valores por defecto de los parámetros de configuración del protocolo AODV definidos en el *RFC 3561, AODV, 10 Configuration Parameters*

A continuación, se muestra un ejemplo de este comportamiento en una simulación en la que *static1* no consigue alcanzar a *static2* y este nunca recibe un RREQ.

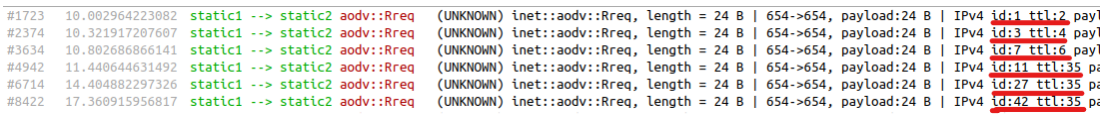

Figura 2.8: Paquetes RREQ enviados de *static1* con destino objetivo *static2*

En nuestro escenario, **¿Cuándo recibe *static2* su primer RREQ?** El nodo *static2* recibe su primer RREQ en t=10.812224972086s, este RREQ tiene un TTL=6, aunque cuando lo recibe *static2* tiene un TTL=2, lo que quiere decir que la ruta está conformada por 4 nodos intermedios.

```
** Event #4001  t=10.812224972086  AodvNetwork.static2.aodv (Aodv, id=1024)  on aodv::Rreq (inet::Packet, id=2563)
INFO:AODV Route Request arrived with source addr: 8.3.3.7 originator addr: 8.1.1.1 destination addr: 8.2.2.2
DETAIL:Adding new route destination = 8.3.3.7, prefixLength = 32, nextHop = 8.3.3.7, metric = 1, interface = wlan0
INFO (Ipv4RoutingTable)AodvNetwork.static2.ipv4.routingTable:add route ??? 8.3.3.7/32 gw:8.3.3.7 metric:1 if:wlan0
DETAIL:Adding new route destination = 8.1.1.1, prefixLength = 32, nextHop = 8.3.3.7, metric = 5, interface = wlan0
INFO (Ipv4RoutingTable)AodvNetwork.static2.ipv4.routingTable:add route ??? 8.1.1.1/32 gw:8.3.3.7 metric:5 if:wlan0
INFO:I am the destination node for which the route was requested
INFO:Sending Route Reply to 8.1.1.1
```

Figura 2.9: Log de la recepción del paquete RREQ en el nodo *static2*

En el log podemos ver, que *static2* añade una ruta hacia *static1* en la que especifica como siguiente salto al nodo *mobile[7]* con dirección IP 8.3.3.7, que es el nodo del que recibió el paquete RREQ (el último nodo de la ruta). A continuación, *static2* responderá con un paquete RREP.

2.3 Pregunta 3

Elija un nodo de la zona intermedia (i.e., fuera del alcance directo de static1 y de static2) por el que vayan a pasar los paquetes UDP tras el establecimiento de la ruta. Muestre la tabla de enrutamiento justo antes y justo después de recibir el primer RREQ. Explique las entradas que se crean. ¿Para qué ruta son útiles esas entradas?

Observaremos la tabla de enrutamiento del nodo *mobile[2]*.

Antes de recibir el paquete, el nodo *mobile[2]* tiene simplemente una ruta para su dirección de loopback (entrada [2]) y las rutas por defecto para cualquier paquete que provenga de la red 8.0.0.0/14 y 8.0.0.0/8 (entradas [0] y [1]).

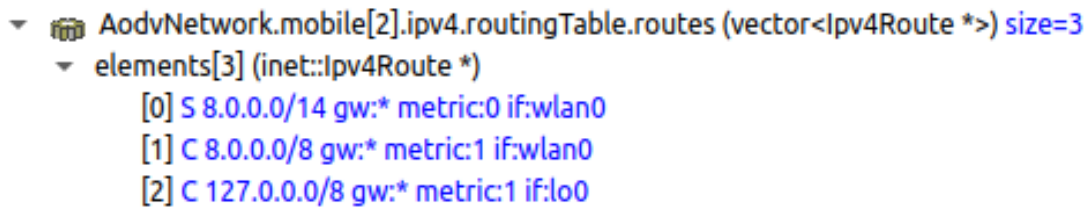

Figura 2.10: Tabla de enrutamiento del nodo *mobile[2]* antes de recibir el primer paquete RREQ

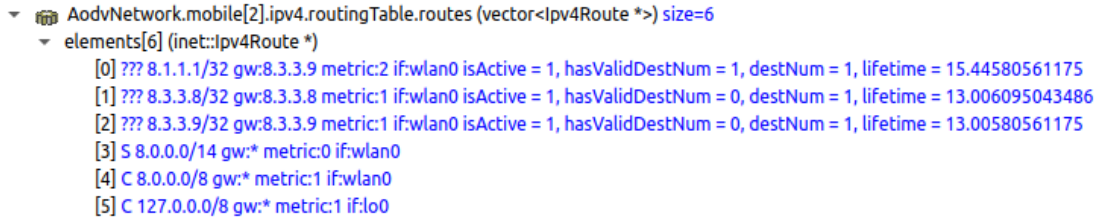

Figura 2.11: Tabla de enrutamiento del nodo *mobile[2]* después de recibir el primer paquete RREQ

Tras recibir el paquete RREQ el nodo *mobile[2]* crea las siguientes entradas:

0 : Ruta hacia el nodo *static1*, los paquetes con destino 8.1.1.1, serán enviados al nodo *mobile[9]* (8.3.3.9), el campo ***metric*** indica que *static1* está a dos saltos (accedemos a él a través de *mobile[9]*).

1 : Ruta hacia el nodo *mobile[8]* (8.3.3.8), al que podemos acceder directamente (***metric***=1).

2 : Ruta hacia el nodo *mobile[9]* (8.3.3.9), al que podemos acceder directamente (***metric***=1).

Las tres entradas tienen el campo ***isActive*** con valor 1, que indica que esa entrada está activa, el campo ***lifetime*** nos indica el tiempo de expiración de esa entrada, pasado este tiempo la ruta habrá expirado o se habrá borrado, este campo se actualiza cada vez que se utiliza dicha entrada, se actualizará a un valor que, como mínimo, tiene que ser mayor que el tiempo actual más el ACTIVE_ROUTE_TIMEOUT (3000ms, ver Figura 2.7).

Por último, el campo ***destNum*** es el número de secuencia (timestamp) de esa entrada, que indica lo reciente qué es, este campo se corresponde con el campo ***originatorSeqNum*** del paquete RREQ recibido, si nos fijamos solo se indica cómo válido para la ruta correspondiente al nodo *static1* (entrada [0]), que es quién originó el paquete RREQ.

Lo que pasa realmente es que el nodo *mobile[2]* recibe primero el paquete RREQ a través del nodo *mobile[9]*, cuando lo recibe, crea las entradas correspondientes, [0] y [2], al *Reverse Path* (ruta hacia el originador del paquete, *static1*) y reenvía el paquete (a la dirección de broadcast). A continuación, vuelve a recibir el paquete RREQ a través del nodo *mobile[8]*, crea una entrada para este nodo y, como ya ha reenviado el paquete una vez, ahora no lo reenvía, por esta razón es muy útil el campo *rreqId* de un paquete RREQ, para que los nodos intermedios sepan qué paquetes RREQ ya han reenviado previamente.

2.4 Pregunta 4

Muestre la tabla de enrutamiento de ese mismo nodo justo antes y justo después de recibir el RREP de respuesta. Explique las entradas creadas por el RREP. ¿Para qué ruta son útiles estas entradas?

Simularemos hasta t=10.81s aproximadamente, instante en el que el nodo *static2* recibe el primer paquete RREQ pero aún no ha enviado su paquete RREP de respuesta.

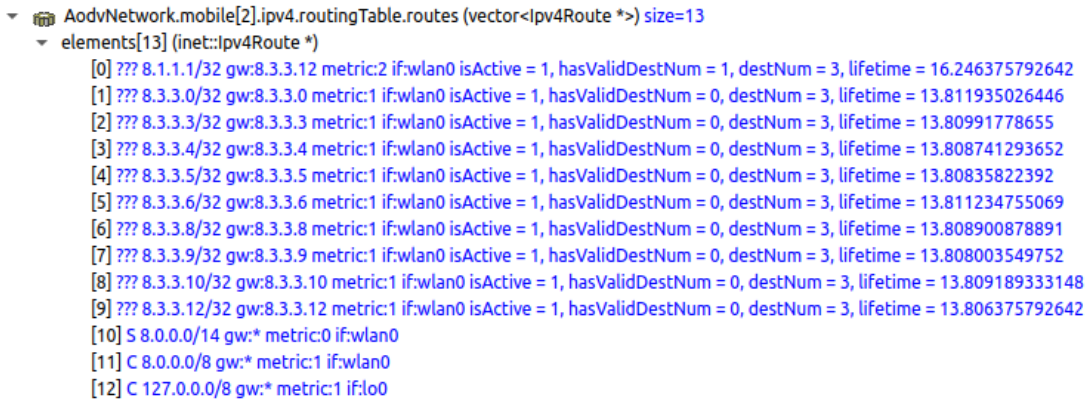

Figura 2.12: Tabla de enrutamiento del nodo *mobile[2]* antes de recibir el primer paquete RREP

Ahora, al haber pasado más tiempo, vemos que la tabla de enrutamiento ha crecido, el nodo *mobile[2]* creó una entrada para cada nodo que le envió un paquete RREQ como había hecho con *mobile[8]* en el ejemplo explicado previamente (Sección 2.11 página 15).

Además, podemos ver que actualizó la entrada correspondiente al nodo *static1*, ahora es más reciente (*destNum*=3) y en vez de usar como gateway al nodo *mobile[9]*, ahora usa al nodo *mobile[12]*, esto significa que el nodo *mobile[12]* fue el primero en enviarle el tercer paquete RREQ con TTL=6 y *originatorSeqNum*=3.

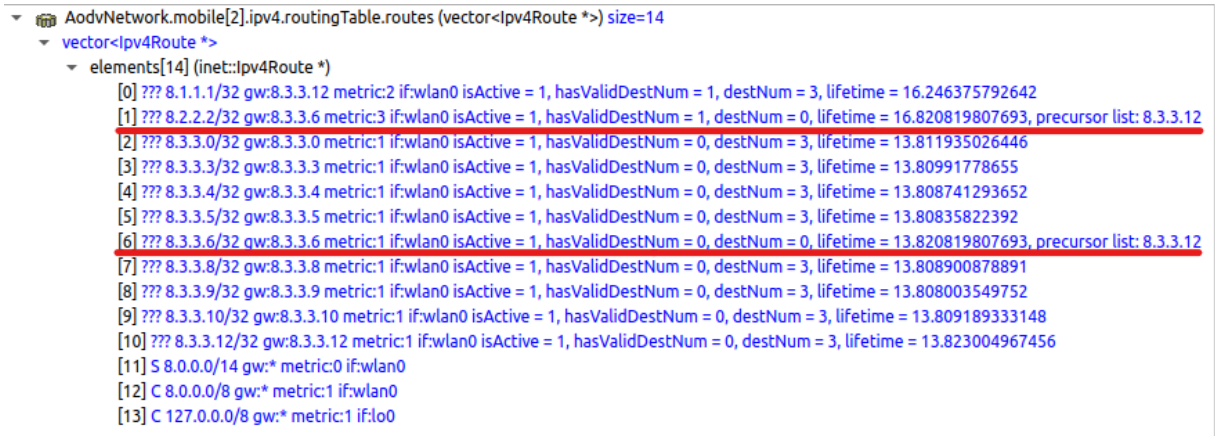

Figura 2.13: Tabla de enrutamiento del nodo *mobile[2]* después de recibir el primer paquete RREP

Ahora, una vez recibido el paquete RREP, se crea una entrada (entrada [1]) hacia el nodo objetivo (*static2*), con dirección IP 8.2.2.2, donde se especifica una lista de nodos precursores a él mismo, es decir, los nodos que tendrán como siguiente salto a *mobile[2]* en su entrada con destino *static2*, esta lista se indica con el objetivo de conocer los nodos anteriores en la ruta, de forma que, cuando se utilice esta entrada, además de actualizar el campo *lifetime* de esta, se actualizará también el *lifetime* de la ruta de vuelta, es decir, las entradas correspondientes a los nodos precursores, en este caso, la entrada [10] que corresponde al nodo precursor *mobile[12]*, este mecanismo se utiliza para facilitar la comunicación bidireccional y que las entradas sean simétricas.

Al campo *destNum*, se le asigna el valor 0, que se corresponde con el valor del campo *destSeqNum* del paquete RREP recibido, en este caso, este número de secuencia (timestamp) comienza a contar en 0, al contrario que el campo *originatorSeqNum* que, como vimos previamente, empieza a contar en 1, pero el funcionamiento es el mismo.

Cuando el RREP se envía hacia *static1*, es cuando todos los nodos crean su entrada correspondiente al nodo *static2*, basándose en el nodo del que reciben el RREP, en este caso *mobile[6]*, el funcionamiento es similar al de la creación del *Reverse Path* cuando se está enviando el paquete RREQ.

Por otra parte, el nodo *mobile[2]*, también actualiza su entrada correspondiente al nodo *mobile[6]*, siguiente salto hacia *static2*, añadiendo la lista de nodos precursores y actualizando su *destNum* al mismo valor que el de la nueva entrada (0).

2.5 Pregunta 5

¿Qué ruta sigue el RREP hasta llegar a static1? ¿Qué IP destino utilizan los RREP? ¿En qué se diferencia del RREQ en cuanto a propagación? Muestre una captura del escenario en la que se vea la ruta seguida por UdpBasicAppData-0 mostrada por el visualizer.

Los paquetes RREP, a diferencia de los paquetes RREQ, se envían a un destino unicast en vez de broadcast, los paquetes se irán enviando al nodo correspondiente al *Reverse Path* que cada nodo intermedio fue almacenando en su tabla de enrutamiento cuando recibían un paquete RREQ.

En este caso el paquete RREP sigue la ruta:

static2 –> mobile[7] –> mobile[6] –> mobile[2] –> mobile[12] –> static1

Como habíamos mencionado previamente (Sección 2.2 página 14), la ruta está conformada por 4 nodos intermedios.

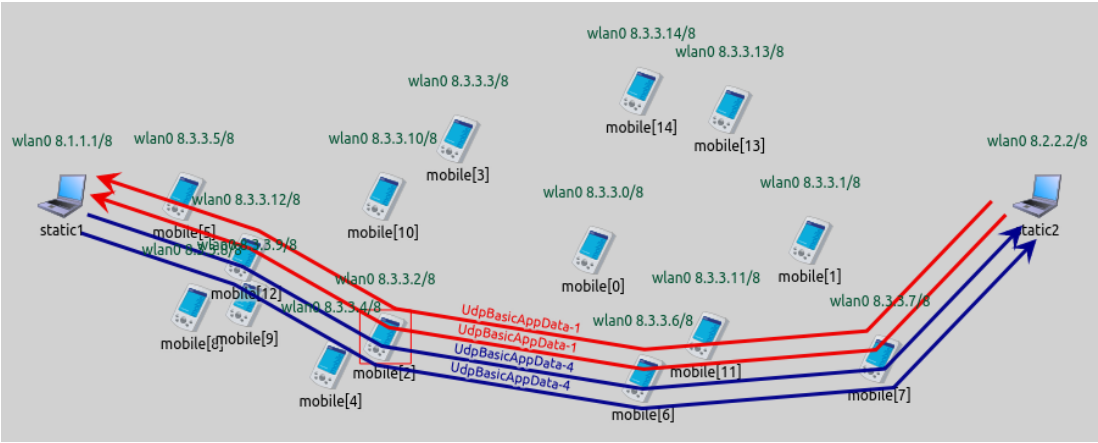

Figura 2.14: Ruta seguida por el paquete *UdpBasicAppData-0*

2.6 Pregunta 6

¿En qué instante recibe el paquete UdpBasicAppData-0 el nodo static2? ¿Cuánto tiempo transcurre desde el startTime? ¿A qué se debe el retardo?

El nodo *static2* recibe el paquete en *UdpBasicAppData-0* t=10.829362889087s, habien-

do transcurrido 0.83s (aproximadamente) desde el startTime (10s), este retardo se debe al establecimiento de la ruta entre los nodos *static1* y *static2* y al tiempo que tarda en llegar el paquete de un nodo a otro, es decir, en pasar entre los nodos intermedios, aunque esto último casi no afecta, el nodo *static1* envía el paquete en 10.825099873452s, por tanto, el envío del paquete solo añade 0.004s=4ms de retardo mientras que el establecimiento de la ruta añade unos 0.825s=825ms.

```
** Event #5640  t=10.825099873452  AodvNetwork.static1.wlan[0].radio (Ieee80211ScalarRadio, id=971)  on UdpBasicAppData-0 (inet::Packet, id=196011)
INFO:Transmission started: (inet::physicallayer::Signal)UdpBasicAppData-0 WHOLE as Ieee80211ScalarTransmission, mode = { Ieee80211ErpOfdmMode }, ch
INFO:Changing radio transmission state from IDLE to TRANSMITTING.
INFO:Changing radio transmitted signal part from NONE to WHOLE.
```

Figura 2.15: Log del envío del paquete *UdpBasicAppData-0* desde el nodo *static1*

```
** Event #6940  t=10.829362889087  AodvNetwork.static2.udp (Udp, id=1019)  on UdpBasicAppData-0 (inet::Packet, id=198357)
INFO:Packet UdpBasicAppData-0 received from network, dest port 5001
INFO:Sending payload up to socket sockId=17
```

Figura 2.16: Log de la recepción del paquete *UdpBasicAppData-0* en el nodo *static2*

2.7 Pregunta 7

Añada las siguientes líneas a la configuración para simular la caída de un nodo en t=13s:

****.hasStatus = true**
***.scenarioManager.script = xml("<scenario>**
<at t='13'><shutdown module='mobile[k]'/></at></scenario>")

Donde k es el número de nodo de la ruta activa en ese instante inmediatamente anterior a static2. ¿Cómo se da cuenta de la caída el nodo anterior? ¿Cuánto tarda en notificar al resto de nodos? Muestre una captura del log del nodo que genera el RERR y el contenido del paquete explicando los campos importantes.

El nodo anterior, *mobile[6]*, intenta envíar el paquete *UdpBasicAppData-15* hacia *static2* a través de *mobile[7]* (siguiente salto), que es el nodo que está caído.

Un nodo puede generar un mensaje RERR por tres razones:

1. Si detecta un fallo de enlace para el siguiente salto de una ruta activa en su tabla de enrutamiento mientras transmite datos (y la reparación de la ruta, si se intentó, no tuvo éxito).

2. Si recibe un paquete de datos destinado a un nodo para el que no tiene una ruta activa y no está siendo reparado (si el nodo está utilizando *Local Repair*).

3. Si recibe un RERR de un vecino para una o más rutas activas.

```
** Event #22657  t=13.023442675911  AodvNetwork.mobile[6].wlan[0].mac.dcf (Dcf, id=441)
INFO:Frame sequence aborted.
INFO:Data/Mgmt frame transmission failed
INFO:Incremented station SRC: stationShortRetryCounter = 7.
WARN:Retry limit reached for (inet::Packet)UdpBasicAppData-15 (164 B) [[inet::ieee80211:
INFO:Dropping frame UdpBasicAppData-15, because retry limit is reached.
DETAIL (Aodv)AodvNetwork.mobile[6].aodv:Received link break signal
DETAIL (Aodv)AodvNetwork.mobile[6].aodv:Marking route to 8.2.2.2 as inactive
DETAIL (Aodv)AodvNetwork.mobile[6].aodv:Marking route to 8.3.3.7 as inactive
INFO (Aodv)AodvNetwork.mobile[6].aodv:Broadcasting Route Error message with TTL=1
INFO (Dcaf)AodvNetwork.mobile[6].wlan[0].mac.dcf.channelAccess:Channel released.
```

Figura 2.17: Log del séptimo intento de envío del paquete *UdpBasicAppData-15* desde el nodo *mobile[6]* en la capa de enlace

Aquí estamos en el primer caso, como podemos ver *mobile[6]*, tras 7 intentos de envío del paquete *UdpBasicAppData-15*, alcanza el RETRY_LIMIT (7), entonces, detecta que se rompió el enlace con el siguiente salto de la ruta, *mobile[7]*, por tanto, marca las entradas correspondientes a este nodo como inactivas y envía un paquete RERR para notificar a sus vecinos.

```
** Event #22658  t=13.027479270713  AodvNetwork.mobile[6].aodv (Aodv, id=429)  on selfmsg aodv-send-jitter (inet::aodv::PacketHolderMessage, id=66558)
** Event #22659  t=13.027479270713  AodvNetwork.mobile[6].udp (Udp, id=425)  on aodv::Rerr (inet::Packet, id=66557)
INFO:Sending app packet aodv::Rerr over ipv4.
** Event #22660  t=13.027479270713  AodvNetwork.mobile[6].ipv4.ip (Ipv4, id=471)  on aodv::Rerr (inet::Packet, id=66557)
INFO:Received (inet::Packet)aodv::Rerr (36 B) [[inet::UdpHeader, port:654->654, payloadLength:28 B, length = 8 B | inet::aodv::Rerr, length = 28 B]] from upper layer.
DETAIL:Sending datagram 'aodv::Rerr' with destination = 255.255.255.255
DETAIL:destination address is broadcast, sending packet to broadcast MAC address
INFO:Sending (inet::Packet)aodv::Rerr (56 B) [[inet::Ipv4Header, length = 20 B | inet::UdpHeader, port:654->654, payloadLength:28 B, length = 8 B | inet::aodv::Rerr, length = 28 B]] to output interface = wlan0.
```

Figura 2.18: Log de la generación y envío del paquete RERR desde *mobile[6]*

En los paquetes RERR, sólo vemos un campo relevante a comentar:

- ***unreachableNodes***: Contiene una lista de los nodos que son inalcanzables debido al fallo de enlace, para cada uno de estos nodos almacena:
  - ***addr***: Dirección IP del nodo
  - ***seqNum***: Número de secuencia de la entrada en la tabla de enrutamiento, este campo se corresponde con el campo ***destNum*** de la entrada correspondiente a ese nodo en la tabla de enrutamiento. Aunque si el nodo en concreto, es el nodo destino de la ruta, es decir, el destino final no un nodo intermedio, en este caso, *static2*, su ***destNum*** se incrementa en 1.

  Como podemos ver se registran como inalcanzables *mobile[7]* y, consecuentemente, *static2*, con sus correspondientes números de secuencia.

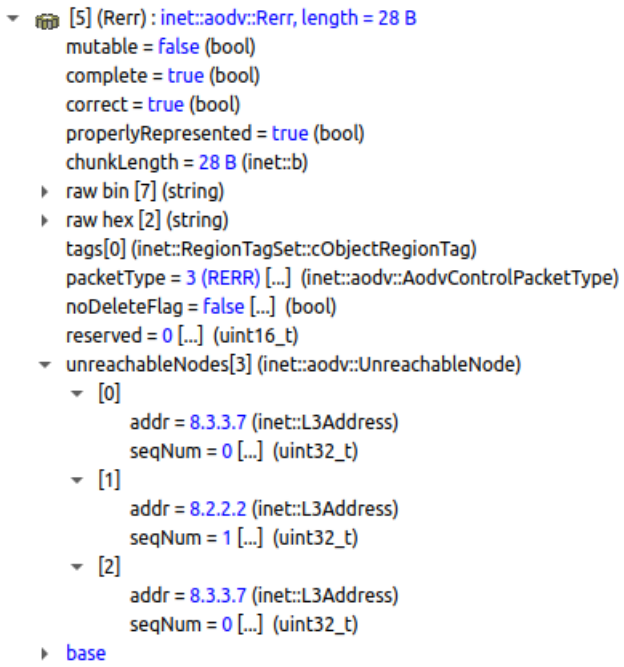

Figura 2.19: Contenido del paquete RERR enviado por *mobile[6]*

Los campos ***reserved*** y ***packetType*** no se comentan porque son campos triviales, el primero se envía como 0 y es ignorado en la recepción del paquete y el segundo siempre será 3 para los paquetes RERR, indicando que el paquete es de este tipo.

ACLARACIÓN: Esta es la vista que nos ofrece OMNET++, en realidad, los paquetes RERR no especifican con un único campo los nodos que quedan inalcanzables, en la estructura real de un paquete RERR, se indica la dirección IP del nodo que queda inalcanzable en el campo ***Unreachable Destination IP Address*** y su número de secuencia en el campo ***Unreachable Destination Sequence Number***. En caso de quedar más de un nodo inalcanzable, se utilizarán los campos ***Additional Unreachable Destination IP Addresses*** y ***Additional Unreachable Destination Sequence Numbers***. También se utilizará el campo ***destCount*** para indicar el número de nodos que han quedado inalcanzables que, como mínimo, ha de ser uno.

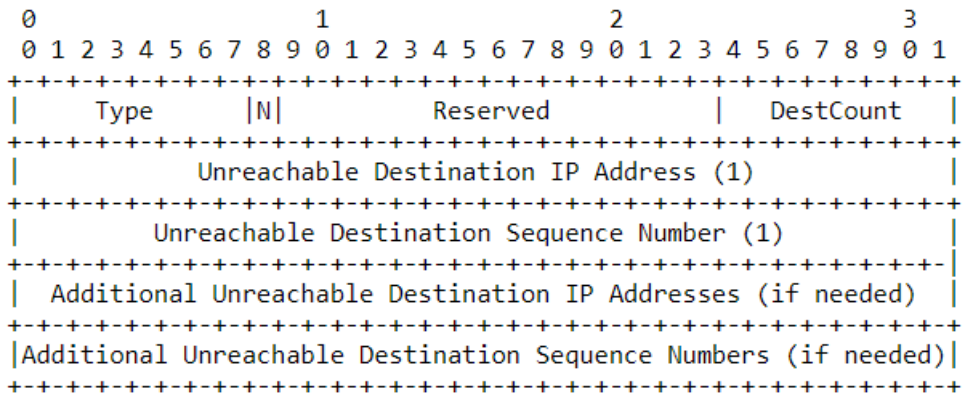

Figura 2.20: Estructura de un paquete RERR según el *RFC 3561, AODV, 5.3. Route Error (RERR) Message Format*

2.8 Pregunta 8

¿Qué IP tiene como destino el RERR? ¿Qué TTL tiene? ¿Es reenviado por todos los nodos que lo reciben? ¿Cómo se propaga por la red?

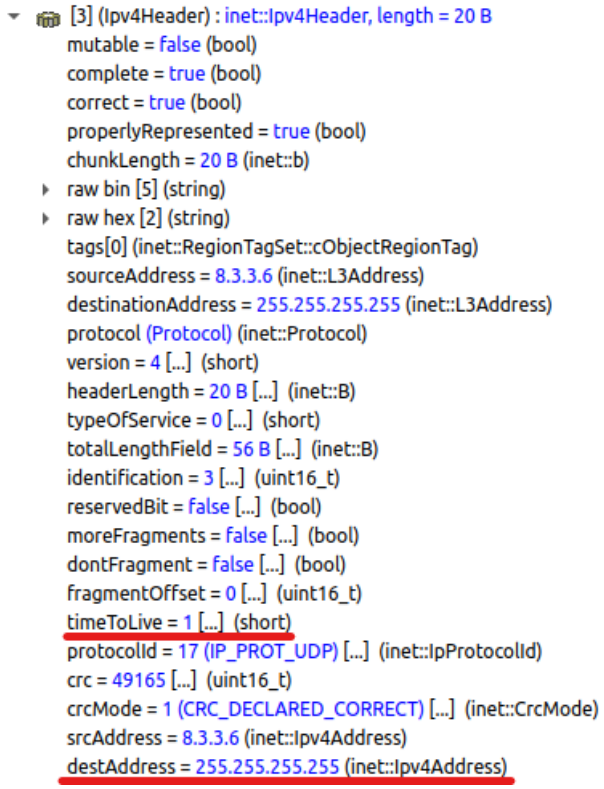

Figura 2.21: Contenido del paquete RERR enviado por *mobile[6]* en la capa IP

En la capa IP, el paquete es enviado como un broadcast con un TTL=1, lo que significa que el paquete no puede ser retransmitido, sin embargo, aquellos nodos que pertenezcan a la ruta entre *static1* y *static2* generarán un nuevo paquete RERR y lo enviaran como broadcast también, de forma que, todos los nodos recibirán el paquete RERR y así podrán actualizar sus tablas de enrutamiento, pero solo aquellos que reciban un paquete RERR y se corresponda con alguna de sus rutas activas, generarán un nuevo paquete RERR y lo enviarán a la dirección de broadcast.

En este caso, los nodos que pertenecen a la ruta y generan nuevos paquetes RERR son *mobile[2]* y *mobile[12]*. A continuación, podemos ver la diferencia entre el log de un nodo que no pertenece a la ruta (*mobile[11]*) y el log de uno que sí pertenece (*mobile[2]*) al recibir el paquete RERR.

```
** Event #22689  t=13.027537487289  AodvNetwork.mobile[11].udp (Udp, id=730)  on aodv::Rerr (inet::Packet, id=100605)
INFO:Packet aodv::Rerr received from network, dest port 654
INFO:Sending payload up to socket sockId=11
** Event #22690  t=13.027537487289  AodvNetwork.mobile[11].aodv (Aodv, id=734)  on aodv::Rerr (inet::Packet, id=100605)
INFO:AODV Route Error arrived with source addr: 8.3.3.6
```

Figura 2.22: Log de la recepción del paquete RERR en *mobile[11]*

```
** Event #22707  t=13.027537977616  AodvNetwork.mobile[2].udp (Udp, id=181)  on aodv::Rerr (inet::Packet, id=100611)
INFO:Packet aodv::Rerr received from network, dest port 654
INFO:Sending payload up to socket sockId=2
** Event #22708  t=13.027537977616  AodvNetwork.mobile[2].aodv (Aodv, id=185)  on aodv::Rerr (inet::Packet, id=100611)
INFO:AODV Route Error arrived with source addr: 8.3.3.6
INFO:Sending RERR to inform our neighbors about link breaks.
```

Figura 2.23: Log de la recepción del paquete RERR en *mobile[2]*

2.9 Pregunta 9

¿Qué hace un nodo al recibir un RERR? Muestre capturas de la tabla de enrutamiento antes y después de recibirlo y explique en qué cambia. ¿Qué hace static1 en los instantes posteriores a recibir el RERR?

Cuando un nodo recibe un paquete RERR, comprueba si alguno de los nodos que han quedado inalcanzables coincide con alguna de sus rutas activas y, en tal caso, marca estas entradas como inactivas.

```
AodvNetwork.mobile[2].ipv4.routingTable.routes (vector<Ipv4Route *>) size=14
  vector<Ipv4Route *>
    elements[14] (inet::Ipv4Route *)
      [0] ??? 8.1.1.1/32 gw:8.3.3.12 metric:2 if:wlan0 isActive = 1, hasValidDestNum = 1, destNum = 3, lifetime = 16.246375792642
      [1] ??? 8.2.2.2/32 gw:8.3.3.6 metric:3 if:wlan0 isActive = 1, hasValidDestNum = 1, destNum = 0, lifetime = 16.820819807693, precursor list: 8.3.3.12
      [2] ??? 8.3.3.0/32 gw:8.3.3.0 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 3, lifetime = 13.811935026446
      [3] ??? 8.3.3.3/32 gw:8.3.3.3 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 3, lifetime = 13.80991778655
      [4] ??? 8.3.3.4/32 gw:8.3.3.4 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 3, lifetime = 13.808741293652
      [5] ??? 8.3.3.5/32 gw:8.3.3.5 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 3, lifetime = 13.80835822392
      [6] ??? 8.3.3.6/32 gw:8.3.3.6 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 0, lifetime = 16.000418968908, precursor list: 8.3.3.12
      [7] ??? 8.3.3.8/32 gw:8.3.3.8 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 3, lifetime = 13.808900878891
      [8] ??? 8.3.3.9/32 gw:8.3.3.9 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 3, lifetime = 13.808003549752
      [9] ??? 8.3.3.10/32 gw:8.3.3.10 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 3, lifetime = 13.809189333148
      [10] ??? 8.3.3.12/32 gw:8.3.3.12 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 3, lifetime = 16.000418968908
      [11] S 8.0.0.0/14 gw:* metric:0 if:wlan0
      [12] C 8.0.0.0/8 gw:* metric:1 if:wlan0
      [13] C 127.0.0.0/8 gw:* metric:1 if:lo0
```

Figura 2.24: Tabla de enrutamiento de *mobile[2]* antes de recibir el paquete RERR

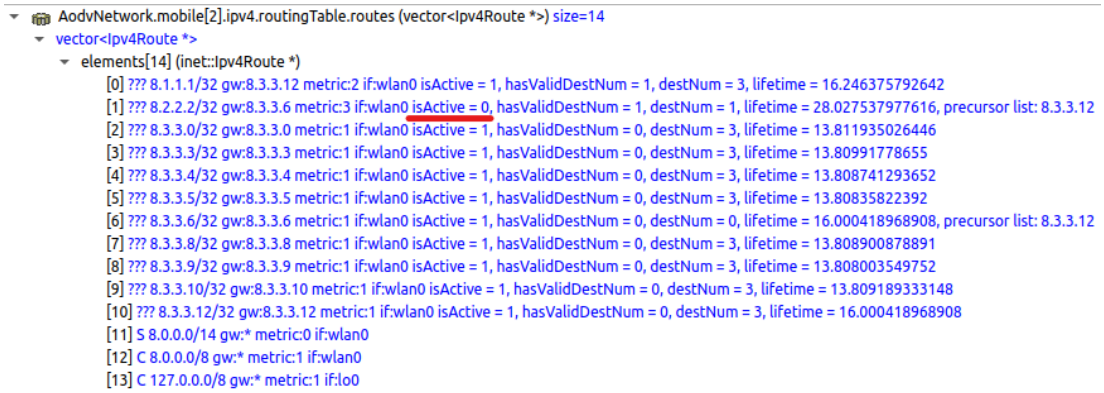

Figura 2.25: Tabla de enrutamiento de *mobile[2]* después de recibir el paquete RERR

¡OJO! Las rutas que coincidan serán actualizadas siempre y cuando el número de secuencia asociado al nodo que ha quedado inalcanzable en el RERR sea mayor o igual (con menor coste) que el número de secuencia de la entrada de la tabla. Además, si el número de secuencia del paquete RERR es mayor, además de marcar la entrada como inactiva actualizaremos este número, en el ejemplo anterior, se actualiza el ***destNum*** de la entrada correspondiente a *static2* de 0 a 1.

Por ejemplo, si observamos la tabla de enrutamiento de *mobile[11]*, vemos que sigue exactamente igual antes y después de recibir el paquete, a pesar de que tiene una entrada activa para *mobile[7]*, que es uno de los nodos que se marca como inalcanzable, esto ocurre porque en el paquete RERR se indica el nodo con un ***seqNum***=0, mientras que la entrada correspondiente de la tabla de enrutamiento de *mobile[11]* tiene un ***destNum***=3, más reciente, por tanto, no se actualiza.

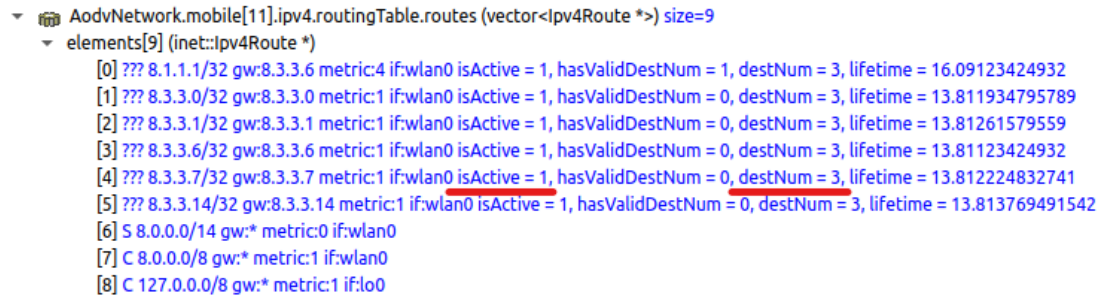

Figura 2.26: Tabla de enrutamiento de *mobile[11]* después de recibir el paquete RERR

Cuando el nodo *static1* reciba el paquete RERR, marcará la entrada correspondiente a *static2* como inactiva y actualizará su número de secuencia, entonces, cuando intente enviar el próximo paquete UDP no va a encontrar ninguna ruta activa para ese destino y enviará

un paquete RREQ para descubrir una nueva ruta hacia *static2*. Este paquete RREQ tendrá asignado como ***destSeqNum*** un número mayor o igual que el número de secuencia de la entrada que corresponde a *static2*, que acaba de ser actualizado al recibir el paquete RERR.

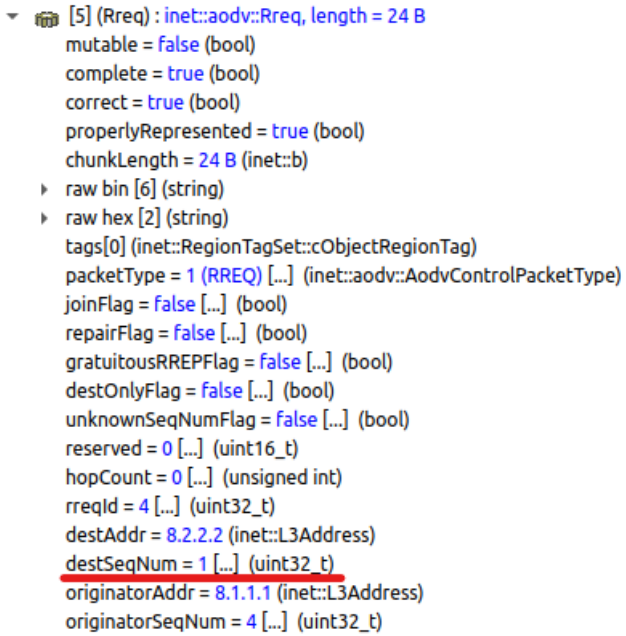

Figura 2.27: Contenido del paquete RREQ generado por *static1* en t=13.200166111932s

Efectivamente, el valor del campo ***destNum*** de la entrada correspondiente a *static2* en la tabla de enrutamiento de *static1* era igual a 0. Tras recibir el paquete RERR, *static1* actualizó el ***destNum*** de esa entrada a 1 (además de marcarla como inactiva). Por último, cuando envió el paquete RREQ nuevo, asignó el valor de ***destNum*** al campo ***destSeqNum*** del paquete RREQ.

2.10 Pregunta 10

Simule ahora el escenario con DSDV con el mismo número de nodos móviles y la misma potencia que en el apartado 1. ¿En qué instante se realiza ahora la primera transmisión (de cualquier tipo de paquete)? ¿Qué diferencia hay con AODV? Muestre el contenido del paquete explicando los campos que considere relevantes.

Ahora el primer paquete se envía en t=0.056712975726s, a diferencia de AODV, que no enviaba ningún paquete hasta t=10s, instante en el que comenzaba la transmisión UDP.

AODV es un protocolo reactivo, es decir, determina la ruta hacia un nodo concreto en el instante en el que quiere enviar un paquete a ese nodo, en este caso, en t=10s, cuando *static1* quiere enviar el primer paquete UDP a *static2*, envía un paquete RREQ para determinar la ruta hacia este.

Por otro lado, DSDV es un protocolo proactivo, es decir, precalcula las rutas a todos los posibles destinos antes de necesitarlas y va actualizando estas rutas periódicamente.

El primer paquete enviado en la simulación es un mensaje *Hello*, un nodo envía este paquete para informar a sus vecinos de las rutas que tiene en su tabla de enrutamiento, de forma que estos puedan mantener sus tablas de enrutamiento actualizadas. Estos paquetes se suelen enviar cuando el nodo ha realizado cambios en su tabla de enrutamiento (una entrada deja de estar disponible, se encontró una entrada de menor coste,...) o periódicamente cada 5s (*helloInterval* por defecto), para mantener las tablas de enrutamiento actualizadas.

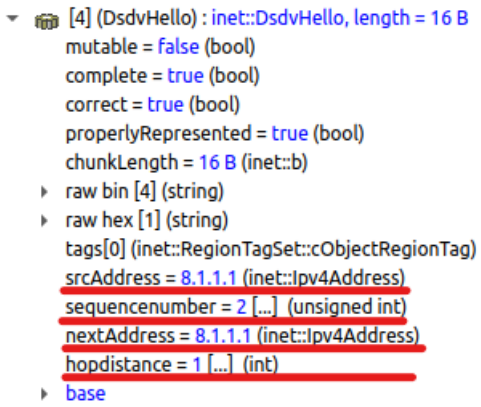

Figura 2.28: Contenido del mensaje *Hello* enviado por *static1*

En el contenido del paquete, el nodo *static1* indica la información necesaria para que sus vecinos puedan crear una entrada en su tabla de enrutamiento al recibir el paquete:

- ***srcAddress***: Indica el nodo origen del paquete *Hello*, es decir, quién está enviando el mensaje *Hello* para proporcionar su información de enrutamiento. Este campo se corresponderá con el destino para el que se ve crear una entrada en la tabla de enrutamiento de los nodos que reciban el paquete. En este caso, se corresponde con la dirección IP del nodo *static1*, que es quién envia el paquete.

- ***sequencenumber***: Número de secuencia (timestamp) que indica lo reciente que es una ruta. A diferencia de AODV, el número de secuencia comienza a contar en 2, se realiza esta implementación porque en DSDV si el número de secuencia es par, indica que el nodo está disponible/activo y, si es impar, indica que el nodo está no disponible/inactivo. En este caso, como es el primer paquete enviado por ese nodo y el nodo está activo, el número de secuencia es igual a 2.

- ***nextAddress***: Indica la dirección de siguiente salto, lo que corresponderá con el campo ***gw*** de la entrada que se va a crear. En este caso, como el paquete lo está enviando el propio *static1*, está enviando el paquete a sus vecinos, nodos alcanzables por él mismo, por tanto el siguiente salto es él mismo (no se necesita un nodo intermedio).

- ***hopdistance***: Distancia o número de saltos necesarios para alcanzar el nodo especificado en ***srcAddress***, se corresponderá con el campo ***metric*** de la entrada que se va a crear. En este caso, como se mencionó previamente, los nodos vecinos que reciban este paquete podran acceder directamente a *static1*, por tanto, este está a un salto.

Figura 2.29: Contenido del mensaje *Hello* enviado por *static1* en la capa IP

A nivel IP, vemos que el paquete es un broadcast normal y corriente, con dirección de origen la de *static1* y un TTL=32, que es el TTL por defecto para INET. Este paquete lo recibirán todos los nodos alcanzables directamente por *static1*, es decir, los que estén suficientemente cerca para recibir la señal.

2.11 Pregunta 11

Muestre algún ejemplo de paquete Hello con valor de hopdistance mayor que 1 y explique el significado de los campos srcAddress y nextAddress utilizando para explicarlos una captura de la tabla de enrutamiento del nodo que lo genera.

Cualquier paquete *Hello* que sea enviado desde un nodo X y contenga la información de enrutamiento de un nodo Y, tendrá un valor de *hopdistance* mayor que 1, porque lo recibirán nodos que no pueden alcanzar directamente a X o que, si pueden alcanzar directamente a X, ya tendrán una ruta hacia este con menor coste/*hopdistance* e ignorarán la recibida.

Inmediatamente después de que *static1* envíe su paquete *Hello*, el nodo *mobile[5]* envía un paquete *Hello* con la información de enrutamiento de *static1*.

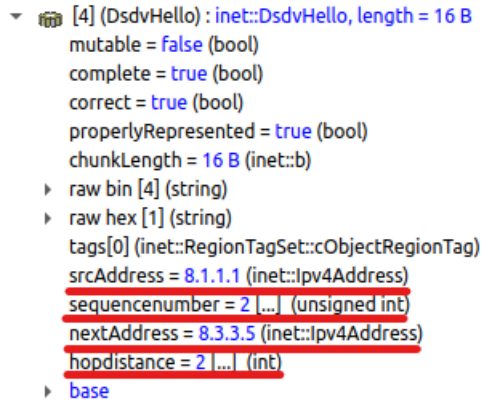

Figura 2.30: Contenido del mensaje *Hello* enviado por *mobile[5]*

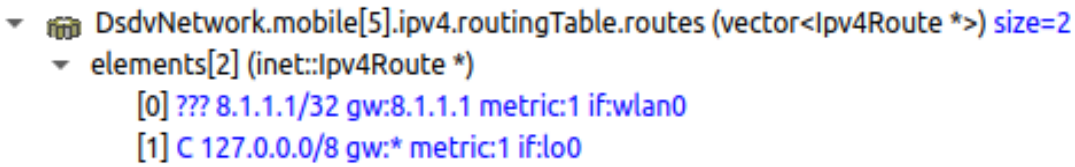

Figura 2.31: Tabla de enrutamiento del nodo *mobile[5]*

Explicación de los campos:

- ***srcAddress***: La dirección de origen seguirá siendo la misma, porque aunque el paquete lo envíe *mobile[5]*, el contenido que se envía es la ruta **hacia *static1***, por eso es su dirección IP la que ocupa este campo. Si nos fijamos en la tabla de enrutamiento de *mobile[5]*, coincide con la única entrada que tiene en su tabla (obviando la de loopback).

- ***sequencenumber***: El número de secuencia será el mismo, ya que estamos hablando de la misma ruta, no de una más reciente.

- ***nextAddress***: Ahora la dirección de siguiente salto será el propio *mobile[5]*, esto se debe a que *mobile[5]* tiene acceso directo a *static1*, podemos verlo en la entrada [0] de su tabla de enrutamiento, donde indica que *static1* está a un salto (*metric*=1), entonces, todos los nodos vecinos de *mobile[5]* que van a recibir este paquete y no tienen acceso directo a *static1*, pueden acceder a este a través de *mobile[5]*, es decir, *mobile[5]* es el siguiente salto en la ruta hacia *static1* para ellos.

- ***hopdistance***: Ahora la distancia o número de saltos que necesitarán los nodos que reciban el paquete para alcanzar a *static1* será igual a 2 (*mobile[5] –> static1*).

ACLARACIÓN: También habrá nodos con acceso directo a *static1* que recibirán este paquete, pero se habla solo de los que no tienen acceso directo, porque los que lo tienen, ya tendrán una entrada en su tabla de enrutamiento hacia *static1* y de menor coste, por tanto, no actualizarán sus tablas de enrutamiento (tal como se mencionó previamente).

2.12 Pregunta 12

Explique la actualización que se produce en la tabla de enrutamiento del nodo que lo recibe relacionándola con el contenido del paquete mostrado en la cuestión anterior

Aquí, dependiendo de lo que acabamos de comentar en la cuestión anterior, existen dos casos.

El nodo no tiene acceso directo a *static1*

En este caso, hemos escogido al nodo *mobile[10]*, que no tiene acceso directo a *static1* (no está lo suficientemente cerca), por tanto, al recibir el paquete, creará una entrada para *static1* (porque aún no tiene ninguna) o, relacionandolo con el paquete, creará una entrada para el nodo indicado en el campo ***srcAddress***. A esta entrada se le asignará como ***gw*** el valor del campo ***nextAddress*** del paquete recibido. Además, en el campo ***metric*** (coste), se asignará el valor del campo ***hopdistance*** del paquete recibido. En las figuras de debajo podemos ver como estos valores coinciden y el log de *mobile[10]* cuando recibe el paquete *Hello*.

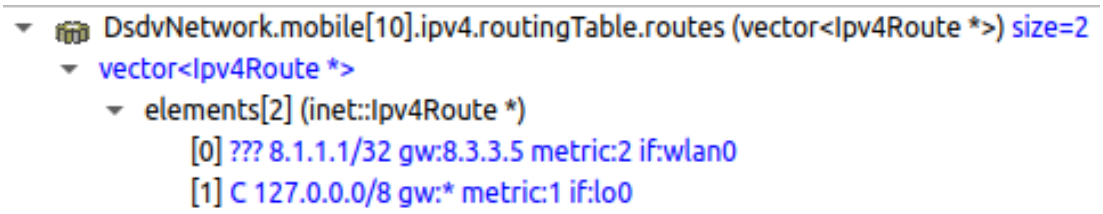

Figura 2.32: Tabla de enrutamiento del nodo *mobile[10]*

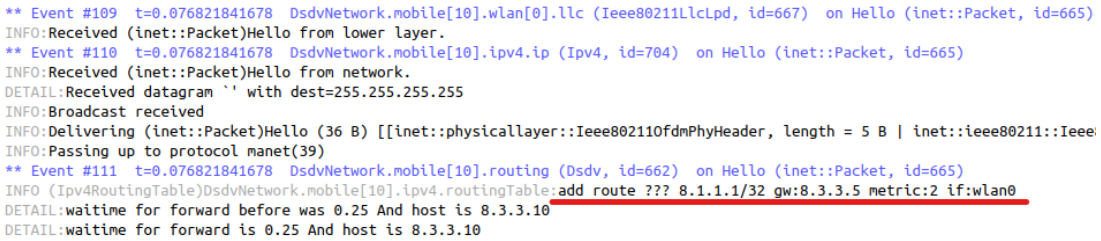

Figura 2.33: Log de la recepción del paquete *Hello* en *mobile[10]*

El nodo tiene acceso directo a *static1*

En este caso, hemos escogido al nodo *mobile[12]* que tiene acceso directo a *static1*. Cuando este nodo reciba el paquete *Hello*, comparará el campo ***srcAddress*** con sus entradas y verá que ya tiene una entrada para ese destino (*static1*). A continuación, comparará el campo ***sequencenumber*** para ver si es mayor que su número de secuencia para el nodo *static1*, en este caso, verá que es el mismo (2). Por último, comparará el campo ***hopdistance*** del paquete con el campo ***metric*** de la entrada correspondiente y verá que la entrada que tiene es de menor coste (***metric*** < ***hopdistance***), por tanto, no actualizará la entrada. A continuación, podemos ver que la tabla de enrutamiento de *mobile[12]* no fue actualizada tras recibir el paquete y el log de la recepción del paquete, en el que se ve que no hace nada.

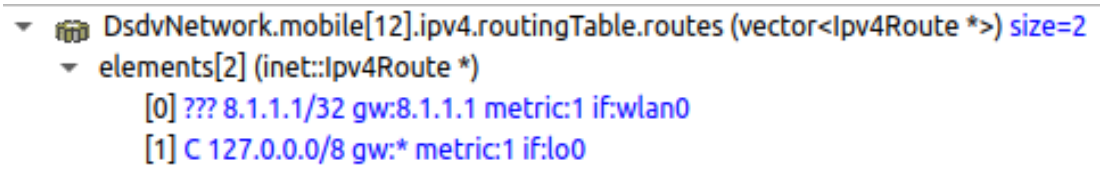

Figura 2.34: Tabla de enrutamiento del nodo *mobile[12]*

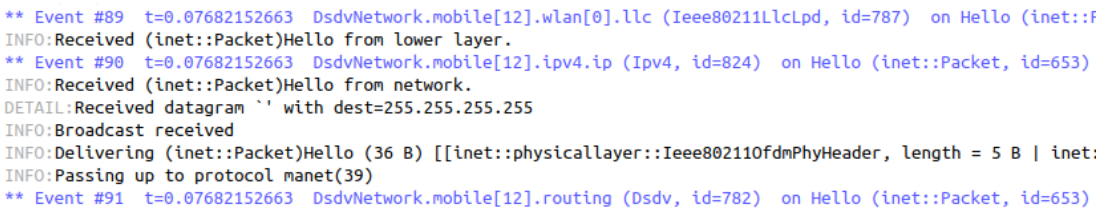

Figura 2.35: Log de la recepción del paquete *Hello* en *mobile[12]*

2.13 Pregunta 13

¿En qué instante llega ahora el paquete UdpBasicAppData-0 a static2? ¿Llega antes o después que en AODV? ¿Por qué?

Ahora el paquete *UdpBasicAppData-0* llega en t=10.00519469523s mientras que en AODV llegaba en t=10.829362889087s, es decir, una diferencia de unos 0.824s=824ms. Esta diferencia se debe al cálculo de la ruta hacia *static2*, en AODV, la ruta comienza a calcularse en t=10s, cuando el nodo *static1* quiere transmitir, el cálculo de la ruta añade un retardo de unos 825ms (calculado en Sección 2.6 página 18), que coincide (aproximadamente) con la diferencia de retardo entre AODV y DSDV.

Por otro lado, en DSDV, cuando *static1* quiere transmirtir, la ruta hacia *static2* ya ha sido calculada previamente mediante el intercambio de mensajes *Hello*, entonces se ahorra estos 824/825ms de retardo.

```
** Event #42033  t=10.00519469523  DsdvNetwork.static2.ipv4.ip (Ipv4, id=1066)  on UdpBasicAppData-0 (inet::Packet, id=45428)
INFO:Received (inet::Packet)UdpBasicAppData-0 from network.
DETAIL:Received datagram `' with dest=8.2.2.2
INFO:Delivering (inet::Packet)UdpBasicAppData-0 (128 B) [[inet::physicallayer::Ieee80211OfdmPhyHeader, length = 5 B | inet::i
INFO:Passing up to protocol udp(57)
** Event #42034  t=10.00519469523  DsdvNetwork.static2.udp (Udp, id=1019)  on UdpBasicAppData-0 (inet::Packet, id=45428)
INFO:Packet UdpBasicAppData-0 received from network, dest port 5001
INFO:Sending payload up to socket sockId=1
```

Figura 2.36: Log de la recepción del paquete *UdpBasicAppData-0* en *static2*

2.14 Pregunta 14

Vuelva a simular la caída del último nodo de la ruta hacia static2 en t=13s (es posible que el nodo sea diferente al de la cuestión 7). ¿Se notifica de alguna manera al resto de nodos? ¿Cómo se repara la ruta en este caso?

El último nodo de la ruta ahora es *mobile[1]*, este será el nodo que simule la caída. El problema aquí será que, a diferencia de AODV, cuando el enlace entre *mobile[0]* (nodo anterior de la ruta) y *mobile[1]* falle, *mobile[0]* alcanzará el límite de intentos de envío del datagrama en la capa de enlace, RETRY_LIMIT (7), como en AODV, pero no informará, solo descartará el datagrama UDP, entonces, *static1* enviará el siguiente datagrama UDP con normalidad y volverá a suceder lo mismo continuamente.

¿Cuándo se solucionará el problema?

Cuando, debido a las actualizaciones periódicas, los nodos intercambien paquetes *Hello* y actualicen sus rutas, esto no sucederá hasta t=15.073363604843s, instante en el que el nodo *static1* envía un paquete *Hello*, sin embargo, el problema no se soluciona hasta que se encuentre una ruta alternativa hacia *static2*, esto no ocurre hasta t=17.204600497678s, instante en que el nodo *static2* vuelve a recibir paquetes UDP porque se encuentra una ruta alternativa gracias al intercambio de paquetes *Hello*.

Ruta antigua: *static1 –> mobile[8] –> mobile[10] –> mobile[0] –> mobile[1] –> static2*

Ruta nueva: *static1 –> mobile[12] –> mobile[2] –> mobile[6] –> mobile[7] –> static2*

En conclusión, no se notifica de ninguna manera que el nodo *mobile[1]* ha caído, si no que se espera a que el problema se solucione por sí solo durante el intercambio periódico de paquetes *Hello*, esto demuestra que DSDV no funciona bien en escenarios con cambios de topología, ya que tardó 4.2s (desde t=13s hasta t=17.2s) en recuperarse de la caída de un nodo y provocó que se perdiesen 20 datagramas UDP (desde *UdpBasicAppData-15* hasta *UdpBasicAppData-35*, ambos incluidos).

```
** Event #77950  t=13.027282657061  DsdvNetwork.mobile[0].wlan[0].mac.dcf (Dcf, id=75)  on selfmsg startRxTimeout (omnetpp::cMessage, id=515)
INFO:Frame sequence aborted.
INFO:Data/Mgmt frame transmission failed
INFO:Incremented station SRC: stationShortRetryCounter = 7.
WARN:Retry limit reached for (inet::Packet)UdpBasicAppData-15 (164 B) [[inet::ieee80211::Ieee80211DataHeader, length = 24 B | inet::Ieee8022L
INFO:Dropping frame UdpBasicAppData-15, because retry limit is reached.
INFO (Dcaf)DsdvNetwork.mobile[0].wlan[0].mac.dcf.channelAccess:Channel released.
```

Figura 2.37: Primer paquete perdido tras la caída del nodo *mobile[1]*

```
** Event #108630  t=17.204600497678  DsdvNetwork.static2.udp (Udp, id=1036)  on UdpBasicAppData-36 (inet::Packet, id=250588)
INFO:Packet UdpBasicAppData-36 received from network, dest port 5001
INFO:Sending payload up to socket sockId=1
```

Figura 2.38: Primer paquete enviado con éxito tras la caída del nodo *mobile[1]*

2.15 Pregunta 15

Simule AODV durante 500s con 15 nodos móviles a una velocidad de 3m/s y con una potencia de 2mW para todos los nodos (sin la caída del nodo de los apartados 7 y 14). ¿Qué porcentaje de paquetes llega desde static1 a static2? ¿Qué porcentaje llega desde static2 a static1? Explique los valores.

Figura 2.39: Paquetes recibidos y enviados desde el nodo *static1*

Figura 2.40: Paquetes recibidos y enviados (echo) desde el nodo *static2*

A partir de las figuras de arriba podemos calcular estos porcentajes:

- **Porcentaje de paquetes que llega desde static1 a static2** = echoed(*static2*) / sent(*static1*) = 2402 / 2451 = 0.98 = **98%**
- **Porcentaje de paquetes que llega desde static2 a static1** = rcvd(*static1*) / echoed(*static2*) = 2402 / 2402 = 1 = **100%**

Como podemos ver, el rendimiento de la red es muy bueno, prácticamente no se pierde ningún paquete, esto se debe a que este escenario, en el que hay cambios de topología

constantes (los nodos móviles se mueven bastante) y un nodo, *static1*, se comunica con un conjunto pequeño de nodos, en concreto solo con *static2*, es idóneo para utilizar protocolos de enrutamiento reactivos, como es AODV.

Seguramente los pocos paquetes que se han perdido, se habrán perdido durante cambios de topología, en los que la transmisión de los datagramas UDP falle y se envie un RERR para actualizar la ruta entre *static1* y *static2*.

2.16 Pregunta 16

Haga lo mismo con DSDV. Explique las diferencias con respecto a AODV en porcentajes de ida y de vuelta. ¿Qué diferencia hay en cantidad de tráfico con AODV? (Para medir el tráfico utilice los valores fwd, up mostrados en el módulo ipv4 de un nodo móvil al azar haciendo doble click sobre él.)

Figura 2.41: Paquetes recibidos y enviados desde el nodo *static1*

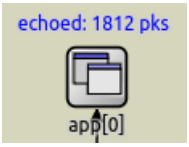

Figura 2.42: Paquetes recibidos y enviados (echo) desde el nodo *static2*

Cálculo de porcentajes:

- **Porcentaje de paquetes que llega desde static1 a static2** = echoed(*static2*) / sent(*static1*) = 1812 / 2451 = 0.74 = **74%**
- **Porcentaje de paquetes que llega desde static2 a static1** = rcvd(*static1*) / echoed(*static2*) = 1590 / 1812 = 0.88 = **88%**

Como podemos ver, el rendimiento de la red ha bajado mucho con DSDV, esto se debe a que este escenario no funciona bien con protocolos proactivos, como es DSDV, los protocolos proactivos funcionarán mejor en redes con pocos cambios de topología, ya vimos previamente

lo que afecta un cambio de topología a DSDV (Sección 2.14 página 34), y en las que un nodo suele comunicarse con muchos nodos distintos, todo lo contrario a este escenario.

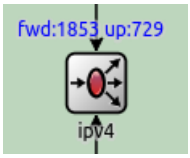

Figura 2.43: AODV - Cantidad de tráfico (a nivel IP) en el nodo *mobile[8]*

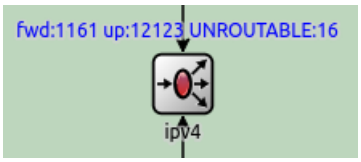

Figura 2.44: DSDV - Cantidad de tráfico (a nivel IP) en el nodo *mobile[8]*

La cantidad de tráfico en DSDV es desmesuradamente más grande, en concreto, los paquetes *up*, es decir, los paquetes que se envían a capas superiores (UDP), esto se debe a la cantidad de paquetes *Hello* generados durante toda la simulación.

Aquí se ve reflejada una de las desventajas de los protocolos proactivos y es que pueden aumentar mucho la sobrecarga provocada por el enrutamiento en la red, ya que se están generando mensajes de control continuamente (de forma periódica).

2.17 Pregunta 17

Repita la simulación de DSDV modificando el parámetro helloInterval a 10s y a 2s (el valor por defecto es 5s). ¿Cómo afecta cada valor al porcentaje de paquetes perdidos? ¿Por qué? ¿Cómo afecta a la cantidad de tráfico?

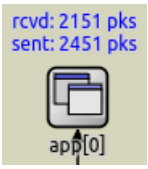

Figura 2.45: Paquetes recibidos y enviados desde el nodo *static1* (*helloInterval* = 2s)

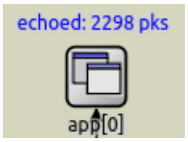

Figura 2.46: Paquetes recibidos y enviados (echo) desde el nodo *static2* (*helloInterval* = 2s)

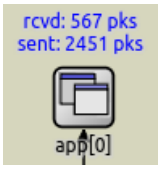

Figura 2.47: Paquetes recibidos y enviados desde el nodo *static1* (*helloInterval* = 10s)

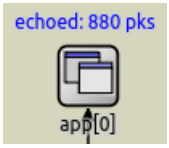

Figura 2.48: Paquetes recibidos y enviados (echo) desde el nodo *static2* (*helloInterval* = 10s)

Cálculo de porcentajes:

- **Porcentaje de paquetes que llega desde static1 a static2 (*helloInterval*=2s)** = echoed(*static2*) / sent(*static1*) = 2298 / 2451 = 0.94 = **94%**
- **Porcentaje de paquetes que llega desde static1 a static2 (*helloInterval*=10s)** = echoed(*static2*) / sent(*static1*) = 880 / 2451 = 0.36 = **36%**
- **Porcentaje de paquetes que llega desde static2 a static1 (*helloInterval*=2s)** = rcvd(*static1*) / echoed(*static2*) = 2151 / 2298 = 0.94 = **94%**
- **Porcentaje de paquetes que llega desde static2 a static1 (*helloInterval*=10s)** = rcvd(*static1*) / echoed(*static2*) = 567 / 880 = 0.64 = **64%**

Se ve claramente que reducir el *helloInterval* beneficia al rendimiento de la red, la cantidad de paquetes perdidos es mucho menor. Se pierde un 6% de los paquetes de ida frente a un 64% aumentando el *helloInterval*, por otro lado, se pierde otro 6% de vuelta frente a un 36% aumentando el *helloInterval*.

¿Por qué pasa esto?

La mayoría de los paquetes se perderán durante cambios de topología, en los que los nodos se hayan movido y la ruta que se estaba utilizando para enviar los datagramas UDP

ya no sea válida, para que esta ruta se actualice lo que tiene que pasar es que, debido a las actualizaciones periódicas, los nodos comiencen a intercambiar paquetes *Hello* y actualicen sus rutas, por tanto, cuanto menor sea el *helloInterval*, menor será el tiempo que tardará en suceder esto y, consecuentemente, menor será el tiempo en el que se estén perdiendo paquetes.

Comentar también, que esto afecta más a la ida que a la vuelta porque, si hay un cambio de topología durante la ida, el datagrama UDP se descartará y el nodo *static1* seguirá enviando datagramas UDP que se van a perder mientras los nodos intermedios no actualicen sus rutas. Sin embargo, si esto ocurre durante la vuelta, el datagrama UDP de respuesta será descartado y se perderá, pero el nodo *static2* no seguirá generando paquetes UDP de respuesta nuevos, porque no tiene ningún paquete UDP nuevo al que responder, por tanto, solo se perderá ese paquete.

¡OJO! De todas formas, si que podría darse el caso en el que el cambio de topología solo afecte a la ruta de vuelta de los datagramas UDP, ya que en DSDV las rutas de ida y de vuelta de los datagramas no tienen porqué coincidir y podría darse el caso de que el nodo *static2* siga recibiendo datagramas UDP, respondiendo a ellos y que esos paquetes se estén perdiendo porque la ruta de vuelta actual no sea válida pero la de ida si.