



TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOGÍAS DA INFORMACIÓN



Práctica 3: MANETs en INET

Estudiante 1: Óscar Olveira Miniño

Estudiante 2: Alejandro Javier Herrero Arango

A Coruña, diciembre de 2024.

Índice general

1 AODV	1
1.1 Ejercicio 1.1	1
1.2 Ejercicio 1.2	3
1.3 Ejercicio 1.3	4
1.4 Ejercicio 1.4	5
1.5 Ejercicio 1.5	7
1.6 Ejercicio 1.6	8
1.7 Ejercicio 1.7	9
1.8 Ejercicio 1.8	10
2 DSDV	12
2.1 Ejercicio 2.1	12
2.2 Ejercicio 2.2	13
2.3 Ejercicio 2.3	13
2.4 Ejercicio 2.4	14
2.5 Ejercicio 2.5	16
2.6 Ejercicio 2.6	17
3 AODV vs. DSDV	20
3.1 Ejercicio 3.1	20
3.2 Ejercicio 3.2	20
3.3 Ejercicio 3.3	20
3.4 Ejercicio 3.4	21
3.5 Ejercicio 3.5	23
4 AODV vs. DSDV: gráficas	24
4.1 Ejercicio 4.1	24
4.2 Ejercicio 4.2	25
4.3 Ejercicio 4.3	26
4.4 Ejercicio 4.4	26

Índice de figuras

1.1	Logs que muestran el envío del primer RREQ y los nodos que lo reciben	2
1.2	Nodos que reenvían el primer RREQ (A la izquierda, mobile[10]; A la derecha, mobile[12])	2
1.3	Disminución del TTL del segundo RREQ al ser reenviado por los nodos	2
1.4	Tabla de enrutamiento de mobile[7] antes de recibir el primer RREQ	3
1.5	Tabla de enrutamiento de mobile[7] después de recibir el primer RREQ	3
1.6	Tabla de enrutamiento de mobile[7] antes de recibir el primer RREP	4
1.7	Tabla de enrutamiento de mobile[7] después de recibir el primer RREP	4
1.8	Escenario en t=15 con todos los nodos en funcionamiento	5
1.9	Detalle de los nodos cercanos a static2 en t=15 con mobile[0] caído	5
1.10	Log de detección de la caída de mobile[0]	6
1.11	Log del envío del RERR a broadcast	6
1.12	Logs de envío del RERR enviado por mobile[9]	6
1.13	Detalle del paquete de error enviado por mobile[2] en t=15,037	7
1.14	Logs de envío del RERR producido por mobile[2]	8
1.15	Tabla de enrutamiento de mobile[1] antes de recibir el RERR	9
1.16	Tabla de enrutamiento de mobile[1] después de recibir el RERR	9
1.17	RREQ enviado por static1 tras la recepción del RREP	10
1.18	Primer RREQ enviado por static1	11
2.1	Log del nodo que manda el primer Hello con hopdistance 3	12
2.2	Tabla de enrutamiento nodo 8	13
2.3	Tabla de enrutamiento nodo 5 antes recibir Hello de nodo 8	14
2.4	Tabla de enrutamiento nodo 9 antes recibir Hello de nodo 8	15
2.5	Tabla de enrutamiento nodo 5 antes recibir Hello de nodo 8	15
2.6	Tabla de enrutamiento nodo 9 antes recibir Hello de nodo 8	16
2.7	Ruta antes de la caída	17
2.8	Mensaje del nodo 1 cuando no es capaz de enviar los UDP datas al detectar un nodo caído	17
2.9	Ruta nueva después de la caída del nodo 2	17
2.10	Stati2 mandando mensaje Hello	18
2.11	Nodo 11 antes de recibir Hello de static2	18
2.12	Nodo 11 después de recibir Hello de static2	19
3.1	Static1 a los 300s en AODV	21
3.2	Static2 a los 300s en AODV	22
3.3	Static1 a los 300s en DSDV	22
3.4	Static2 a los 300s en DSDV	23

4.1	Gráfica paquetes recibidos en static2 en función de la potencia	24
4.2	Gráfica paquetes recibidos en static2 en función número de nodos	25
4.3	Gráfica paquetes recibidos en static2 en función de la velocidad	26

Capítulo 1

AODV

Nota

La semilla usada durante la práctica es **2701**, a no ser que se especifique en un ejercicio, una semilla concreta

1.1 Ejercicio 1.1

1.1.1 ¿Qué nodos reenvían el primer paquete RREQ enviado por static1? ¿Y el segundo RREQ? ¿Por qué?

El primer paquete RREQ enviado por static1 es procesado y reenviado únicamente por los nodos mobile[10] y mobile [12], a pesar de enviarse con intención de alcanzar todos los nodos de la red (Figura 1.1). Esto ocurre porque ese primer paquete parte de static1 con un TTL igual a 2. Mobile[10] y mobile[12] son los únicos que se encuentran en un posición dentro del rango de alcance de static1 y que, por tanto, reciben el RREQ con potencia suficiente para intentar procesarlo, reduciéndose el TTL en 1. Después, lo reenvían de forma que, cuando llega a los siguientes nodos, el TTL se reduce 1 más, cayendo a 0 y frenándose el proceso de nuevos reenvíos.

Este comportamiento está justificado en el [RFC 3561 sobre AODV](#), que dice que TTL_START suele ser 1, aunque se recomienda que sea al menos 2 cuando los mensajes se usan para conectividad local. Además, el TTL_INCREMENT por defecto es 2, por lo que en la siguiente iteración el TTL sería 4.

Debido a lo comentado anteriormente, el segundo RREQ es reenviado por 10 y 12 primero (TTL = 3). Gracias a mobile[10] llega a mobile[3] y mobile[1] (TTL = 2), que lo reenvían de nuevo. Finalmente, mobile[7] y mobile[2] lo reenvían una última vez (TTL = 1). Llegados a este punto, el TTL con el que alcanza los siguientes nodos es 0, por lo que ese segundo RREQ ya no se reenviaría más.

```

** Event #1719 t=10.002964223082 Manet.static1.wlan[0].radio (Ieee80211ScalarRadio, id=79) on aodv::Rreq (inet::Packet, id=1642)
INFO: Transmission started: (inet::physicallayer::WirelessSignal)aodv::Rreq (58 us 99 B) (inet::Packet)aodv::Rreq (99 B) (inet::SequenceNumber)
INFO: Changing radio transmission state from IDLE to TRANSMITTING.
INFO: Changing radio transmitted signal part from NONE to WHOLE.
** Event #1720 t=10.002964556393 Manet.mobile[10].wlan[0].radio (Ieee80211ScalarRadio, id=860) on aodv::Rreq (inet::physicallayer)
INFO: Reception started: attempting (inet::physicallayer::WirelessSignal)aodv::Rreq (58 us 99 B) (inet::Packet)aodv::Rreq (99 B) (inet::PhysicalLayer)
INFO: Changing radio reception state from IDLE to RECEIVING.
INFO: Changing radio received signal part from NONE to WHOLE.
** Event #1721 t=10.002965000243 Manet.mobile[10].wlan[0].radio (Ieee80211ScalarRadio, id=990) on aodv::Rreq (inet::physicallayer)
INFO: Reception started: attempting (inet::physicallayer::WirelessSignal)aodv::Rreq (58 us 99 B) (inet::Packet)aodv::Rreq (99 B) (inet::PhysicalLayer)
INFO: Changing radio reception state from IDLE to RECEIVING.
INFO: Changing radio received signal part from NONE to WHOLE.
** Event #1722 t=10.002965127308 Manet.mobile[1].wlan[0].radio (Ieee80211ScalarRadio, id=275) on aodv::Rreq (inet::physicallayer)
INFO: Reception started: not attempting (inet::physicallayer::WirelessSignal)aodv::Rreq (58 us 99 B) (inet::Packet)aodv::Rreq (99 B)
** Event #1723 t=10.002965194789 Manet.mobile[3].wlan[0].radio (Ieee80211ScalarRadio, id=405) on aodv::Rreq (inet::physicallayer)
INFO: Reception started: not attempting (inet::physicallayer::WirelessSignal)aodv::Rreq (58 us 99 B) (inet::Packet)aodv::Rreq (99 B)
** Event #1724 t=10.002965069556 Manet.mobile[7].wlan[0].radio (Ieee80211ScalarRadio, id=665) on aodv::Rreq (inet::physicallayer)
INFO: Reception started: not attempting (inet::physicallayer::WirelessSignal)aodv::Rreq (58 us 99 B) (inet::Packet)aodv::Rreq (99 B)
** Event #1725 t=10.002965764806 Manet.mobile[2].wlan[0].radio (Ieee80211ScalarRadio, id=340) on aodv::Rreq (inet::physicallayer)
INFO: Reception started: not attempting (inet::physicallayer::WirelessSignal)aodv::Rreq (58 us 99 B) (inet::Packet)aodv::Rreq (99 B)
** Event #1726 t=10.002965801036 Manet.mobile[11].wlan[0].radio (Ieee80211ScalarRadio, id=925) on aodv::Rreq (inet::physicallayer)
INFO: Reception started: not attempting (inet::physicallayer::WirelessSignal)aodv::Rreq (58 us 99 B) (inet::Packet)aodv::Rreq (99 B)
** Event #1727 t=10.002966033624 Manet.mobile[6].wlan[0].radio (Ieee80211ScalarRadio, id=600) on aodv::Rreq (inet::physicallayer)
INFO: Reception started: not attempting (inet::physicallayer::WirelessSignal)aodv::Rreq (58 us 99 B) (inet::Packet)aodv::Rreq (99 B)
** Event #1728 t=10.002966245726 Manet.mobile[13].wlan[0].radio (Ieee80211ScalarRadio, id=1055) on aodv::Rreq (inet::physicallayer)
INFO: Reception started: not attempting (inet::physicallayer::WirelessSignal)aodv::Rreq (58 us 99 B) (inet::Packet)aodv::Rreq (99 B)
** Event #1729 t=10.002966273179 Manet.mobile[0].wlan[0].radio (Ieee80211ScalarRadio, id=210) on aodv::Rreq (inet::physicallayer)
INFO: Reception started: not attempting (inet::physicallayer::WirelessSignal)aodv::Rreq (58 us 99 B) (inet::Packet)aodv::Rreq (99 B)
** Event #1730 t=10.002966524973 Manet.mobile[5].wlan[0].radio (Ieee80211ScalarRadio, id=535) on aodv::Rreq (inet::physicallayer)

```

Figura 1.1: Logs que muestran el envío del primer RREQ y los nodos que lo reciben

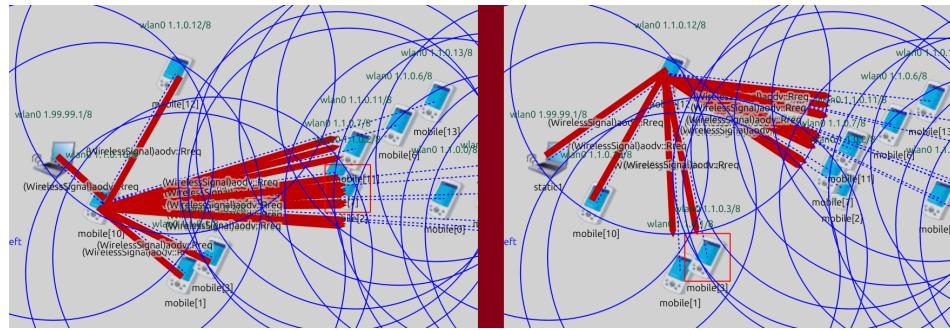


Figura 1.2: Nodos que reenvían el primer RREQ (A la izquierda, mobile[10]; A la derecha, mobile[12])

```

** Event #1949 t=10.32 Manet.static1.routingApp (Aodv, id=70)
INFO: We didn't get any Route Reply within RREP timeout
INFO: Sending a Route Request with target 1.99.99.2 and TTL= 4

** Event #1985 t=10.324517210677 Manet.mobile[10].routingApp (Aodv, id=851)
INFO: AODV Route Request arrived with source addr: 1.99.99.1 originator addr: 1.1.0.10
DETAIL: Updating existing route: destination = 1.99.99.1, prefixLength = 32, n
DETAIL: Route updated: destination = 1.99.99.1, prefixLength = 32, nextHop = 1
DETAIL: Updating existing route: destination = 1.99.99.1, prefixLength = 32, n
DETAIL: Route updated: destination = 1.99.99.1, prefixLength = 32, nextHop = 1
INFO: Forwarding the Route Request message with TTL= 3

** Event #2044 t=10.325939057474 Manet.mobile[1].routingApp (Aodv, id=266)
INFO: AODV Route Request arrived with source addr: 1.1.0.10 originator addr: 1
DETAIL: Updating existing route: destination = 1.1.0.10, prefixLength = 32, n
DETAIL: Route updated: destination = 1.1.0.10, prefixLength = 32, nextHop = 1.
DETAIL: Updating existing route: destination = 1.99.99.1, prefixLength = 32, n
DETAIL: Route updated: destination = 1.99.99.1, prefixLength = 32, nextHop = 1
INFO: Forwarding the Route Request message with TTL= 2

** Event #2219 t=10.330544624389 Manet.mobile[2].routingApp (Aodv, id=331)
INFO: AODV Route Request arrived with source addr: 1.1.0.3 originator addr: 1
DETAIL: Adding new route destination = 1.1.0.3, prefixLength = 32, nextHop =
INFO (Ipv4RoutingTable)Manet.mobile[2].ipv4.routingTable: add route ??? 1.1.0.3
DETAIL: Adding new route destination = 1.99.99.1, prefixLength = 32, nextHop
INFO (Ipv4RoutingTable)Manet.mobile[2].ipv4.routingTable: add route ??? 1.99.99.1
INFO: Forwarding the Route Request message with TTL= 1

```

Figura 1.3: Disminución del TTL del segundo RREQ al ser reenviado por los nodos

1.2 Ejercicio 1.2

- 1.2.1 Elige el nodo intermedio de la ruta que sigue el primer paquete RREQ que llega a static2. Muestra su tabla de enrutamiento (vector <Ipv4route *> dentro del módulo ipv4.routingTable) justo antes y justo después de recibir el primer RREQ. Explica las diferencias y cómo se crean las entradas que aparecen (incluyendo los campos más importantes).**

Desde el principio de la simulación hasta que llega a mobile[7] el primer RREQ, que es el segundo en términos globales, la tabla de enrutamiento contiene únicamente la dirección del segmento de red, con máscara de subred /8 y /9, y la de localhost (Figura 1.4).

Tras recibir el primer RREQ, la tabla de mobile[7] se puebla con dos nuevas entradas (Figura 1.5). La primera de ellas es la dirección de mobile[3], quien le envía ese primer route request y la segunda es la ruta al origen mediante mobile[3]. Estas rutas nuevas contienen más información en la tabla que las anteriores de acuerdo a las necesidades de AODV. Ambas se generan con la recepción del RREQ y se actualizarán con los RREQ siguientes recibidos por mobile[7]. Vemos como la dirección de destino es la de mobile[3] para una y la de static1 para la otra, mientras que las dos tienen el mismo *gateway* (mobile[3]). Otra diferencia apreciable está en la métrica, que es 3 para el segundo caso debido a que hay tres saltos hasta static1. Finalmente, cabe destacar que ambas tienen el mismo número de secuencia que se incrementa con cada RREQ para indicar que la información es actualizada, pero la primera entrada se marca como inválida para evitar posibles bucles y mantener solo las rutas necesarias (static1 a través de mobile[3]).

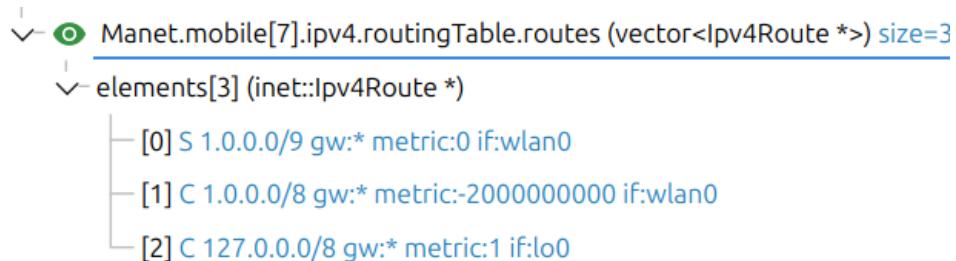


Figura 1.4: Tabla de enrutamiento de mobile[7] antes de recibir el primer RREQ



Figura 1.5: Tabla de enrutamiento de mobile[7] después de recibir el primer RREQ

1.3 Ejercicio 1.3

1.3.1 Haz lo mismo justo antes y justo después del primer RREP.

Como se ve en la figura 1.6, antes de que llegue a mobile[7] el primer RREP la tabla de enrutamiento de encuentra mucho más llena que cuando hablábamos del RREQ. Ahora la tabla contiene las direcciones de todos los vecinos directos (Alcanzables en un salto) de mobile[7], además de la ruta a static1.

Por otra parte, la segunda imagen (Figura 1.7) introduce una entrada más, la de static2. Esta entrada se crea gracias a la recepción del RREP, que intenta rehacer el camino de vuelta al origen desde el destino siguiendo la ruta más corta. Como esta ruta sí es necesaria mantenerla y no es de un vecino de mobile[7], se marca como válida en hasValidDestNum. Podemos ver también que la ruta a mobile[0] y la ruta a static2 tienen un número de secuencia 0. Este número ha sido generado por static2 al enviar el RREP y significa que la ruta es válida y es la de máxima frescura, por lo que es la que será usada para comunicar static1 con static2 (El número de secuencia en AODV es un entero sin signo, por lo que el 0 realmente es el mayor valor representable). Aunque en menor importancia, también podemos fijarnos en que el lifetime de la nueva entrada es mayor que el resto, esto es debido a que se generó más tarde que las otras (Lifetime = Instante actual + ART).

```

[0] ??? 1.1.0.0/32 gw:1.1.0.0 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 4, lifetime = 14.451979941873
[1] ??? 1.1.0.2/32 gw:1.1.0.2 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 4, lifetime = 14.45415136536
[2] ??? 1.1.0.3/32 gw:1.1.0.3 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 4, lifetime = 14.450015640549
[3] ??? 1.1.0.5/32 gw:1.1.0.5 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 4, lifetime = 14.45390328499
[4] ??? 1.1.0.6/32 gw:1.1.0.6 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 4, lifetime = 14.453334632851
[5] ??? 1.1.0.11/32 gw:1.1.0.11 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 4, lifetime = 14.453527559494
[6] ??? 1.1.0.13/32 gw:1.1.0.13 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 4, lifetime = 14.45484453818
[7] ??? 1.99.99.1/32 gw:1.1.0.3 metric:3 if:wlan0 isActive = 1, hasValidDestNum = 1, destNum = 4, lifetime = 16.810015640549
[8] S 1.0.0.0/9 gw:* metric:0 if:wlan0
[9] C 1.0.0.0/8 gw:* metric:-2000000000 if:wlan0
[10] C 127.0.0.0/8 gw:* metric:1 if:lo0

```

Figura 1.6: Tabla de enrutamiento de mobile[7] antes de recibir el primer RREP

```

[0] ??? 1.1.0.0/32 gw:1.1.0.0 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, lifetime = 14.456681608003, precursor list: 1.1.0.3
[1] ??? 1.1.0.2/32 gw:1.1.0.2 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 4, lifetime = 14.45415136536
[2] ??? 1.1.0.3/32 gw:1.1.0.3 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 4, lifetime = 14.450015640549
[3] ??? 1.1.0.5/32 gw:1.1.0.5 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 4, lifetime = 14.45390328499
[4] ??? 1.1.0.6/32 gw:1.1.0.6 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 4, lifetime = 14.453334632851
[5] ??? 1.1.0.11/32 gw:1.1.0.11 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 4, lifetime = 14.453527559494
[6] ??? 1.1.0.13/32 gw:1.1.0.13 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 4, lifetime = 14.45484453818
[7] ??? 1.99.99.1/32 gw:1.1.0.3 metric:3 if:wlan0 isActive = 1, hasValidDestNum = 1, destNum = 4, lifetime = 16.810015640549
[8] ??? 1.99.99.2/32 gw:1.1.0.0 metric:3 if:wlan0 isActive = 1, hasValidDestNum = 1, destNum = 0, lifetime = 17.456681608003, precursor list: 1.1.0.3
[9] S 1.0.0.0/9 gw:* metric:0 if:wlan0
[10] C 1.0.0.0/8 gw:* metric:-2000000000 if:wlan0
[11] C 127.0.0.0/8 gw:* metric:1 if:lo0

```

Figura 1.7: Tabla de enrutamiento de mobile[7] después de recibir el primer RREP

1.4 Ejercicio 1.4

- 1.4.1 Tras aplicar la nueva configuración, ¿Cuál es el primer nodo en darse cuenta de la caída? ¿Cómo? Muestra una captura del log del nodo que se da cuenta que muestre el motivo. ¿Notifica este nodo la caída del nodo?**

Primero de todo, debe aclararse que, como se observa en la figura 1.8, en el escenario producido con la semilla actual, en $t=15$, los nodos más alejados de static1 que pueden producir una ruta alternativa son mobile[0] y mobile[5], ya que si tirásemos el más cercano a static2 se perdería la comunicación al no haber otro nodo en su rango.

Tras aplicar la configuración para mobile[0] podemos observar en la simulación que este nodo aparece marcado como caído, pero sigue existiendo una ruta hasta static2 (Figura 1.9).

Ahora, como vemos en la figura 1.10, la caída de mobile[0] es detectada rápidamente por mobile[2], que es el nodo previo en el camino hacia static2. Mobile[2] intenta contactar durante un período de tiempo con mobile[0], pero al no recibir respuesta, lo acaba detectando como ruta inválida.

También podemos ver en los logs (Figura 1.11) que mobile[2] notifica a todos los nodos posibles mediante broadcast que mobile[0] no está disponible. En la figura 1.12 se muestra cómo mobile[2] transmite el paquete RERR y los miembros a su alcance lo reciben.



Figura 1.8: Escenario en $t=15$ con todos los nodos en funcionamiento

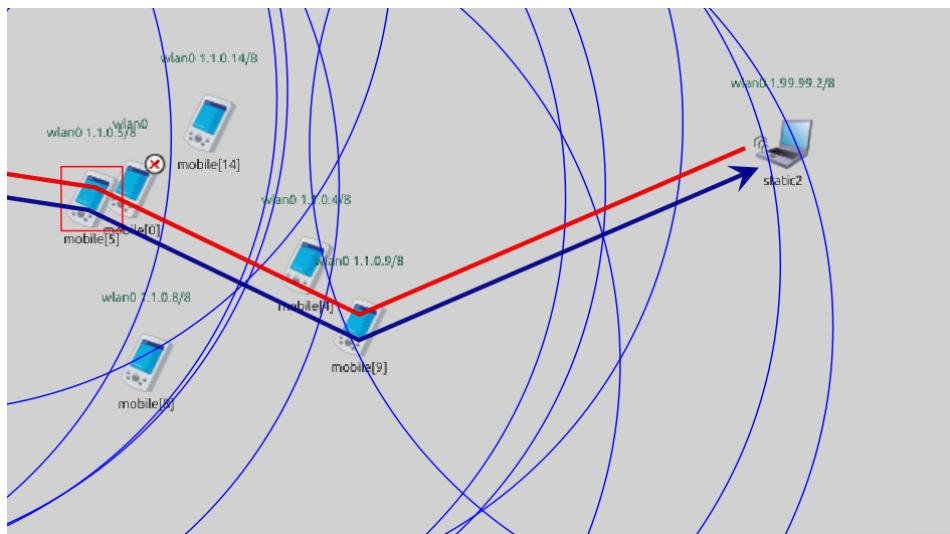


Figura 1.9: Detalle de los nodos cercanos a static2 en $t=15$ con mobile[0] caído

```
** Event #174699 t=15.032462833376 Manet.mobile[2].wlan[0].mac.dcf (Dcf, id=353) on selfmsg startRxTimeout (omnetpp::cMessage, id=904817)
INFO: Frame sequence aborted.
INFO: Data/Mgmt frame transmission failed
INFO: Incremented station SRC: stationShortRetryCounter = 7.
WARN: Retry limit reached for (inet::Packet)UdpBasicAppData-125 (164 B) (inet::SequenceChunk) length = 164 B.
INFO: Dropping frame UdpBasicAppData-125, because retry limit is reached.
DETAIL (Aodv)Manet.mobile[2].routingApp: Received link break signal
DETAIL (Aodv)Manet.mobile[2].routingApp: Marking route to 1.1.0.0 as inactive
DETAIL (Aodv)Manet.mobile[2].routingApp: Marking route to 1.99.99.2 as inactive
INFO: (Dcf)Manet.mobile[2].wlan[0].mac.dcf.channelAccess: Channel released.
```

Figura 1.10: Log de detección de la caída de mobile[0]

```
** Event #174702 t=15.037068131364 Manet.mobile[2].ipv4.ip (Ipv4, id=383) on aodv::Rerr (inet::Packet, id=1129330)
INFO: Received (inet::Packet)aodv::Rerr (36 B) (inet::SequenceChunk) length = 36 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::SequenceChunk) length = 56 B to output interface = wlan0.
```

Figura 1.11: Log del envío del RERR a broadcast

```
** Event #174708 t=15.037068131364 Manet.mobile[2].radio (Ieee80211ScalarRadio, id=348) on aodv::Rerr (inet::Packet, id=1129333)
INFO: Transmission started: (inet::physicallayer::WirelessSignal)aodv::Rerr (58 us 99 B) (inet::Packet)aodv::Rerr (99 B) (inet::SequenceChunk) length = 99 B WHILE a:
INFO: Changing radio transmission state from IDLE to TRANSMITTING.
INFO: Changing radio transmission signal path from NONE to IEEE.
INFO: Event #174708 t=15.037068131364 Manet.mobile[2].radio (Ieee80211ScalarRadio, id=683) on aodv::Rerr (inet::physicallayer::WirelessSignal, id=1129352)
INFO: Reception started: attempting (inet::physicallayer::WirelessSignal)aodv::Rerr (58 us 99 B) (inet::Packet)aodv::Rerr (99 B) (inet::SequenceChunk) length = 99 B
INFO: Changing radio reception state from IDLE to RECEIVING.
** Event #174710 t=15.037068373969 Manet.mobile[6].wlan[0].radio (Ieee80211ScalarRadio, id=616) on aodv::Rerr (inet::physicallayer::WirelessSignal, id=1129350)
INFO: Reception started: attempting (inet::physicallayer::WirelessSignal)aodv::Rerr (58 us 99 B) (inet::Packet)aodv::Rerr (99 B) (inet::SequenceChunk) length = 99 B
INFO: Changing radio reception state from IDLE to RECEIVING.
INFO: Event #174711 t=15.037068373971 Manet.mobile[6].wlan[0].radio (Ieee80211ScalarRadio, id=951) on aodv::Rerr (inet::physicallayer::WirelessSignal, id=1129360)
INFO: Reception started: attempting (inet::physicallayer::WirelessSignal)aodv::Rerr (58 us 99 B) (inet::Packet)aodv::Rerr (99 B) (inet::SequenceChunk) length = 99 B
INFO: Changing radio reception state from IDLE to RECEIVING.
** Event #174712 t=15.037068673345 Manet.mobile[11].wlan[0].radio (Ieee80211ScalarRadio, id=1085) on aodv::Rerr (inet::physicallayer::WirelessSignal, id=1129364)
INFO: Reception started: attempting (inet::physicallayer::WirelessSignal)aodv::Rerr (58 us 99 B) (inet::Packet)aodv::Rerr (99 B) (inet::SequenceChunk) length = 99 B
INFO: Changing radio reception state from IDLE to RECEIVING.
** Event #174713 t=15.037068780358 Manet.mobile[3].wlan[0].radio (Ieee80211ScalarRadio, id=415) on aodv::Rerr (inet::physicallayer::WirelessSignal, id=1129344)
INFO: Reception started: attempting (inet::physicallayer::WirelessSignal)aodv::Rerr (58 us 99 B) (inet::Packet)aodv::Rerr (99 B) (inet::SequenceChunk) length = 99 B
INFO: Changing radio reception state from IDLE to RECEIVING.
** Event #174714 t=15.037068840713 Manet.mobile[5].wlan[0].radio (Ieee80211ScalarRadio, id=540) on aodv::Rerr (inet::physicallayer::WirelessSignal, id=1129348)
INFO: Reception started: attempting (inet::physicallayer::WirelessSignal)aodv::Rerr (58 us 99 B) (inet::Packet)aodv::Rerr (99 B) (inet::SequenceChunk) length = 99 B
INFO: Changing radio reception state from IDLE to RECEIVING.
** Event #174715 t=15.037068866232 Manet.mobile[0].wlan[0].radio (Ieee80211ScalarRadio, id=214) on aodv::Rerr (inet::physicallayer::WirelessSignal, id=1129340)
** Event #174716 t=15.037068899043 Manet.mobile[12].wlan[0].radio (Ieee80211ScalarRadio, id=1018) on aodv::Rerr (inet::physicallayer::WirelessSignal, id=1129362)
INFO: Reception started: attempting (inet::physicallayer::WirelessSignal)aodv::Rerr (58 us 99 B) (inet::Packet)aodv::Rerr (99 B) (inet::SequenceChunk) length = 99 B
INFO: Changing radio reception state from IDLE to RECEIVING.
INFO: Event #174717 t=15.037068927741 Manet.mobile[1].wlan[0].radio (Ieee80211ScalarRadio, id=281) on aodv::Rerr (inet::physicallayer::WirelessSignal, id=1129342)
INFO: Reception started: attempting (inet::physicallayer::WirelessSignal)aodv::Rerr (58 us 99 B) (inet::Packet)aodv::Rerr (99 B) (inet::SequenceChunk) length = 99 B
INFO: Changing radio reception state from IDLE to RECEIVING.
** Event #174718 t=15.03706914973 Manet.mobile[8].wlan[0].radio (Ieee80211ScalarRadio, id=750) on aodv::Rerr (inet::physicallayer::WirelessSignal, id=1129354)
INFO: Reception started: attempting (inet::physicallayer::WirelessSignal)aodv::Rerr (58 us 99 B) (inet::Packet)aodv::Rerr (99 B) (inet::SequenceChunk) length = 99 B
INFO: Reception started: attempting (inet::physicallayer::WirelessSignal)aodv::Rerr (58 us 99 B) (inet::Packet)aodv::Rerr (99 B) (inet::SequenceChunk) length = 99 B
** Event #174719 t=15.037069238071 Manet.mobile[1].wlan[0].radio (Ieee80211ScalarRadio, id=1152) on aodv::Rerr (inet::physicallayer::WirelessSignal, id=1129349)
INFO: Reception started: not attempting (inet::physicallayer::WirelessSignal)aodv::Rerr (58 us 99 B) (inet::Packet)aodv::Rerr (99 B) (inet::SequenceChunk) length = 99 B
** Event #174720 t=15.037069238074 Manet.mobile[4].wlan[0].radio (Ieee80211ScalarRadio, id=482) on aodv::Rerr (inet::physicallayer::WirelessSignal, id=1129346)
INFO: Reception started: not attempting (inet::physicallayer::WirelessSignal)aodv::Rerr (58 us 99 B) (inet::Packet)aodv::Rerr (99 B) (inet::SequenceChunk) length = 99 B
** Event #174721 t=15.037069288979 Manet.mobile[10].wlan[0].radio (Ieee80211ScalarRadio, id=884) on aodv::Rerr (inet::physicallayer::WirelessSignal, id=1129358)
INFO: Reception started: not attempting (inet::physicallayer::WirelessSignal)aodv::Rerr (58 us 99 B) (inet::Packet)aodv::Rerr (99 B) (inet::SequenceChunk) length = 99 B
```

Figura 1.12: Logs de envío del RERR enviado por mobile[9]

1.5 Ejercicio 1.5

1.5.1 Muestra el contenido del paquete RERR en Wireshark explicando los campos más importantes. ¿Qué IP tiene como destino? ¿Por qué?

La figura 1.13 muestra en detalle el paquete RERR enviado por mobile[2] tras detectar la no disponibilidad de mobile[0].

Como características notables, se puede indicar que el paquete se envía siguiendo el protocolo UDP, desde la dirección IP de mobile[2] (1.1.0.2) a toda la red (255.255.255.255) a través del puerto 654 con TTL=1, por lo que no es reenviado por sus vecinos. En cuanto a la cabecera AODV, se indica el tipo de paquete (Route Error) y las direcciones no alcanzables, que incluyen mobile[0] (Duplicado por problemas de INET) y, por consiguiente, static2.

Tiene como IP de destino la 255.255.255.255 debido a que, según el RFC 3561, es el comportamiento general y el que se toma cuando hay dos o más nodos vecinos en la lista de precursores de al menos uno de los destinos inalcanzables. Si hubiera un único vecino en la lista de precursores de mobile[0], podría enviarse como unicast.

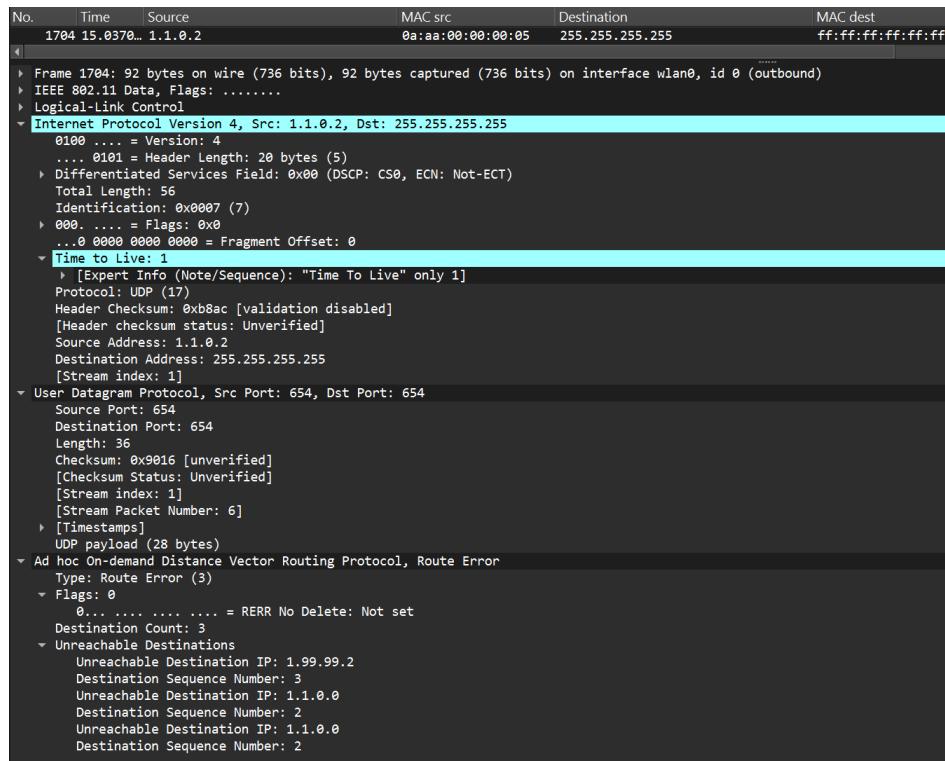


Figura 1.13: Detalle del paquete de error enviado por mobile[2] en t=15,037

1.6 Ejercicio 1.6

1.6.1 Explica cómo se propaga el RERR por la red. ¿Qué nodos lo reenvían? ¿Cómo sabe un nodo si debe reenviar el RERR?

El RERR se propaga, generalmente, mediante broadcast, debido a que los nodos afectados a los que se debe avisar son más que uno. El RERR es recibido por todos los nodos al alcance del emisor, que lo procesan y comprueban si las *Unreachable destinations* del RERR coinciden con alguna entrada contenida en sus tablas de enrutamiento. Si es el caso, los nodos afectados reenvían el RERR.

Por ejemplo, el primer RERR (Enviado por mobile[2]) es reenviado por los nodos que contienen la ruta hacia static2. Como vemos en la figura 1.14, después de salir de mobile[2], es reenviado por mobile[1] y mobile[10] hasta alcanzar static1.

Event#	Time	Relevant Hops	Name	TxUpdate? / Sour	Length / Destina
#174708	15.037'068'131'364	mobile[2] → static1	aodv::Rerr	1.1.0.2:654	255.255.255.255:654
#174708	15.037'068'131'364	mobile[2] → static2	aodv::Rerr	1.1.0.2:654	255.255.255.255:654
#174708	15.037'068'131'364	mobile[2] → mobile[0]	aodv::Rerr	1.1.0.2:654	255.255.255.255:654
#174708	15.037'068'131'364	mobile[2] → mobile[1]	aodv::Rerr	1.1.0.2:654	255.255.255.255:654
#174708	15.037'068'131'364	mobile[2] → mobile[3]	aodv::Rerr	1.1.0.2:654	255.255.255.255:654
#174708	15.037'068'131'364	mobile[2] → mobile[4]	aodv::Rerr	1.1.0.2:654	255.255.255.255:654
#174708	15.037'068'131'364	mobile[2] → mobile[5]	aodv::Rerr	1.1.0.2:654	255.255.255.255:654
#174708	15.037'068'131'364	mobile[2] → mobile[6]	aodv::Rerr	1.1.0.2:654	255.255.255.255:654
#174708	15.037'068'131'364	mobile[2] → mobile[7]	aodv::Rerr	1.1.0.2:654	255.255.255.255:654
#174708	15.037'068'131'364	mobile[2] → mobile[8]	aodv::Rerr	1.1.0.2:654	255.255.255.255:654
#174708	15.037'068'131'364	mobile[2] → mobile[9]	aodv::Rerr	1.1.0.2:654	255.255.255.255:654
#174708	15.037'068'131'364	mobile[2] → mobile[10]	aodv::Rerr	1.1.0.2:654	255.255.255.255:654
#174708	15.037'068'131'364	mobile[2] → mobile[11]	aodv::Rerr	1.1.0.2:654	255.255.255.255:654
#174708	15.037'068'131'364	mobile[2] → mobile[12]	aodv::Rerr	1.1.0.2:654	255.255.255.255:654
#174708	15.037'068'131'364	mobile[2] → mobile[13]	aodv::Rerr	1.1.0.2:654	255.255.255.255:654
#174708	15.037'068'131'364	mobile[2] → mobile[14]	aodv::Rerr	1.1.0.2:654	255.255.255.255:654
#174789	15.037'476'927'355	mobile[1] → static1	aodv::Rerr	1.1.0.1:654	255.255.255.255:654
#174789	15.037'476'927'355	mobile[1] → static2	aodv::Rerr	1.1.0.1:654	255.255.255.255:654
#174789	15.037'476'927'355	mobile[1] → mobile[0]	aodv::Rerr	1.1.0.1:654	255.255.255.255:654
#174789	15.037'476'927'355	mobile[1] → mobile[2]	aodv::Rerr	1.1.0.1:654	255.255.255.255:654
#174789	15.037'476'927'355	mobile[1] → mobile[3]	aodv::Rerr	1.1.0.1:654	255.255.255.255:654
#174789	15.037'476'927'355	mobile[1] → mobile[4]	aodv::Rerr	1.1.0.1:654	255.255.255.255:654
#174789	15.037'476'927'355	mobile[1] → mobile[5]	aodv::Rerr	1.1.0.1:654	255.255.255.255:654
#174789	15.037'476'927'355	mobile[1] → mobile[6]	aodv::Rerr	1.1.0.1:654	255.255.255.255:654
#174789	15.037'476'927'355	mobile[1] → mobile[7]	aodv::Rerr	1.1.0.1:654	255.255.255.255:654
#174789	15.037'476'927'355	mobile[1] → mobile[8]	aodv::Rerr	1.1.0.1:654	255.255.255.255:654
#174789	15.037'476'927'355	mobile[1] → mobile[9]	aodv::Rerr	1.1.0.1:654	255.255.255.255:654
#174789	15.037'476'927'355	mobile[1] → mobile[10]	aodv::Rerr	1.1.0.1:654	255.255.255.255:654
#174789	15.037'476'927'355	mobile[1] → mobile[11]	aodv::Rerr	1.1.0.1:654	255.255.255.255:654
#174789	15.037'476'927'355	mobile[1] → mobile[12]	aodv::Rerr	1.1.0.1:654	255.255.255.255:654
#174789	15.037'476'927'355	mobile[1] → mobile[13]	aodv::Rerr	1.1.0.1:654	255.255.255.255:654
#174789	15.037'476'927'355	mobile[1] → mobile[14]	aodv::Rerr	1.1.0.1:654	255.255.255.255:654
#174845	15.037'621'482'775	mobile[10] → static1	aodv::Rerr	1.1.0.10:654	255.255.255.255:654
#174845	15.037'621'482'775	mobile[10] → static2	aodv::Rerr	1.1.0.10:654	255.255.255.255:654
#174845	15.037'621'482'775	mobile[10] → mobile[0]	aodv::Rerr	1.1.0.10:654	255.255.255.255:654
#174845	15.037'621'482'775	mobile[10] → mobile[1]	aodv::Rerr	1.1.0.10:654	255.255.255.255:654
#174845	15.037'621'482'775	mobile[10] → mobile[10]	aodv::Rerr	1.1.0.10:654	255.255.255.255:654
#174845	15.037'621'482'775	mobile[10] → mobile[11]	aodv::Rerr	1.1.0.10:654	255.255.255.255:654
#174845	15.037'621'482'775	mobile[10] → mobile[12]	aodv::Rerr	1.1.0.10:654	255.255.255.255:654
#174845	15.037'621'482'775	mobile[10] → mobile[13]	aodv::Rerr	1.1.0.10:654	255.255.255.255:654
#174845	15.037'621'482'775	mobile[10] → mobile[14]	aodv::Rerr	1.1.0.10:654	255.255.255.255:654
#174910	15.041'471'330'806	static1 → static2	aodv::Rreq	1.99.99.1:654	255.255.255.255:654
#174910	15.041'471'330'806	static1 → mobile[0]	aodv::Rreq	1.99.99.1:654	255.255.255.255:654

Figura 1.14: Logs de envío del RERR producido por mobile[2]

1.7 Ejercicio 1.7

1.7.1 Muestra capturas de la tabla de enrutamiento de un nodo antes y después de recibir un RERR y explica en qué cambia.

Los nodos que pertenecen a la lista de predecesores del nodo caído y a la ruta entre static1 y static2 y, por tanto, posibles de usar para este ejercicio (Se apreciarán cambios en su tabla de enrutamiento) son mobile[2], mobile[1] y mobile[10] (Y poor supuesto static1). Para esta demostración usamos mobile[1] como ejemplo.

En el caso de mobile[1], las diferencias entre antes^{1.15} y después^{1.16} de la recepción del RERR radica en la entrada [5], que es la ruta a static2. Vemos que la ruta se marca como inactiva (isActive=0) y, con la llegada del RERR, se actualiza en uno el número de secuencia de destino en esa entrada (Ya lo había incrementado mobile[2]) y se actualiza también el lifetime al valor incluido en el paquete (Generado en mobile[2] como instante actual+ART).

```
Manet.mobile[1].ipv4.routingTable.routes (vector<Ipv4Route *>) size=9
elements[9] (inet::Ipv4Route *)
[0] ??? 1.1.0.2/32 gw:1.1.0.2 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 2, lifetime = 18.000459036489, precursor list: 1.1.0.10
[1] ??? 1.1.0.3/32 gw:1.1.0.3 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 6, lifetime = 17.330309359265
[2] ??? 1.1.0.4/32 gw:1.1.0.3 metric:4 if:wlan0 isActive = 1, hasValidDestNum = 1, destNum = 2, lifetime = 19.573104020434
[3] ??? 1.1.0.10/32 gw:1.1.0.10 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 6, lifetime = 18.000459036489
[4] ??? 1.99.99.1/32 gw:1.1.0.10 metric:2 if:wlan0 isActive = 1, hasValidDestNum = 1, destNum = 6, lifetime = 19.766484881925
[5] ??? 1.99.99.2/32 gw:1.1.0.2 metric:4 if:wlan0 isActive = 1, hasValidDestNum = 1, destNum = 2, lifetime = 19.735638712179, precursor list: 1.1.0.10
[6] S 1.0.0.0/9 gw:* metric:0 if:wlan0
[7] C 1.0.0.0/8 gw:* metric:-2000000000 if:wlan0
[8] C 127.0.0.0/8 gw:* metric:1 if:lo0
```

Figura 1.15: Tabla de enrutamiento de mobile[1] antes de recibir el RERR

```
Manet.mobile[1].ipv4.routingTable.routes (vector<Ipv4Route *>) size=9
elements[9] (inet::Ipv4Route *)
[0] ??? 1.1.0.2/32 gw:1.1.0.2 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 2, lifetime = 18.000459036489, precursor list: 1.1.0.10
[1] ??? 1.1.0.3/32 gw:1.1.0.3 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 6, lifetime = 17.330309359265
[2] ??? 1.1.0.4/32 gw:1.1.0.3 metric:4 if:wlan0 isActive = 1, hasValidDestNum = 1, destNum = 2, lifetime = 19.573104020434
[3] ??? 1.1.0.10/32 gw:1.1.0.10 metric:1 if:wlan0 isActive = 1, hasValidDestNum = 0, destNum = 6, lifetime = 18.000459036489
[4] ??? 1.99.99.1/32 gw:1.1.0.10 metric:2 if:wlan0 isActive = 1, hasValidDestNum = 1, destNum = 6, lifetime = 19.766484881925
[5] ??? 1.99.99.2/32 gw:1.1.0.2 metric:4 if:wlan0 isActive = 0, hasValidDestNum = 1, destNum = 3, lifetime = 30.037126927355, precursor list: 1.1.0.10
[6] S 1.0.0.0/9 gw:* metric:0 if:wlan0
[7] C 1.0.0.0/8 gw:* metric:-2000000000 if:wlan0
[8] C 127.0.0.0/8 gw:* metric:1 if:lo0
```

Figura 1.16: Tabla de enrutamiento de mobile[1] después de recibir el RERR

1.8 Ejercicio 1.8

1.8.1 ¿Qué hace static1 al recibir el RERR? Muestra el contenido del siguiente RREQ en Wireshark. ¿En qué cambia con respecto al de la pregunta 1?

Al recibir el último RERR, enviado desde mobile[10], static1 inicia un nuevo descubrimiento de ruta para intentar alcanzar static2 utilizando un número de secuencia de destino igual o mayor al recibido en el RERR, que ya había sido incrementado.

Como se observa al comparar la figura 1.17 con la figura 1.18, el contenido de los paquetes es prácticamente idéntico, diferenciándose principalmente en que, en el primer RREQ se incluye una flag indicando que aún no se conoce un primer número de secuencia de destino, mientras que el RREQ tras el RERR ya viene con un número establecido.

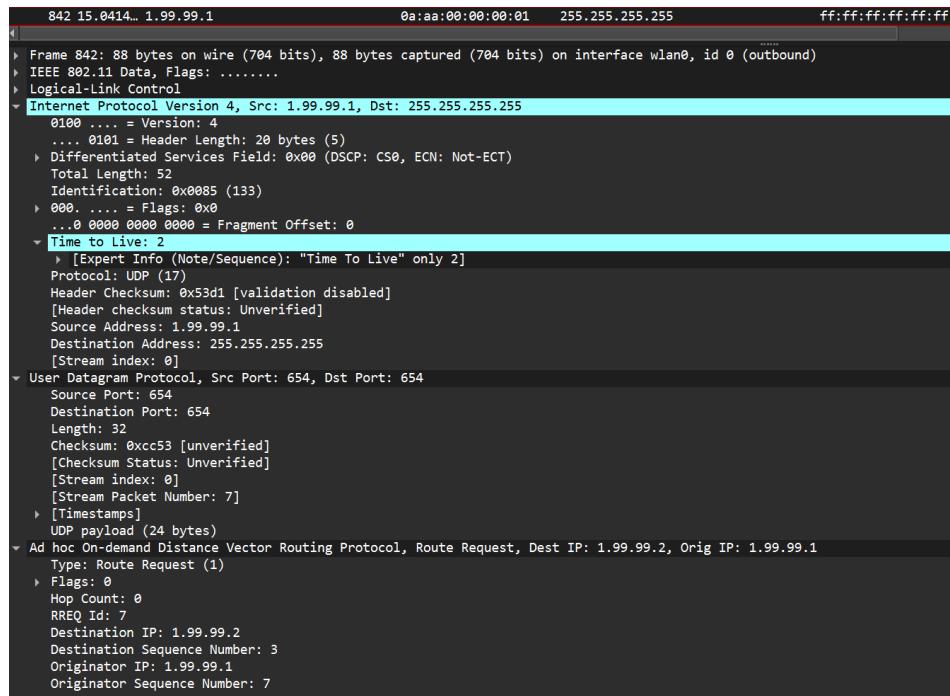


Figura 1.17: RREQ enviado por static1 tras a la recepción del RREP

```

1 10.0029... 1.99.99.1          0a:aa:00:00:00:01  255.255.255.255          ff:ff:ff:ff:ff:ff
|> Frame 1: 88 bytes on wire (704 bits), 88 bytes captured (704 bits) on interface wlan0, id 0 (outbound)
|> IEEE 802.11 Data, Flags: .....
|> Logical-Link Control
|> Internet Protocol Version 4, Src: 1.99.99.1, Dst: 255.255.255.255
|  0100 .... = Version: 4
|  .... 0101 = Header Length: 20 bytes (5)
|> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
|  Total Length: 52
|  Identification: 0x0001 (1)
|  000. .... = Flags: 0x0
|  ... 0 0000 0000 0000 = Fragment Offset: 0
|> Time to Live: 2
|  > [Expert Info (Note/Sequence): "Time To Live" only 2]
|  Protocol: UDP (17)
|  Header Checksum: 0x5455 [validation disabled]
|  [Header checksum status: Unverified]
|  Source Address: 1.99.99.1
|  Destination Address: 255.255.255.255
|  [Stream index: 0]
|> User Datagram Protocol, Src Port: 654, Dst Port: 654
|  Source Port: 654
|  Destination Port: 654
|  Length: 32
|  Checksum: 0xcc5a [unverified]
|  [Checksum Status: Unverified]
|  [Stream index: 0]
|  [Stream Packet Number: 1]
|> [Timestamps]
|  UDP payload (24 bytes)
|> Ad hoc On-demand Distance Vector Routing Protocol, Route Request, Dest IP: 1.99.99.2, Orig IP: 1.99.99.1
|  Type: Route Request (1)
|> Flags: 2048, RREQ Unknown Sequence Number
|  0... .... .... .... = RREQ Join: Not set
|  .0.. .... .... .... = RREQ Repair: Not set
|  ..0. .... .... .... = RREQ Gratuitous RREP: Not set
|  ...0. .... .... .... = RREQ Destination only: Not set
|  .... 1... .... .... = RREQ Unknown Sequence Number: Set
|  Hop Count: 0
|  RREQ Id: 1
|  Destination IP: 1.99.99.2
|  Destination Sequence Number: 0
|  Originator IP: 1.99.99.1
|  Originator Sequence Number: 1

```

Figura 1.18: Primer RREQ enviado por static1

Capítulo 2

DSDV

2.1 Ejercicio 2.1

- 2.1.1 Avanza la simulación hasta el instante $t = 7$ s. Busca el primer paquete Hello transmitido a partir a ese instante con un valor de hopdistance de al menos 3 y muestra una captura del contenido. Explica el significado de los campos srcAddress y nextAddress, utilizando para explicarlos una captura de la tabla de enrutamiento del nodo que está transmitiendo el paquete (i.e., no el que consta en srcAddress)

```
** Event #46284 t=7.0078877216 Manet.radioMedium (Ieee80211ScalarRadioMedium, id=3) on selfmsg removeNonInterferingTransmissions (omnetpp::cMessage, id=307768)
** Event #46285 t=7.0136600356 Manet.mobile[8].ipv4.ip (Ipv4, id=766) on Hello (inet::Packet, id=353225)
INFO: Received (inet::Packet)Hello (16 B) (inet::DsdvHello) srcAddress = 1.1.0.7, sequencenumber = 6, nextAddress = 1.1.0.8, hopdistance = 3 from upper layer.
DETAIL: Sending datagram 'Hello' with destination = 255.255.255.255
DETAIL: destination address is broadcast, sending packet to broadcast MAC address
INFO: Sending (inet::Packet)Hello (36 B) (inet::SequenceChunk) length = 36 B to output interface = wlan0.
INFO: (MessageDispatcher)Manet.mobile[8].ipv4.l0: Dispatching packet to interface, interfaceId = 101, inGate = (omnetpp::cGate)in[1] --> ip.queueOut, outGate = (omnetpp::cGate)out[0]
INFO: (MessageDispatcher)Manet.mobile[8].nl: Dispatching packet to interface, interfaceId = 101, inGate = (omnetpp::cGate)in[0] --> ipv4.ifOut, outGate = (omnetpp::cGate)out[1]
INFO: (MessageDispatcher)Manet.mobile[8].ch: Dispatching packet to interface, interfaceId = 101, inGate = (omnetpp::cGate)in[1] --> nl.out[1], outGate = (omnetpp::cGate)out[0]
INFO: (MessageDispatcher)Manet.mobile[8].cb: Dispatching packet to interface, interfaceId = 101, inGate = (omnetpp::cGate)in[0] --> cb.out[0], outGate = (omnetpp::cGate)out[1]
INFO: (MessageDispatcher)Manet.mobile[8].bl: Dispatching packet to interface, interfaceId = 101, inGate = (omnetpp::cGate)in[0] --> bl.out[1], outGate = (omnetpp::cGate)out[0]
** Event #46286 t=7.0136600356 Manet.mobile[8].llc (Ieee80211llcLpd, id=727) on Hello (inet::Packet, id=353225)
INFO: Received (inet::Packet)Hello (36 B) (inet::SequenceChunk) length = 36 B from upper layer.
** Event #46287 t=7.0136600356 Manet.mobile[8].wlan[0].mac (Ieee80211Mac, id=730) on Hello (inet::Packet, id=353225)
INFO: Frame (inet::Packet)Hello (72 B) (inet::SequenceChunk) length = 72 B received from higher layer, receiver = FF-FF-FF-FF-FF-FF
INFO: (Dcf)Manet.mobile[8].wlan[0].mac.dcf: Processing upper frame: Hello
INFO: (PendingQueue)Manet.mobile[8].wlan[0].mac.dcf.channelAccess.pendingQueue: Pushing packet, packet = (Packet)Hello (72 B) [Ieee80211DataHeader, address4 = 00-00-00-00-00-00]
DETAIL: (Dcf)Manet.mobile[8].wlan[0].mac.dcf: Requesting channel
INFO: (Contention)Manet.mobile[8].wlan[0].mac.dcf.channelAccess.contention: Starting contention: cw = 31, slots = 5, slotTime = 0.00002, ifts = 0.00005, eifs = 0.00005
DETAIL: (Contention)Manet.mobile[8].wlan[0].mac.dcf.channelAccess.contention: Scheduling contention end: backoffSlots = 5, slotTime = 0.00002, lastBusyTime = 6.9978862;
```

Figura 2.1: Log del nodo que manda el primer Hello con hopdistance 3

Como se puede ver la imagen, el nodo que manda el primer mensaje Hello con hopdistance 3 es el nodo 8. Vamos a fijarnos en su tabla de enrutamiento:

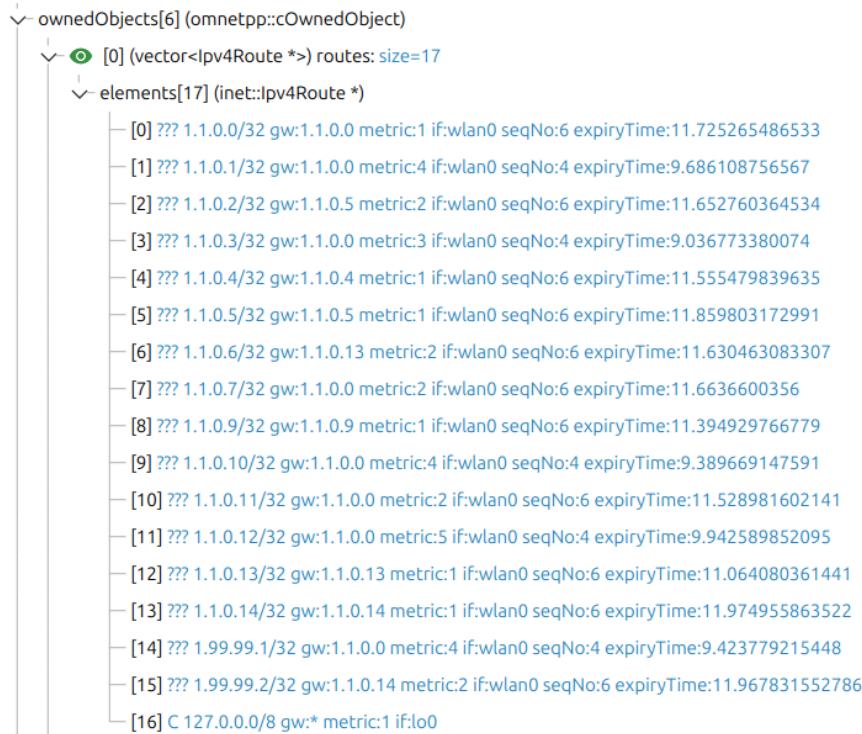


Figura 2.2: Tabla de enrutamiento nodo 8

Los mensajes Hello contienen las entradas de la tabla de enrutamiento de ese nodo. En Inet no se manda toda las entradas de la tabla en un mensaje Hello, si no que manda una o algunas veces varias entradas. Según la imagen 2.1, el campo srcAddress es el nodo que generó el paquete Hello, en este caso es el 1.1.0.7. El campo nextAddress es la siguiente dirección que va a retransmitir el Hello, en este caso 1.1.0.8. Como 1.1.0.8 (nodo 8) tiene en su tabla de enrutamiento una métrica de 2 para 1.1.0.7 (nodo 7) (imagen 2.2), va a sumar 1 para transmitir el Hello a los vecinos indicando que el nodo destino está a 3 saltos del nodo 7.

2.2 Ejercicio 2.2

2.2.1 ¿Qué valor tiene de sequencenumber? ¿Qué quiere decir ese valor?

Como se puede ver en la imagen 2.1, el campo sequencenumber tiene valor 6. Este valor ayuda a identificar entradas obsoletas o inválidas (un nodo se ha desplazado y ya no es alcanzable) con el fin de evitar bucles. Para ver si una ruta es válida o inválida llega con ver si este valor es par (ruta disponible) o impar (ruta no disponible/caída). De esta forma, nos indica como de actualizado está la información de la ruta, el estado de la ruta y evitar posibles conflictos en la red.

2.3 Ejercicio 2.3

2.3.1 ¿Cómo se modifica? ¿Qué nodo lo modifica, y cuándo lo hace?

Este valor lo modifica el propio nodo en su tabla de enrutamiento, incrementándolo en 2 cuando la ruta que es válida, pero si un nodo identifica un nodo caído (o que se desplazó), el nodo que identifica la caída puede cambiar en la entrada de la tabla, el valor del sequencenumber de otro nodo indicando que es inválido. Esto se indica incrementando el valor 1 número (las rutas válidas tienen valor par, si incrementamos a 1, pasa a ser impar y por lo tanto indicando que ya no es un nodo válido).

Cuando un nodo descubre una ruta mejor hacia el destino, actualiza este valor. Esto ocurre si el nuevo número de secuencia recibido es mayor (y par), con respecto a lo que tiene el guardado en su tabla de enrutamiento, por lo registra en su tabla y actualiza el nodo que recibió la información (en este caso mobile[8]). En el caso de una ruta no válida, genera un sequencenumber impar y establece el costo de la ruta como infinito para indicar que esa ruta no se puede usar.

El caso de una ruta inválida, es el otro nodo el que lo actualiza. Esto ocurre básicamente porque en el caso de que el nodo que identifica otro nodo inválido se moviera y un nodo en su tabla de enrutamiento tuviera registrado el nodo inválido con un sequencenumber impar, este al ver que si que es alcanzable, como el nodo inválido tiene un sequencenumber mayor, el otro nodo puede actualizarlo ya como valido y no pasaría nada.

2.4 Ejercicio 2.4

2.4.1 Muestra la tabla de enrutamiento del nodo que recibe el Hello de la pregunta anterior justo antes y justo después de recibirlo, relacionándola con el contenido del paquete. Si se actualiza la tabla, explica por qué se actualiza y las entradas que se crean. Si no se actualiza, explica por qué no se actualiza y di qué entrada se crearía (destino, gateway, métrica) si se actualizase con la información del paquete.

```

    ownedObjects[6] (omnetpp::cOwnedObject)
      routes: size=17
      elements[17] (inet::Ipv4Route *)
        [0] ??? 1.1.0.0/32 gw:1.1.0.0 metric:1 if:wlan0 seqNo:6 expiryTime:11.725265439917
        [1] ??? 1.1.0.1/32 gw:1.1.0.6 metric:4 if:wlan0 seqNo:4 expiryTime:9.636109101984
        [2] ??? 1.1.0.2/32 gw:1.1.0.2 metric:1 if:wlan0 seqNo:6 expiryTime:11.612706143526
        [3] ??? 1.1.0.3/32 gw:1.1.0.0 metric:3 if:wlan0 seqNo:4 expiryTime:9.03677337288
        [4] ??? 1.1.0.4/32 gw:1.1.0.4 metric:1 if:wlan0 seqNo:6 expiryTime:11.555479905357
        [5] ??? 1.1.0.6/32 gw:1.1.0.6 metric:1 if:wlan0 seqNo:6 expiryTime:11.440408774046
        [6] ??? 1.1.0.7/32 gw:1.1.0.0 metric:2 if:wlan0 seqNo:6 expiryTime:11.663659989892
        [7] ??? 1.1.0.8/32 gw:1.1.0.8 metric:1 if:wlan0 seqNo:6 expiryTime:11.656227369884
        [8] ??? 1.1.0.9/32 gw:1.1.0.9 metric:1 if:wlan0 seqNo:6 expiryTime:11.394929821113
        [9] ??? 1.1.0.10/32 gw:1.1.0.2 metric:4 if:wlan0 seqNo:4 expiryTime:9.239669155454
        [10] ??? 1.1.0.11/32 gw:1.1.0.6 metric:2 if:wlan0 seqNo:6 expiryTime:11.378981594991
        [11] ??? 1.1.0.12/32 gw:1.1.0.2 metric:5 if:wlan0 seqNo:4 expiryTime:9.642589853498
        [12] ??? 1.1.0.13/32 gw:1.1.0.13 metric:1 if:wlan0 seqNo:6 expiryTime:11.064080150948
        [13] ??? 1.1.0.14/32 gw:1.1.0.14 metric:1 if:wlan0 seqNo:6 expiryTime:11.974955704862
        [14] ??? 1.99.99.1/32 gw:1.1.0.0 metric:4 if:wlan0 seqNo:4 expiryTime:9.423779202605
        [15] ??? 1.99.99.2/32 gw:1.1.0.14 metric:2 if:wlan0 seqNo:6 expiryTime:11.967831393972
        [16] C 127.0.0.0/8 gw:* metric:1 if:lo0
  
```

Figura 2.3: Tabla de enrutamiento nodo 5 antes recibir Hello de nodo 8

```

    [0] (vector<Ipv4Route *>) routes: size=17
      elements[17] (inet::Ipv4Route *)
        [0] ??? 1.1.0.0/32 gw:1.1.0.8 metric:2 if:wlan0 seqNo:6 expiryTime:11.805319946345
        [1] ??? 1.1.0.1/32 gw:1.1.0.14 metric:5 if:wlan0 seqNo:4 expiryTime:9.846163654168
        [2] ??? 1.1.0.2/32 gw:1.1.0.5 metric:2 if:wlan0 seqNo:6 expiryTime:11.652760664461
        [3] ??? 1.1.0.3/32 gw:1.1.0.8 metric:4 if:wlan0 seqNo:4 expiryTime:9.356827881887
        [4] ??? 1.1.0.4/32 gw:1.1.0.4 metric:1 if:wlan0 seqNo:6 expiryTime:11.555479669497
        [5] ??? 1.1.0.5/32 gw:1.1.0.5 metric:1 if:wlan0 seqNo:6 expiryTime:11.859803475901
        [6] ??? 1.1.0.6/32 gw:1.1.0.14 metric:2 if:wlan0 seqNo:6 expiryTime:11.880463338296
        [7] ??? 1.1.0.7/32 gw:1.1.0.5 metric:3 if:wlan0 seqNo:6 expiryTime:11.833714510839
        [8] ??? 1.1.0.8/32 gw:1.1.0.8 metric:1 if:wlan0 seqNo:6 expiryTime:11.656227611185
        [9] ??? 1.1.0.10/32 gw:1.1.0.8 metric:5 if:wlan0 seqNo:4 expiryTime:9.479723647211
        [10] ??? 1.1.0.11/32 gw:1.1.0.5 metric:3 if:wlan0 seqNo:4 expiryTime:9.031369608311
        [11] ??? 1.1.0.12/32 gw:1.1.0.5 metric:6 if:wlan0 seqNo:4 expiryTime:9.862644374339
        [12] ??? 1.1.0.13/32 gw:1.1.0.8 metric:2 if:wlan0 seqNo:6 expiryTime:11.114134832725
        [13] ??? 1.1.0.14/32 gw:1.1.0.14 metric:1 if:wlan0 seqNo:6 expiryTime:11.974955876923
        [14] ??? 1.99.99.1/32 gw:1.1.0.14 metric:5 if:wlan0 seqNo:4 expiryTime:9.574342829771
        [15] ??? 1.99.99.2/32 gw:1.99.99.2 metric:1 if:wlan0 seqNo:6 expiryTime:11.607777005146
        [16] C 127.0.0.0/8 gw:* metric:1 if:lo0

```

Figura 2.4: Tabla de enrutamiento nodo 9 antes recibir Hello de nodo 8

En las anteriores imágenes (2.3 y 2.4) podemos ver las tablas de enrutamiento antes de recibir el Hello, de los nodos 5 y 9, que son nodos que está dentro del rango del nodo 8. Ambas tienen para la entrada al nodo 7 un numero de secuencia 6. En el nodo 5 tiene una métrica de 2 y el gateway es el nodo 0, mientras que el nodo 9 tiene una métrica de 3 y el gateway es el nodo 5.

En las siguientes imágenes vamos a ver las tablas de enrutamiento de los dos nodos mencionados anteriormente, pero después de haber recibido el mensaje Hello:

```

    ownedObjects[6] (omnetpp::cOwnedObject)
      [0] (vector<Ipv4Route *>) routes: size=17
        elements[17] (inet::Ipv4Route *)
          [0] ??? 1.1.0.0/32 gw:1.1.0.0 metric:1 if:wlan0 seqNo:6 expiryTime:11.725265439917
          [1] ??? 1.1.0.1/32 gw:1.1.0.6 metric:4 if:wlan0 seqNo:4 expiryTime:9.636109101984
          [2] ??? 1.1.0.2/32 gw:1.1.0.2 metric:1 if:wlan0 seqNo:6 expiryTime:11.612706143526
          [3] ??? 1.1.0.3/32 gw:1.1.0.0 metric:3 if:wlan0 seqNo:4 expiryTime:9.03677337288
          [4] ??? 1.1.0.4/32 gw:1.1.0.4 metric:1 if:wlan0 seqNo:6 expiryTime:11.555479905357
          [5] ??? 1.1.0.6/32 gw:1.1.0.6 metric:1 if:wlan0 seqNo:6 expiryTime:11.440408774046
          [6] ??? 1.1.0.7/32 gw:1.1.0.0 metric:2 if:wlan0 seqNo:6 expiryTime:11.663659989892
          [7] ??? 1.1.0.8/32 gw:1.1.0.8 metric:1 if:wlan0 seqNo:6 expiryTime:11.656227369884
          [8] ??? 1.1.0.9/32 gw:1.1.0.9 metric:1 if:wlan0 seqNo:6 expiryTime:11.394929821113
          [9] ??? 1.1.0.10/32 gw:1.1.0.2 metric:4 if:wlan0 seqNo:4 expiryTime:9.239669155454
          [10] ??? 1.1.0.11/32 gw:1.1.0.6 metric:2 if:wlan0 seqNo:6 expiryTime:11.378981594991
          [11] ??? 1.1.0.12/32 gw:1.1.0.2 metric:5 if:wlan0 seqNo:4 expiryTime:9.642589853498
          [12] ??? 1.1.0.13/32 gw:1.1.0.13 metric:1 if:wlan0 seqNo:6 expiryTime:11.064080150948
          [13] ??? 1.1.0.14/32 gw:1.1.0.14 metric:1 if:wlan0 seqNo:6 expiryTime:11.974955704862
          [14] ??? 1.99.99.1/32 gw:1.1.0.0 metric:4 if:wlan0 seqNo:4 expiryTime:9.423779202605
          [15] ??? 1.99.99.2/32 gw:1.1.0.14 metric:2 if:wlan0 seqNo:6 expiryTime:11.967831393972
          [16] C 127.0.0.0/8 gw:* metric:1 if:lo0

```

Figura 2.5: Tabla de enrutamiento nodo 5 antes recibir Hello de nodo 8

```

    [0] (vector<Ipv4Route *>) routes: size=17
    elements[17] (inet::Ipv4Route *)
        [0] ??? 1.1.0.0/32 gw:1.1.0.8 metric:2 if:wlan0 seqNo:6 expiryTime:11.805319946345
        [1] ??? 1.1.0.1/32 gw:1.1.0.14 metric:5 if:wlan0 seqNo:4 expiryTime:9.846163654168
        [2] ??? 1.1.0.2/32 gw:1.1.0.5 metric:2 if:wlan0 seqNo:6 expiryTime:11.652760664461
        [3] ??? 1.1.0.3/32 gw:1.1.0.8 metric:4 if:wlan0 seqNo:4 expiryTime:9.356827881887
        [4] ??? 1.1.0.4/32 gw:1.1.0.4 metric:1 if:wlan0 seqNo:6 expiryTime:11.555479669497
        [5] ??? 1.1.0.5/32 gw:1.1.0.5 metric:1 if:wlan0 seqNo:6 expiryTime:11.859803475901
        [6] ??? 1.1.0.6/32 gw:1.1.0.14 metric:2 if:wlan0 seqNo:6 expiryTime:11.880463338296
        [7] ??? 1.1.0.7/32 gw:1.1.0.5 metric:3 if:wlan0 seqNo:6 expiryTime:11.833714510839
        [8] ??? 1.1.0.8/32 gw:1.1.0.8 metric:1 if:wlan0 seqNo:6 expiryTime:11.656227611185
        [9] ??? 1.1.0.10/32 gw:1.1.0.8 metric:5 if:wlan0 seqNo:4 expiryTime:9.479723647211
        [10] ??? 1.1.0.11/32 gw:1.1.0.5 metric:3 if:wlan0 seqNo:4 expiryTime:9.031369608311
        [11] ??? 1.1.0.12/32 gw:1.1.0.5 metric:6 if:wlan0 seqNo:4 expiryTime:9.862644374339
        [12] ??? 1.1.0.13/32 gw:1.1.0.8 metric:2 if:wlan0 seqNo:6 expiryTime:11.114134832725
        [13] ??? 1.1.0.14/32 gw:1.1.0.14 metric:1 if:wlan0 seqNo:6 expiryTime:11.974955876923
        [14] ??? 1.99.99.1/32 gw:1.1.0.14 metric:5 if:wlan0 seqNo:4 expiryTime:9.574342829771
        [15] ??? 1.99.99.2/32 gw:1.99.99.2 metric:1 if:wlan0 seqNo:6 expiryTime:11.607777005146
        [16] C 127.0.0.0/8 gw:* metric:1 if:lo0

```

Figura 2.6: Tabla de enrutamiento nodo 9 antes recibir Hello de nodo 8

Como podemos ver, ninguna de las tablas de enrutamiento han cambiado. Esto pasó ya que el mensaje Hello que ha mandado el nodo 8, manda una entrada de su tabla de enrutamiento con un sequencenumber igual al que tienen los otros nodos en su tabla de enrutamiento, por lo que no se actualiza nada.

Como se ve en la figura 2.2, la entrada que tiene el nodo 8 para el nodo 7 es el mismo gateway que tiene el nodo 5 y 9. En el caso de la métrica, como el nodo 5 tiene hopdistance mayor o igual al que guarda (marca hopdistance 3 y en el nodo 5 y 9 tiene una métrica de 2 y 3 respectivamente), tampoco lo actualizan.

En el caso de que recibieran un sequencenumber mayor y par, no se crearía ninguna entrada, solamente se actualizaría cambiando la métrica. Para el caso del hopdistance, si los nodos 5 y 9 guardasen un número mayor que el que recibieran, significaría que el nodo 8 estaría más cerca por lo que se actualizaría el valor. El gateway en el caso de que se actualizase la entrada de alguno de los nodos, el gateway pasaría a ser el del nodo 8.

En el caso de que recibiera un sequencenumber mayor y impar, tampoco se crearía ninguna nueva entrada, solamente se actualizaría ese valor para saber que el nodo es inalcanzable, aunque en el caso de que otro nodo ajeno le mandara un Hello con la entrada de ese nodo válido, lo volvería a actualizar.

2.5 Ejercicio 2.5

2.5.1 Avanza hasta la caída del nodo en t = 15 s. Ten en cuenta que la ruta en ese momento puede ser diferente a la de AODV, y por lo tanto el nodo a desactivar también. ¿Cuál es el primer nodo en darse cuenta de la caída? ¿Notifica la caída del nodo de alguna forma?

Hay que aclarar que hemos cambiado de semilla ya que con DSDV no funcionaba bien con la semilla que usamos en los anteriores apartados. La semilla a utilizar en este apartado y para el siguiente es 1865.

Ruta antes de la caída:

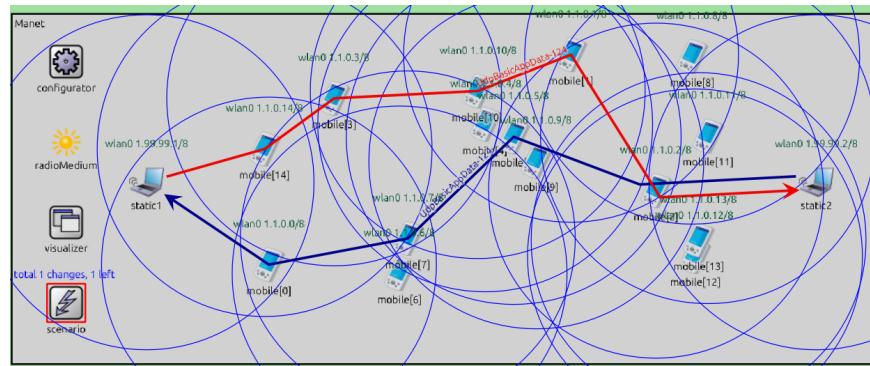


Figura 2.7: Ruta antes de la caída

Como nodo 2 es el nodo más próximo a static2 que establece la ruta, vamos a provocar su caída. Cuando el nodo está caído, vemos que el nodo 1, que es el nodo que está en la ruta y alcanza a nodo 2 manda varios UDP datos hasta que llega al límite (solo envía 6 veces, a la séptima para). Esto se puede ver en la siguiente imagen:

```

INFO: Frame sequence aborted.
INFO: Data/Mgmt frame transmission failed
INFO: Incremented station SRC: stationShortRetryCounter = 7.
WARN: Retry limit reached for (inet::Packet)UdpBasicAppData-125 (164 B) (inet
INFO: Dropping frame UdpBasicAppData-125, because retry limit is reached.
INFO (Dcaf)Manet.mobile[1].wlan[0].mac.dcf.channelAccess: Channel released.

```

Figura 2.8: Mensaje del nodo 1 cuando no es capaz de enviar los UDP datos al detectar un nodo caído

Esto quiere decir que el nodo 1 es el primero en ver que el nodo 2 está caído, ya que no puede procesar los paquetes UDP. El problema es que INET no tiene forma de notificar la caída, es decir, en las tablas de enrutamiento no cambia el sequencenumber a un número impar para indicar qué nodo ya no es válido, por lo que no hay forma de notificar eso de una forma eficaz.

2.6 Ejercicio 2.6

2.6.1 ¿Cómo se repara la ruta entre static1 y static2? ¿En qué momento?

La ruta de static1 a static2 se repara cuando todos los nodos mandan sus mensajes Hello actualizando la tabla de enrutamiento y todo el mundo conoce el cambio de topología que hay en la red gracias a los mensajes UDP Data. Es decir, cuando las tablas de enrutamiento y topología de la red están actualizadas, los nodos crean una ruta. La nueva ruta se puede ver en la siguiente imagen:

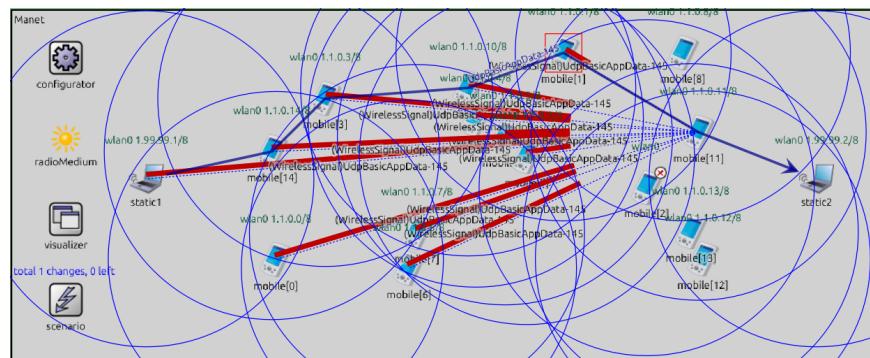


Figura 2.9: Ruta nueva después de la caída del nodo 2

Por ejemplo, antes de establecer la nueva ruta, si vemos la tabla de enrutamiento de un nodo antes de que static2 enviara un Hello y después de recibir ese Hello, vemos que ha cambiado:

```

15.558 602 015 839 mobile[1] → mobile[12] UdpBasicAppData-133 1.99.99.1:1025 1.99.99.2:7 UDP DATA 175 B (UNKNOWN) (inet::ApplicationPacket) sequenceNumber = 1
15.558 602 015 839 mobile[1] → mobile[13] UdpBasicAppData-133 1.99.99.1:1025 1.99.99.2:7 UDP DATA 175 B (UNKNOWN) (inet::ApplicationPacket) sequenceNumber = 1
15.558 602 015 839 mobile[1] → mobile[14] UdpBasicAppData-133 1.99.99.1:1025 1.99.99.2:7 UDP DATA 175 B (UNKNOWN) (inet::ApplicationPacket) sequenceNumber = 1
15.558 853 325 700 static2 → static1 Hello 1.99.99.2 255.255.255.255 IPv4 DATA 91 B (UNKNOWN) (inet::DsdvHello) srcAddress = 1.1.0.13, sequencenumber = 12, seq
15.558 853 325 700 static2 → mobile[0] Hello 1.99.99.2 255.255.255.255 IPv4 DATA 91 B (UNKNOWN) (inet::DsdvHello) srcAddress = 1.1.0.13, sequencenumber = 12,
15.558 853 325 700 static2 → mobile[1] Hello 1.99.99.2 255.255.255.255 IPv4 DATA 91 B (UNKNOWN) (inet::DsdvHello) srcAddress = 1.1.0.13, sequencenumber = 12,
15.558 853 325 700 static2 → mobile[2] Hello 1.99.99.2 255.255.255.255 IPv4 DATA 91 B (UNKNOWN) (inet::DsdvHello) srcAddress = 1.1.0.13, sequencenumber = 12,
15.558 853 325 700 static2 → mobile[3] Hello 1.99.99.2 255.255.255.255 IPv4 DATA 91 B (UNKNOWN) (inet::DsdvHello) srcAddress = 1.1.0.13, sequencenumber = 12,
15.558 853 325 700 static2 → mobile[4] Hello 1.99.99.2 255.255.255.255 IPv4 DATA 91 B (UNKNOWN) (inet::DsdvHello) srcAddress = 1.1.0.13, sequencenumber = 12,
15.558 853 325 700 static2 → mobile[5] Hello 1.99.99.2 255.255.255.255 IPv4 DATA 91 B (UNKNOWN) (inet::DsdvHello) srcAddress = 1.1.0.13, sequencenumber = 12,
15.558 853 325 700 static2 → mobile[6] Hello 1.99.99.2 255.255.255.255 IPv4 DATA 91 B (UNKNOWN) (inet::DsdvHello) srcAddress = 1.1.0.13, sequencenumber = 12,
15.558 853 325 700 static2 → mobile[7] Hello 1.99.99.2 255.255.255.255 IPv4 DATA 91 B (UNKNOWN) (inet::DsdvHello) srcAddress = 1.1.0.13, sequencenumber = 12,
15.558 853 325 700 static2 → mobile[8] Hello 1.99.99.2 255.255.255.255 IPv4 DATA 91 B (UNKNOWN) (inet::DsdvHello) srcAddress = 1.1.0.13, sequencenumber = 12,
15.558 853 325 700 static2 → mobile[9] Hello 1.99.99.2 255.255.255.255 IPv4 DATA 91 B (UNKNOWN) (inet::DsdvHello) srcAddress = 1.1.0.13, sequencenumber = 12,
15.558 853 325 700 static2 → mobile[10] Hello 1.99.99.2 255.255.255.255 IPv4 DATA 91 B (UNKNOWN) (inet::DsdvHello) srcAddress = 1.1.0.13, sequencenumber = 12,
15.558 853 325 700 static2 → mobile[11] Hello 1.99.99.2 255.255.255.255 IPv4 DATA 91 B (UNKNOWN) (inet::DsdvHello) srcAddress = 1.1.0.13, sequencenumber = 12,
15.558 853 325 700 static2 → mobile[12] Hello 1.99.99.2 255.255.255.255 IPv4 DATA 91 B (UNKNOWN) (inet::DsdvHello) srcAddress = 1.1.0.13, sequencenumber = 12,
15.558 853 325 700 static2 → mobile[13] Hello 1.99.99.2 255.255.255.255 IPv4 DATA 91 B (UNKNOWN) (inet::DsdvHello) srcAddress = 1.1.0.13, sequencenumber = 12,
15.558 853 325 700 static2 → mobile[14] Hello 1.99.99.2 255.255.255.255 IPv4 DATA 91 B (UNKNOWN) (inet::DsdvHello) srcAddress = 1.1.0.13, sequencenumber = 12,
15.560 static1 → static2 UdpBasicAppData-139 1.99.99.1:1025 1.99.99.2:7 UDP DATA 175 B (UNKNOWN) (inet::ApplicationPacket) sequenceNumber = 139 | 1025-
15.560 static1 → mobile[0] UdpBasicAppData-139 1.99.99.1:1025 1.99.99.2:7 UDP DATA 175 B (UNKNOWN) (inet::ApplicationPacket) sequenceNumber = 139 | 102

```

Figura 2.10: Stati2 mandando mensaje Hello



Figura 2.11: Nodo 11 antes de recibir Hello de static2

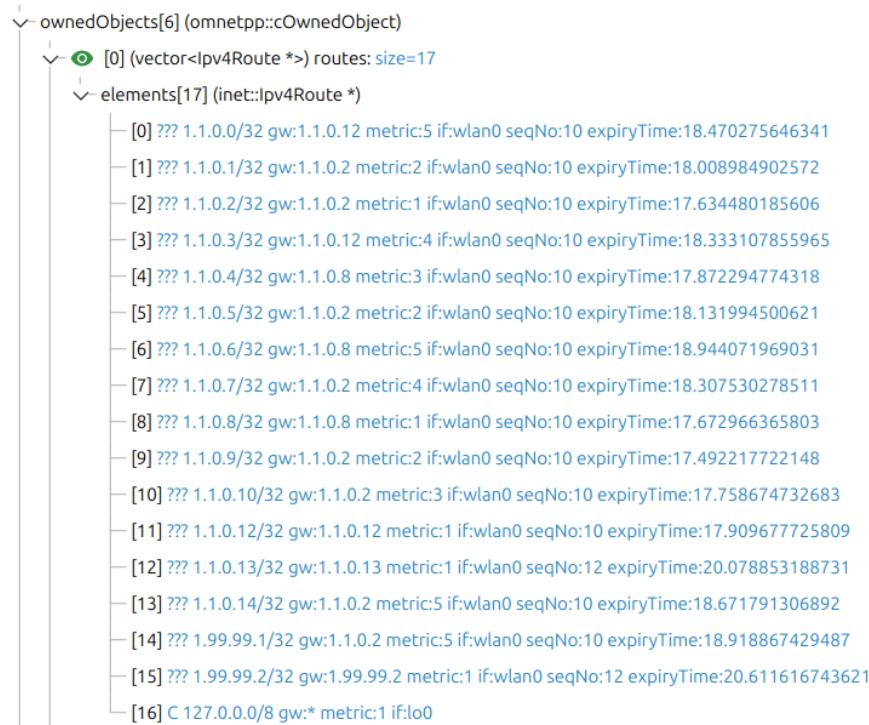


Figura 2.12: Nodo 11 después de recibir Hello de static2

Si comparamos la imagen 2.11 con 2.12 la entrada de static2 en el nodo 11 ha cambiado. Este paso se repite para todos los demás nodos y, una vez tienen la tabla de enrutamiento actualizada, se establecería la ruta que aparece en la imagen 2.9

Capítulo 3

AODV vs. DSDV

3.1 Ejercicio 3.1

3.1.1 ¿En qué instante se realiza la primera transmisión (de cualquier tipo de paquete) con AODV? ¿Y con DSDV? ¿Por qué?

La primera transmisión de AODV se hace a los 10.002 segundos. En cambio, DSDV transmite el primer paquete a los 0.056 segundos.

La diferencia que hay es debido a que DSDV es un protocolo proactivo, por lo que va actualizando cada poco tiempo las tablas de enrutamiento para así mantener la información fresca. En cambio AODV es un protocolo reactivo, solo manda paquetes cuando es necesario (en el .ini se especifica que no mandamos tramas UDP hasta los 10s, por lo que AODV no va a empezar a transmitir paquetería hasta ese instante).

3.2 Ejercicio 3.2

3.2.1 ¿En qué instante recibe static2 el datagrama UdpBasicAppData-0 con AODV? ¿Y con DSDV? ¿Por qué?

Con AODV static2 recibe el datagrama a los 11.4526 segundos, mientras que con DSDV lo recibe a los 10.0030 segundos. Esto ocurre porque en AODV los nodos tienen que ir actualizando la tabla de enrutamiento para poder encontrar la ruta y esto lleva un tiempo extra, en cambio en DSDV al tener todos los nodos sus tablas actualizadas, a la hora que querer establecer la ruta ya lo hace al instante.

3.3 Ejercicio 3.3

3.3.1 ¿Cuántos paquetes se pierden como consecuencia de la caída del nodo en AODV? ¿Y en DSDV? ¿A qué se debe la diferencia? (Nota: para calcular el número de paquetes perdidos avanza hasta un instante posterior a la caída en el que en ambos escenarios static2 vuelva a recibir datagramas y calcula la diferencia entre el número de paquetes recibidos en static2 con y sin la caída del nodo).

Para el caso de DSDV tuvimos que usar la semilla 65634 para poder ver con claridad la pérdida de paquetes y así hacer la comparación. En AODV mantenemos la semilla inicial.

Hemos hecho las simulaciones en ambos casos hasta los 22 segundos y en la tabla 3.1 se ven los resultados:

Protocolo	Sin fallo	Con fallo	% pérdidas
AODV	325	323	0.6%
DSDV	254	156	38.5%

Cuadro 3.1: Tabla comparativa pérdida paquetes al caer un nodo

Como vemos, en AODV la pérdida de paquetes es irrisoria con respecto a DSDV. Esto se debe a que en AODV, cuando un nodo detecta un fallo en la ruta, este envía un RERR a todos los nodos para informarlos del error por lo que se empieza a buscar inmediatamente otra ruta alternativa. En cambio en DSDV, cuando un nodo detecta una ruta inválida, no se genera otra ruta hasta que todos los nodos tengan su tabla de enrutamiento actualizada por lo que esto se traduce a una pérdida significativa de paquetes ya que tardan en actualizar sus tablas un cierto período de tiempo.

3.4 Ejercicio 3.4

3.4.1 Con la caída del nodo desactivada, simula AODV y DSDV durante 300 s (sim-time-limit). Muestra capturas de los nodos estáticos a nivel de aplicación (doble click sobre el nodo) al final de la simulación. ¿Qué porcentaje de los UdpBasicAppData enviados por static1 llega a static2? Explica los valores y las diferencias observadas

Primero vamos a ver lo que pasa en AODV, para eso vamos a fijarnos en las siguientes imágenes:

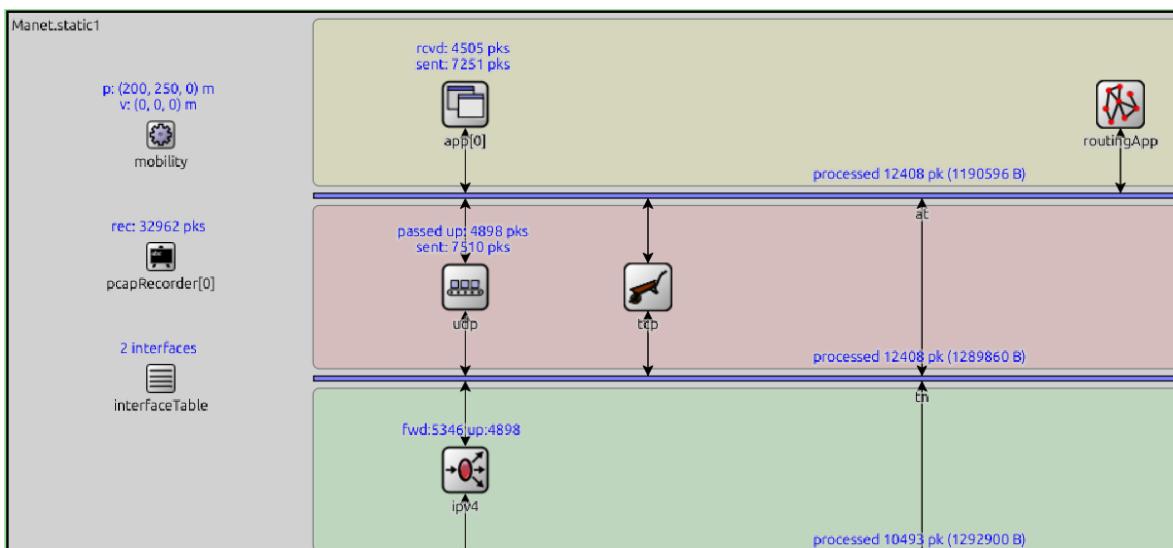


Figura 3.1: Static1 a los 300s en AODV

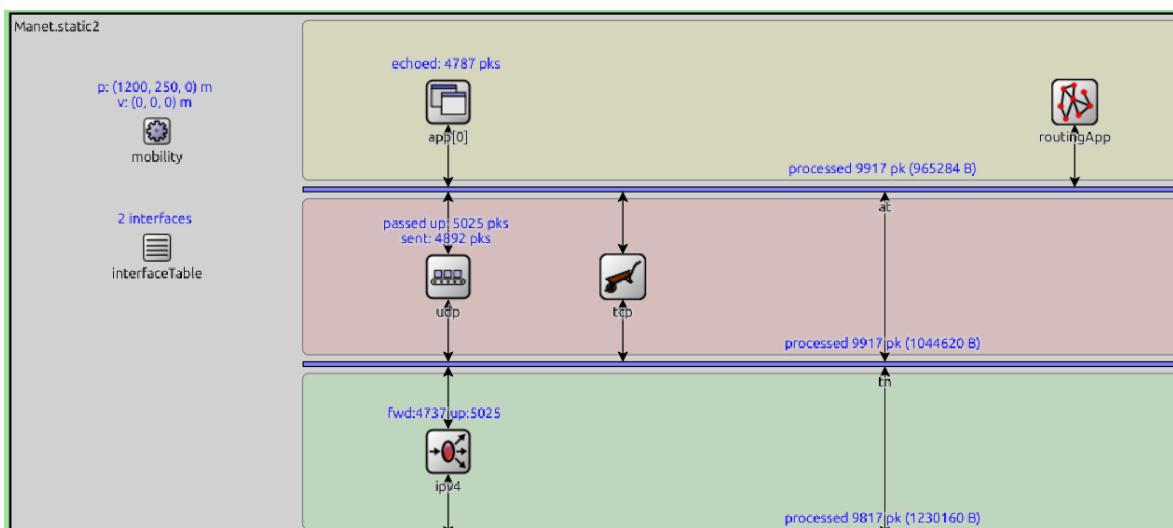


Figura 3.2: Static2 a los 300s en AODV

Como vemos, en la imagen 3.1, static1 manda 7251 paquetes y static2 recibe 4787 (imagen 3.2), por lo que se traduce a que a static2 le llega el 66% de los paquetes.

Ahora vamos a ver lo que pasa en DSDV:

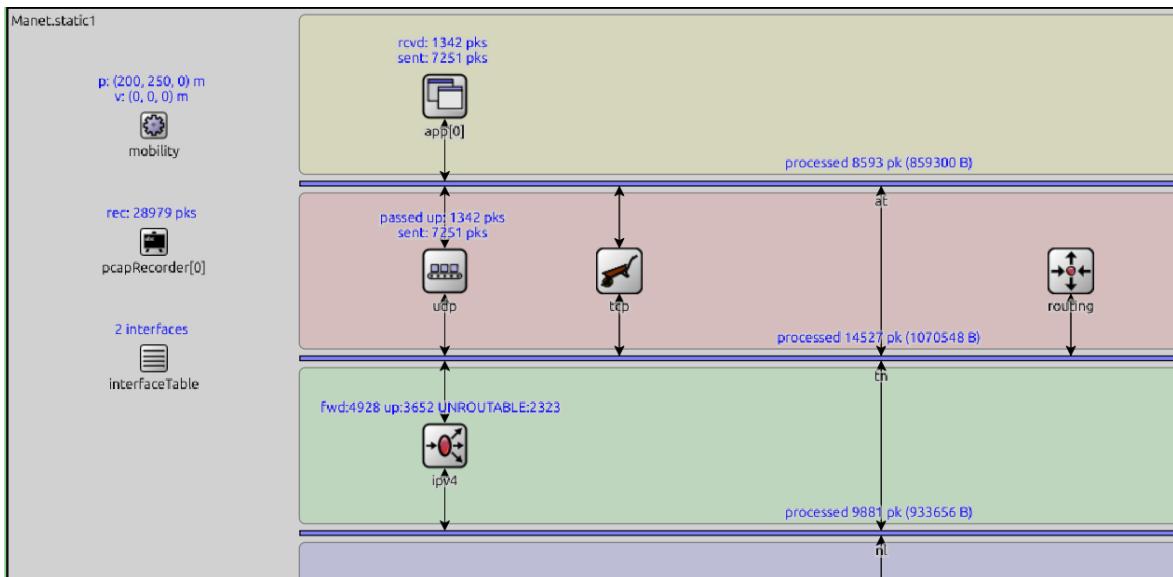


Figura 3.3: Static1 a los 300s en DSDV

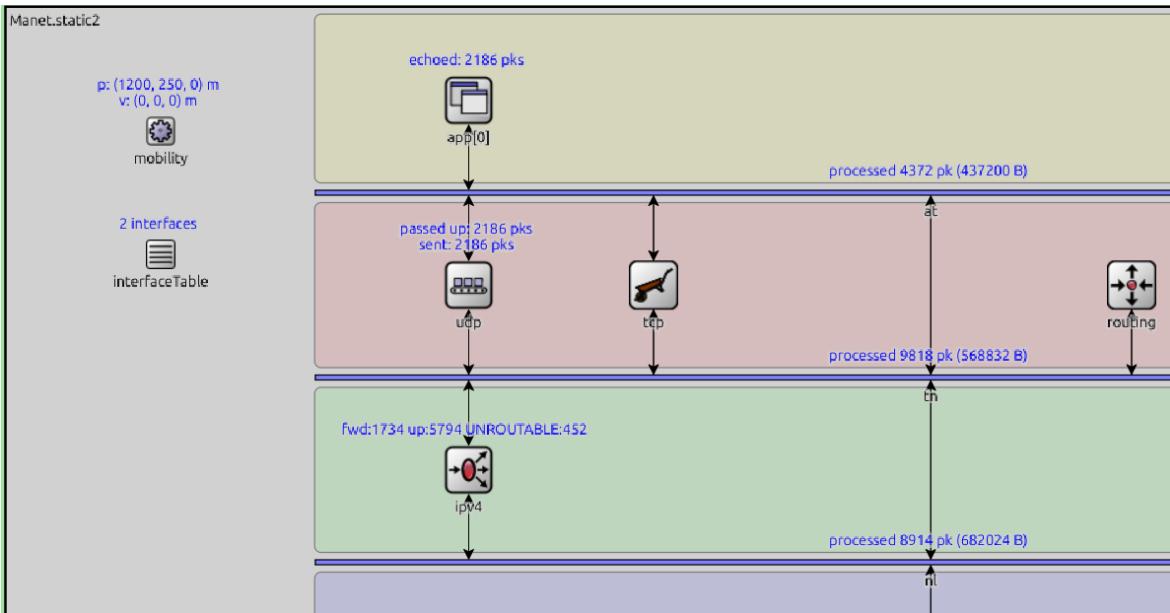


Figura 3.4: Static2 a los 300s en DSDV

Si vemos la imagen 3.3, static1 manda 7251, mientras que a static2 le llegan 2186 (ver imagen 3.4), por lo que solamente le llega a static2 el 30%.

La diferencia que hay entre AODV y DSDV es muy grande. Concretamente, DSDV pierde el doble de los paquetes con respecto a AODV. Esto se debe a la constante actualización de las tablas de enrutamiento que hay en DSDV, por lo que cambia constantemente de ruta, mientras que en AODV cuando detecta una ruta, se mantiene todo el tiempo activa hasta que no sea válida. Además, en DSDV cuando se cambia de ruta o deja de ser válida, tiene que volver a hacer el proceso de reconstrucción y esto se traduce en un período de tiempo que static2 no recibe ningún paquete de static1.

3.5 Ejercicio 3.5

3.5.1 ¿Qué porcentaje de los paquetes devueltos por static2 llegan a static1? Explica los valores y las diferencias observadas.

Para el caso de AODV, static2 hace echo de los 4787 paquetes que recibe y static1 recibe 4505 paquetes (imagen 3.1), por lo que esto lleva a que static1 recibe el 94% de los paquetes que le manda static2.

En DSDV, static2 manda 2186 paquetes y static1 recibe 1342 paquetes (imagen 3.3), por lo que se traduce a que static1 recibe el 61% de los paquetes mandados por static2.

La diferencia que hay entre AODV y DSDV es dada a como funciona cada protocolo. AODV por cada RREQ que se manda entre los nodos, hay un RREP haciendo la ruta inversa por la misma ruta que se ha establecido de ida, por lo que es más difícil de que haya pérdida de paquetes. En cambio, en DSDV al no haber un control en la ruta inversa, si algún nodo no es capaz de poder devolver el paquete, se pierde entonces hasta que tenga una ruta de vuelta actualizada/disponible. También hay que comentar que en DSDV hay una mayor congestión en el tráfico, por lo que se puede provocar colisiones y sobrecargas.

AODV vs. DSDV: gráficas

4.1 Ejercicio 4.1

- 4.1.1 Obtén una gráfica que muestre el número de paquetes recibidos por static2 en función de la potencia tanto para AODV como para DSDV

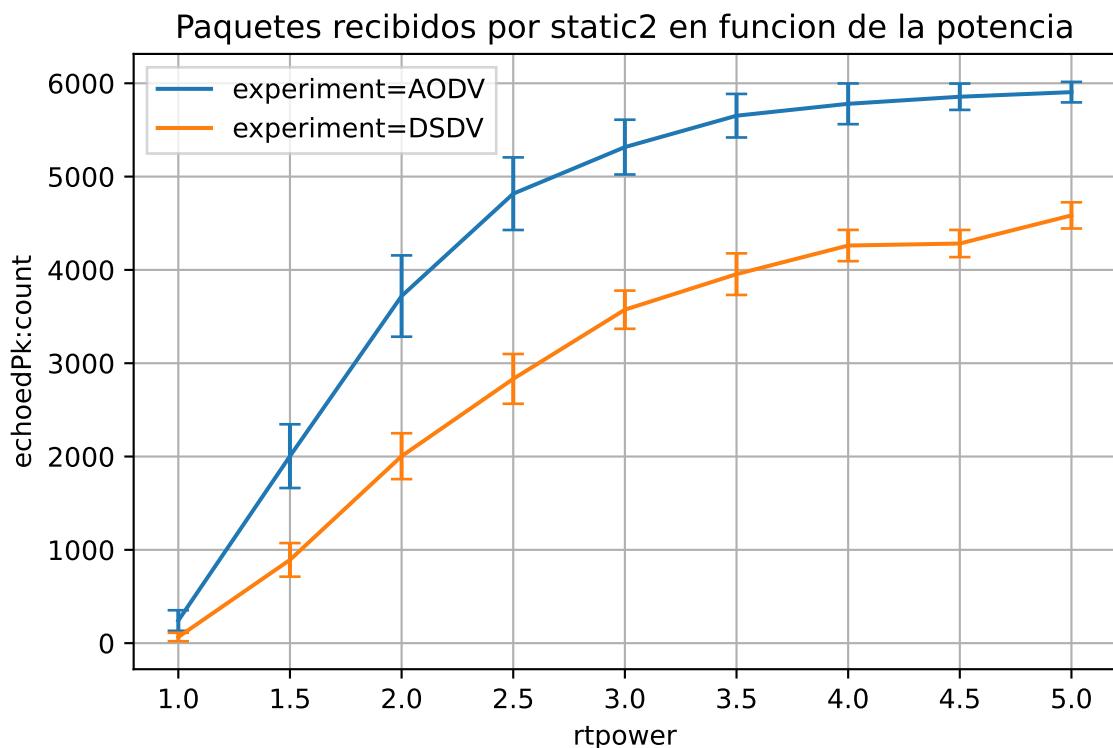


Figura 4.1: Gráfica paquetes recibidos en static2 en función de la potencia

El gráfico 4.1 muestra que el nodo final static2 recibe más paquetes con AODV que con DSDV según aumenta la potencia de transmisión. Esto ocurre debido a las diferencias en la forma de enrutamiento de ambos protocolos.

AODV actúa de forma reactiva, de forma que establece rutas únicamente cuando se necesitan. A medida que aumenta la potencia de transmisión, se incrementa el rango de cobertura y el número de rutas alternativas (Es bastante improbable que se pierdan rutas, pero pueden existir nuevas rutas más óptimas). Sin embargo, AODV se queda con la ruta que conoce y sabe que funciona, a no ser que se produzca un error, lo cual no es habitual al aumentar la potencia.

Sin embargo, DSDV es un protocolo proactivo, por lo que mantiene una tabla de enrutamiento actualizada en todo momento, lo que introduce una mayor sobrecarga debido al intercambio constante de actualizaciones. Además, este mecanismo necesita que todos los nodos recompongan sus tablas de enrutamiento con mensajes hello después de un cambio de un error por cambios en la topología, lo que hace que se pierda tiempo en ese proceso en vez de seguir enviando paquetes según la red se vuelve más compleja.

4.2 Ejercicio 4.2

4.2.1 Obtén una gráfica similar para el número de nodos. Explica lo observado.?

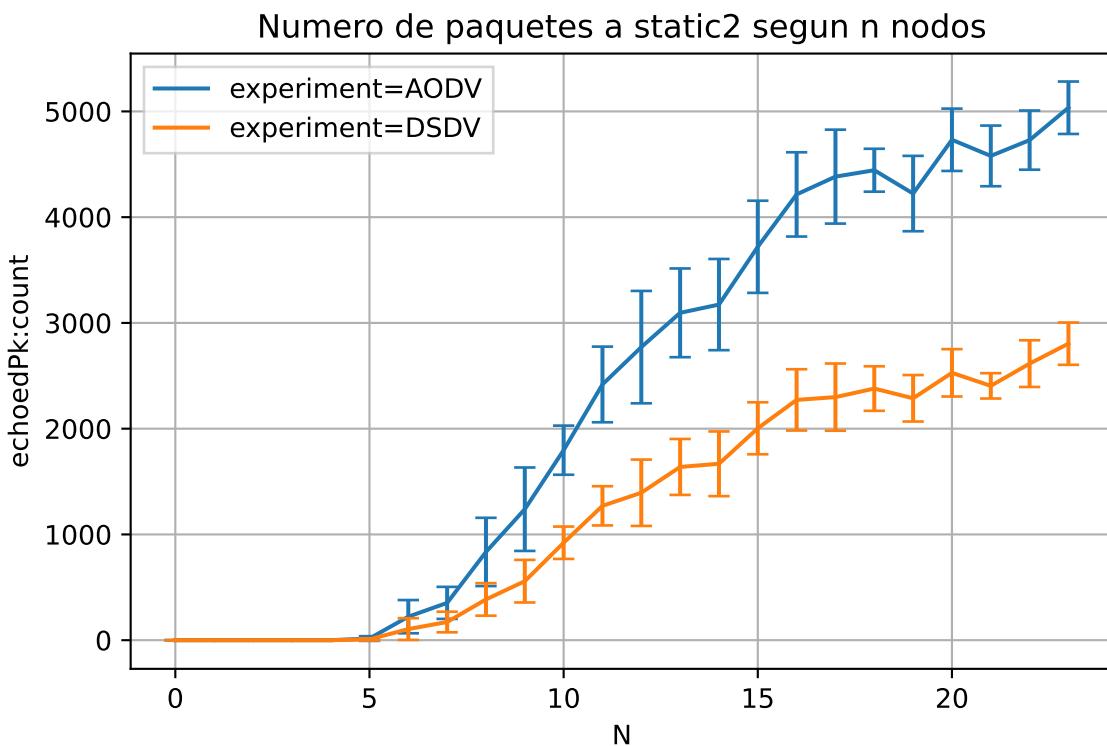


Figura 4.2: Gráfica paquetes recibidos en static2 en función número de nodos

Hay que destacar que en el .ini tenemos que llegue a 30 nodos pero por cuestiones de tiempo paramos en 23 ambos.

Como podemos ver en la gráfica 4.2, en AODV le llega a static2 mas paquetes que en DSDV. También hay que decir que en los dos aumenta pero AODV más rápido que en DSDV. La diferencia es:

En AODV en realidad no importa si aumenta el numero de nodos ya que una vez establece una ruta, a no ser que caiga, no va a dejar de usarla a no se que se rompa.

En el caso de DSDV, le llega a static2 menos paquetes ya que al ir aumentando el número de nodos, los nodos vecinos ven que hay nuevos nodos por lo que van actualizando la tabla de enrutamiento entonces no se establece una ruta fija hasta que las tablas estean actualizadas.

También cabe destacar que por mucho que se aumente el número de nodos, va llegar un punto que para ambos protocolos el número de paquetes que le llega static2 va a ser constante.

4.3 Ejercicio 4.3

4.3.1 Obtén una gráfica como las anteriores para la velocidad. Explica lo observado.

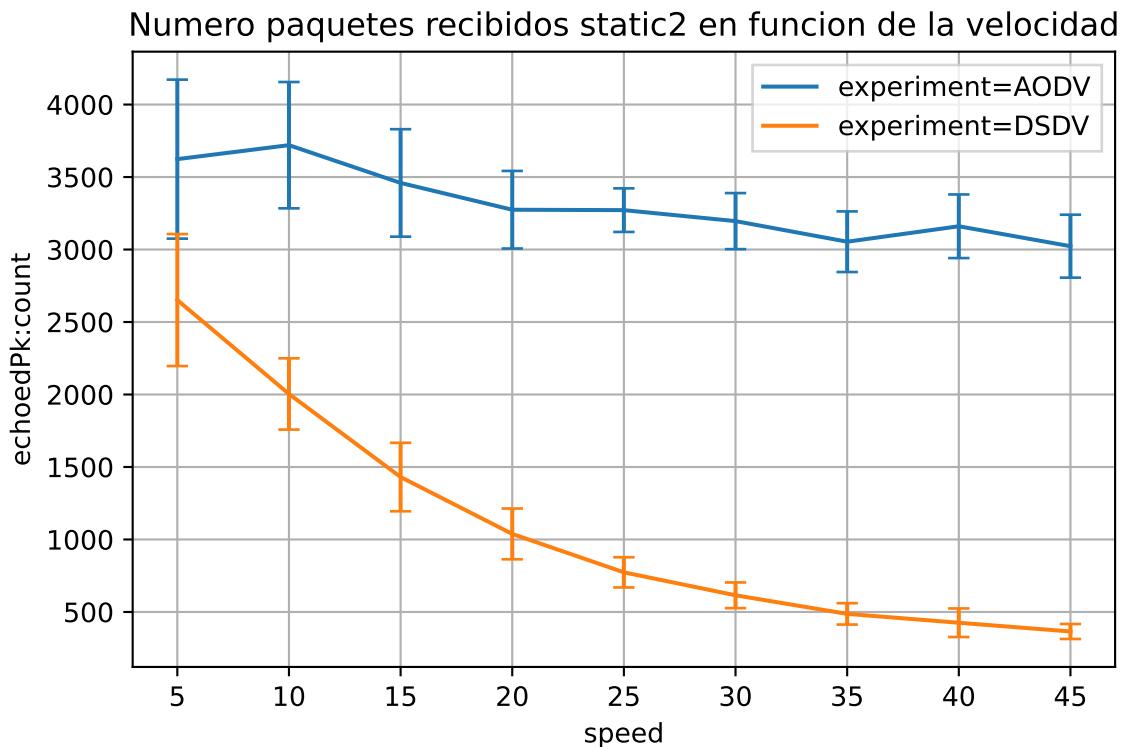


Figura 4.3: Gráfica paquetes recibidos en static2 en función de la velocidad

Como se puede ver en la gráfica 4.3, en AODV se mantiene más o menos el número de paquetes que le llega a static2, mientras que en DSDV a medida que la velocidad de los nodos aumenta, baja drásticamente los paquetes que le llegan a static2. Esto ocurre ya que en AODV, aunque los nodos se muevan mucho, esto no afecta a las rutas ya que aunque un nodo no sea alcanzable, si lo es con otro por lo que siempre va haber una ruta y se va restablecer de forma rápida. En cambio en DSDV el movimiento continuo y aumento de velocidad hace que empeore la situación. Esto ocurre ya que en DSDV para que se establezca una ruta tienen que estar en todos los nodos, sus tablas de enrutamiento actualizadas, por lo que el movimiento continuo de los nodos dificulta la estabilización de las tablas por lo que no se establece una ruta y los paquetes no le llegan a static2.

4.4 Ejercicio 4.4

4.4.1 Para DSDV, obtén una gráfica del porcentaje de paquetes perdidos con diferentes valores de helloInterval. Explica lo observado.