



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

---

<b>1 AODV</b>	<b>1</b>
1.1 Ejercicio 1.1 . . . . .	1
1.2 Ejercicio 1.2 . . . . .	2
1.3 Ejercicio 1.3 . . . . .	2
1.4 Ejercicio 1.4 . . . . .	2
1.5 Ejercicio 1.5 . . . . .	3
1.6 Ejercicio 1.6 . . . . .	3
1.7 Ejercicio 1.7 . . . . .	3
1.8 Ejercicio 1.8 . . . . .	3
<b>2 DSDV</b>	<b>4</b>
2.1 Ejercicio 2.1 . . . . .	4
2.2 Ejercicio 2.2 . . . . .	5
2.3 Ejercicio 2.3 . . . . .	5
2.4 Ejercicio 2.4 . . . . .	6
2.5 Ejercicio 2.5 . . . . .	8
2.6 Ejercicio 2.6 . . . . .	9
<b>3 AODV vs. DSDV</b>	<b>12</b>
3.1 Ejercicio 3.1 . . . . .	12
3.2 Ejercicio 3.2 . . . . .	12
3.3 Ejercicio 3.3 . . . . .	12
3.4 Ejercicio 3.4 . . . . .	13
3.5 Ejercicio 3.5 . . . . .	15

# Índice de figuras

---

1.1	Logs que muestran el envío del primer RREQ y los nodos que lo reciben . . . . .	1
1.2	Nodos que reenvían el primer RREQ (A la izquierda, mobile[10]; A la derecha, mobile[12]) . . . . .	2
2.1	Log del nodo que manda el primer Hello con hopdistance 3 . . . . .	4
2.2	Tabla de enrutamiento nodo 8 . . . . .	5
2.3	Tabla de enrutamiento nodo 5 antes recibir Hello de nodo 8 . . . . .	6
2.4	Tabla de enrutamiento nodo 9 antes recibir Hello de nodo 8 . . . . .	7
2.5	Tabla de enrutamiento nodo 5 antes recibir Hello de nodo 8 . . . . .	7
2.6	Tabla de enrutamiento nodo 9 antes recibir Hello de nodo 8 . . . . .	8
2.7	Ruta antes de la caída . . . . .	9
2.8	Mensaje del nodo 1 cuando no es capaz de enviar los UDP datas al detectar un nodo caído . . . . .	9
2.9	Ruta nueva después de la caída del nodo 2 . . . . .	9
2.10	Stati2 mandando mensaje Hello . . . . .	10
2.11	Nodo 11 antes de recibir Hello de static2 . . . . .	10
2.12	Nodo 11 después de recibir Hello de static2 . . . . .	11
3.1	Static1 a los 300s en AODV . . . . .	13
3.2	Static2 a los 300s en AODV . . . . .	14
3.3	Static1 a los 300s en DSDV . . . . .	14
3.4	Static2 a los 300s en DSDV . . . . .	15

# Capítulo 1

## AODV

### 1.1 Ejercicio 1.1

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

SEMILLA A USAR DURANTE LA PRACTICA: 2701

El primer paquete RREQ enviado por static1 es recibido 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 el primer envío contiene un TTL igual a 2 (aparece en la documentación de Inet), por lo que solamente van a responder los dispositivos a los que le llegue un TTL > 1 para así poder hacer el reenvío.

El segundo RREQ es reenviado por 10, 12, 3, 1, 7, 2. Al hacer el segundo reenvío, el TTL del paquete pasa a ser 4 (según la documentación de INET, en los siguientes paquetes, el TTL se suma 2 con respecto al anterior ya que el objetivo es poder llegar lo más lejos posible). Como ahora el TTL es 4, pasa por 10 y 12 otra vez (el TTL pasa a ser 3), 12 no alcanza ningún objetivo pero 10 logra mandar ese paquete a 1 y 3 (el TTL pasa a ser 2) y como 3 llega alcanzar a 7 y 2, se los envía también. Llegados a este punto, el TTL es 1 por lo que ya que se acabaría todos los posibles reenvíos del segundo RREQ.

```
** 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::Seql
INFO: Changing radio transmission state from IDLE to TRANSMITTING.
INFO: Changing radio transmitted signal part from NONE to WHOLE.
** Event #1720 t=10.002964558393 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) (ir
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[12].wlan[0].radio (Ieee80211ScalarRadio, id=990) on aodv::Rreq (inet::physicalayer
INFO: Reception started: attempting (inet::physicallayer::WirelessSignal)aodv::Rreq (58 us 99 B) (inet::Packet)aodv::Rreq (99 B) (ir
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::physicalayer
INFO: Reception started: not attempting (inet::physicalayer::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::physicalayer
INFO: Reception started: not attempting (inet::physicalayer::WirelessSignal)aodv::Rreq (58 us 99 B) (inet::Packet)aodv::Rreq (99 B);
** Event #1724 t=10.002965706956 Manet.mobile[7].wlan[0].radio (Ieee80211ScalarRadio, id=665) on aodv::Rreq (inet::physicalayer
INFO: Reception started: not attempting (inet::physicalayer::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::physicalayer
INFO: Reception started: not attempting (inet::physicalayer::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::physicalayer
INFO: Reception started: not attempting (inet::physicalayer::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::physicalayer
INFO: Reception started: not attempting (inet::physicalayer::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::physicalayer
INFO: Reception started: not attempting (inet::physicalayer::WirelessSignal)aodv::Rreq (58 us 99 B) (inet::Packet)aodv::Rreq (99 B;
** Event #1729 t=10.002966271747 Manet.mobile[0].wlan[0].radio (Ieee80211ScalarRadio, id=210) on aodv::Rreq (inet::physicalayer
INFO: Reception started: not attempting (inet::physicalayer::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::physicalayer
```

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

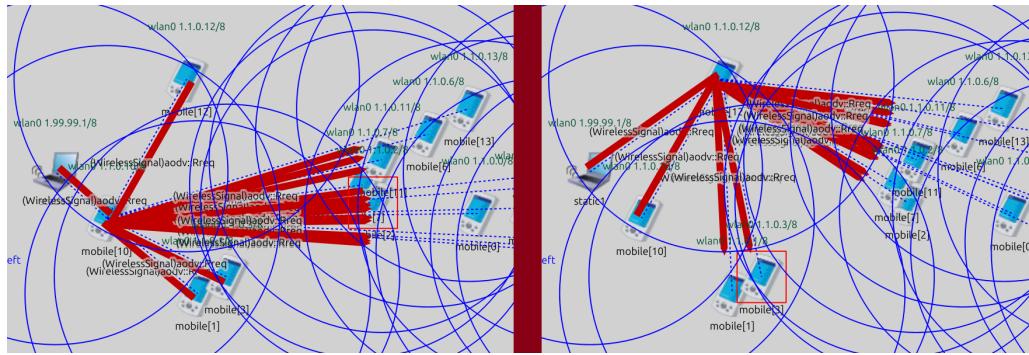


Figura 1.2: Nodos que reenvían el primer RREQ (A la izquierda, mobile[10]; A la derecha, mobile[12])

## 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).

## 1.3 Ejercicio 1.3

- 1.3.1** Haz lo mismo justo antes y justo después del 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?

Hay que aclarar que con la semilla que usamos, a partir de  $t=15$  el nodo alternativo para establecer la ruta se alejaba un poco de static2 por lo que hicimos las pruebas en  $t=12s$ .

**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?

**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?

**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.

**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?

# 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].li: 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:

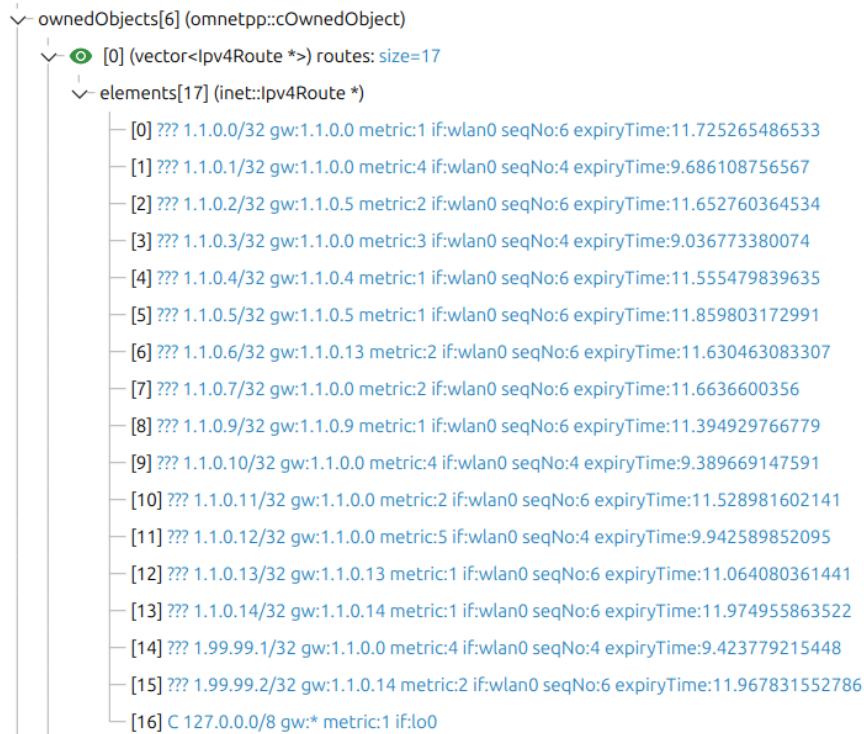


Figura 2.2: Tabla de enrutamiento nodo 8

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 y el campo nextAddress es la siguiente dirección que va a retransmitir la trama, 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), va a sumar 1 para transmitirlo 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 evita 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), ese nodo 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 respeto 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 invalido con un sequencenumber impar, este al ver que si que es alcanzable, como el nodo invalido 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)
      ↴ [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 2.3: 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



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 aieno 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:

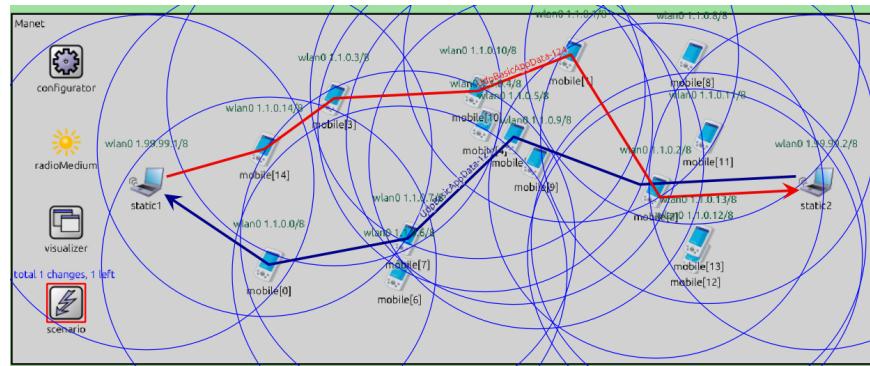


Figura 2.7: Ruta antes de la caída

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 que nodo ya no es válido por lo que no hay forma de notificar eso de una forma eficiente.

## 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:

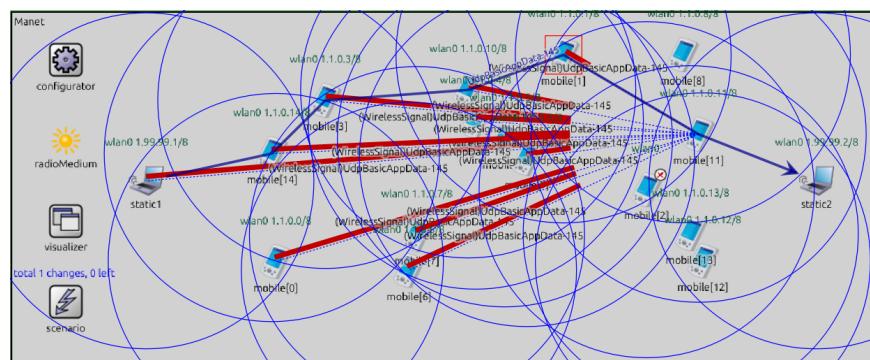


Figura 2.9: Ruta nueva después de la caída del nodo 2

Por ejemplo, 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

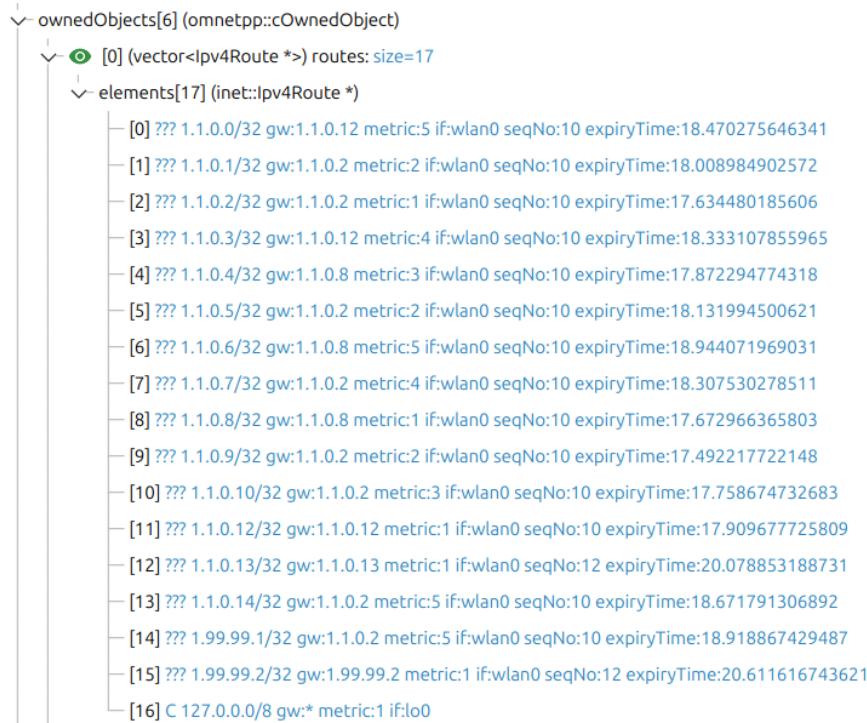


Figura 2.12: Nodo 11 después de recibir Hello de static2

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

ELIMINAR ESTO ELIMINAR ESTO ELIMINAR ESTO non recibe toda a tabla de enrutamiento cando si que deberia, algunas veces os nodos mandan unha entrada da sua tabla, outras veces manda varias entradas..

## 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 dado 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.

### 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 vuelve 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.

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 irrisorio 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:

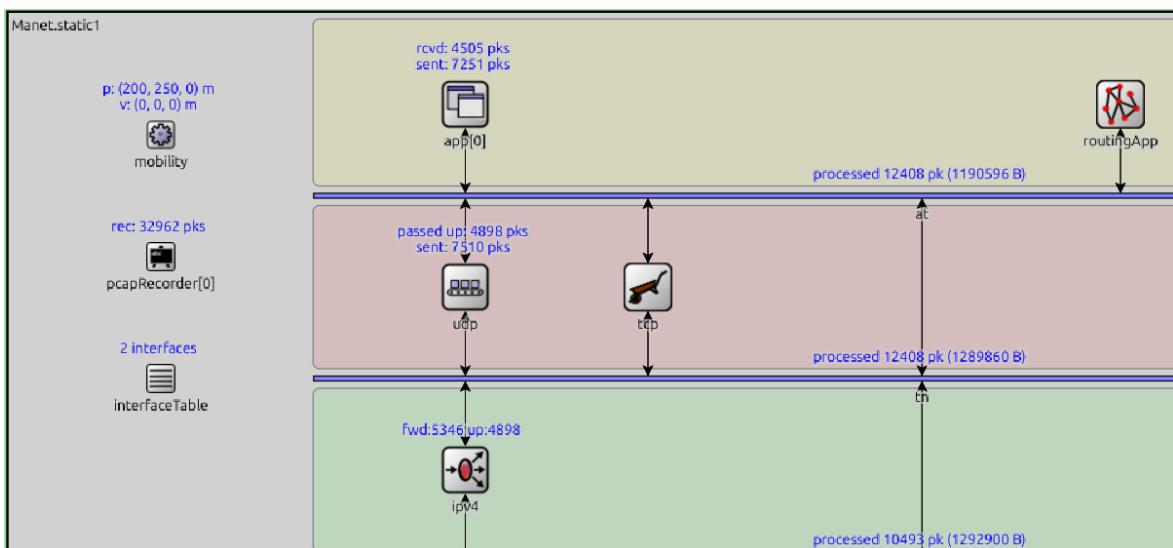


Figura 3.1: Static1 a los 300s en AODV

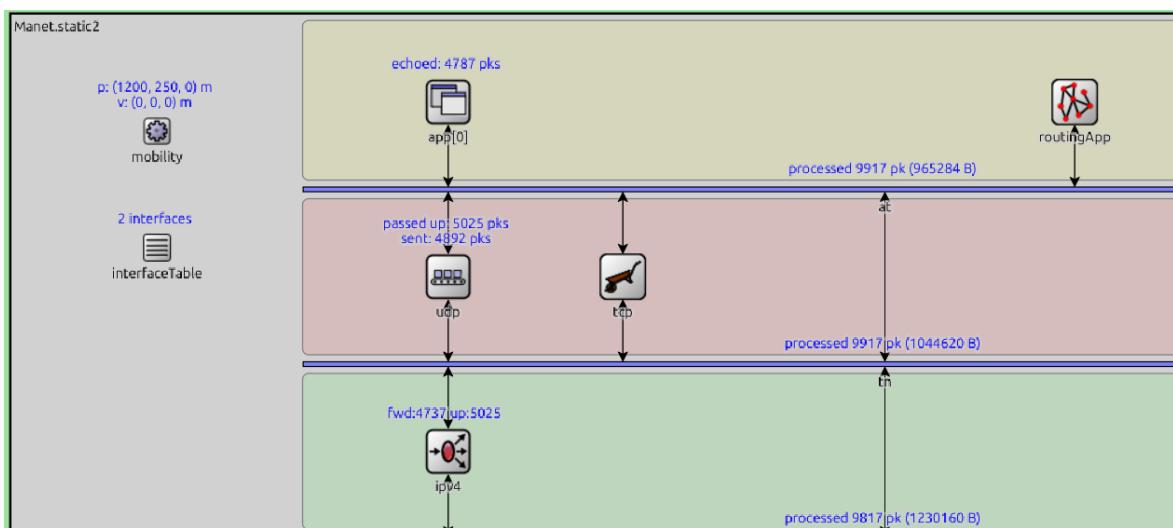


Figura 3.2: Static2 a los 300s en AODV

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:

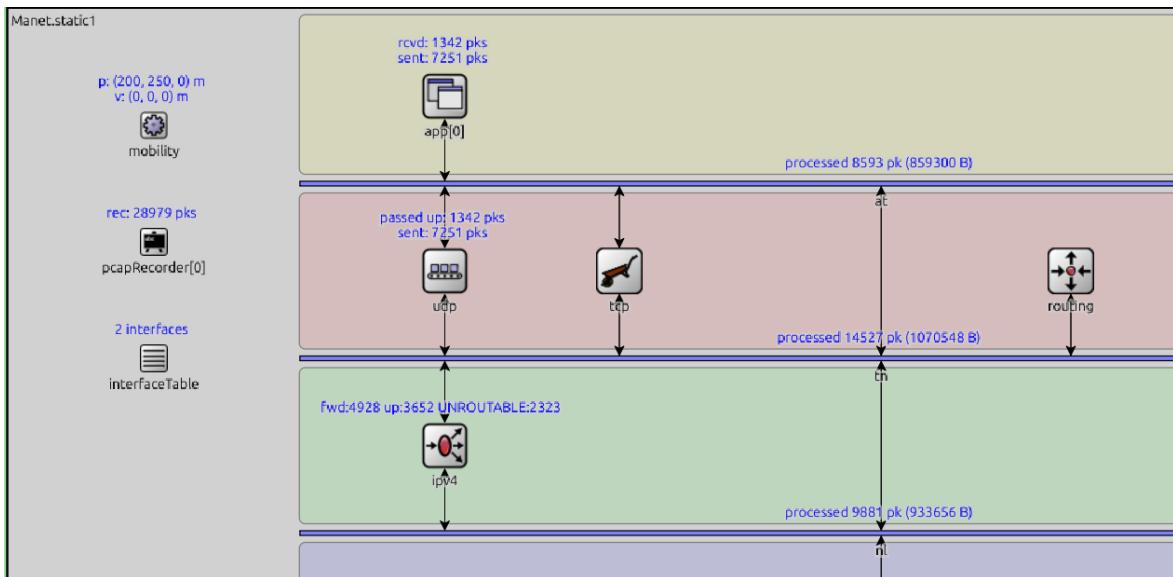


Figura 3.3: Static1 a los 300s en DSDV

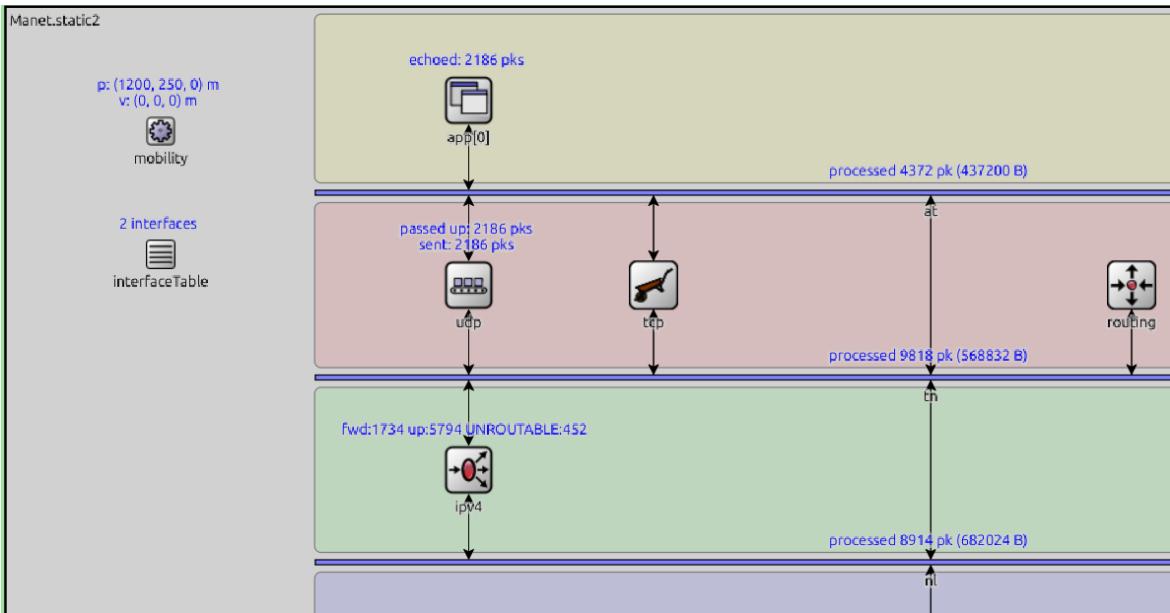


Figura 3.4: Static2 a los 300s en DSDV

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 reciba 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 destacar que en DSDV hay una mayor congestión en el tráfico por lo que se puede provocar colisiones y sobrecargas.