Fragmentación y Mezcla de redes ad-hoc utilizando el protocolo ANTop sobre IPv6

Tesis de grado

Ingeniería Electrónica

Marzo 2018

Tesista: Pablo Damián Torrado

Tutor: Dr. Ing. José Ignacio Alvarez-Hamelin

Índice general

Índice de figuras **7**

Índice de cuadros **13**

1. Introducción **1**
1.1. Objetivos 1
1.2. Organización del documento 2

2. Estado del arte **3**
2.1. B.A.T.M.A.N (Better Approach To Mobile Adhoc Networking 3
2.1.1. Descubrimientos de rutas 4
2.1.2. Mantenimientos de rutas 5
2.1.3. Procesamiento y retransmisión 6
2.2. Babel 6
2.2.1. Descubrimientos de rutas 7
2.2.2. Mantenimientos de rutas 9
2.3. ANTop (Adjacent Network Topoly) 10
2.3.1. Ruteo Indirecto 10
2.3.2. Direccionamiento en un Hipercubo 10
2.3.3. Conexión de nodos 13
2.3.4. Desconexión de nodos 20
2.3.5. Ruteo 20
2.3.6. Resolución de nombres, servicio *Rendez Vous* 24
2.3.7. Ruteo de paquetes *Rendez Vous* 28
2.3.8. Paquetes de control ANTop 29

3. Implementación ANTop en SO Linux 4.10.0 **35**
3.1. Interacción con el sistema operativo 36
3.1.1. IOCTL (Input / Output Control) 36
3.1.2. Netlink 37
3.1.3. Netfilter 39
3.1.4. Capa de enlace 41
3.2. Kernel 42
3.3. Direccionamiento en ANTop sobre IPv6 46
3.4. Prueba de funcionamiento 48

4. Fragmentación y mezcla **53**
4.1. Numeración de red 53
4.1.1. Propósito y campo *número de red* 54
4.1.2. Algoritmo de asignación de *número de red* 58
4.2. Fragmentación 60
4.2.1. Definición y detección 61
4.2.2. Proceso de asignación de direcciones 66
4.3. Mezcla 74
4.3.1. Definición y detección 74
4.3.2. Proceso de asignación de direcciones 77

5. Simulación de redes **101**
5.1. Mininet 101
5.1.1. Mininet Wifi 102
5.2. Simulaciones 106
5.2.1. Redes aleatorias 106
5.2.2. Fragmentación de una red 114
5.2.3. Mezcla de dos redes 132

6. Conclusiones **147**
6.1. Trabajos futuros 149

A. Scripts de simulación **151**
A.1. Script de simulación de redes aleatorias 151
A.2. Script de simulación de fragmentación topología 1 153
A.3. Script de simulación de mezcla topología 1 155

Bibliografía **157**

Índice de figuras

2.1. Desconexión de enlace entre nodos en una red BATMAN. . 5
2.2. Esquema de tabla de rutas de los nodos de una red Babel . 8
2.3. Modelo de ruteo indirecto 11
2.4. Direccionamiento basado en hipercubo 12
2.5. Paquete de control genérico ANTop 29
2.6. Encabezado opcional *Additional Address* 29
2.7. Paquete de control *Rendez Vous* 32

3.1. Direccionamiento sockets Netlink 38
3.2. Conexión espacio usuario al kernel 39
3.3. Conexión espacio usuario al espacio usuario 39
3.4. Escucha del espacio usuario las notificaciones del kernel . . 40
3.5. Diagramas de ganchos de Netfilter 41
3.6. Vínculos de módulos del kernel, Fuente Figura 2.1 de [12] . 44
3.7. Segmentación de dirección IPv6. 47
3.8. Formato de dirección Global Unicast IPv6. 47
3.9. Configuración de la placa inalámbrica del *nodo 1*. 48
3.10. Tabla de rutas del nodo1. 49
3.11. Configuración de la placa inalámbrica del *nodo 1* , al correr ANTop. 49
3.12. Tabla de rutas del *nodo 1*, al correr ANTop. 50
3.13. PAP del *nodo 1* recibido por el *nodo 2*. 50
3.14. Configuración de la placa inalámbrica del *nodo 2* , al correr ANTop. 51
3.15. Tabla de rutas del *nodo 2*, al correr ANTop. 51
3.16. Tabla de vecinos del *nodo 1*. 51

3.17. Tabla de vecinos del *nodo 2*. 52

4.1. Dos redes ANTop independientes 54
4.2. Dos redes ANTop independientes mezcladas 55
4.3. Paquete de control genérico ANTop con encabezado opcional 57
4.4. Encabezado opcional modificado 58
4.5. Formato general de dirección MAC 59
4.6. Red ANTop antes de la fragmentación 62
4.7. Red ANTop fragmentada 63
4.8. Fragmentación de red ANTop con enlace secundario 64
4.9. Direccionamiento de red ANTop fragmentada 67
4.10. Envío de paquetes FAR desde el nodo detector 70
4.11. Envío de paquetes FAR y FAN de nodos intermedios. . . . 72
4.12. Dos redes ANTop independientes y numeradas 75
4.13. Mezcla de dos redes ANTop numeradas 76
4.14. Resultado propuesto de la mezcla de dos redes ANTop . . . 78
4.15. Primer caso de tratamiento de dirección del mecanismo 1 de mezcla 80
4.16. Segundo caso de tratamiento de dirección del mecanismo 1 de mezcla 82
4.17. Tercer caso de tratamiento de dirección del mecanismo 1 de mezcla 85
4.18. Paquete de control genérico con encabezado opcional con el campo “Lengh“ habilitado. 88
4.19. Tratamiento de direcciones mecanismo 2 90
4.20. Envío del paquete MAR1 del *par emisor* al *par receptor*. . . 94
4.21. Envíos de MAR1 y MAR2 del par recepto, y confirmación MAN1 al nodo emisor. 95
4.22. Propagación de mensaje MAR1, MAR2 y sus confirmaciones. 97

5.1. Componentes Mininet Wifi, Fuente Figura 1.1 de [18] . . . 102
5.2. Componentes y conexiones en dos host creados con Mininet Wifi, Fuente Figura 1.2 de [18] 103
5.3. Cantidad de vecinos 108
5.4. Cantidad de entradas máximas en la tabla de ruteo 109
5.5. Cantidad de entradas en la tabla *Rendez Vous* 110

5.6. Nodos cercanos: Posición aleatoria de los nodos en cada simulación. 112
5.7. Nodos lejanos: Posición aleatoria de los nodos en cada simulación. 113
5.8. Topología 1, t=0. 115
5.9. Topología 1, t=100s. 116
5.10. Topología 1, ping desde `sta11` a la dirección *2001:db8::c000:0:0:1* antes de la fragmentación. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*. 117
5.11. Topología 1, ping desde `sta8` a la dirección *2001:db8::1* antes de la fragmentación. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*. 118
5.12. Topología 1, ping desde `sta8` a la dirección *2001:db8::8000:0:0:1* antes de la fragmentación. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*. 119
5.13. Topología 1, t=300s. Fragmentación en Red 1 a la Izquierda y Red 2 a la derecha. 120
5.14. Topología 1, captura de paquetes de control FAR y FAN en `sta8` al momento de la fragmentación. En la figura de la izquierda se muestra a `sta8` enviando un paquete FAR a `sta13`, y la figura de la derecha la confirmación FAN de `sta13` a `sta8` 121
5.15. Topología 1, captura de paquetes de control FAR y FAN en `sta8` al momento de la fragmentación. En la figura de la izquierda se muestra a `sta8` enviando un paquete FAR a `sta14`, y la figura de la derecha la confirmación FAN de `sta14` a `sta8` 121
5.16. Topología 1, captura de paquetes de control FAR y FAN en `sta8` al momento de la fragmentación. En la figura de la izquierda se muestra a `sta8` enviando un paquete FAR a `sta11`, y la figura de la derecha la confirmación FAN de `sta11` a `sta8` 122
5.17. Topología 1, ping desde `sta11` a la dirección *2001:db8::c000:0:0:1* luego de la fragmentación. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*. 123

5.18. Topología 1, ping desde `sta8` a la dirección *2001:db8::8000:0:0:1* luego de la fragmentación. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*. 124
5.19. Topología 2, t=0. 125
5.20. Topología 2, t=150s. 126
5.21. Topología 2, ping desde `sta7` a la dirección *2001:db8::1* antes de la fragmentación. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*. 127
5.22. Topología 2, t=400s. Fragmentación en Red 1 abajo y Red 2 arriba. 128
5.23. Topología 2, ping desde `sta7` a la dirección *2001:db8::1* después de la fragmentación. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*. 129
5.24. Topología 2, ruteo de paquetes ICMPv6 *Echo Request* en `sta10` producto del *ping* de `sta3` a `sta12`, luego de la fragmentación. 130
5.25. Topología 2, ruteo de paquetes ICMPv6 *Echo Reply* en `sta10` producto del *ping* de `sta3` a `sta12`, luego de la fragmentación. 131
5.26. Topología 1, t=0. 133
5.27. Topología 1, t=200s. Mezcla de redes, Red 2 es absorbida por la Red 1. 134
5.28. Topología 1, captura de paquetes de control MAR1, MAR2, MAN1 y MAN2 en `sta6` al momento de la mezcla. En la figura de la izquierda se muestra a `sta15` enviando un paquete MAR1 en modo multicast, y la figura de la derecha la confirmación MAN1 de `sta6` a `sta15` 135
5.29. Topología 1, captura de paquetes de control MAR1, MAR2, MAN1 y MAN2 en `sta6` al momento de la mezcla. En la figura de la izquierda se muestra a `sta6` enviando un paquete MAR2 a `sta4`, y la figura de la derecha la confirmación MAN2 de `sta4` a `sta6` 135
5.30. Topología 1, captura de paquetes de control MAR1, MAR2, MAN1 y MAN2 en `sta6` al momento de la mezcla. En la figura de la izquierda se muestra a `sta8` enviando un paquete MAR2 a `sta6`, y la figura de la derecha la confirmación MAN2 de `sta6` a `sta8` 136

5.31. Topología 1, ping desde `sta8` al nodo `sta15` después de la mezcla. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*. 138
5.32. Topología 1, ruteo de paquetes ICMPv6 *Echo Request* en `sta6` producto del *ping* de `sta8` a `sta15`, luego de mezcla. . 139
5.33. Topología 1, ruteo de paquetes ICMPv6 *Echo Reply* en `sta6` producto del *ping* de `sta8` a `sta15`, luego de mezcla. 140
5.34. Topología 2, t=0. 141
5.35. Topología 2, t=350s. Mezcla de redes, Red 2 es absorbida por la Red 1. 142
5.36. Topología 2, t=500s. 143
5.37. Topología 2, ping desde `sta1` al nodo `sta2` después de la mezcla. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*. 144
5.38. Topología 2, ping desde `sta7` al nodo `sta2` después de la mezcla. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*. 145
5.39. Topología 2, ping desde `sta7` al nodo `sta3` después de la mezcla. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*. 146

Índice de cuadros

5.1. Tabla de direcciones MAC por Mininet Wifi 115
5.2. Topología 1, tabla de direcciones relativas antes de la fragmentación. 116
5.3. Topología 1, tabla de direcciones relativas de la Red 1 resultante de la fragmentación. 119
5.4. Topología 1, tabla de direcciones relativas de la Red 2 resultante de la fragmentación. 120
5.5. Topología 2, tabla de direcciones relativas antes de la fragmentación. 125
5.6. Topología 2, tabla de direcciones relativas de la Red 1 resultante de la fragmentación. 127
5.7. Topología 2, tabla de direcciones relativas de la Red 2 resultante de la fragmentación. 128
5.8. Topología 1, tabla de direcciones relativas de la Red 1 antes de la mezcla. 133
5.9. Topología 1, tabla de direcciones relativas de la Red 2 antes de la mezcla. 133
5.10. Topología 1, tabla de direcciones relativas de la Red 1 después de la mezcla. 137
5.11. Topología 2, tabla de direcciones relativas de la Red 1 antes de la mezcla. 141
5.12. Topología 2, tabla de direcciones relativas de la Red 2 antes de la mezcla. 142
5.13. Topología 2, tabla de direcciones relativas de la Red 1 después de la mezcla. 144

Capítulo 1

Introducción

En este capítulo se presenta un resumen general de lo que se tratará en este trabajo de Tesis. Aquí se podrá encontrar dos secciones, la primera de ellas discute las motivaciones y objetivos a alcanzar, y luego una reseña de la organización de como se tratarán los temas a lo largo de este trabajo.

1.1. Objetivos

La motivación del trabajo presente, es el estudio de mezcla y fragmentación de redes *ad hoc* usando el protocolo ANTop *(Adjacent Network Topologies)*. Este último fue desarrollado en el artículo [1] y en trabajos de tesis de Alejandro Marcu [2], Gastón Tejia [3] y Matias Campolo [4].

Para lograr cumplir con la meta, se evaluará los trabajos hasta el momento del protocolo ANTop adaptándolos a los sistemas más actuales de su implementación y posteriormente se buscará las distintas herramientas de simulación para reproducir ciertos esquemas que serán necesarios para nuestro estudio. Por último se pretende buscar una solución con respecto a la fragmentación y mezcla de redes *ad hoc* utilizando ANTop.

Una vez definido el esquema más eficiente de nuestra solución, se realizarán pruebas de funcionamiento y se analizarán resultados y conclusiones.

1.2. Organización del documento

A lo largo del trabajo se desarrollan capítulos con distintos temas que integran esta tesis. Para ello se da una reseña de cada uno.

En el capítulo 2 se presentan protocolos de ruteos para redes *ad hoc* en el cual tienen una RFC que los respalda. Luego se describe brevemente el protocolo ANTop, haciendo mención de los trabajos realizados hasta el momento.

El capítulo 3 se presenta la implementación ANTop sobre sistema operativo desarrollada en el trabajo de tesis [4]. Se mencionan las herramientas utilizadas para interactuar con las funciones de red del sistema operativo. Se trabaja sobre el desarrollo para lograr su funcionamiento en sistemas operativos actuales y se mencionan los cambios de la implementación y se justifican las mejora. Finalmente se realiza una prueba física del protocolo para constatar lo desarrollado.

Luego, en el capítulo 4, se discute los problemas en la fragmentación y mezcla de redes ANTop. Se analiza las opciones para mitigar los problemas y se propone una solución. Tanto en la fragmentación y en la mezcla, se explica y justifica todas la decisiones tomadas. Se ilustran los algoritmos de la solución y se desarrollan nuevos concepto que serán incorporados en el protocolo ANTop y en su respectiva implementación en un sistema operativo Linux 4.10.0.

En el capítulo 5, se presentan las pruebas realizadas para corroborar el funcionamiento de los algoritmos desarrollados en el capítulo 4 por medio de herramienta de simulación de redes *ad hoc*.

Por último, en el capítulo 6 se describen las conclusiones de este trabajo de tesis, nombrando los aportes, dificultades encontradas y indicando los temas que no pudieron ser tratados pero abren puertas para futuros trabajos.

Capítulo 2

Estado del arte

En este capítulo se presentan los protocolos de ruteo para redes *ad hoc* más tratados en la bibliografía, y que son respaldados por un RFC. Para cada uno de ellos se explica la teoría de funcionamiento.

Luego se hará un resumen del protocolo ANTop desarrollados en los trabajos [1], [2], [3] y [4]. Si bien los detalles se pueden consultar en las referencias anteriores, se tratará de abarcar todos los temas que fueron desarrollados hasta antes de este trabajo de tesis.

2.1. B.A.T.M.A.N (Better Approach To Mobile Adhoc Networking

B.A.T.M.A.N. [5] tiene su punto crucial en los conocimientos sobre la descentralización de la mejor ruta a través de la red, no permitiendo que un sólo nodo tenga todos los datos. Esta técnica elimina la necesidad de difundir información relativa a los cambios de red a todos los nodos de la misma. Un nodo sólo guarda información sobre la dirección de otro si hubo recibido algún tipo información de éste, y envía sus datos como ser la dirección y sus vecinos en consecuencia. Así se crea una red de inteligencia colectiva, es decir, que ningún de los nodos tiene la información completa de la topología de red.

Para construir su tabla de enrutamiento, almacena la mejor dirección para llegar a cada nodo. Cuando se habla de dirección, se refiere al enlace de intercambio de un salto que conecta a un nodo con otro. Para cada enlace inalámbrico de un salto, se cuentan los paquetes que llegan de cada nodo. El enlace por el que llegan

más paquetes de un nodo en concreto, será el mejor enlace para enviar paquetes a dicho nodo. De esta manera se comparte el conocimiento de la topología de la red, y cada nodo solo almacena la mejor dirección para destinatario.

2.1.1. Descubrimientos de rutas

Las características principales de este protocolo son.

- Batman distingue entre nodos e interfaces. Un nodo puede tener más de una interfaz formando parte de la red.
- El paquete utilizado para informar a los demás sobre la presencia de un nodo se llama OGM (*Originator Message*).
- Se utilizan número de secuencia para diferenciar información nueva y obsoleta.
- Se hace uso de una ventana deslizante para almacenar los números de secuencias, hasta que son considerados fuera del rango (fuera del borde inferior de la ventana). El tamaño de la ventana es configurable mediante la constante `Windows Size`.
- La cantidad de números de secuencias almacenados en la ventana se usa como métrica para determinar la calidad de los enlaces y rutas.

Un nodo BATMAN utiliza paquetes OGM para anunciar su presencia al resto de la red. Esto lo hace en forma periódica. El intervalo entre la emisión de dos OGM's se define en la variable "*Originator Interval*".

El OGM es retransmitido a todos sus vecinos, esto a su vez lo retransmiten a los suyos. El número de OGM's recibidos de un nodo dado a través de cada vecino se usa para calcular la calidad de la ruta. Es decir, si recibimos OGM's de un nodo determinado a través de varios vecinos, aquel vecino por el que hayamos recibido más OGM's de ese nodo será el siguiente salto en la ruta hacia el nodo emisor del OGM. Los OGM's retransmitido por enlaces pobre o lentos, tardaran más en propagarse o se producirán pérdidas o corrupciones de datos, favoreciendo la perdida y correcta propagación de los OGM's que viajan por buenos enlaces.

Por ejemplo, la figura 2.1, G recibe el OGM1 de J a través de H e I. Se produce una pérdida de datos a través de I y el OGM2 sólo le llega por H. Como G ha recibido dos OGM's de J a través de H y solo uno a través de I, G designa a H como siguiente salto hacia J. Para saber si se ha recibido un OGM más de una vez, estos llevan asociado un número de secuencia. Un OGM contiene, por lo menos, la IP del emisor, IP del nodo que retransmite el paquete, un TTL (*time to live*) y un número de secuencia generado por el emisor que permite saber si un

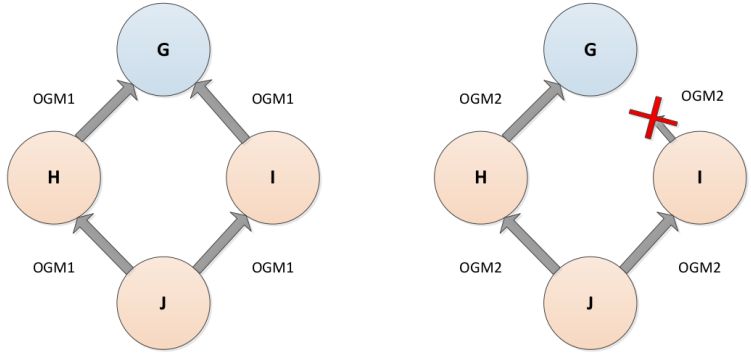

Figura 2.1: Desconexión de enlace entre nodos en una red BATMAN.

paquete ha sido recibido más de una vez. El tiempo de vida (*time to live*) es un concepto usado en redes de computadoras para indicar por cuanto nodos puede pasar un paquete antes de ser descartado por la red o devuelto al origen.

2.1.2. Mantenimientos de rutas

Para registrar y mantener actualizada la información sobre las rutas se utiliza una lista en la que se registran todos los nodos de los que hemos recibido por los menos un OGM. Para ello se guardan, entre otras cosas, su dirección IP, el tiempo transcurrido desde que se recibió el último OGM, su actual número de secuencia y una lista de información de vecinos. En esta lista se mantiene, para cada vecino del nodo, una estructura de ventana deslizante que va almacenando los últimos OGM's recibidos. La ventana deslizante tiene un tamaño determinado por la variable `Windows Size`. En ella se guarda el número de secuencia del último OGM recibido y los "`WindowsSize` -1" números anteriores. Para cada número que este dentro de los límites de las ventanas se refleja si se ha recibido o no el OGM correspondiente. Esto nos sirve para calcular la métrica de la ruta hacia el emisor del OGM a través del vecino asociado a la ventana. Si recibimos un OGM con un número de secuencia dentro de los límites de la ventana, no se actualiza nada. Pero si recibimos un número de secuencia que no está comprendido en la ventana, al actual número de secuencia se le asigna el que hemos recibido y por lo tanto la ventana se mueve. Una ruta hacia un nodo será borrado cuando no se reciba un nuevo OGM de dicho nodo por un tiempo superior a `Windows Size` y al intervalo `Purge Timeout`. Además, si el nodo ofrece un acceso a redes distintas a BATMAN,

deben anexar un mensaje de extensión HNA (*Host Network Annoucement*) para cada red que desee anunciar.

2.1.3. Procesamiento y retransmisión

Cuando un nodo recibe un OGM, lo retransmite como mucho una sola vez solo si ha recibido del vecino designado como el mejor enlace para llegar al emisor del OGM, consiguiendo una dispersión o *Flooding* selectivo. Al recibir un OGM de otro nodo, nos encontramos con dos posibilidades.

- El nodo receptor no tiene el emisor en su tabla de rutas: Se crea una nueva entrada en la tabla de rutas, almacenando la interfaz por la que se recibió y el vecino a través del cual se alcanza el emisor. Esto se hace incluso para los vecinos de un salto.
- El nodo receptor ya tiene el emisor en su tabla de rutas: Si el número de secuencia del OGM ya está contenido en la ventana deslizante asociada al enlace por el que se recibió el OGM, no se actualiza nada. Si el número de secuencia es nuevo (no está contenido en la ventana) se actualizan las ventanas deslizantes que contengan la mayor cantidad de números de secuencia para ese emisor pasara a ser la pasarela hacia dicho nodo.

El nodo retransmitirá el OGM si:

- Ha sido recibido de un vecino de un salto.
- Ha sido recibido por el enlace designado como pasarela hacia el emisor del OGM, y el OGM no es duplicado. UN OGM duplicado es cuando tiene un número de secuencia igual al de un OGM que haya llegado antes.

2.2. Babel

Babel [6] es un protocolo que utiliza técnica basada en vector distancia, inspiradas en protocolos: AODV [7], DSDV [8] y EIGRP [9]. Diseñado para redes inalámbricas *ad hoc*, por lo que es un protocolo robusto en presencia de nodos móviles, impidiendo la formación de bucles sin fin y ofreciendo una rápida convergencia. El protocolo está basado en el algoritmo de *Bellmah-Ford* [10] al que incorpora ciertos refinamientos que previenen la formación de bucles, o al menos, asegura que un bucle desaparecerá después de un cierto tiempo y no volverá aparecer.

2.2.1. Descubrimientos de rutas

A continuación se define las regularidades y conceptos básicos que difieren de Babel a otros protocolos de enrutamiento para redes *ad hoc*.

Nodos e interfaces Al igual que Batman distingue entre nodos e interfaces. Un nodo debe tener más de una interfaz formando parte de la red.

Métrica Babel se basa en la calidad de los enlaces entre nodos para calcular la métrica de una ruta. Se define el costo de un enlace entre A y B como $C(A, B)$ y la métrica de A a un destino S como $D(A)$. Para determinar esta métrica se utilizan los tipos paquetes: *Hello* y IHU (*I Heard You*).

Algoritmo de *Bellman-Ford* [10] Este algoritmo calcula la ruta más corta entre dos nodos en un grafo dirigido y ponderado. Babel se basa en él para el cálculo de rutas. El algoritmo se ejecuta en paralelo para cada nodo, calculando las distancias a los demás nodos de la red. Para la explicación teórica que sigue, se define nodo destino S de referencia para el que se calculara la distancia mínima.

Condición de visibilidad Cuando el enlace entre dos nodos vecinos se rompen, pueden crearse bucles en la actualización de rutas. Para evitar esto, se define la condición de viabilidad como aquella que permite a un nodo rechazar un anuncio de ruta por parte de otro nodo, si dicho anuncio puede crear un bucle sin fin. Las condiciones de viabilidad pueden ser muy diversas, Babel usa la misma que el protocolo EIGRP [9].

Dado un nodo A, se define la distancias de viabilidad de A a S como $FD(A)$, como la mejor métrica que A ha anunciado alguna vez para llegar a S. Es decir, una actualización de ruta $D(B)$ hacia S anunciada por un vecino B de A será viable si $D(B)$ es estrictamente menor que la distancia de viabilidad de A ($FD(A)$). Es decir, que se debe cumplir ($D(B) < FD(A)$).

Número de secuencia Babel utiliza números de secuencia para las rutas, una solución introducida por DSDV [8] y AODV [7]. Además en las materias, cada ruta transporta un número de secuencia, un número entero no decreciente que se propaga inalterado por toda la red. El único que puede incrementar el número de secuencia es el nodo origen. El par (métrica, número de secuencia) se conoce como distancia. De esta forma, podemos saber si la información recibida sobre una ruta es nueva o antigua.

Para cada nodo S destino, un nodo A mantiene los campos de datos. La distancia estimada hacia $D(A)$, y el próximo salto (de entre sus vecinos) para llegar a S, $NH(A)$. Inicialmente $D(A)$ vale infinito y $NH(A)$ esta indefinido. A medida que se van calculando los costos de los enlaces y las métricas de las

rutas, y después de que los nodos hayan anunciado su distancias a S (proceso de convergencia de la red). Tendremos una situación como se muestra en la figura 2.2.

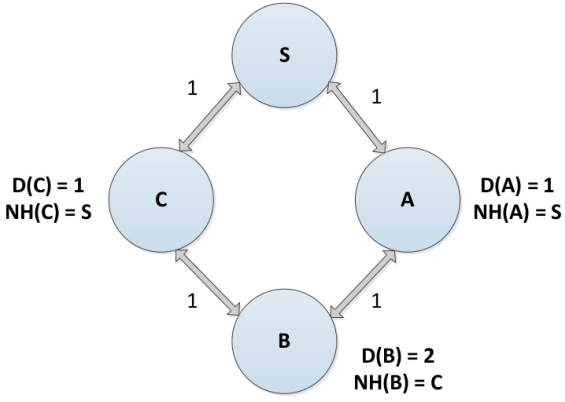

Figura 2.2: Esquema de tabla de rutas de los nodos de una red Babel

Con el fin de advertir su presencia al conjunto de sus vecinos, un nodo envía periódicamente paquete *Hello* por todas sus interfaces, en modo broadcast. La estructura de un paquete *Hello*.

Discriminador Indica que se trata de un paquete.

Longitud Longitud del cuerpo de un mensaje.

Número de secuencia Valor del número de secuencia del paquete para interfaz que lo envía.

Intervalo Límite máximo de tiempo en el que se volverá a enviar otro paquete *Hello*.

Cada nodo mantiene un historial de los paquetes *Hello* recibidos para cada uno de sus vecinos en una estructura de datos de ventana deslizante, al igual que BATMAN. Cuando recibe un paquete *Hello*, el nodo receptor responde con un paquete IHU (*I Heard You*). De esta manera, un nodo A que ha enviado un paquete *Hello*, computa el costo del enlace con un vecino B, que ha respondido con un paquete IHU. El formato del paquete IHU es el siguiente.

Discriminador Indica que se trata de un paquete IHU.

Longitud Longitud del cuerpo del mensaje.

Codificación del campo dirección Indica cual es el formato del campo dirección de destino.

Costo de recepción El costo de recepción, desde el punto de vista del nodo que recibe el IHU. Este valor se calcula a partir del historial de paquete *Hello*.

Intervalo Límite máximo de tiempo en el que volverla a enviar otro paquete IHU.

Dirección Dirección del nodo destino, en el formato especificado en el campo codificación del campo dirección.

2.2.2. Mantenimientos de rutas

Periódicamente, cada nodo B envía una actualización de ruta a todos sus vecinos, con su $D(B)$. Cuando un nodo A recibe su actualización de B, comprueba si B es su siguiente salto para llegar a S. Si es ese el caso y se cumple con la condición de viabilidad, entonces $NH(A) = B$ y $D(A) = C(A, B) + D(B)$. En caso contrario, A compara $C(A, B) + D(B)$ con su valor actual de D(A). Si es menor, es decir, si la ruta es mejor, entonces A asigna $NH(A) = B$ y $D(A) = C(A, B) + D(B)$. Si se rompe un enlace de un nodo A con vecino B, el costo del enlace pasa a valer infinito. Cada nodo mantiene una tabla, en la que se almacena para cada vecino, la siguiente información.

- Interfaz por la que llega el vecino.
- Dirección de la interfaz del vecino.
- Historial de paquetes *Hello* del vecino.
- Costo de transmisión, contenido en el último paquete IHU del vecino.
- Número de secuencia esperado en el próximo paquete *Hello* del vecino.

De esta manera, cada nodo mantiene actualizado el costo de trasmisión con cada uno de sus vecinos. Además, el nodo almacena una tabla de rutas que refleja (entre otras cosas), para cada nodo S, el vecino para llegar a S y la métrica de la ruta. El nodo anuncia en forma periódica y en broadcast dicha tabla a sus vecinos. También hace lo mismo si se produce un cambio significativo en una de sus rutas almacenadas.

2.3. ANTop (Adjacent Network Topoly)

El protocolo ANTop es un protocolo diseñado para redes *ad hoc*, con un esquema descentralizado, ya que todos los nodos conectados tienen igual jerarquía, pudiendo cumplir las mismas funciones, como ser la asignación de direcciones a los nuevos vecinos, resolución de direcciones a partir de un identificador universal o efectuar el ruteo de paquetes.

Las redes *ad hoc* se caracterizan por ser de creación espontánea. En ellas, los nodos gozan de una gran movilidad, con lo cual su estructura cambia permanentemente. Es por esta razón que la dirección dependiente de la ubicación en la red cambiara frecuentemente, volviéndose imprescindible contar con un identificador universal del nodo. Este identificador, sin embargo, no será suficiente para poder rutear un paquete hacia su destino. En respuesta a este requerimiento, surge el concepto de *Distributed Hash Tables* (DHT). Incorporando esta abstracción a la capa de red, ANTop define una técnica de ruteo indirecto, en la cual nodos llamados *Rendez Vous* son responsables de guardar la dirección de red de otros nodos, asociándola con su identificador universal. Como se dijo anteriormente, cualquier nodo que se conecte a la red *ad hoc* podrá actuar como *Rendez Vous*.

2.3.1. Ruteo Indirecto

Cada nodo cuenta con tres direcciones. La primera es la dirección universal U, la cual es conocida por cualquier otro nodo que quiera establecer una comunicación. Al ser totalmente independiente de la capa de red, puede ser cualquier tipo de identificador. La segunda dirección de un nodo, la relativa R, es aquella dependiente de la ubicación en la topología de red y la que nos permitirá rutear los paquetes. Si el nodo se mueve, este identificador cambiará. Finalmente la tercer dirección, la virtual V, se obtiene aplicando una función de *Hash* lineal a la U y será la dirección R del nodo *Rendez Vous* que esta a cargo de proveer el identificador relativo asociado a esa dirección U.

El esquema de ruteo indirecto funciona en la siguiente forma. En la figura 2.3 el nodo origen s, requiere comunicarse con el destino d, pero no sabe su dirección R, tan sólo la U. Por esta razón, s contactara al nodo *Rendez Vous* T quien conoce la dirección R de d, ya que este último se registro con T al conectarse a la red. A continuación T envía un mensaje a s, que contiene la dirección R de d, de modo que los paquetes desde s a d ya pueden ser ruteados.

2.3.2. Direccionamiento en un Hipercubo

Una vez que se tiene la dirección R del destino, se debe rutear el paquete hacia la misma. Aquí se presentan, dos esquemas posibles, ruteo proactivo y ruteo

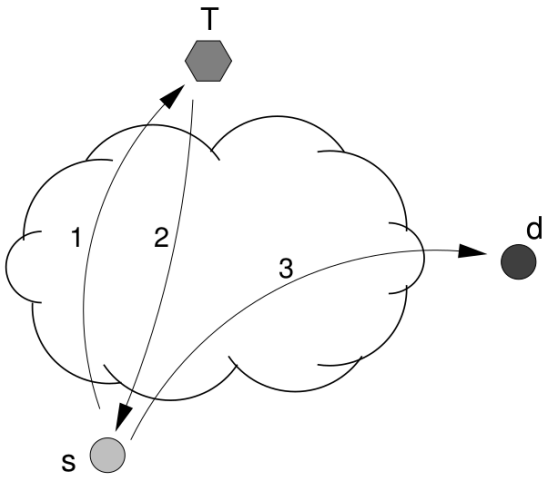

Figura 2.3: Modelo de ruteo indirecto

reactivo. En el primero de ellos las tablas de ruteo se construyen y se mantienen actualizadas constantemente, asegurando una ruta a cada nodo de la red, en todo momento. En el segundo esquema las rutas se generan bajo demanda y se mantienen durante un determinado lapso de tiempo. Dada la naturaleza de las redes *ad hoc*, donde la posición de cada nodo puede variar con frecuencia, el segundo esquema (reactivo) es el que mejor se adapta. Para redes cuya topología cambia frecuentemente, las rutas obtenidas proactivamente pueden ser obsoletas al momento de tener que rutear un paquete hacia un destino en particular, haciendo que los recursos insumidos para conseguirlas hayan sido desperdiciados. El ruteo proactivo se utiliza para aquellos esquemas de redes cuya características más notable es la baja movilidad física de los integrantes, por ejemplo, las redes cableadas.

ANTop, define un esquema de direccionamiento basado en hipercubos. Estos son una generalización de un cubo de tres dimensiones, a un número d. Cada nodo tiene una coordenada de valor 0 o 1 en cada dimensión. Esto implica que el número total de nodos 2^d. Cada nodo esta conectado a aquellos cuyas coordenadas varían sólo en una dimensión. En la figura 2.4 se muestra un hipercubo de dimensión 4. ANTop utiliza las coordenadas de cada nodo en el hipercubo, como la dirección R del mismo.

En la figura 2.4, los vértices negros representan nodos que físicamente están

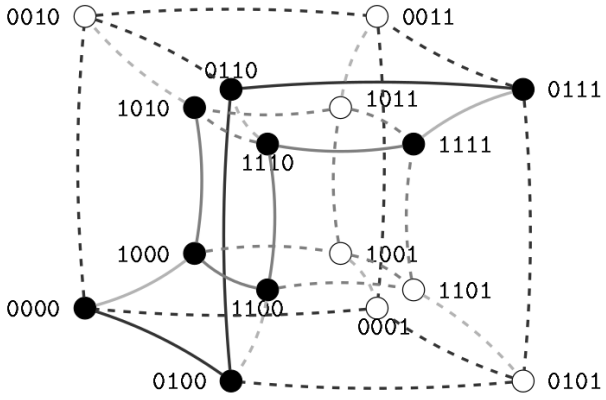

Figura 2.4: Direccionamiento basado en hipercubo

presentes en la red, mientras que los blancos, representan direcciones que aún no fueron asignadas. Se dice en este caso que el hipercubo es incompleto, ya que no todas las direcciones están siendo utilizadas por nodos conectados a la red. Por el contrario, algunas de ellas están disponibles para ser asignadas a nuevos nodos que deseen conectarse a la red. Cuando todas hayan sido asignadas, la red no admitirá la conexión de nuevos nodos.

En este esquema, es fácil ver que la distancia entre dos nodos, definida como la cantidad de dimensiones con coordenadas distintas, se puede obtener aplicando una función XOR a las direcciones. Por ejemplo la distancia entre los nodos 0100 y 0110 es 2. La distancia se puede pensar como la cantidad de nodos mínima por el que un mensaje de un nodo debe pasar para llegar a un destinatario.

Cuando un nuevo nodo esta en el radio de cobertura de alguno de los ya conectados a la red, hará un pedido en modo broadcast para que le sea asignada una dirección de hipercubo. Este pedido será escuchado por los posibles vecinos del nuevo nodo los cuales le ofrecerán una dirección que difiere en solo una coordenada (un bit) respecto de la propia.

Cada nodo de la red contará con, además de una dirección R, una máscara que permitirá definir un espacio de direcciones que ese nodo administra. Tomando como ejemplo un hipercubo de dimensión $d = 4$, el nodo A con dirección y másca-

ra 1000/3, estará cargado de administrar las direcciones 1000 y 1001. Cuando un nuevo nodo desee conectarse a la red, A podrá cederle la dirección 1001/4. En caso de que esto suceda, es decir que efectivamente el nuevo nodo acepte esta propuesta como su dirección R, A aumentará su máscara a un valor de 4. Es en esta forma que A delega parte de su espacio de direcciones con su correspondiente administración a un nuevo nodo que se conecte a al red.

El direccionamiento basado en un hipercubo, presenta un aporte que simplifica el proceso de ruteo. Esto puede verse claramente si se considera el caso de un hipercubo completo, es decir aquel en el que todas las direcciones están siendo usadas por nodos. Para ilustrar esto último se puede considerar como ejemplo el caso en que el nodo 0001, quiere enviar un mensaje al nodo 1101 en un hipercubo de $d = 4$. Es posible ver la distancia entre ambos, que resulta ser de 2. Es decir que 2, es la cantidad de bits que difieren entre las direcciones de dichos nodos. Al existir todos los nodos en la red, es posible saber *a priori* que el camino $0001 \rightarrow 1001 \rightarrow 1101$ será válido para rutear el paquete, y la cantidad de saltos que el mismo transitará es igual a la distancia entre los nodos. Como puede verse, el nodo origen sin disponer aún de ninguna ruta, ya sabe un posible camino al destino, con distancia mínima.

En un hipercubo incompleto, en cambio, algunas direcciones no estarán siendo efectivamente utilizadas por nodos, con lo cual no es posible asegurar un camino a priori. Sin embargo, es posible que algunos caminos con distancia mínima si existan y por lo tanto vale la pena intentar rutear el paquete en esa dirección. Esto mismo es lo que hace ANTop, ya que bajo su esquema de ruteo reactivo, los paquetes se envían en primer instancia a aquel vecino cuya distancia al destino sea menor, al tiempo que este último repetirá el procedimiento para lograr rutear el paquete. Si durante esta exploración, se llega a un callejón sin salida, el paquete será devuelto por donde vino para que el nodo que originalmente envió el paquete en esa dirección pueda enviarlo a su próximo vecino, eligiéndolo por el mismo criterio de distancia al destino.

2.3.3. Conexión de nodos

En el protocolo ANTop no hay ningún servidor central ni jerarquías, por lo tanto todos los nodos tienen la misma funcionalidad. Lo primero que necesita un nodo para conectarse es obtener una dirección primaria, ya que de otra forma no podría ser distinguido de otros nodos. Una vez obtenida, puede comenzar a enviar y recibir paquetes de otros nodos, así como a ceder su espacio de direcciones o adquirir direcciones secundarias para mejorar la conectividad.

Establecimiento de la conexión Cuando un nodo desea conectarse a la red primero tiene que obtener una dirección primaria. Para ello, debe notificar a sus vecinos para que éstos le ofrezcan direcciones primarias. Esto lo hace enviando un paquete de tipo PAR (*Primary Address Request*) en modo broadcast, para que todos sus vecinos lo reciban y respondan a su vez con un paquete PAP (*Primary Address Proposal*), sugiriendo direcciones primarias.

El primer nodo que se conecta a la red, no recibirá respuesta a cambio del envío del paquete PAR. Por esta razón el mismo asignará la primera dirección del hipercubo. en verdad, ANTop prevé una primer dirección R cuyos bits tiene un valor cero en su totalidad. El segundo nodo que quiera conectarse a la red, enviará un paquete PAR, en modo multicast, y cuando el primer nodo participante de la red, recibe dicho pedido, debe responder con un paquete PAN (*Primary Address Notification*), cuyo destino sera la dirección del segundo nodo, es decir que no sera enviado en modo broadcast.

Luego de enviar el PAR, el nodo 2 espera un determinado tiempo para colectar las ofertas de dirección R y vencido dicho intervalo, elige la opción con mayor máscara. Es decir, el espacio de direcciones más grande. En este caso, solo recibirá una respuesta del nodo 1.

Una vez que el segundo nodo acepta la dirección, lo anuncia enviando un paquete PAN al nodo 1, notificando que acepta su propuesta y que éste incremente su máscara a un valor de uno. Para ejemplificarlo, se propone un esquema de espacio de direcciones de ocho bits.

Nodo 1
Dirección 00000000 (primer dirección de hipercubo)
Máscara 1

Nodo 2
Dirección 10000000
Máscara 1

Cuando un tercer nodo se conecta a la red, si el mismo está en el alcance de los dos primeros, recibirá dos opciones de dirección R.

Dirección 01000000
Máscara 2

y,

Dirección 11000000
Máscara 2

La dirección que ofrece el nodo uno surge de asignar un valor de uno al bit más significativo del espacio de hosts ANTop. Es decir, dado la dirección R del nodo 1 es 00000000, con máscara 1, con lo cual ofrece una dirección que solo difiere en un bit, de modo que este byte vale 01000000, con máscara 2. La dirección R es 10000000, con máscara 1, con lo cual ofrece una dirección R para la cual cambia este byte. El mismo adquiere un valor de 11000000, con máscara 2. En este caso las dos opciones tienen la misma máscara ANTop, con lo cual el nodo 3 elegirá alguna de ellas indistintamente.

Cabe destacar que el protocolo ANTop presenta un mecanismo para lidiar con la perdida de paquetes de control. En lo que se refiere a a asignación de direcciones, el mecanismo es el siguiente.

Del algoritmo 1 el nodo que se quiere conectar a la red envía un paquete PAR en modo broadcast. Aquí se dispara un timer, y durante la vida del mismo se colectan respuestas, en forma de paquetes PAP. Estos paquetes UDP, tiene en su payload una estructura cuyos campos son, entre otros, la dirección propuesta, la máscara, el tipo de mensaje ANTop. Cada una de estas estructuras recibidas se guarda en el Espacio usuario en un vector de respuestas. Una vez vencido el timer, se chequean la longitud del vector. Si se recibieron respuestas, se elige la de mayor máscara, como ya se dijo, pero si la longitud del vector es nula, entonces puede haberse perdido el PAR, o sus respuestas, es decir los paquetes PAP. En este caso se envía nuevamente un PAR y se espera el mismo intervalo de tiempo por respuestas. Este proceso se repite un número configurable de veces, y si no se recibió respuestas entonces se considera que es el primer nodo de la red y se asigna el mismo una dirección de hipercubo.

En el caso de que se reciban respuestas, se envía un paquete PAN, informando la dirección elegida. Este paquete se envía en modo broadcast para que todos los nodos que propusieron una direcciones sepan que una fue elegida. El nodo que ofreció la dirección elegida cederá ese espacio de direcciones, y se enviara un paquete de tipo PANC (*Primary Address Notification Confirmation*)

Si el paquete PAN que envía el nodo que quiere conectarse a la red, se perdiese, entonces el nodo que ofreció la dirección elegida no enviará el PANC, y por lo tanto el proceso de conexión del nuevo nodo queda incompleto. Frente a este escenario, el nodo entrante vuelve a iniciar el ciclo de envío de paquetes PAR.

Algorithm 1 Establecimiento de dirección primaria

1: $i = N\_PAR$;
2: **repeat**
3: Enviar paquete PAR;
4: Esperar un tiempo T_PAP, almacenando los paquetes PAP recibidos;
5: $i := i - 1$;
6: **if** *se recibió al menos un paquete PAP con dirección disponible* **then**
7: ir a 10;
8: **end if**
9: **until** $i == 0$
10: **if** *se recibió al menos un paquete PAP con dirección disponible* **then**
11: Elegir entre las direcciones propuestas una con la menor máscara;
12: Enviar un paquete PAN en broadcast notificando la dirección elegida;
13: Esperar a que llegue un paquete PANC o que transcurra un tiempo
14: T_PANC;
15: **if** *se recibió un paquete PANC* **then**
16: La dirección primaria elegida ya puede ser utilizada. Fin del algoritmo;
17: **end if**
18: **else**
19: **if** *se recibió un paquete PAR indicando que no hay direcciones disponibles* **then**
20: No se puede conectar. Fin del algoritmo;
21: **end if**
22: Utilizar la dirección compuesta por todos ceros;
23: **end if**

Dirección estable En el momento que un nodo adquiera una direccione primaria estable, puede que reciba un pedido de dirección primaria (paquete PAR) de algún nodo que se quiere conectar. En el caso de que tenga direcciones disponibles, debe de responder ofreciéndole una dirección (paquete PAP), o diciéndole que se le agotó el espacio de direcciones ene l caso de que no tuviera para ofrecer. Una vez enviado el PAP se espera a recibir un PAN que dará a conocer la dirección elegida. Si este paquete no se recibió en un lapso de tiempo, se debe volver al modo de escucha de pedidos para restablecer el funcionamiento normal, ya que posiblemente algún paquete se perdió. Cuando se recibe un paquete PAN, este puede estar indicando que la dirección elegida es la del nodo o que no es. En el primer caso, el nodo debe ceder el espacio de direcciones, cambiando su máscara y notificando a otros procesos que pudieran estar interesados en conocer este hecho (como el algoritmo de ruteo o la aplicación de *Rendez Vous*) y responder con un paquete PANC al remitente. Tanto si la dirección escogida es la del nodo como si no lo es, se vuelve al modo de escucha de paquetes. El algoritmo 2 resume el ofrecimiento de direcciones primarias, siendo `T_PAN` el tiempo máximo que se espera respuesta.

Algorithm 2 Ofrecimiento de direcciones primarias

```
1: Esperar un paquete PAR;
2: if hay direcciones para ceder then
3:     Enviar un paquete PAP ofreciendo una dirección primaria;
4:     Esperar a recibir un paquete PAN o que transcurra un tiempo T_PAN;
5:     if se recibió un paquete PAN y la dirección elegida es la del nodo then
6:         Ceder el espacio de direcciones;
7:     end if
8:     Enviar un paquete PAP indicando que no se tienen direcciones;
9: end if
```

Una vez que el proceso de conexión de un nodo a finalizado, el mismo tiene una dirección R en el hipercubo y será vecino de todos aquellos nodos que cumplan con dos condiciones:

- Estar adentro del alcance de la placa inalámbrica
- Diferencia de un bit en las direcciones R.

Al momento de rutear un paquete, cada nodo debe tener conocimiento de cuales de sus vecinos están activos en la red, de lo contrario, se perderán paquetes por ser reenviados a nodos que ya no forman parte de la red. Para lograr esto, cada nodo envía periódicamente un paquete de control del tipo *Heart Beat* en modo broadcast. Esto último se logra con un timer `T_HB` como se muestra en el algoritmo

3 al cual se le asocia una función de *callback* que una vez que el mismo se vence, vuelve a dispararlo con el mismo tiempo de vida.

Algorithm 3 Envío de Heart Beats

1: Enviar un paquete HB en broadcast;
2: Esperar un tiempo T_HB;
3: Ir a 1;

Cuando un nodo recibe un *Heart Beat*, chequean en su tabla de vecinos alojada en el Espacio Usuario si ya tiene registrado el origen como un vecino. En caso negativo, agrega una entrada al final de la tabla (generada por una lista de registros), en caso positivo se marca mediante una bandera que un nuevo *Heart Beat* ha sido recibido para ese nodo.

Periódicamente, se recorre la tabla de vecinos y se chequean cuales vecinos no han enviado *Heart Beat*. En esos casos, se da un valor de uno a una variable que representa la cantidad de veces que no se ha recibido este aviso de actividad. Se repite este mecanismo, y se incrementa la variable si corresponde. Cuando llega un *Heart Beat* de ese vecino, la variable toma un valor nulo nuevamente. Cuando la misma llega un determinado, configurable, se considera que el nodo ya no está en la red.

Algorithm 4 Procesamiento de Heart Beats

1: Esperar paquetes HB durante un tiempo T_LHB;
2: Marcar como activos a los vecinos que enviaron paquetes HB;
3: **if** hay un vecino que permanece sigue inactivo luego de N_INACT interacciones **then**
4: Recuperar el espacio de direcciones si el vecino era un hijo;
5: Eliminarlo de la tabla de vecinos;
6: **end if**

El algoritmo 4 se ejecuta indefinidamente. El valor `T_LHB` es el tiempo durante el que se esperan paquetes HB de los vecinos. El `N_INACT` es la cantidad de veces consecutivas que se tolera no recibir un HB de dar al nodo por desaparecido.

Establecimiento de dirección secundaria En cuanto al ofrecimiento de direcciones secundarias, se debe separar de la asignación de direcciones primarias para no demorar la conexión más de lo necesario. Se podría eventualmente enviar

un pedido de direcciones secundarias y responderlo; sin embargo, dado que una vez conectados los nodos se envían periódicamente paquetes HB, al recibirlo se puede comprobar si hay alguna dirección secundaria para asignarse y obtener conectividad con el vecino que envió el paquete. Si es posible asignar una dirección, entonces se envía un paquete SAP (*Secondary Address Proposal*), y se espera un paquete SAN (*Secondary Address Notification*) en respuesta.

El paquete SAP contiene la dirección secundaria con el cual el nodo que lo reciba, tendrá conectividad con el emisor del paquete. En el caso que el vecino responda con el paquete SAN, el emisor del paquete SAP se auto asigna la dirección secundaria creando un enlace secundario. Luego de haber enviado un paquete SAP como propuesta de dirección secundaria, se espera un tiempo `T_SAN` en el cual no se enviará paquetes SAP como propuesta a otros nodos, si no que se espera la confirmación SAN del vecino con el que se está negociando un enlace secundario. En el algoritmo 5 detalla el procesamiento de HB con propuesta secundaria.

Algorithm 5 Procesamiento de HB con propuesta de direcciones secundarias

```
1: Esperar paquetes HB durante un tiempo T_LHB;
2: Marcar como activos a los vecinos que enviaron paquetes HB;
3: if hay un vecino que permanece sigue inactivo luego de N_INACT interacciones
   then
4:     Recuperar el espacio de direcciones si el vecino era un hijo;
5:     Eliminarlo de la tabla de vecinos;
6: end if
7: if hay algún nodo al que se puede establecer enlace secundario then
8:     Enviarle un paquete SAP con la dirección secundaria;
9:     Esperar un paquete SAN o que transcurra un tiempo T_SAN;
10:    if se recibió un paquete SAN then
11:        Asignar la dirección secundaria propuesta;
12:        Marcar al nodo que emitió el paquete SAN como el vecino enlazado por
   medio de dirección secundaria;
13:    end if
14: end if
```

El nodo que recibe el paquete SAP, a parte de reenviar el mensaje de confirmación SAN, agrega como vecino la dirección secundaria propuesta y lo marca con conectividad por enlace secundario como se muestra en el algoritmo 6.

Algorithm 6 Procesamiento de SAP

1: Agregar la dirección propuesta en la tabla de vecinos;
2: Marcar al vecino agregado como enlace secundario;
3: Enviar un paquete SAN al emisor del paquete SAP;

2.3.4. Desconexión de nodos

Cuando un nodo se desconecta, sus vecinos dejarán de recibir sus *Hear Beat*. Al cabo de un determinado tiempo, ellos consideran que el mismo abandonó la red, y lo eliminara de su tabla de vecinos. Quien sea el padre del nodo, recuperará el espacio de direcciones cedido cuando el mismo se conectó, de modo tal que puede ser utilizado nuevamente. Cuando un nuevo nodo intenta conectarse dicho espacio es el primero en ofrecerse. Una vez que todos los recuperados fueron cedidos, se continua con la sesión del resto de las direcciones.

2.3.5. Ruteo

Cuando un paquete llega al nodo, el mismo es enviado al Espacio Usuario. En el caso de que el destino sea el nodo local, entonces no se ejecuta ninguna acción y se permite que el mismo continué su viaje a lo largo de la pila de protocolos de red. Si, por ejemplo, en el caso recién mencionado, el paquete es tráfico de control ANTop, entonces el mismo seguirá su curso normal, y será recibido por la función de lectura asociada al socket UDP que escucha en el puerto ANTop.

Si por el contrario, el paquete tiene como destino una dirección R distinta a la propia, entonces sucede que este nodo se trata de un eslabón en la cadena de ruteo entre origen y destino. En este caso se deberá chequear en la tabla de ruteo del SO si se tiene una entrada con el destino apropiado. De no ser así, se deberá buscar una. Esto también es valido cuando el tráfico se genera localmente y debe ser enviado a un destino particular.

Al obtener una copia de la tabla de ruteo en el Espacio Usuario, se recorre la misma con el fin de determinar si existe una ruta para llegar al destino del paquete. Si este no fuera el caso se dispara el mecanismo de búsqueda de ruta.

A grandes rasgos, el mecanismo de búsqueda, se trata de una exploración de los vecinos. Básicamente se envía el paquete hacia el que tiene menor distancia al destino y, aunque no se tiene seguridad de que exista una ruta efectiva a través de ese vecino, la misma es instalada con el primer intento de exploración.

Luego de agregada la tabla, se configura un timer que invoca el borrado de la misma una vez que este expire. De esta forma se asegura la búsqueda de una nueva ruta luego de un tiempo, y por ende no rutear paquetes por caminos que ya no son validas.

Una de las situaciones usuales que surgirá con el esquema de ruteo de AN-Top es que al recibir un paquete que deba ser ruteado hacia un cierto destino, no se tengan más vecinos disponibles que en el último salto por el que pasó. En esa situación el mismo será devuelto para que el nodo que lo envió continué la búsqueda a través de otros vecinos. Con el fin de llevar un registro de los vecinos que el paquete ya visitó, se utiliza un *visited bitmap* el cual se compone de un vector de banderas por cada ruta instalada en la tabla. La cantidad de banderas es igual para todos los vectores . cada una de ellas hace referencia a un vecino disponible, y se implementa como un bit que tendrá el valor de uno si ese vecino ya fue visitado al intentar rutear un paquete al destino o cero en el caso contrario.

Se puede resumir el algoritmo de ruteo en dos algoritmos, el 7 y 8. El algoritmo 7, el primer parámetro, M es el que debe rutear, que se puede escribir también como *M(x,z)*, siendo x el nodo de origen y z el de destino. El segundo parámetro, w, es el vecino por el cual se recibió el paquete. Y el segundo algoritmo es el 8, tomando parámetros del primer algoritmo.

En los pasos 2 a 5, se verifica si el paquete está marcado como devuelto, lo cual ocurre si no se encontró ninguna ruta yendo por ese vecino. Por ello, se llama a la función 8, que busca enviar al paquete por otra ruta. Los pasos 6 a 9 detectan bucles. Para ello, buscan si un paquete con el mismo origen y destino pasó previamente con una distancia al origen menor, en cuyo caso el paquete es enviado de vuelta por el vecino del que provino. El paso 10 se agrega la entrada inversa si no estaba; es decir, se agrega la ruta para ir al nodo de origen.

El paso 11 busca la entrada directa, que indica por que vecino se debe ir hacia el destino. En los pasos 12 a 16, si existe esta entrada, se manda el paquete por el vecino especificado. Una vez que la ruta se encuentra correctamente establecida, éste es el flujo de programa que siguen los paquetes subsecuentes. En cambio, si no se encuentra la ruta, se utiliza la función 8 para que mande el paquete por el vecino más conveniente.

En el paso 17 se asocia el origen con el destino y la distancia al origen para poder detectar bucles.

El algoritmo 8 se utiliza para determinar por que vecino se debe enviar el paquete la primera vez que llega un paquete con ese destino, o cuando uno es devuelto porque no se encontró ruta. En los pasos 1 a 4, si no hay *Visited Bitmap* (lo cual ocurre cuando se envía un paquete a un destino por primera vez), este es creado y se inicializa el timeout. En los pasos 5 a 10, si no se encontró ruta por

Algorithm 7 route(M,w)

1: v recibe un mensaje *M(x,z)* del vecino w;
2: **if** *M está marcado como devuelto* **then**
3: *sendToNextNeighbor(M,w)*;
4: Fin del algoritmo;
5: **end if**
6: **if** se encuentra en la tabla de pares a (x, z) con distancia al origen menor que M **then**
7: Devolver el paquete por donde vino;
8: Fin del algoritmo;
9: **end if**
10: Buscar en la tabla de ruteo $v \rightarrow x : w, n$ (entrada inversa y agregarla si no se encuentra;
11: Buscar en la tabla de ruteo $v \rightarrow z : y, m$;
12: **if** *se encontraron entradas* **then**
13: Enviar el paquete por la entrada con mínima distancia;
14: **else**
15: *sendToNextNeighbor(M,w)*
16: **end if**
17: Agregar en la tabla de pares a (x, z) si no estaba anteriormente;

Algorithm 8 sendToNextNeighbor(M,w)

1: **if** *no existe el Visited Bitmap de vecinos para* $v \rightarrow z$ **then**
2: Crear el *Visited Bitmap*, y marcar el vecino w como visitado;
3: Iniciar Bitmap Timer (entrada) eliminará solo el *Visited Bitmap* de la entrada;
4: **end if**
5: **if** *están todo los bits marcados* **then**
6: Marcar en la entrada de la tabla de ruteo que no hay ruta para z;
7: Buscar en la tabla de ruteo $v \rightarrow x : y, n$;
8: Devolver el paquete a y;
9: Fin del algoritmo;
10: **if** *hay* $NBP + 1$ *bits marcados* **then**
11: NextHop = primer bit padre;
12: **else**
13: **if** *el primer bit esta marcado* **then**
14: NextHop = siguiente bit no marcado;
15: **end if**
16: **if** *el primer bit no esta marcado* **then**
17: NextHop = nit no marcado correspondiente al vecino con mínima distancia al destino en el *Bitmap*;
18: **end if**
19: **end if**
20: Iniciar Route Timer (entrada) eliminará toda la entrada en la tabla de ruteo;
21: Marcar NextHop en el *Visited Bitmap* y enviar el paquete por ese nodo;
22: **end if**

ninguno de los vecinos, el paquete es devuelto por el nodo desde donde provino. Los pasos 11 a 19 determinan cuál es el vecino a visitar. El paso 20 inicializa el timeout para que la entrada se borre transcurrido un lapso de tiempo. En el último paso, se marca el vecino por el que se está enviando el paquete como visitado y se envía.

2.3.6. Resolución de nombres, servicio *Rendez Vous*

Una de las características más distinguidas de las redes *ad hoc* es que los nodos que en ellas participan gozan de gran movilidad. Es por esta razón que la dirección dependiente de la ubicación en la red, cambiara frecuentemente. Esto último se vuelve incomodo cuando se intenta establecer una comunicación con el nodo, ya que se debe hacer un seguimiento de esta dirección. Para independizar la identificación del nodo, de su dirección R, se utiliza la dirección U. La misma será una cadena de caracteres, por ejemplo *nodo_antop_pablo*.

Cada nodo que ingresa a la red, deberá registrarse en un servidor *Rendez Vous*. La información de resolución de nombres, se encuentra dispersa a lo largo de toda la red, es decir, que no hay una entidad centralizada que lo administre.

Cada nodo utilizará función *Hash* aplicada a su dirección U (la cual es asignada por el usuario del nodo), y como resultado obtendrá una serie de direcciones que serán los servidores *Rendez Vous* en los cuales deberá registrarse. Con este esquema se obtiene redundancia de la información en caso de que un nodo se desconecte en forma imprevista, ya que la información de resolución de nombres en el contenida, se encuentra replicada en un número de servidores *Rendez Vous*, que dependerá de la dimensión de la red.

Para concretar el registro de la dirección en el servidor *Rendez Vous*, el nodo entrante envía un paquete de control ANTop, en la misma forma que un PAR, solo que en lugar de ser un multicast a todos los nodos, se envía a un grupo de nodos en particular. Este registro se envía periódicamente, con lo cual la información se mantiene actualizada. Para estos registros también se asocia un timer que luego de vencido borra la entrada.

Cuando el usuario de una PC que participa en una red ANTop, desea entablar una conversación a nivel de aplicación de otro nodo, debe hacer referencia a su IP, o bien a su nombre de host.

El registro de direcciones podría implementarse a nivel de capa de red; pero haciéndolo en la capa de aplicación se tiene la ventaja adicional de poder utilizar la capa de transporte de abajo, la cual permitiría tener una comunicación más confiable si se usa TCP. Como desventaja, las otras aplicaciones en el nodo deberán comunicarse primero con esta aplicación para resolver la dirección de red. En realidad, se utilizan dos aplicaciones, una aplicación cliente y una servidor. La primera se encarga de hacer pedidos de direcciones y de mantenerlos en memoria

un cierto tiempo, mientras que la segunda resuelve los pedidos, registra y desregistra nuevos nodos.

Cliente *Rendez Vous* Si el cliente de *Rendez Vous* solamente realizará la resolución de direcciones, entonces las aplicaciones se tendrían que encargar de mandar el paquete una vez recibida la resolución. El principal inconveniente es que un pedido de resolución puede tomar un tiempo relativamente largo, por lo que muy probablemente la aplicación no quiera quedarse bloqueada esperando que llegue una respuesta para poder continuar, aunque por supuesto que hasta que no reciba respuesta no puede enviar el paquete. Además, debería lidiar con una caché de direcciones para no volver a efectuar la consulta cada vez que sea necesario. Por ello, es conveniente que el cliente *Rendez Vous* se encargue de todas estas gestiones, dándole a las otras aplicaciones un medio más sencillo para que puedan enviar paquetes. Entonces, la interfaz que provee esta aplicación es la función $send(U, M)$, siendo U la dirección universal a la que se manda el paquete, y M el paquete a enviar. Esta función se muestra en el algoritmo 9.

Algorithm 9 send(U,M)

1: Buscar el par (U, V) en el caché;
2: **if** *se encontró* (U, V) **then**
3: Enviar M a la dirección V;
4: **else**
5: $E = H(U)$;
6: Enviar un paquete *Address Lookup* a E pasándole la dirección U;
7: Agregar (U, M) en el vector *WaitQueue*;
8: **end if**

Si la dirección U se encuentra en la caché, se envía el paquete a la dirección de red correspondiente. Si no se encuentra, se manda el pedido de resolución y se agregan la dirección U y el mensaje a un vector (*WaitQueue*) para que cuando se resuelva esta dirección, se envíe el paquete.

Cuando la aplicación recibe un paquete, eventualmente se resolvió una dirección, por lo que debe agregar este valor a la caché y revisar el vector *Wait Queue* para enviar los paquetes cuya dirección U se acaba de resolver. Esto se muestra en el algoritmo 10.

Las aplicaciones de hipercubo deberán localizar a la aplicación *Client Rendez Vous* y utilizar la función $send(U, M)$ para enviar los paquetes, delegando en ella toda la responsabilidad de resolución de direcciones.

Algorithm 10 receive(M)

```
1: if M es un paquete de tipo Address Solve then
2:     if el flag solved de M vale verdadero then
3:         Agregar en el caché (U, V);
4:         for cada par (Um, M) en WaitQueue do
5:             if Um == U then
6:                 enviar M a la dirección V;
7:             end if
8:         end for
9:     else
10:        Notificar mediante un mensaje interno que U no se puede resolver;
11:    end if
12: end if
```

Servidor *Rendez Vous* El servidor de *Rendez Vous* debe cumplir con las siguientes funciones.

- resolver pedidos de resolución.
- tomar pedidos de registración y des-registración de otros nodos y reflejarlos en la tabla de resolución.
- recibir tablas de resolución de otros nodos e incorporarlas.
- registrar al nodo en el *Rendez Vous* cuando este se conecta.
- des-registrar al nodo en el *Rendez Vous* cuando se desconecta y delegar su tabla de resolución.
- enviar un fragmento de la tabla de resolución cuando se cede una parte del espacio de direcciones.

Los tres primeros items ocurren siempre cuando un mensaje llega a esta aplicación, es por ello que estas operaciones pueden ser efectuadas en la función receive de la aplicación, como se muestra en el algoritmo 11 .

Este algoritmo realiza distintas operaciones según el tipo de paquete recibido. Ninguna de estas operaciones contiene bucles, saltos o esperas, por lo que el algoritmo debe finalizar. Las tres últimas funciones que debe realizar el servidor *Rendez Vous* ocurren cuando el nodo se conecta y desconecta de la red o cuando se ceden direcciones, por lo que pueden ser realizados cuando se reciben mensajes internos. El algoritmo 12 muestra estas operaciones.

Este algoritmo realiza distintas operaciones según el tipo de mensaje recibido. Para el tipo *Address-Given* se realiza un bucle, pero dado que éste recorre una

Algorithm 11 receive(M)

```
1: if M es un paquete de tipo Register then
2:     agregar (U, V) en la tabla de resolución;
3: else
4:     if M es un paquete de tipo Deregister then
5:         Eliminar (U, V) de la tabla de resolución;
6:     else
7:         if M es un paquete de tipo Address Solve then
8:             Buscar U en la tabla de resolución;
9:         end if
10:    end if
11:    if se encontró U en la tabla de resolución then
12:        Responder con un paquete Address Lookup con la dirección de red correspondiente;
13:    else
14:        Responder con un paquete Address Lookup con Solved = false;
15:    end if
16:    if M es un paquete de tipo Lookup Table then
17:        Agregar la tabla recibida a la tabla de resolución;
18:        Enviar un paquete Lookup Table Received al remitente, identificando las entradas recibidas;
19:    else
20:        if M es un paquete de tipo Lookup Table Received then
21:            Eliminar de la tabla de resolución las entradas indicadas;
22:        end if
23:        if el nodo esta en proceso de desconexión then
24:            mandar un mensaje interno ReadyForDisc;
25:        end if
26:    end if
27: end if
```

Algorithm 12 onMessageReceived(msg)

```
1: E = H(U);
2: if msg es de tipo Connected then
3:     Enviar un paquete Register a E con (U, V);
4: else
5:     if msg es de tipo WillDisconnect then
6:         Mandar un mensaje interno WaitMe;
7:         Enviar un paquete Lookup Table con la tabla de resolución al padre;
8:         Enviar un paquete Deregister a E con (U, V);
9:     end if
10:    if msg es de tipo AddressGiven then
11:        for cada elemento (Ur, Vr) de la tabla de resolución do
12:            if H(Ur) esta en el espacio de direcciones cedido then
13:                agregar (Ur, Vr) en el vector SendTable;
14:            end if
15:        end for
16:        Enviar un paquete Lookup Table con todos los elementos de SendTable
       al nodo que se le cedió el espacio de direcciones;
17:    end if
18: end if
```

tabla, finaliza en tantos ciclos como elementos tenga la tabla. Las otras operaciones no tienen bucles, saltos o esperas, asegurando la finalización del algoritmo.

2.3.7. Ruteo de paquetes *Rendez Vous*

En el caso de ese tipo de paquetes, el servidor *Rendez Vous* destino se determina mediante una función de *Hash*. Por esta razón es probable que esa dirección no haya sido asignada a ningún nodo aún. Será el dueño del espacio de direcciones de dicho destino el que deba procesar el paquete. En ese caso, se chequea si el destino se encuentra en el espacio de direcciones que administra el nodo.

Para el ruteo de paquetes de *Rendez Vous*, al calcular la distancia se contempla la máscara del nodo. Por ejemplo, si el destino es 110111 y un nodo tiene dirección 010000/2, entonces la distancia utilizada en el ruteo de datos es 4, ya que hay 4 bits distintos. En cambio, en el ruteo de *Rendez Vous*, para calcular la distancia se toman solamente los dos primeros bits de la dirección (dado por la máscara), y por lo tanto la distancia es 1.

2.3.8. Paquetes de control ANTop

En la implementación de ANTop se define un paquete de control genérico, cuyos campos tomarán distintos valores en función de la función requerida. El paquete de control genérico esta mostrado en la figura 2.5.

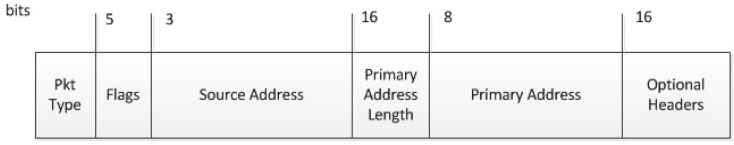

Figura 2.5: Paquete de control genérico ANTop

Figura 2.6: Encabezado opcional *Additional Address*

packet type El tipo del paquete en 5 bits, permitiendo hasta 32 tipos distintos.

flag 3 fags de uso general, que cada tipo de paquete puede asignar para lo que necesite.

total length Longitud total del paquete, incluyendo encabezados opcionales.

A continuación se detalla el valor de cada uno de los campos de un paquete de control genérico para cumplir con las diferentes instancias del proceso de conexión de nodos.

1. **Paquete PAR *(Primary Address Request)***

packet type Este campo toma el valor de 1.

source address La dirección IPv6 asignada a la interfaz inalámbrica, antes de correr el demonio ANTop.

2. **Paquete PAP** ***(Primary Address Proposal)***

   packet type Este campo toma el valor de 2.

   primary address La dirección IPv6 del nodo, asignada para participar en la red ANTop.

   primary address length La dimensión del hipercubo.

   optional headers Se incluye un encabezado opcional de topo *Additional Address*.

   El encabezado opcional contiene los siguientes valores,

   mask El valor de la máscara ANTop si se tiene una dirección R que ofrecer, ó cero en otro caso.

   address La dirección IPv6 que contiene embebida la dirección R ofrecida al nodo que quiere conectarse.

3. **Paquete PAN** ***(Primary Address Notification)***

   packet type Este campo toma el valor de 3.

   primary address La dirección IPv6 elegida para participar en la red ANTop.

   source address La dirección IPv6 de la interfaz inalámbrica antes de conectarse a la red ANTop.

   optional headers Se incluye un encabezado opcional de topo *Additional Address*.

El encabezado Additional Address tiene los siguientes valores en sus campos,

address La dirección de aquel nodo que ofreció la dirección elegida.

4. **Paquete PANC** ***(Primary Address Notification Confirmation)***

   packet type Este campo toma el valor de 4.

   Para el encabezado opcional se tiene solamente el siguiente valor.

   address La dirección de aquel nodo que ofreció la dirección elegida, que debería concordar con la del nodo que envía el paquete PANC.

   Para el proceso de cesión de direcciones secundarias, se utiliza un paquete SAP y un SAN.

5. **Paquete SAP** ***(Secondary Address Proposal)***

   packet type Este campo lleva el valor de 7.

   source address La dirección IPv6 del nodo que ofrece la dirección secundaria.
   primary address length La dimensión del hipercubo.

   optional headers Se envía un encabezado Additional Address.

   El encabezado Additional Address tiene los siguientes valores en sus campos,

   address La dirección IPv6 con la dirección R secundaria embebida.

   mask La máscara ANTop de la dirección secundaria.

6. **Paquete SAN** ***(Secondary Address Notification)***

   packet type Este campo lleva el valor de 8.

source address La dirección IPv6 del nodo que confirma la dirección secundaria.

optional headers Se envía un encabezado Additional Address.

Para el encabezado opcional se tiene solamente el siguiente valor,

address La dirección IPv6 con la dirección R secundaria embebida.

mask La máscara ANTop de la dirección secundaria.

7. **Paquete HB *(Hear Beat)***

packet type Este campo lleva el valor de 8.

primary address La dirección IPv6 del nodo que envía el HB.

optional headers Se envía un encabezado Additional Address.

A partir de aquí se mostrarán los paquetes de control *Rendez Vous* utilizados. En la figura 2.7 se muestran el paquete de control de *Rendez Vous*

Figura 2.7: Paquete de control *Rendez Vous*

1. **Paquete Register**

packet type Este campo lleva el valor de 9.

primary address La dirección IPv6 del nodo que se registra.

universal address La dirección *U* del nodo que se registra.

2. **Paquete Address Lookup**

   packet type Este campo lleva el valor de 10.

   source address La dirección IPv6 del nodo que pide la resolución.

   universal address Dirección *U* a resolver.

3. **Paquete Address Solve**

   packet type Este campo lleva el valor de 11.

   flags Este campo lleva el valor de 1 si la dirección fue resuelta, cero en otro caso.

   primary address La dirección IPv6 que se corresponde con la dirección *U* de la búsqueda.

   universal address Dirección *U* a resolver.

Capítulo 3

Implementación ANTop en SO Linux 4.10.0

En trabajo anteriores se abordo las especificaciones de ANTop, así como el diseño de un simulador, tanto para la versión reactiva del protocolo, como para la proactiva. Luego se diseño e implemento una aplicación que corra en sistema operativo Linux, mas específicamente la distribución Ubuntu basada en Debian. Esta última aplicación fue desarrollada en versión de núcleo (kernel) de sistemas GNU menores 2.6, las versiones mas actuales de los sistemas operativos Ubuntu (versión 16.04) utilizan el núcleo 4.10.0, en el cual sufrió una serie de modificaciones incompatibles con las versiones anteriores. Ante dicho inconveniente, una de las principales metas de este capítulo es la implementación de la aplicación ANTop en un sistema operativo Ubuntu 16.04.

En la sección 3.1, se presentan las herramientas utilizadas para la interacción entre el núcleo del sistema con el espacio usuario disponibles en sistema operativo Linux. Las herramientas presentes, son aquellas que fueron implementadas en la aplicación ANTop del trabajo de tesis [4].

Luego, en la sección 3.2 se presenta en detalle las funciones del núcleo de un sistemas Linux. En principio se da una reseña de las tareas que realiza el núcleo y se nombra el concepto de los módulos kernel y las ventajas operativas que este aporta. Al final de la sección se pretende compilar el código de la aplicación ANTop en un sistema operativo Ubuntu 16.04.

En la sección 3.3, se analiza las direcciones IPv6 que serán usada en la apli-

cación ANTop, justificando las decisiones tomadas.

Por último, en la sección 3.4 se realiza un ensayo entre dos computadoras corriendo la aplicación ANTop en un sistema operativo Ubuntu 16.04.

3.1. Interacción con el sistema operativo

La memoria de sistema en Linux se divide en dos regiones. *Espacio de Kernel* y *Espacio de Usuario*. La primera es en donde se ejecuta el núcleo del sistema operativo, es decir el Kernel. Desde allí, este último provee sus servicios. Por otro lado el Espacio de Usuario es la región de memoria del sistema operativo en la que corren los Procesos de usuario, es decir las instancias de los programas.

En el Kernel corren los servicios de bajo nivel, los cuales interactúan directamente con el hardware. Es allí donde, por ejemplo, se ejecutan los drivers de los dispositivos. Por esta razón, este código para cada computadora en forma particular. ANTop corre en el Espacio de Usuario con funcionalidades del Kernel mediantes interfaces creadas para tal fin.

La interacción con el Kernel se da en tres puntos básicos,

1. **Tabla de ruteo** El proceso de enrutamiento, ANTop consulta y edita la tabla de ruteo del nodo que corre el protocolo. A lo largo de este algoritmo se vuelve imprescindible la interacción con esta tabla. Por ende se genera canal de interacción del Espacio del Usuario con el Kernel para la gestión de dicha tabla.
2. **Interfaz de red** ANTop establece un esquema de asignación de direcciones R a la topología de la red, es posible ver que se necesitara poder definir desde la capa de usuario dicho parámetro de configuración determinado por el protocolo en base a las respuestas que reciba de los nodos vecinos. Por esta razón, también se vuelve un requerimiento contar con una herramienta que permita obtener y modificar la configuración
3. **Manejo de paquetes** Más adelante se vera que ANTop trabaja con paquetes de control que interactuan entre los nodos de la red y que son procesados en el espacio usuario transportados desde el espacio kernel.

3.1.1. IOCTL (Input / Output Control)

La opción para cubrir la interacción con la tabla de ruteo y la configuración de interfaz de red es IOCTL. El nombre de esta herramienta hace referencia a *Input / Output Control*. Se trata básicamente de una interfaz que permite comunicarse con el controlador del dispositivo a controlar mediante un conjunto de instrucciones

definidas como etiquetas en los archivos encabezados del código fuente del driver en cuestión. Esta función logra controlar una variedad de mecanismos del sistema operativo. Esta característica se debe a que la misma se implementa como una llamada de sistema que multiplexa diferentes comandos hacia la función del espacio de Kernel correspondiente. Una llamadas IOCTL tiene tres argumentos:

1. un archivo descriptor (o socket)
2. un identificador de comando
3. un argumento

3.1.2. Netlink

Netlink [11] es una herramienta que permite manejar módulos, es decir código que puede ser cargado o eliminado bajo demanda, en el sistema operativo, sin necesidad de recompilar el Kernel. Esto permite extender los servicios que brinda el mismo, sin siquiera tener que reiniciar el sistema. Provee un canal de comunicación bidireccional entre los procesos del espacio usuario y los módulos del espacio kernel. Consiste en una interfaz basada en sockets para los procesos y una *Application Programming Interface* (API) en el kernel para los módulos.

Los sockets, se deben definir algunos parámetros que los identifica. En el trabajo de tesis [4] se podrá consultar en detalle la configuración de dichos sockets y las distintas funcionalidades.

La dirección netlink (puerto) consiste en un entero de 32 bits. El puerto 0 (cero) está reservado para el kernel y se refiere al socket del lado del kernel de cada familia de protocolos netlink. Otros números de puerto usualmente se refieren a los espacios de usuario de los sockets, aunque esto no se aplica.

En la figura 3.1 se ilustra tres aplicaciones y el lado del núcleo que expone dos zócalos laterales del núcleo. Muestra los casos comunes de uso de netlink.

- Espacio del usuario al kernel.
- Espacio del usuario al espacio del usuario.
- Escuchando las notificaciones de multidifusión del kernel.

Espacio del usuario al kernel

La forma más común de uso de netlink es que una aplicación de espacio de usuario envíe peticiones al kernel y procese la respuesta que es un mensaje de error o una notificación de éxito.

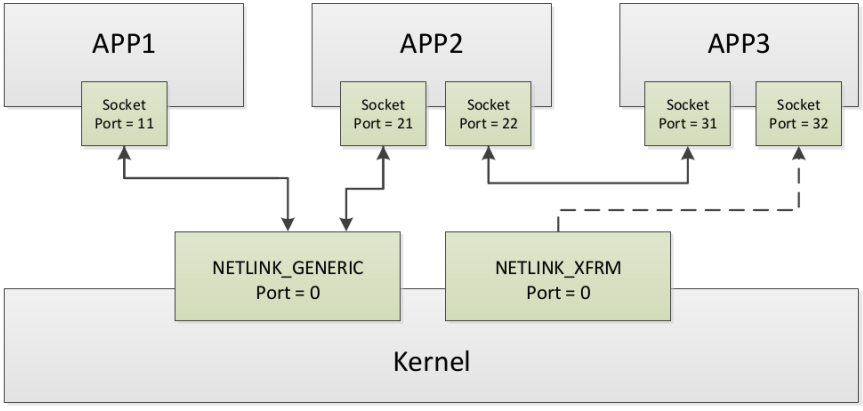

Figura 3.1: Direccionamiento sockets Netlink

Espacio del usuario al espacio del usuario

Netlink también puede utilizarse como un mecanismo IPC (Inter Porcess Communication) para comunicarse directamente entre aplicaciones de espacio de usuario. La comunicación no se limita a dos pares, cualquier número de compañeros puede comunicarse entre sí y capacidades de multidifusión permiten llegar a múltiples pares con un solo mensaje.

Para que los sockets sean visibles entre sí, ambos sockets deben crearse para la misma familia de protocolos netlink.

Espacio del usuario que escucha las notificaciones del kernel

Esta forma de comunicación de netlink se encuentra típicamente en los demonios de espacio de usuario que necesitan actuar en ciertos eventos del kernel. Tales demonios normalmente mantendrán un socket de netlink suscrito a un grupo de multidifusión que es usado por el kernel para notificar a las partes interesadas del espacio del usuario sobre eventos específicos.

Se prefiere el uso de multidifusión sobre el direccionamiento directo debido a la flexibilidad en el intercambio del componente de espacio de usuario en cualquier momento sin la notificación del kernel.

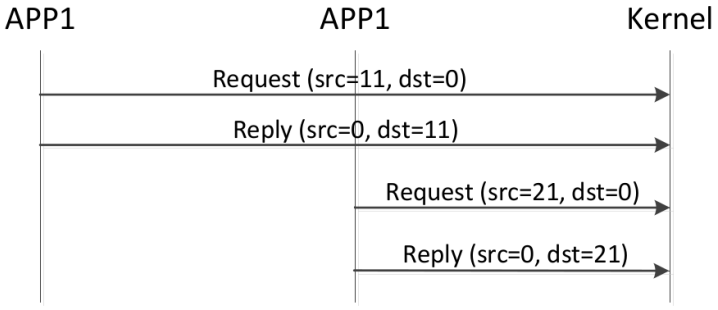

Figura 3.2: Conexión espacio usuario al kernel

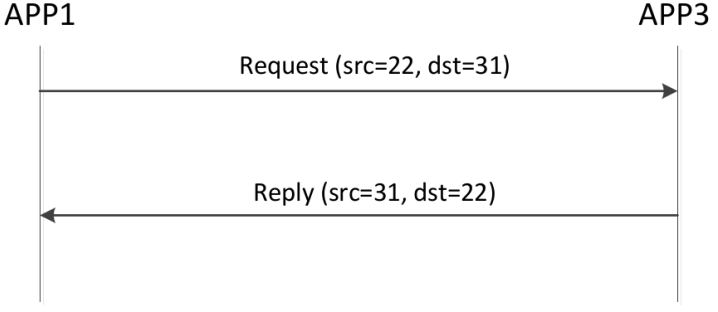

Figura 3.3: Conexión espacio usuario al espacio usuario

3.1.3. Netfilter

Netfilter está compuesto por una serie de ganchos ubicados a lo largo de la estructura de protocolos de red de un sistema Linux. En cada gancho es posible registrar un módulo kernel, que realice algún manejo de paquetes en diferentes instancias del procesamiento del mismo por parte de los protocolos de red. Un ejemplo de uso de esta herramienta de manejo de paquetes es Iptables, un firewall para Linux. Iptables utiliza Netfilter como motor de captura de paquetes. Netfilter inserta cinco ganchos asociados al protocolo IP en Linux.

Netfilter inserta cinco ganchos asociados al protocolo IP para versión 4 y 6 en Linux como muestra en la siguiente figura 3.5.

PREROUTING: Todos los paquetes que llegan al nodo, pasan por este gancho,

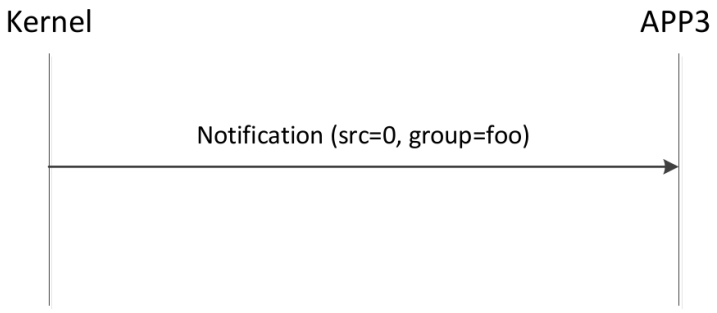

Figura 3.4: Escucha del espacio usuario las notificaciones del kernel

que se encuentra antes de la decisión de ruteo y después de los chequeos de sanidad del encabezado IP.

LOCAL INPUT: Todos los paquetes que tienen como destino la computadora local llegan a este gancho. Este es el último para aquellos paquetes dirigidos al nodo.

FORWARD: Por aquí pasan los paquetes que son ruteados por este nodo como un próximo salto,pero que tienen un destino diferente.

LOCAL OUTPUT: Este es el primer gancho en el camino de salida de aquellos paquetes que se originan en el nodo local.

POST ROUTING: Este gancho se ubica luego de la decisión de ruteo. Absolutamente todos los paquetes salientes del nodo pasan por este gancho.

Es posible registrar funciones de *callback* en cada uno de los ganchos. Es decir, funciones que se ejecutan automáticamente cuando un paquete atraviesa dicho gancho. De ese modo es posible realizar algún tipo de manejo de paquetes. Para cada registro de función se establece una prioridad, ya que es posible vincular varias de ellas a un mismo gancho.

Las funciones de callback pueden devolver diferentes valores, que se corresponden con diferentes accionar a tomar.

ACCEPT: Permite que el paquete siga atravesando la estructura del protocolo de red.

DROP: Descarta el paquete.

QUEUE: envía el paquete al Espacio de Usuario, de forma tal que un programa corriendo alli se encarga de manejarlo.

STOLEN: Detiene el paquete hasta que se cumple una condición. Esta acción se usa normalmente para colectar paquetes fragmentados.

REPEAT: Fuerza el paquete a ingresar nuevamente al gancho

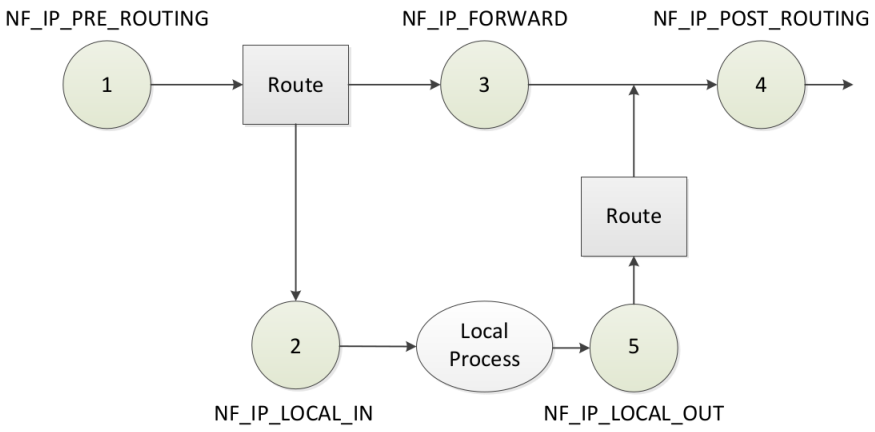

Figura 3.5: Diagramas de ganchos de Netfilter

3.1.4. Capa de enlace

El protocolo utilizado es IEEE 802.11 [13]. En la sección 5.2.1 del citado documento, se define el concepto de *Independent Basic Service Set LAN (IBSS LAN)*, para representar una red en la cual las estaciones se conectan directamente sin necesidad de ninguna entidad de control. También se indica que a este tipo de topología se le suele dar el nombre de *ad hoc network*.

Para que este servicio funcione correctamente, las estaciones (aquí serían los nodos de la red ANTop) deben trabajar en el mismo canal y compartir del mismo *Service Set Identifier* (SSID) para la red *ad hoc* a la que se conectan, de lo contrario no tendrán visibilidad a nivel de capa de enlace. Cada nodo que se conecte a la red ANTop deberá usar el mismo SSID, por ejemplo, *antop*. De esta manera, cuando un paquete contenga una dirección MAC de broadcast como destino, todos los nodos que estén dentro del alcance del que envío el paquete lo recibirán.

En el caso del estándar IEEE 802.11g [14], se define una velocidad de conexión máxima de 54 Mbps, para redes estructuradas. Sin embargo, para las redes *ad hoc*, el estándar no requiere alcanzar dicha tasa de transferencia, sino que será obligatorio alcanzar los 11 Mbps que define la norma b. Quedando, finalmente, la tasa máxima de transferencia en decisión del fabricante que proporcione el controlador del dispositivo, en este caso para un sistema operativo Linux.

3.2. Kernel

Tareas del kernel En un nivel puramente técnico, el kernel (núcleo) es una capa intermedia entre el hardware y el software en un sistema Linux. Su propósito es pasar solicitudes de aplicación al hardware y actuar como un controlador de bajo nivel para dirigir los dispositivos y componentes del sistema. Sin embargo, hay otras maneras interesantes de ver al kernel.

- El kernel puede considerarse como una máquina mejorada que, a la vista de la aplicación, abstrae la computadora a un nivel alto. Por ejemplo, cuando el kernel se dirige a un disco duro, debe decidir qué ruta usar para copiar datos del disco a la memoria, donde residen los datos, qué comandos debe ser enviado al disco a través de qué ruta de acceso, y así sucesivamente. Las aplicaciones, por otro lado, sólo necesitan emitir la orden de que los datos se van a transferir. La forma en que se hace esto es irrelevante para el solicitante. Los detalles son abstraídos por el kernel. Los programas de aplicación no tienen contacto con el hardware en sí, sólo con el núcleo, que, para ellos, representa el nivel más bajo en el jerarquía que conocen.
- La visualización del kernel como administrador de recursos se justifica cuando varios programas se ejecutan simultáneamente en un sistema. En este caso, el kernel es una instancia que comparte los recursos disponibles: tiempo de CPU, espacio en disco, conexiones de red, etc., entre los distintos procesos del sistema garantizando al mismo tiempo la integridad del sistema.
- Otra vista del kernel es como una biblioteca que proporciona una serie de comandos orientados al sistema. Como es generalmente conocidas, las llamadas de sistema se usan para enviar solicitudes al ordenador; con la ayuda de la biblioteca C estándar, estos aparecen en los programas de aplicación como funciones normales que se invocan de la misma manera que cualquier otra función.

Módulo kernel Los módulos Kernel son fragmentos de código que se pueden cargar y descargar en el kernel según la demanda. Extienden la funcionalidad del kernel sin necesidad de reiniciar el sistema. Por ejemplo, un tipo de módulo es el controlador de dispositivo, que permite al núcleo acceder al hardware conectado al sistema. Sin módulo, tendríamos que construir núcleos monolíticos y agregar nueva funcionalidad directamente a la imagen del kernel. Además de tener núcleos más grandes, esto tiene la desventaja de requerir reconstruir y reiniciar el kernel cada vez que deseamos una nueva funcionalidad. En Linux se puede observar los módulos que están cargados en el kernel ejecutando el comando `lsmod`, que obtiene su información leyendo el archivo `/proc/modules`. Los módulos kernel pueden ser invocados por la función *module_init* y ser removidos con la función *module_exit*.

Diferencia entre módulo kernel y una aplicación Antes de ir más lejos, vale la pena subrayar las diferencias entre un módulo kernel y una aplicación. La mayoría de las aplicaciones pequeñas y medianas realizan una sola tarea desde le principio hasta el final, cada módulo kernel sólo se registra para atender futuras solicitudes, y su función de inicialización terminan inmediatamente. En otras palabras la tarea de inicialización del módulo es preparase para la invocación posterior de las funciones del módulo; es como si el módulo estuviera diciendo *Aqui estoy, y esto es lo que puedo hacer*. La función de salida del módulo kernel se invoca justo antes de descargar el módulo, es decir, el módulo estaría diciendo *Ya no estoy allí, no me pidas que haga nada más*. Este tipo de enfoque de la programación es similar a la programación dirigida por eventos. Otra diferencia importante entre las aplicaciones dirigidas por eventos y el código kernel está en la función de salida de un módulo, es decir, que una aplicación que termina puede ser perezosa en liberar recursos o mismo lo evita. En cambio la función de salida del módulo kernel, debe ser cuidadoso a la hora de deshacer los recurso que acumulo la función inicial en la carga del módulo, o mismo estos quedaran hasta que se reinicie el sistema. En el caso de una aplicación puede llamar a funciones que no tiene definida en su código pero si las puede resolver utilizando bibliotecas de funciones apropiadas. Por otro lado, el módulo kernel está vinculado solo al kernel y las únicas funciones que puede llamar son las exportadas por el kernel, es decir que no hay bibliotecas para enlazar. Por ejemplo, en la figura 3.6 se muestra como las funciones *calls* y las demás funciones son usados en un módulo para agregar nuevas funcionalidades al correr el kernel.

Dado que ninguna biblioteca está vinculada a los módulos, los archivos origen no deben incluir los archivos de cabecera (*header*). Solamente las funciones que son realmente parte del kernel mismo pueden ser utilizadas en los módulos del kernel. Cualquier elemento relacionado con el kernel se declara en los *header* encontrados en el *kernel source tree*.

El *kernel source tree* es un directorio que contiene toda la fuente del kernel (*kernel source*).

Fuente de kernel (kernel source) Como lo indica su nombre, la fuente del kernel, es el conjunto de directorio con sus respectivos archivos de cabeceras llamados *header*, por el cual definen el kernel. Todo sistema Linux, es un desarrollo open source en constante mejora, por ende hay distintas versiones existentes. A lo largo de los años, las distintas versiones de Linux en cualquier distribución, fueron desarrollando nuevas versiones de fuente de kernel.

Las fuente kernel tiene un simple sistema de numeración de sus versiones. Se diferencia dos tipo de versiones de las existentes, una llamada estable y otra en

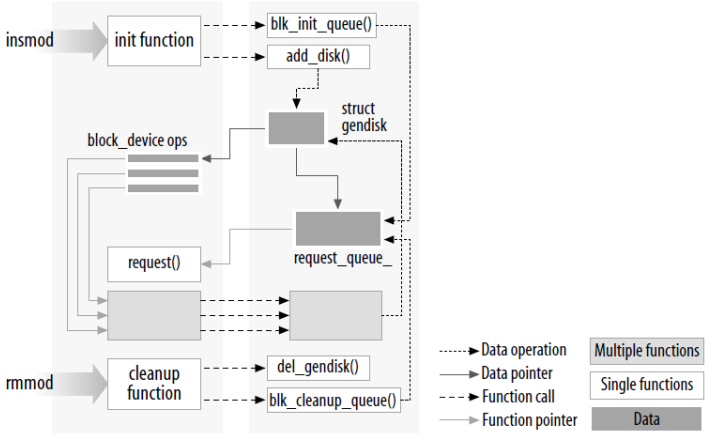

Figura 3.6: Vínculos de módulos del kernel, Fuente Figura 2.1 de [12]

desarrollo. Las versiones estable están dadas como por ejemplo la versión 2.0.30, en cambio las versiones en desarrolladas están denominadas de tal forma que su terminación numeral es distinta de cero, por ejemplo, 2.1.42.

Las fuentes se encuentran disponibles en `https://wwww.kernel.org`.

En el trabajo de tesis presente, a parte del estudio de la fragmentación y mezcla de redes ANTop, se deberá disponer de una versión actualizada de la implementación de ANTop del trabajo de tesis [4]. Dicha implementación fue construida en base a un sistema Linux Ubuntu de distribución Debian utilizando fuentes de kernel versión 2.6.0. Las últimas versiones disponibles del sistema operativo Ubuntu utiliza una versión de kernel 4.10.0, en el cual sufrieron cambios en los *headers files* en comparación la versión 2.6.0. Una de las metas de este capítulo es reacondicionar la implementación ANTop existente y llevarlo al funcionamiento sobre una versión de sistema operativo actual. Para ello es necesario introducirnos en la compilación de los módulos kernel del código fuente de la implementación ANTop.

Compilar módulo kernel Como primer paso, necesitamos ver un poco como deben ser construidos los módulos. El proceso de generación de módulos difiere significativamente de aquel utilizado para aplicaciones de espacio usuario. El kernel es un gran programa autónomo con requisitos detallados y explícitos sobre la forma en que sus piezas se juntan. Una de las cuestiones a tener en cuenta es que si se

requiere crear módulos cargables para un kernel 4.10.0 se debe de disponer el directorio de archivos fuentes del kernel instalado en el sistema.

Para crear un *makefile* para el módulo *kantop.c* se hace en principio la siguiente linea.

```
obj-m:= kantop.o
```

La linea anterior indica que hay un módulo que se construirá a partir del archivo *kantop.o*. El módulo resultante se llama *kantop.ko* después de haber sido construido el archivo objeto. Si en cambio, hay un módulo llamado *modulo.ko* que se genera a partir de dos fuentes llamados por ejemplo *file1.c* y *file2.c*, se define la siguiente linea.

```
obj-m:= modulo.o
module-objs:= file1.o file2.o
```

Como se mostró anteriormente, para que un *makefile* funcione debe ser invocado dentro del contexto del sistema de compilación del kernel más grande. Si el árbol de directorio de origen del kernel se encuentra en el directorio `/lib/modules/kernel-4.10`, el comando `make` necesario para construir su módulo (escrito en el directorio que contiene la fuente del módulo y *makefile*) sería la siguiente linea.

```
make -c /lib/modules/kernel-4.10 M=$(pwd) modules
```

Este comando comienza cambiando su directorio al que se proporciona con la opción `-c` (es decir, el directorio origen del kernel). Allí se encuentra el *makefile* de alto nivel del kernel. La opción `M` hace que el *makefile* vuelve a su directorio de fuente del módulo antes de intentar construir el destino de los módulos. Este objetivo, a su vez, se refiere a la lista de módulos encontrados en la variable `obj-m`, que hemos puesto a *modulo.o* en el ejemplo anterior.

Escribir el comando `make` anterior puede resultar tedioso después de un tiempo, por lo que los desarrolladores del kernel han desarrollado una especie de idioma *makefile*, que facilita la construcción de los módulos fuera del árbol de directorios de la fuente kernel.

```
objm:=kantop.o

KVERSION = $(shell uname -r)

all:
```

```
make -c /lib/modules/$(KVERSION)/build M=$(shell pwd) modules

clean:
make -c /lib/modules/$(KVERSION)/build M=$(shell pwd) clean
```

Estas últimas lineas es el *makefile* que se realizó para construir el módulo kernel de la implementación ANTop con fuente del kernel versión 4.10. Esto último corrige y soluciona el problema de compilación en la construcción de los módulos kernel de la implementación, que en cambio funcionaba en la versión 2.6.0.

Las versiones anteriores al kernel 2.6.0 requerían que se importara mucho sobre estos ajustes, que normalmente se almacena en el *makefile*. Aunque organizados jerárquicamente, muchos ajustes redundantes se acumulaban en el *makefile* de subnivel y los hacían grandes y bastante difícil de mantener. Afortunadamente, existe una nueva forma de hacer estas cosas, llamada *kbuild*, y el proceso de construcción de módulos cargables externos ahora está totalmente integrado en el mecanismo estándar de compilación kernel.

Volviendo al *makefile* escrito antes, se denota el comando `KVERSION = $(shell uname -r)` que se encargará de resolver la versión de la fuente kernel con la que cuenta el sistema local. Luego en la linea definida como `all` se define el comando `make` que invoca el sistema de compilación del kernel. Por último la linea definida como `clear`, se encargará de remover el módulo kernel en el momento que se lo solicite.

3.3. Direccionamiento en ANTop sobre IPv6

Tal lo visto en la introducción a ANTop en la sección 2, cada nodo en la red tiene una dirección R dependiente de la topología, que representará las coordenadas de ese nodo en un hipercubo de dimensión n, y que se usará para rutear el tráfico entre los nodos, mediante el criterio de mínima distancia al destino. En la presente tesis, las direcciones R adquieren un formato IPv6 como se puede apreciar en el trabajo de tesis [4], pero aun así se deberá intervenir en la elección de un tipo de dirección que nos permita trabajar y distribuir la implementación ANTop acordes a las normas.

En principio las direcciones R deben cumplir con el formato de *unicast* definido por IPv6 [15]. Se muestra en la siguiente figura 3.7 las distintas direcciones *unicast* definidas.

En el trabajo [4], se discutió las opciones del tipo de dirección a utilizar como dirección R en el protocolo. En principio se analizó la posibilidad de utilizar el

Address type	Binary prefix	IPv6 notation
Unspecified	00...0 (128 bits)	::/128
Loopback	00...1 (128 bits)	::1/128
Multicast	11111111	FF00::/8
Link-Local unicast	1111111010	FE80::/10
Global Unicast	(everything else)	

Figura 3.7: Segmentación de dirección IPv6.

tipo de direcciones *link local*. Esto último resolvía varias cuestiones con el funcionamiento del protocolo, como ser el mecanismo de asignación de dirección local en el cual garantizaba la unicidad de dirección entre los demás direcciones de dispositivos vecinos. Sin embargo no fue posible utilizar la configuración debido a la manera que el sistema operativo maneja los pedidos de DNS, los cuales tiene gran relevancia en el funcionamiento del servicio *Rendez Vous* que incorpora ANTop.

Al no poder utilizar direcciones del tipo *link local* debido al problema de resolución de nombres, se debió optar por las de tipo *global unicast*. En la figura 3.8 se muestra los campos que componen una dirección *global unicast*, uno de ellos se utiliza para identificar el sitio.

Figura 3.8: Formato de dirección Global Unicast IPv6.

En el caso del presente trabajo, se tiene una red *ad hoc* en la cual el sitio es único. Por esta razón, se puede pensar en un formato IPv6 con rango global, sin la necesidad de destinar m cantidad de bits a la identificación de la subred. En este modelo, será el campo *Interface ID* el que alberga la dirección R del nodo ANTop. Este es el esquema de mapeo que se emplea para llevar las direcciones R a direcciones IPv6.

Todavía falta definir el valor que tomará el campo *Global Routing Prefix* y *Subnet ID*. Dependiendo del campo *Global Routing Prefix*, existen distintos entes reguladores que gestionan la asignación de direcciones públicas que son identifica-

das en dicho campo. Por ejemplo para el prefijo y dirección *2001:0000::/23* dicho espacio está designado para el IANA (*Internet Assigned Numbers Authority*), es decir, que si necesito direcciones públicas IPv6 del tipo *global unicast* una de las opciones es comunicarse con dicha organización y solicitarlas. Entonces, a la hora de elegir el contenido de los campos *Global Routing Prefix* y *Subnet ID*, debemos de solicitar alguna de las organizaciones que gestionan las direcciones *global unicast*. Sin embargo la otra manera es de utilizar las direcciones IPv6 reservadas para documentación definidas en la RFC 3849 [16]. Esta última tiene el prefijo *2001:db8::/32*, el resto de los bits se podrá utilizar como la identificación de la dirección R en la implementación del protocolo ANTop.

3.4. Prueba de funcionamiento

En las secciones anteriores, se mencionaron y justificaron los cambios realizados en la implementación ANTop. En esta sección se realiza una prueba de funcionamiento. Si bien en el capítulo 5 se realizan simulaciones y pruebas de redes ANTop, el fin de esta sección es comprobar que la aplicación del protocolo funcione para un sistema operativo Ubuntu versión 16.04, y además las direcciones R sean representadas como se describió en la sección anterior 3.3.

La siguiente prueba consistirá en iniciar la red ANTop corriendo la aplicación ANTop en una PC que llamaremos *nodo 1*. Luego, se conecta una segunda PC a la red ANTop iniciada por el *nodo 1*, que llamaremos *nodo 2*. Este último se asignará una dirección por medio del proceso de conexión de nodos 2.3.3.

En el momento que el *nodo 1* se conecte a la red tiene la siguiente configuración para su interfaz de red.

```
wlan0     Link encap:Ethernet  HWaddr 70:77:81:bc:3d:9d
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: 2001:db8::a/64 Scope:Global
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:13 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1278 (1.2 KB)  TX bytes:572 (572.0 B)
```

Figura 3.9: Configuración de la placa inalámbrica del *nodo 1*.

Inicialmente, el nodo presenta una dirección que tiene que ser configurada por

el usuario. A dicha dirección inicial se destinarán los paquetes PAP provenientes de los nodos que ya están conectados a la red ANTop. En este caso la dirección elegida al configurar la conexión inalámbrica *ad hoc* es *2001:db8::a* con máscara 64. La tabla de rutas se muestra en la figura 3.10.

```
Kernel IPv6 routing table
Destination                    Next Hop                   Flag Met Ref Use If
2001:db8::/64                  ::                         U    256 0     0 wlan0
::/0                           ::                         !n   -1  1     1 lo
::1/128                        ::                         Un   0   1     0 lo
2001:db8::a/128                ::                         Un   0   1     0 lo
ff00::/8                       ::                         U    256 3    15 wlan0
::/0                           ::                         !n   -1  1     1 lo
```

Figura 3.10: Tabla de rutas del nodo1.

En este punto se ejecuta el demonio ANTop en el espacio usuario. Según lo discutido en la sección 2.3.3, se asigna la primer dirección de la red del espacio del hipercubo.

```
wlan0     Link encap:Ethernet  HWaddr 70:77:81:bc:3d:9d
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: 2001:db8::1/64 Scope:Global
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:15 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1438 (1.4 KB)  TX bytes:2076 (2.0 KB)
```

Figura 3.11: Configuración de la placa inalámbrica del *nodo 1* , al correr ANTop.

Se debe aclarar que la máscara de red de la interfaz vale 64, que es distinto a la máscara de ANTop, lo cual en este caso vale 0 y esta definida en una variable de espacio usuario.

En la imagen 3.12 se presentan la tabla de ruteo luego de obtener la nueva dirección IPv6.

Ahora se conecta el *nodo 2* a la red. El mismo envía una paquete PAR en

```
Kernel IPv6 routing table
Destination                    Next Hop                   Flag Met Ref Use If
2001:db8::/64                  ::                         U    256 0     0 wlan0
::/0                           ::                         !n   -1  1     1 lo
::1/128                        ::                         Un   0   1     0 lo
2001:db8::/128                 ::                         Un   0   1     0 lo
2001:db8::1/128                ::                         Un   0   1     0 lo
ff00::/8                       ::                         U    256 4   228 wlan0
::/0                           ::                         !n   -1  1     1 lo
```

Figura 3.12: Tabla de rutas del *nodo 1*, al correr ANTop.

modo broadcast, y recibe un PAP del *nodo 1* con la propuesta de la siguiente dirección del hipercubo con máscara 2, es decir, que le propone la dirección *2001:db8::8000:0:0:1*. El *nodo 2* al recibir solo el PAP del *nodo 1*, como se muestra en la figura 3.13, acepta la direcciones propuesta. Luego del proceso de conexión, el *nodo 2* se configura la dirección de la interfaz inalámbrica como se muestra en la figura 3.14.

```
PAR sent. Waiting for PAPs.
ANTOP SOCKET READ
se recibio un paquete de tipo PAP
The pkt will be processed
Receive PAP pkt_type: 2
PKT TYPE: 2
MASK: 1
PRIM ADDR: 2001:db8:0:0:0:0:0:1
```

Figura 3.13: PAP del *nodo 1* recibido por el *nodo 2*.

Para una máscara de red de 64, se dispone de 64 bits para representar a cada nodo, es decir, 8 bytes. En las notación utilizada, cada byte es representado por dos dígitos hexadecimal. En este caso el primer dígito de la dirección R del *nodo 2* tiene el valor de 8, lo cual implica que solo el bit más significativo toma el valor de 1. Esto es lo esperado según planteado en el mecanismo de asignación de direcciones ANTop.

Finalmente la tabla de ruteo resultante se muestra en la figura 3.15.

En el momento que cada uno de los nodos genere el paquete HB, al recibirlo el otro actualizará la tabla de vecinos. En la figura 3.16 se presenta la tabla de vecino

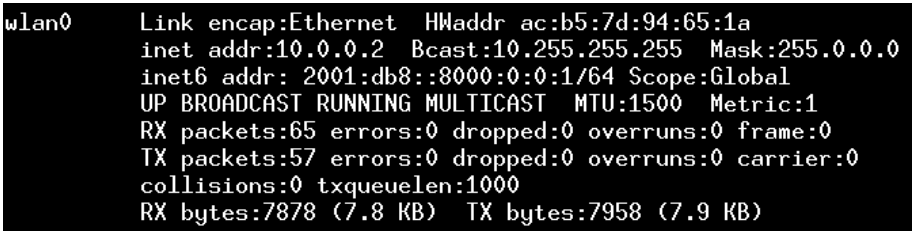

Figura 3.14: Configuración de la placa inalámbrica del *nodo 2* , al correr ANTop.

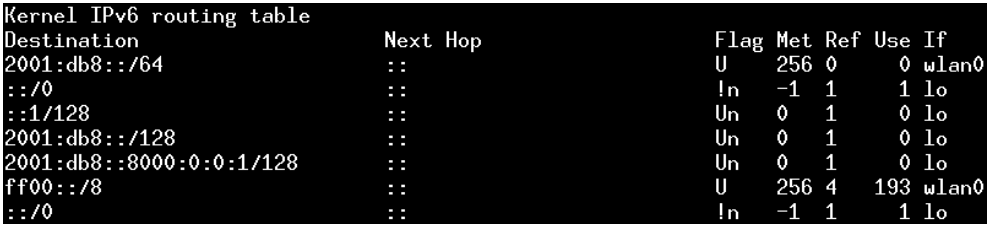

Figura 3.15: Tabla de rutas del *nodo 2*, al correr ANTop.

del *nodo 1* en el cual solo tiene como vecino la dirección *2001:db8::8000:0:0:1* y máscara 1 que corresponde a la dirección *R* primaria y máscara del *nodo 2*. Este último, por otro lado, recibe el HB del *nodo 1* y actualiza la tabla de vecinos que corresponde a la dirección *R 2001:db8::1* y máscara 1 como se muestra en la figura 3.17.

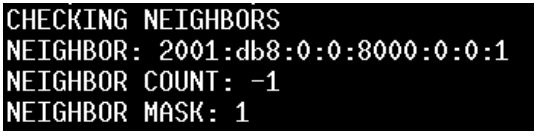

Figura 3.16: Tabla de vecinos del *nodo 1*.

El campo `NEIGHBOR COUNT` es la cantidad de periodos consecutivos, durante los cuales no se ha recibido un HB. En estos casos, el valor de -1 indica que no se ha perdido ninguno.

```
CHECKING NEIGHBORS
NEIGHBOR: 2001:db8:0:0:0:0:0:1
NEIGHBOR COUNT: -1
NEIGHBOR MASK: 1
```

Figura 3.17: Tabla de vecinos del *nodo 2*.

Capítulo 4

Fragmentación y mezcla

En el capítulo presente se estudia la fragmentación y mezcla de redes *ad hoc* utilizando el protocolo ANTop.

Antes del análisis de cualquiera de las dos situaciones que se plantea, primero en la sección 4.1 se incorpora un nuevo concepto, que consiste de numerar las redes en el cual se integrará al protocolo ANTop. Este último concepto tiene gran influencia en la detección de mezclas de redes ANTop. Se detalla los algoritmos y nuevo campos del protocolo que se integrarán en la implementación.

Luego en la sección 4.2 se tratará la fragmentación de una red ANTop, cuyo estrategia será primero el análisis de la detección seguido de una propuesta para mitigar los problemas encontrados en dicha situación.

Por último, en la sección 4.3 se analiza la mezcla de dos redes ANTop, en donde se definen los algoritmos de detección y transmisión de mensaje a los nodos afectados en la mezcla.

4.1. Numeración de red

En este capítulo como se mencionó, el objetivo central es el estudio de fragmentación y mezcla de redes ANTop. En primer lugar para definir estas situaciones, es necesario que se identifiquen las redes ANTop que son administradas independientemente. Para ello se define en esta sección el concepto de *número de red*.

En la sub sección 4.1.1, se muestra una problemática que define el propósito

de este nuevo concepto, y luego se presenta el campo *número de red*. En la última sub sección 4.1.2 se presenta el algoritmo de asignación de *número de red* que será incorporado en el protocolo ANTop.

4.1.1. Propósito y campo *número de red*

Supongamos el siguiente esquema de la figura 4.1.

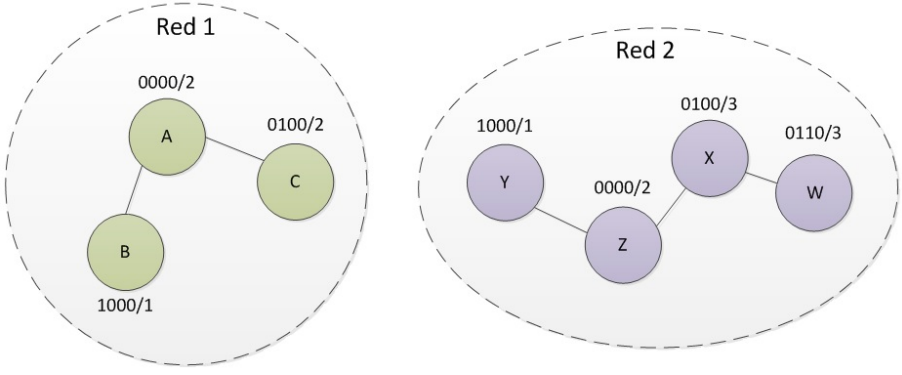

Figura 4.1: Dos redes ANTop independientes

Aquí se presentan dos redes, en el cual una de las redes está compuesto por los nodos verdes que llamaremos Red 1 y la otra por los nodos violetas que llamaremos Red 2.

Antes de analizar que ocurre en la mezcla entre nodos de diferentes redes, vamos a analizar como ocurrieron las conexiones y la asignación de las direcciones en cada red.

Supongamos que inicialmente se encuentran dos nodos distanciados, el nodo A y el nodo Z. El nodo A al correr ANTop envía paquetes PAR en modo broadcast para iniciar su conexión en la red. Luego de haberse consumido el *timer* de espera de paquetes PAP, es decir que no recibe ninguna propuesta de dirección, asume que no hay una red ANTop en su entorno del alcance de la placa inalámbrica, y se asigna la primer dirección de hipercubo 0000 y máscara 1. Luego el nodo B, que se encuentra dentro del radio de alcance del nodo A, iniciar su conexión enviando la solicitud del paquete PAR y recibiendo una respuesta PAP del nodo A proponiéndole la dirección 1000 y máscara 2. Este primero lo acepta confirmándolo con el envío de un paquete PAN en modo broadcast y en cual el nodo A envía un

nuevo mensaje PANC de confirmación dando concluida el proceso de asignación vista en la sección 2.3.3. El tercer nodo en conectar es el C. Suponiendo en principio que solo tiene en su alcance al nodo A quien será el que le asigna la dirección 0100 y máscara 3 con el mismo proceso que se vio anteriormente para el nodo B. Entonces el nodo A aumentaría el valor de 1 a su máscara tomando el valor de 3, cediendo parte de su espacio al nodo C en este último proceso.

Hasta acá se conforma la Red 1.

Teniendo de base la formación y la asignación de direcciones de la Red 1, se puede ver fácilmente como ocurrió la asignación de direcciones de la Red 2. En primer lugar el nodo en conectarse es el Z que se asigna la primer dirección de hipercubo. Sin entrar en detalles, Z le asigna la primer dirección 1000 y máscara 1 a Y, y luego la dirección 0100 y máscara 2 a X. Por último W que tiene solo a su alcance al nodo X, al conectarse este último le asigna la dirección 0110 con máscara 3, dejando así conformada la Red 2. Si suponemos que el nodo Y físicamente se moviliza sin perder la conectividad con el nodo Z hacia el sentido del nodo C, llegará a un punto en el cual los nodos C y Y tendrán conectividad a partir de que uno esté dentro del radio de alcance del otro. En la figura 4.2 describe lo mencionado.

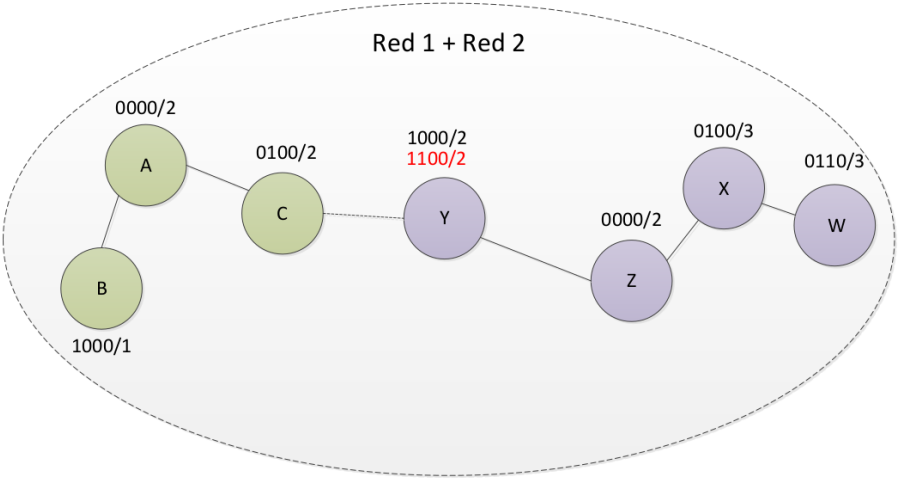

Figura 4.2: Dos redes ANTop independientes mezcladas

Todos los nodos envían un paquete de control *Heart Beat*, que anuncia la exis-

tencia a los nodos vecinos. En el momento que el nodo Y procesa un paquete HB del nodo C, este lo considera como un nodo más de su red y se entera que tiene una dirección con distancia 2 a si mismo. Por dicho motivo le envía un mensaje de SAP para proponerle su dirección secundaria 1100 y máscara 2 como vecino, y así tener conectividad a nivel ANTop. El nodo C al recibir la propuesta, responde con un mensaje SAN notificando la confirmación. De aquí en más el nodo Y enviará HB tanto por la dirección primaria como por la secundaria.

Entonces el registro de la tablas de vecinos de los nodos C y Y, quedan de la siguiente forma.

Vecinos del nodo C:
0000
1100

Vecinos del nodo Y:
0000
0100

La distribución de direccionamiento basado en hipercubo pierde sentido, solo por el hecho de que hay nodos que comparten la misma dirección R en la red resultante de la mezcla de la red 1 y 2.

Por ejemplo, el nodo Y perdería comunicación con el nodo X. Esto es producto de que el nodo X comparte la misma dirección R que el nodo C, pero este último es un vecino del nodo Y. Entonces todo mensaje con dirección destino 0100, desde el nodo Y, lo estará enviando al nodo C. En resumen, la resultante de la mezcla de las redes destruyen el concepto de direccionamiento por hipercubo y la teoría propuesta por el protocolo ANTop.

A lo largo de este capítulo se estudiará la fragmentación y mezcla por separado y las acciones que las redes deben tomar para responder a la problemáticas descriptas. Pero antes incorporaremos un concepto nuevo que será la numeración de las redes ANTop. Esto nos permitirá detectar la mezcla de dos o más redes.

El concepto de numeración de redes, es básicamente disponer dentro del encabezado de algunos paquetes de control ANTop, un campo que se almacene el *número de red* al que pertenece dicho nodo.

Volviendo al ejemplo de la figura 4.1, a la Red 1 la numeramos como red número 1 y la Red 2 como número 2. Retomando el desplazamiento del nodo Y

hasta estar al alcance del nodo C sin perder conexión con el nodo Z, en el momento que el nodo Y procesa el paquete HB del nodo C, este lee que el mensaje HB tiene un *número de red* distinto al del propio y se podrá interpretar que es un nodo de otra red. Esto último es una especie de alarma para el nodo Y, en el cual podrá disparar un mecanismo que mantenga la consistencia e integridad de las características de las redes ANTop propuestas por el protocolo.

En la sección 2.3, se presentó un resumen del protocolo ANTop, en donde se describió los paquetes de control de utilizados y las funciones que abordan. Los paquetes de control PAR, PAP, PAN, PANC, SAP, SAN y HB tienen una estructura genérica con un encabezado opcional como se muestra en la figura 4.3.

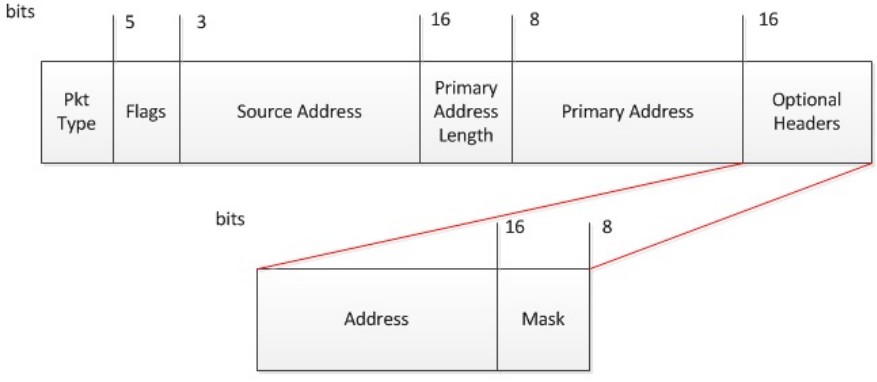

Figura 4.3: Paquete de control genérico ANTop con encabezado opcional

El encabezado opcional es usado para múltiples propósitos según la función a cumplir que tiene el paquete de control ANTop. Por ejemplo, se utiliza el campo *Address* y *Mask* en el mensaje PAP, para transmitir la dirección R y máscara propuesta al nodo que lo recibe y quiere iniciar una sesión en la red ANTop.

Para la numeración de red, se agrega un campo más en el encabezado opcional del tamaño de un byte. El encabezado opcional resultante se muestra en la figura 4.4.

Figura 4.4: Encabezado opcional modificado

4.1.2. Algoritmo de asignación de *número de red*

En primer lugar se discute el valor que tiene que tener el *número de red* de una red ANTop. Las características con las que tiene que contar es propia de la unicidad del valor en entre las redes posibles. Es decir, el *número de red* de cada red tiene que ser único e irrepetible.

Cuando un nodo quiere formar parte de una red ANTop, este en primer lugar envía un mensaje PAR para recibir propuestas de direcciones aceptadas y disponible por dicha red. Pero si el nodo está fuera del alcance de cualquier red ANTop, se creará una nueva red ANTop asignándose al mismo la primer dirección de hipercubo. Entonces la estrategia de asignación de *número de red* estará atada a características únicas de cada primer nodo en conectarse a la red, o así también alguna identificación numeral de la misma, como puede ser la dirección MAC de la placa wireless.

En las redes de computadoras, la dirección MAC (*Media Access Control*) es un identificador de 48 bits (6 bloques de dos caracteres hexadecimales, 4 bits por carácter, o sea 8 bits por bloque) que corresponde de forma única a una tarjeta o dispositivo de red. Se conoce también como dirección física, y es única para cada dispositivo. Está determinada y configurada por el IEEE (los últimos 24 bits) y el fabricante (primeros 24 bits) utilizando el *Organizationally Unique Identifier* (Identificador Único Organizacional). En la figura 4.5 se muestra el formato.

El valor de la dirección MAC es el resultado directo de las normas implementadas por el IEEE (*Institute Electrical Electronics Engineers*) para proveedores con el objetivo de garantizar direcciones únicas para cada dispositivo ethernet. Las normas establecidas por el IEEE obligan a los proveedores de dispositivos ethernet a registrarse en el IEEE. El IEEE le asigna a cada proveedor un código de 3 bytes, denominado Identificador único organizacional (OUI).El IEEE obliga

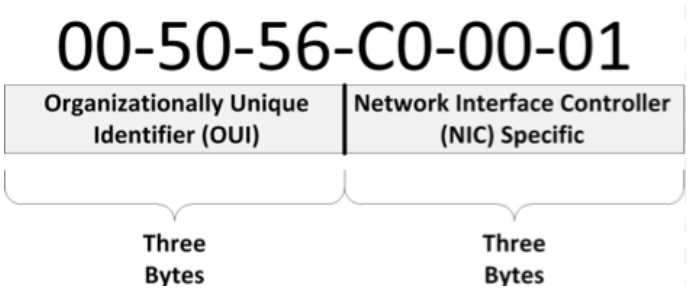

Figura 4.5: Formato general de dirección MAC

a los proveedores a respetar dos normas simples: Todas las direcciones MAC asignadas a una NIC (Network Interface Controller) u otro dispositivo ethernet deben utilizar el OUI que se le asignó a dicho proveedor como los 3 primeros bytes. Se les debe asignar un valor exclusivo (código del fabricante o número de serie) a todas las direcciones MAC con el mismo OUI (Identificador exclusivo de organización) en los últimos 3 bytes.

En resumen, los bytes correspondientes al NIC, son únicos en cada interfaz wireless de los nodos en una red ANTop. El campo *número de red* en el encabezado opcional tiene tamaño de 1 byte, entonces una buena identificación para una red ANTop sería de elegir algunos de los tres bytes del campo NIC de la dirección MAC del nodo que tiene la primer dirección del hipercubo. Esto último se aplica a dicho nodo ya que sería el primer nodo en conectarse a la red, o mejor dicho, el que crea una nueva red que luego evolucionará en cantidad de colaboradores, siendo éste el que le asigne una identificación a la red.

En este trabajo se elije la numeración del campo *número de red* al primer byte del NIC, es decir, al cuarto byte de la dirección MAC de la interfaz wireless del nodo. Cabe señalar que la implementación de la numeración de red en el código del protocolo, se podrá configurar cualquiera de los bytes de la dirección MAC como valor de *número de red*.

Anteriormente mencionamos brevemente que hay tres casos de asignación del valor del campo *número de red* del nodo al conectarse a la red ANTop.

1. El nodo al conectarse envía un mensaje PAR, y espera la recepción de propuesta de dirección de mensajes PAP un tiempo hasta un *timeout* con-

figurable del protocolo. Si no se recolecta ningún mensaje PAP en ese lapso de tiempo, el nodo se autoconfigura la primer dirección de hipercubo y éste a su vez se asigna el *número de red* propio que será el cuarto byte de la dirección MAC del mismo.

2. El nodo al conectarse envía un mensaje PAR, y espera la recepción de propuesta de dirección de mensajes PAP un tiempo hasta un *timeout*. Se recolecta uno o más mensaje PAP con propuesta de dirección a la red ANTop. Del mensaje PAP de la dirección elegida para asignarse, se toma el campo de *número de red* del encabezado opcional, y se asigna dicho valor como *número de red* del nodo mismo luego de haber recibido el mensaje PANC de confirmación de uso de la dirección primaria. El *número de red* resultante es el valor del cuarto byte de la dirección MAC del nodo que tiene asignada la primer dirección de hipercubo de la red ANTop a la que se conectó y de ahora en más pertenece.
3. El nodo envía un mensaje PAR para iniciar sesión en un red ANTop, y recibe un mensaje PAP indicando que no hay direcciones disponibles. El nodo no se puede conectar y finaliza el algoritmo de procedimiento del protocolo ANTop. En este caso el nodo no llega asignarse ningún valor de *número de red* ya que ANTop deja de correr en el dispositivo.

El algoritmo de asignación de *número de red* está acompañado del algoritmo de asignación de dirección primaria. En decir, que acompaña los tiempos y procesos de asignación de la misma manera como se genera la dirección primaria descripto en el algoritmo 1. El algoritmo del establecimiento de dirección primaria se puede consultar también en los trabajos de tesis de Alejandro Marcu [2] y Matias Campolo [4].

La asignación de *número de red* se describe en el algoritmo 13.

4.2. Fragmentación

En la sección 4.1, se discutió la necesidad de numerar las redes ANTop que son gestionadas en forma independiente entre sí y se justificó en base a una mezcla de dos redes ANTop. En principio la numeración de las redes ANTop, desde el punto de vista de una fragmentación queda sin efecto, ya que en la detección no será necesario, pero igualmente se tendrá en cuenta para conservar el concepto ante alguna futura mezcla.

En la sub sección 4.2.1 se define la fragmentación y el modo de detección utilizado. Luego en la sub sección 4.2.2 se describe el proceso de asignación de direcciones de la red fragmentada.

Algorithm 13 Asignación de número de red

```
1: if se recibió al menos un paquete PAP con dirección disponible then
2:     Esperar a que llegue un paquete PANC o que transcurra un tiempo
3:     T_PANC;
4:     if se recibió un paquete PANC then
5:         Se asigna al número de red el valor del campo de número de red
6:         del mensaje PAP con menor máscara;
7:     end if
8: else
9:     if se recibió un paquete PAR indicando que no hay direcciones disponibles
   then
10:        No se puede conectar y no se asigna ningún valor de número de red;
11:        Fin del algoritmo;
12:    end if
13:    Al número de red se asigna el valor del cuarto byte de la dirección MAC
14:    del dispositivo;
15: end if
```

4.2.1. Definición y detección

Antes de comenzar con la teoría de detección de fragmentación, primero se repasa el concepto de tabla de vecino (*neighbor table*). La tabla de vecinos es un registro del protocolo que se actualiza a medida que se procesa un paquete HB. Los HB son los paquetes de control que cada nodo emite para que el nodo vecino tenga conocimiento de la existencia del mismo. El nodo al recibir un paquete HB, se toma la molestia de verificar la dirección R del nodo que emitió dicho paquete, y si éste está a una distancia 1, lo agrega al registro tabla de vecinos. Luego una dirección del registro de la tabla de vecinos, puede ser eliminada cuando habiendo ocurrido tres ciclos consecutivos de no haber recibido paquete HB de dicha dirección en cuestión, éste sería eliminado de la tabla.
La detección de fragmentación, se enfocará en el concepto de tablas de vecinos para saber cuando disparar el mecanismo que mitigue potenciales problemas.

En la figura 4.6 se muestra una red ANTop cuya estructura se presentan las tablas de vecinos de cada uno de los nodos. Haciendo foco en el nodo C, se observa que el mismo en un ciclo recibe paquetes HB de los nodos A, D y E. Por ende, el C tiene registrado las dirección R de cada uno de ellos en su tabla de vecinos. Lo mismo ocurre con otro nodo, por ejemplo, el nodo B tiene en su tabla de vecino al nodo A que es el único que tiene conexión y del cual solo recibió paquete HB.

Ahora si se supone que el nodo A se desplaza físicamente hacia la izquierda

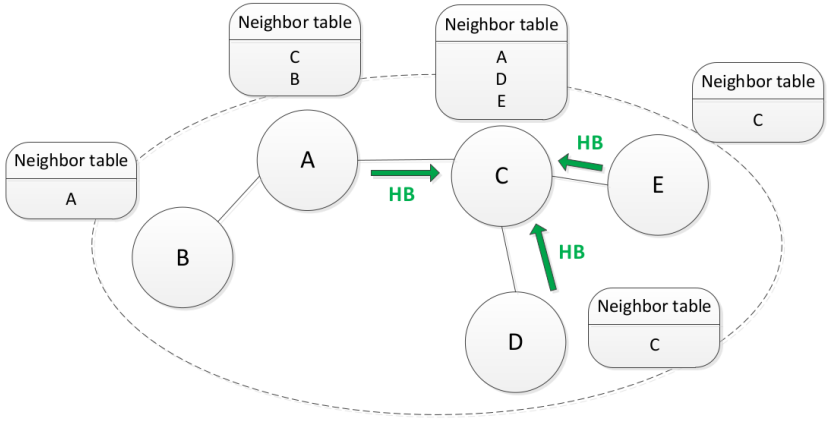

Figura 4.6: Red ANTop antes de la fragmentación

a tal punto de salir del alcance del nodo C y perder conexión con este último, el nodo C en principio va mantener en su tabla de registros de vecino a la dirección R del nodo A. Así también el nodo A va mantener en su tabla de vecinos la dirección R del nodo C. Luego de tres ciclos de envíos de HB, cuyo *timer* del ciclo es configurable en la implementación del protocolo, tanto el nodo A como el nodo C, se van a eliminar de sus tablas de vecinos respectivamente, como se muestra en las tablas resultantes de la figura 4.7.

Se ve claro que la fragmentación ocurrió, y esquemáticamente se observa dos redes separadas como resultado. Pero de las dos redes resultante, una tiene que tomar el rol de disparar el mecanismo de fragmentación que más adelante del capítulo se tratará, y la otra directamente no tomará acción alguna. Antes de definir la diferencia entre las partes fragmentadas, se vuelve a repasar el significado del nodo *padre* cuyo concepto fue desarrollado en el trabajo [2].

El nodo *padre* de un nodo en cuestión, es aquel en el cual le proporcionó el espacio de direcciones que se tuvo cuando este se conectó a la red. Otra definición, puede ser aquel nodo que es vecino y que tiene menor distancia a la primera dirección del hipercubo entre todos los vecinos que tiene registrado en la tabla de vecinos. Por ejemplo, un nodo cuyo dos vecinos tienen dirección R 1000 y 1110, entonces el nodo con dirección 1000 es el nodo *padre*.

Suponiendo que en la figura 4.7, el nodo A es el *padre* del nodo C, entonces la

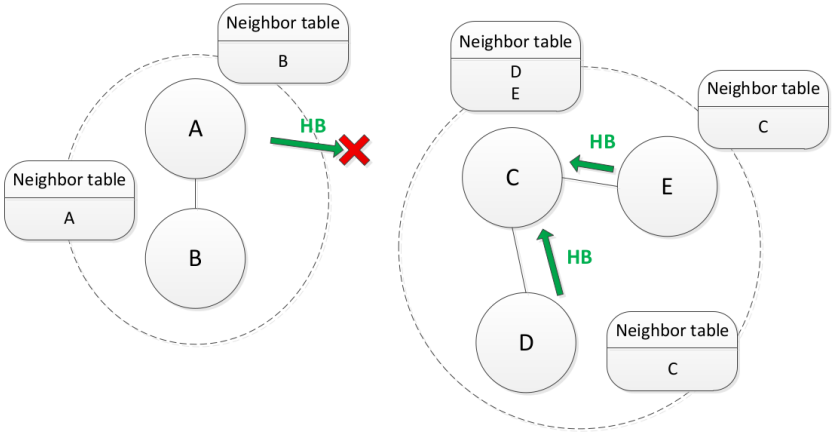

Figura 4.7: Red ANTop fragmentada

red fragmentada será aquella en el cual está compuesta por el nodo C, D y E. Es decir, que dicha red se disparará el mecanismo de fragmentación que en principio será de renombrar la red, empezando por el C que se asignará la primer dirección del hipercubo para garantizar todo el espacio de direcciones disponibles.

Hasta ahora se puede definir que un nodo detecta fragmentación, cuando este pierde conexión con su nodo *padre*. Pero no es condición necesaria, ya que el protocolo ANTop tiene la opción de ceder direcciones secundarias y se puede dar el caso de la figura 4.8.

Si se supone que el nodo A tiene asignada la primer dirección del hipercubo, y a su vez es el nodo *padre* del C, cuando este último pierda conexión con el A, sería erróneo considerar la red conformada por C, D y E fragmentada de la red de A y B. Gráficamente se muestra que luego de que el C elimina de la tabla de vecinos a su *padre* el nodo A, aún hay un vinculo que no permite la fragmentación por el enlace secundario formado entre el nodo B y D. Entonces, dado este caso, luego de que el nodo detecte perdida de conexión con su nodo *padre*, se debe disparar un mecanismo de exploración para asegurar que no haya formado enlaces secundarios que no permiten la fragmentación.

Una solución para la exploración de enlaces secundarios, podría ser aprovechando ciertas características desarrolladas del protocolo ANTop. Una de ellas el ruteo de paquetes de un nodo a otro de la red. En los trabajos [2] y [3], se

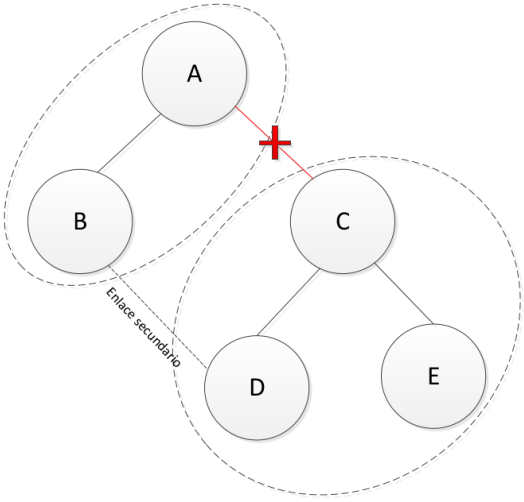

Figura 4.8: Fragmentación de red ANTop con enlace secundario

describieron algoritmos de ruteo reactivo y proactivo respectivamente. Los algoritmos desarrollados de ruteo, ahorra desarrollar un mecanismo de exploración de enlaces secundarios en particular. La estrategia consistirá de generar un paquete ICMP *Echo Request* con dirección destino la primer dirección del hipercubo, y si se recibe la respuesta de ICMP *Echo Reply* antes de que se consuma el *time out*, se podrá considerar que existe un enlace secundario y concluir que no hay fragmentación. En caso contrario, al no recibir el paquete ICMP *Echo Reply*, se considerará que efectivamente hay fragmentación y entonces se podrá proceder a ejecutar el mecanismo de renombramiento de las direcciones de la red fragmentada.

El detector de fragmentación, algoritmo 14, es un proceso que corre cada vez que el nodo reciba un paquete HB, en donde deberá de actualizar la tabla de vecino pero también debe de chequear si el nodo *padre* sigue siendo el mismo. En la sección 4.3 se analizará la mezcla de dos redes ANTop, en el cual a priori una de las dos redes renombrará a la otra, es decir, que en un proceso de mezcla puede que el nodo sea afectado y cambie su dirección R, acompañado del cambio de vecinos y también del nodo *padre*. Ni más lejos, mismo en un proceso de fragmentación, se mencionó que la sub red que resulta fragmentada, también se renombrará y como consecuencia el nodo *padre* podría cambiar. Entonces, es necesario chequear constantemente el nodo *padre*, recorriendo la tabla de vecinos y comparando las

distancias a la dirección *root* del hipercubo (primera dirección del hipercubo) con la distancia que tiene el nodo *padre* actual con el *root* del hipercubo. En el caso que un vecino tenga menor distancia al *root* que el nodo *padre* actual, entonces el vecino en cuestión será designado como el nodo *padre*. Luego de actualizar e identificar el nodo *padre*, se recorre nuevamente la tabla de vecinos para chequear que el nodo *padre* actual esté en la tabla de vecinos. De no ser así, se deberá chequear por medio del proceso de exploración de enlaces secundario, algoritmo 15, la existencia de un camino al *root* del hipercubo. Si en este último no existe un camino al *root* por un enlace secundario, entonces efectivamente se produjo una fragmentación y se corre el mecanismo *Fragmentación* definido en el algoritmo 16 que se explicará en detalle más adelante.

Algorithm 14 Detector de Fragmentación

Require: paquete HB
Ensure: None
1: **if** *se recibió un paquete HB* **then**
2: Se procesa el paquete HB;
3: Se actualiza la tabla de vecinos;
4: **goto** linea 7;
5: **else** return;
6: **end if**
7: **for** *si no se llego al final de la tabla de vecinos* **do**
8: **if** *el vecino tiene menor distancia a la raiz que el padre actual* **then**
9: Agrego el vecino como padre;
10: **end if**
11: **end for**
12: **for** *si no se llego al final de la tabla de vecinos* **do**
13: **if** *si la dirección del vecino es igual a la dirección del padre* **then** flag=1;
14: break;
15: **else** flag=0;
16: **end if**
17: **end for**
18: **if** *flag no está seteado en uno* **then**
19: **if** *ExploracionEnlaceSecundario() devuelve 0* **then**
20: *Fragmentación()*;
21: return;
22: **end if**
23: **else** return;
24: **end if**

Algorithm 15 ExploracionEnlaceSecundario()

Require: None
Ensure: None
1: Enviar mensaje ICMP Echo Request a la dirección R compuesta todos los bits en cero;
2: Espera un tiempo TIMEOUT;
3: **if** *se recibió un mensaje ICMP Echo Reply* **then**
4: return 1;
5: **else**
6: return 0;
7: **end if**

4.2.2. Proceso de asignación de direcciones

Luego de haber definido la detección de fragmentación de una red ANTop, nos queda determinar que debe hacer la red fragmentada. En principio una red fragmentada con las direcciones R y máscara que fueron definida antes de la fragmentación, se podrían conservar y cumplir con las características de una red ANTop. De esto último, uno de los problemas es que la red fragmentada conserva el mismo *número de red* que la red original, y como se verá más adelante, este campo será importante que se diferencie entre redes independientes ante una posible futura mezcla de redes ANTop. Entonces es inevitable disparar un mecanismo que informe a todos los nodos de la red fragmentada para que cambie alguno de sus parámetros del protocolo. Con respecto a esto último, se podrá aprovechar el mecanismo para renombrar los nodos de la red fragmentada de tal forma que el nodo que detectó la fragmentación y disparó el mecanismo, se auto asigne la primer dirección del hipercubo. Luego este último que avise a los nodos subordinados para que cambie su dirección de una forma que se tratará en la siguiente explicación.

En la figura 4.9 se muestra una red ANTop, conformada por los nodos A,B,C,D y E. En el momento que el nodo C pierde conexión con el nodo A, los nodos C,D y E forman la red fragmentada.

El nodo C es el que detectará la fragmentación y será el encargado de disparar el mecanismo de fragmentación y auto asignarse la primer dirección del espacio del hipercubo, es decir, la dirección R 0000. El nodo C envía un mensaje a los nodos D y E para que se ajusten sus parámetros del protocolo. En el caso de un nodo subordinado al nodo que dispara el mecanismo de fragmentación, deberá asignarse una dirección R de tal forma que desplace la jerarquía del espacio de direcciones dependiendo del espacio que empezó a administrar el nodo que detec-

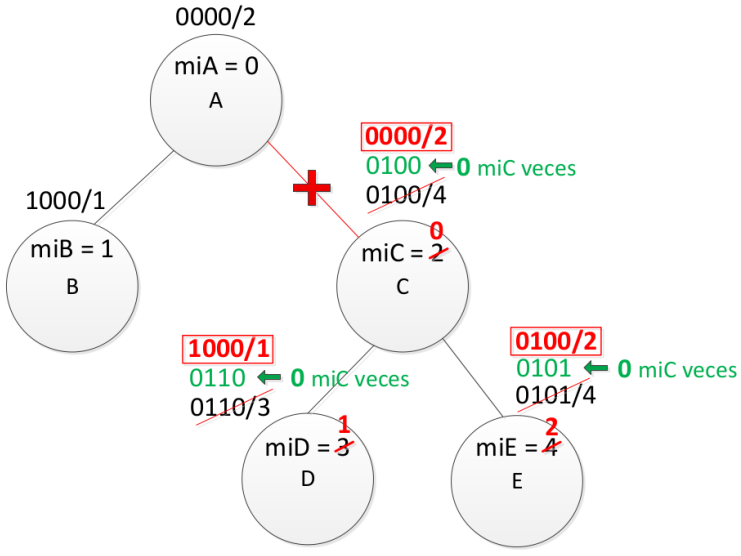

Figura 4.9: Direccionamiento de red ANTop fragmentada

to la fragmentación, en este caso el nodo C. Una forma es que las direcciones de los nodos subordinados se desplacen los bits a izquierda agregando ceros desde la derecha hacia la izquierda. Esta última operación se la conoce como la operación de *shift* de ceros a izquierda. La cantidad de operación *shift* de ceros a izquierda dependerá de un valor en particular que conceptualmente lo podemos llamar como *máscara inicial (mi)* del nodo que dispara el mecanismo de fragmentación.

La *mi* de cada nodo, es la máscara que el nodo obtuvo junto con la dirección R primaria cuando se conectó a la red por medio del proceso de conexión con el intercambio de paquetes PAR, PAP, PAN y PANC. Por ejemplo, la *mi* del nodo E es el valor 4, ya que es la máscara que le ofreció el nodo C cuando este se conectó a la red.

En el esquema planteado, las dirección que resulta en el nodo D es la dirección R 1000, que básicamente resulta de realizar un desplazamiento a izquierda agregando ceros la cantidad de *miC* veces. Lo mismo ocurre con el nodo E, que de la dirección 0101, realizando la operación de *shift* de ceros a izquierda la cantidad de *miC* veces, o sea 2 veces, se obtiene la dirección R resultante 0100.

Una de las ventajas de asignar nuevas direcciones R a los nodos de la red fragmentada, es de agrandar el espacio de direcciones para tener la máxima disponibilidad posible. También las máscara de cada nodo, se deberá setear para fijar el espacio de direcciones que pertenece en la nueva red resultante de la fragmentación. Por ejemplo, el nodo C, obtiene la máscara 2 ya que tiene dos nodos subordinados (D y E). El nodo D tiene máscara 1 ya que el espacio resultante con respecto al espacio del nodo C, difiere del primer bit. Por otro lado, se puede obtener el valor de la máscara resultante, por medio de una simple operación de resta entre la máscara antes de la fragmentación menos la *miC* del nodo C. En el caso del nodo E, se puede obtener la máscara resultante, restando la máscara 4 antes de la fragmentación, menos la *miC* del valor de 2, obteniendo así el valor de 2 la máscara resultante.

Luego del proceso de fragmentación, el valor de *mi* de cada nodo, se debe setear nuevamente, debido que en un futuro puede ocurrir alguna fragmentación o mezcla. En esta última situación, se verá más adelante que la *mi* juega un rol importante en el cálculo de los parámetros resultantes de cada nodo en una mezcla. Como se mencionó antes, cuando el nodo que detecta la fragmentación, amplía su espacio de direcciones las veces según el valor de *mi*, entonces todos los nodos subordinado deben de hacer lo mismo, obteniendo su *mi* luego de la fragmentación por medio de la operación de resta entre la *mi* y la *miC* antes de la fragmentación. Por ejemplo, el nodo D, la *miD* luego de la fragmentación se obtiene de la resta de 3 de la *miD* antes de la fragmentación y 1 de la *miC* también antes del proceso de fragmentación, o sea 1.

Tratamiento de dirección relativa y máscara En resumen el tratamiento que tendrán los nodos que pertenecen a la red fragmentada, estará descripta por los siguientes puntos. Al nodo que detecta la fragmentación y dispara el mecanismo de renombramiento de los parámetros, lo llamaremos *nodo detector*. En el caso de la figura 4.9, el *nodo detector* es el C. El *nodo detector* al detectar fragmentación, dispara el mecanismo de fragmentación enviando mensaje a los nodos subordinados, y luego estos últimos también lo hace con sus subordinados, hasta cubrir la estructura del árbol de direccionamiento. La información trasmitida es la máscara inicial del *nodo detector*.

1. La dirección R resultante se obtiene de la operación de *shift* de cero a izquierda de la dirección R del nodo, la cantidad de veces el valor de $mi_{\ nodo\ detector}$.

2. La máscara resultante, se puede calcular con la resta de la máscara propio del nodo antes de la fragmentación menos la $mi_{\ nodo\ detector}$.

3. La máscara inicial del nodo en cuestión, se obtiene realizando la resta entre la máscara inicial antes de la fragmentación menos la $mi_{\ nodo\ detector}$.

Paquetes de control En la figura 4.4 se definió la estructura de los paquetes de control utilizados en el protocolo. En el proceso de fragmentación, a partir del *nodo detector* y sus nodos subordinado recibirán mensaje de control que transmiten la información necesaria para que puedan cambiar su dirección R y máscara. Los paquete de control que se utiliza en el proceso de fragmentación sera FAR (*Fragment Address Request*) y FAN (*Fragment Address Notification*).

1. **Paquete FAR *(Fragment Address Request)***

   packet type Este campo toma el valor de 12.

   source address La dirección primaria del nodo antes de la fragmentación.

   primary address La dirección primaria del nodo que envía el paquete.

   optional headers Se incluye un encabezado opcional de tipo *Additional Address*.

   El encabezado opcional contiene los siguientes valores,

   address La dirección del nodo que recibirá el paquete.

   mask La máscara inicial del *nodo detector*.

   red number *Número de red* del *nodo detector*.

2. **Paquete FAN *(Fragment Address Notification)***

   packet type Este campo toma el valor de 13.

   source address La dirección primaria del nodo emisor del paquete, antes de la fragmentación.

optional headers Se envía un encabezado opcional de tipo *Additional Address*.

El encabezado opcional contiene los siguientes valores,

mask La máscara inicial del *nodo detector*.

red number *Número de red* del *nodo detector*.

Algoritmos de envío de mensajes de control

Fragmentación Los mensajes FAR y FAN, se deben enviar a todos los nodos de la red fragmentada, empezando desde el *nodo detector* hasta el final del árbol. En primer lugar se debe analizar el algoritmo que ejecuta el *nodo detector* al momento que detecta que la fragmentación ocurrió.

De la figura 4.10 se visualiza al *nodo detector* con sus vecinos de V1, V2 y Vn (n vecinos). Cuando detecta fragmentación, debe avisar a todo sus nodos vecinos, barriendo su tabla de vecinos y enviando a cada uno de ellos un mensaje FAR.

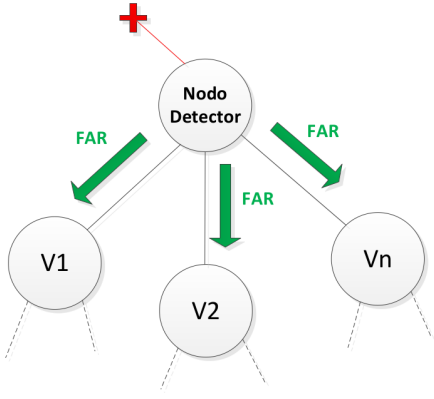

Figura 4.10: Envío de paquetes FAR desde el nodo detector

En primer lugar, el *nodo detector* debe de suspender el servicio de envío HB y los paquetes de registro y resolución a los servidores *Rendez Vous*. En caso con-

trario, con el cambio de dirección R, los paquetes HB son enviados con la nueva dirección como dirección origen, y entonces los vecinos que la reciban pueden pensar que se trate de un nuevo nodo y según el caso pueden generarse dirección secundaria. Luego, el algoritmo fragmentación 16, envía los paquetes FAR a aquellos vecinos que tienen distancia uno. En el caso de tener una dirección secundaria, pueden contener vecinos que estén a distancia uno de la secundaria, pero no de la primaria. La dirección R, máscara y máscara inicial, se asignan de acuerdo al tratamiento descripto en la sub sección 4.2.2.

Algorithm 16 Fragmentación

Require: None
Ensure: None
1: Suspender el envío y procesamiento de paquetes HB;
2: Suspender el servicio de registración y resolución de nombres *Rendez Vous*;
3: Asignar el *número de red* igual al cuarto byte de la dirección MAC;
4: Shift de ceros a izquierda la cantidad de máscara inicial a la dirección primaria;
5: $m = m - máscara\ inicial$;
6: Realizar el mismo tratamiento que la dirección primaria y m a la tabla de direcciones recuperadas;
7: Borrar tabla de rutas;
8: Enviar FAR a todos los vecinos que están a distancia mayor a 1 de la dirección secundaria;
9: Borrar tabla de nombres *Rendez Vous*;
10: Borrar tabla de vecinos;
11: Setear la máscara inicial en cero;
12: return;

Procesamiento FAR En el momento que un nodo reciba un paquete FAR, debe seguir con el envío hasta el final del árbol. Por ende, en el procesamiento del FAR, debe de enviar en principio el paquete FAR a todos los vecinos subordinados si es que tiene, y también el paquete de confirmación FAN al nodo del quien recibió el FAR.

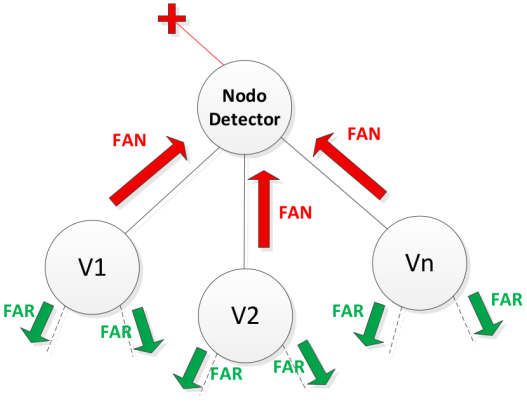

Figura 4.11: Envío de paquetes FAR y FAN de nodos intermedios.

En la figura 4.11 se visualiza los nodos vecinos que recibieron el paquete FAR del *nodo detector* y envía paquetes FAR a sus subordinados. También tiene el deber de notificar al *nodo detector* con el un mensaje FAN de notificación. El algoritmo 17, muestra el procesamiento del FAR como también resuelve la dirección R y máscara que se debe asignar cada nodo.

Procesamiento FAN El nodo que envía un mensaje FAR, como se indico en el algoritmo 17, debe de suspender el servicio de envío HB y los paquetes de registro y resolución a los servidores *Rendez Vous*. La única manera de que se restablezca, es por medio de la recepción de un mensaje FAN de algún nodo subordinado. En el caso que un nodo contenga una dirección secundaria, deberá esperar a que termine el proceso de asignación de la nueva dirección R y máscara, tanto la primaria como la secundaria, para restablecer dichos servicios. Esto último se muestra en el algoritmo 18 de procesamiento de FAN.

Algorithm 17 Procesamiento FAR

Require: Paquete FAR
Ensure: None
Suspender el envío y procesamiento de paquetes HB;
Suspender el servicio de registración y resolución de nombres *Rendez Vous*;
if la dirección destino del FAR es la dirección secundaria **then**
    Enviar confirmación FAN al nodo que envío FAR;
    Shift de ceros a izquierda la cantidad de máscara del paquete FAR a la dirección secundaria;
    $m_{secundaria} = m_{secundaria} - máscara\ FAR$;
    if el tratamiento de dirección primaria ocurrió **then**
        Reanudar servicio de HB y *Rendez Vous*;
    end if
    return;
end if
Agregar como vecino y nodo padre la dirección del nodo que envió FAR;
Enviar confirmación FAN al nodo que envío FAR;
Asignar el *número de red* del campo del paquete FAR;
Shift de ceros a izquierda la cantidad de máscara del paquete FAR a la dirección primaria;
Realizar el mismo tratamiento que la dirección primaria y m a la tabla de direcciones recuperadas;
Borrar tabla de rutas;
Borrar tabla de nombres *Rendez Vous*;
if tiene vecinos subordinados a distancia 1 de la dirección primaria antigua **then**
    Enviar FAR a todos los vecinos que están a distancia 1 de la dirección primaria antigua;
end if
Borrar tabla de vecinos excepto el nodo padre;
$m = m - máscara\ FAR$;
$mi = mi - máscara\ FAR$;
if no se envío paquete FAR **then**
    if no tiene dirección secundaria o el tratamiento de la dirección secundaria ocurrió **then**
        Reanudar servicio de HB y *Rendez Vous*;
    end if
end if
return;

Algorithm 18 Procesamiento FAN

Require: Paquete FAN
Ensure: None
1: **if** si aún no se restableció el servicio de HB y *Rendez Vous* **then**
2: **if** no tiene dirección secundaria o el tratamiento de la dirección primaria y secundaria ocurrió **then**
3: Reanudar servicio de HB y *Rendez Vous*;
4: **end if**
5: **end if**
6: return;

4.3. Mezcla

Al principio del capítulo fundamentamos el uso de un nuevo concepto y la necesidad de numerar las redes ANTop que son gestionadas en forma independientes. Si volvemos al ejemplo de la figura 4.2, se pueden observar dos redes ANTop que luego son mezcladas como muestra en la figura 4.3. El análisis que se realizó a los efecto de la mezcla de dos redes ANTop fueron sin tener en cuenta la numeración de las redes y la posibilidad de disparar un mecanismo que mitigue los problemas que se detectaron en materia de mezcla.

Entonces como primer paso en esta sección, se definirá la detección de mezcla con otra red en la sub sección 4.3.1. Por último, en la sub sección 4.3.2 se define el proceso de asignación de direcciones en la mezcla.

4.3.1. Definición y detección

En la siguiente figura 4.12, se encuentran dos redes ANTop independientes y numeradas. En cada nodo tiene identificada la dirección primaria, la máscara y el *número de red* que es igual a todos los nodos que pertenecen a la misma red. Por ejemplo, el nodo B tiene la dirección primaria 1000, la máscara 1 y el *número de red* $3d$.

Como hicimos en la primer sección de este capítulo, vamos a suponer que uno de los nodos se desplaza físicamente hasta lograr la mezcla de ambas redes. Si el nodo Y se desplaza sin perder vinculo de comunicación con el nodo Z y a su vez entra dentro del alcance del nodo C perteneciente a otra red ANTop, va a intercambiar mensajes de control ANTop pero ahora con un extra de información aportada por el campo *número de red* alojada en el encabezado adicional, que nos permitirá saber a que red pertenece y armar una lógica de decisión según el caso que se presente.

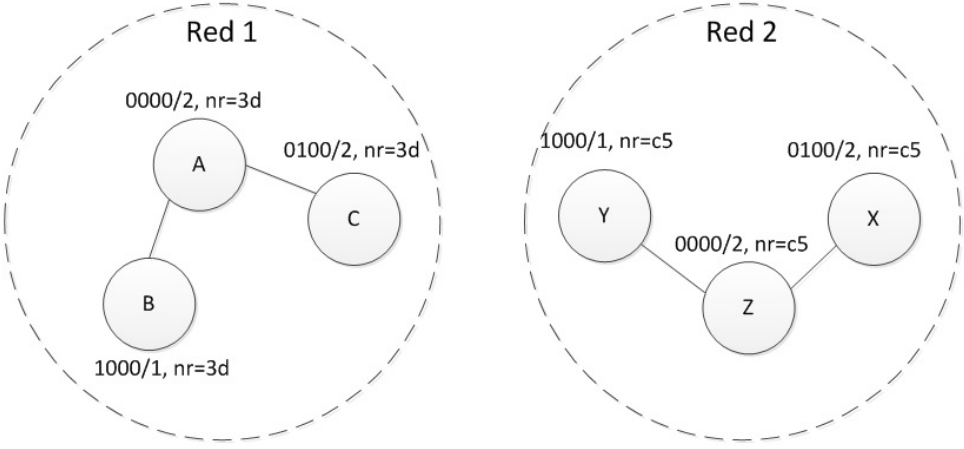

Figura 4.12: Dos redes ANTop independientes y numeradas

Recordando que cada nodo envía periódicamente el paquete de control HB en modo broadcast, en donde cualquier nodo que esté en el radio de alcance pueda recibirlo y procesarlo. En el momento que el nodo Y reciba el mensaje HB del C como se muestra en la figura 4.13, procesa dicha información y se puede llegar a plantear un algoritmo que analice los números de redes de los paquetes HB y cuando este sea diferente al *número de red* local, dispare el mecanismo de mezcla que básicamente consistirá en el renombramiento de direcciones de una red ANTop a otra. El proceso de renombramiento propuesto se tratará más adelante en el capítulo.

Si imaginamos que al mismo tiempo el nodo Y envía un paquete HB en modo boradcast antes de procesar el paquete HB recibido del nodo C, este último también recibirá un paquete HB con un *número de red* diferente al del él mismo. El mecanismo de renombramiento de una red a otra consistirá de que una de las dos tome el papel de renombrar a la otra, y esta última dejarse renombrar. Entonces si volvemos al esquema de la figura 4.13, en el momento que el nodo Y procese el paquete HB, tiene que decidir si comienza el mecanismo de renombramiento de direcciones a la Red 1, o espera que el nodo C procese el paquete HB de él mismo y comience el mecanismo de renombramiento a la Red 2.

En conclusión con el último análisis, es necesario todavía definir en la detección de mezcla que red renombra a la otra. Idealmente se podría pensar que la red

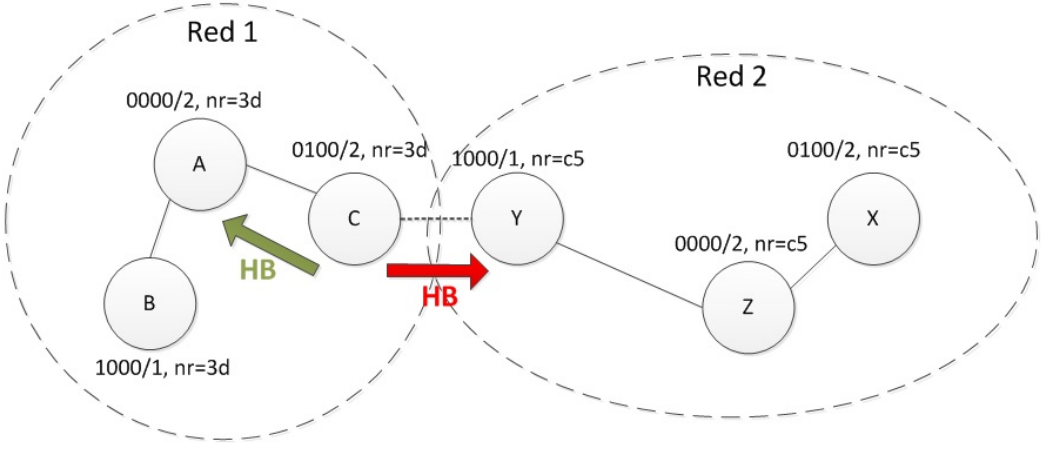

Figura 4.13: Mezcla de dos redes ANTop numeradas

ANTop con mayor cantidad de nodos asociados, renombre a la red ANTop con menor cantidad de nodos asociados. Para ello se necesitará saber la cantidades de nodos de ambas redes. Una manera es estimar la cantidad de nodos de una red a partir de la información que puede aportar el nodo que detecta la mezcla con otra red. En el caso que vimos anteriormente, sería el nodo C para la Red 1, y el nodo Y para la Red 2.

La estimación de la cantidad de nodos, se puede realizar a partir de la información del valor de la máscara del nodo que detecta la mezcla y la cantidad de nodos que tienen registrado aquellos nodos que tiene el papel de servidor de nombres *Rendez Vous*. Luego con dicha información, se deberá intercambiar los datos recolectados entre el par de nodos que se produjo la mezcla y recién ahí decidir quien renombra a quien. Si bien dicha idea se descarto en el trabajo presente, ya que se cuenta con intercambio de paquetes de control para solo reunir información y tener una estimación de la cantidad de nodos que puede diferir de la real. Además dicho estimador se deberá diseñar con bases de probabilidad y estadísticas, que escapa al alcance de este trabajo de tesis, pero que abre la posibilidad de ser tratado en futuros trabajos.

En la decisión de cuales de las redes ANTop dispara el mecanismo de renombramiento en la mezcla, una opción podría ser de forma aleatoria.

El *número de red* de cada red ANTop es el valor del cuarto byte de la dirección MAC de la placa wireless del nodo que tiene la primer dirección de hipercubo. Si recordamos que el valor correspondiente al cuarto, quinto y sexto byte de la dirección MAC de un dispositivo de red, es asignada por el fabricante y tiene correspondencia con el número de serie que a su vez da información de la planta de fabricación, lote al que perteneció, etc. Si ahora pensamos en el dinamismo de una red ANTop en donde las conexiones entre nodos varían en el tiempo, se puede asumir que en el momento de la mezcla de dos redes ANTop los *número de red* que entran en juego pueden resultar en forma aleatoria. Entonces la decisión de que red renombra a la otra puede ser determinada a partir de un simple condición. Por ejemplo, en mezcla de dos redes ANTop, la red que tiene mayor *número de red*, es la que será renombrada por la red que tiene menor *número de red*. Este último ejemplo es el que tomaremos como condición de decisión en el proceso de mezcla.

El algoritmo 19, detector de mezcla, corre siempre que se procesa un paquete HB y según el caso, devuelve un 1, 0 o -1. La función *mezcla()* es la que se encarga de iniciar el procesos de asignación de direcciones en el cual se verá en detalle más adelante.

Algorithm 19 Detector de Mezcla

Require: *Número de red* del paquete HB
Ensure: None
1: **if** *número de red local es menor al número de red del paquete HB* **then**
2: mezcla();
3: return 1;
4: **end if**
5: **if** *número de red local es igual al número de red del paquete HB* **then**
6: return 0;
7: **end if**
8: **if** *número de red local es mayor al número de red del paquete HB* **then**
9: return -1;
10: **end if**

4.3.2. Proceso de asignación de direcciones

Definido la detección de la mezcla de dos redes ANTop, en esta sección se tratará el mecanismo de mezcla que será ejecutada por algunas de las redes. En

principio se planteará la estrategia que se tomará para mitigar dicha mezcla.

Una de las opciones que se comento antes, es la posibilidad de que una red renombre a la otra para que se mantenga la estructura de direcciones R basados en hipercurbo, cuyo concepto es la esencia del protocolo ANTop.
En el esquema mostrado anteriormente de la figura 4.13, en el momento que se mezcla las dos redes ANTop por medio del par de nodos C-Y, según el mecanismo de detección visto anteriormente, el nodo C activa el mecanismo de mezcla renombrando a la Red 2 según la jerarquía de asignación de dirección del mismo a partir de su dirección R 0100/2. El resultado de la mezcla se muestra en la figura 4.14, en donde se agrego un bit más en las direcciones para mostrar en detalle las direcciones asignadas.

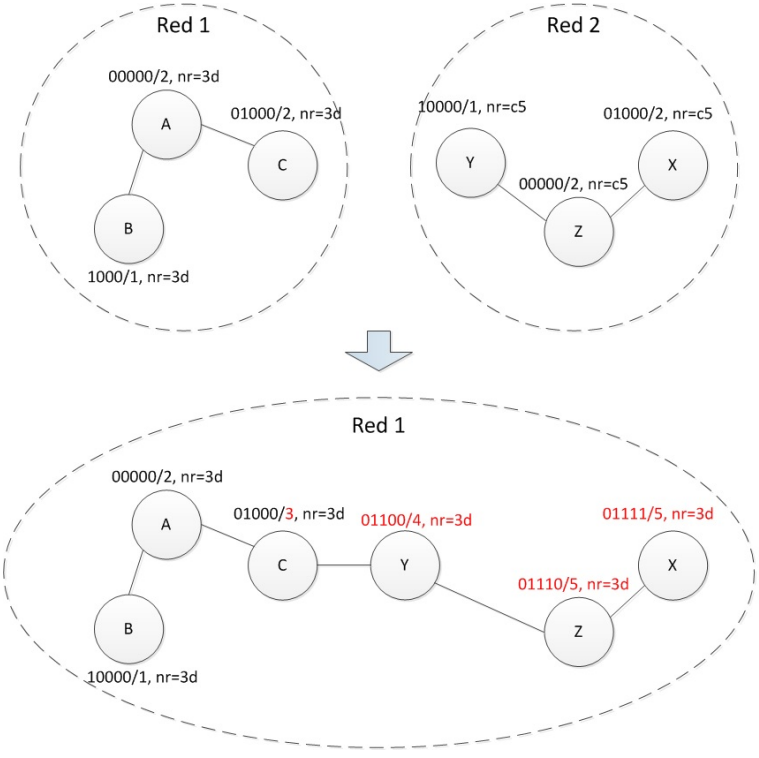

Figura 4.14: Resultado propuesto de la mezcla de dos redes ANTop

En la mezcla se muestran los cambios en color rojo, tanto de direcciones R como también de máscara y *número de red* para los nodos que integraban la Red 2 y el nodo C en particular de la Red 1. La máscara cambian ya que deben respetar el espacio de direcciones asignados según el nodo que le antecede y le sucede. Por ejemplo, el nodo C al ceder el espacio de direcciones 01100 con máscara 3 a Y, debe aumentar en 1 su máscara dando como resultado máscara 3. Luego Y resulta de máscara 4 porque le cede el espacio de direcciones 01110 y máscara 4 al nodo Z. Y así sucesivamente hasta barrer todos los nodos de la Red 2.

El resultado es similar al proceso de asignación de dirección dado primero por la conexión del nodo Y enviando un paquete PAR en modo broadcast y el nodo C asignandole una dirección con un mensaje PAP. Luego el proceso es completado con los mensajes PAN y PANC. Se repite el mismo proceso cuando el segundo nodo en conectarse sea el Z que le llega una propuesta de dirección R a partir de la recepción del mensaje PAP del nodo Y. Y así también la conexión del nodo X con el Z.
Sin embargo lo último descripto, sería como la interpretación del resultado similar a la del mecanismo de mezcla que se propone en esta sección. Es decir, que en la mezcla no habrá paquetes de control PAR, PAP, PAN o PANC para el renombramiento de una de las dos redes, sino que se diseñan nuevos paquetes de control que se definen más adelante.

En el proceso de asignación de direcciones se dividen en dos vetas, es decir, que el mecanismo de mezcla en realidad estará compuesto por dos mecanismos distintos en el que a uno lo llamaremos *mecanismo 1* 4.3.2 y al otro *mecanismo 2* 4.3.2.

Mecanismo 1

Para explicar el mecanismo 1, se plantea los siguientes casos para renombrar una red ANTop en la mezcla. Dichas propuestas irán evolucionando en cada una de ellas.

Primer Caso La idea es encontrar una solución para renombrar ciertos nodos de una red en el proceso de mezcla, sin que la topología y el orden de la estructura de direcciones R formadas entre sí cambie lo menos posible. El primer caso se muestra en la figura 4.15.

Analizando la mezcla de la Red 1 con la Red 2, se observa que ocurre por medio del par de nodos A y B. Se puede suponer que en dicha mezcla la red que será renombrada es la Red 2. Inicialmente en la Red 1 el nodo A tiene asignada la

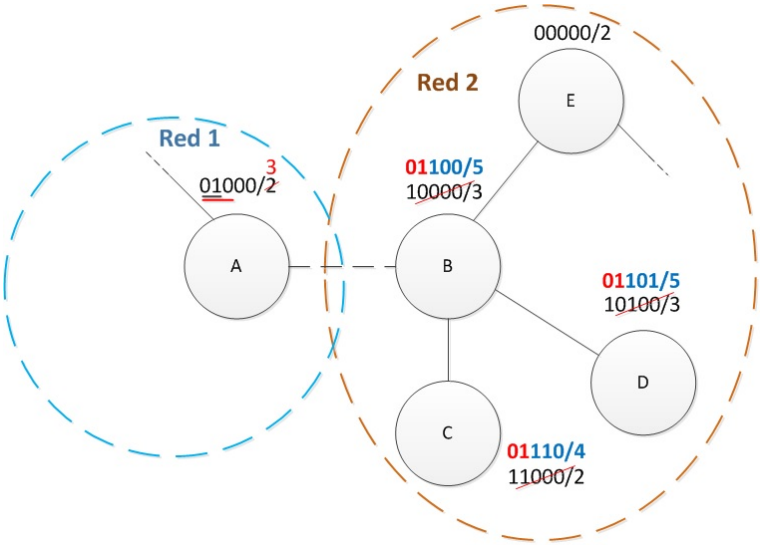

Figura 4.15: Primer caso de tratamiento de dirección del mecanismo 1 de mezcla

dirección R 01000 con máscara 2. La Red 2 a su vez se muestran varios nodos para analizar que direcciones se podrían asignar luego de la mezcla. Se supone que en el momento que el nodo A reciba el paquete HB del B, este ejecuta el mecanismo 1 y le informa al nodo B para que cambie de dirección R de una manera en particular que se analizará en la siguiente explicación.

En la topología del ejemplo planteado, como resultado de la nueva dirección del nodo B, está escrita en color rojo y azul arriba de la dirección 10000/3 antes de la mezcla. Se puede observar que está compuesta por dos cifras diferenciadas por los colores. El prefijo en rojo 01 coincide con el prefijo de la dirección 01000 con máscara 2 del nodo A. Luego el resto de la cifra en azul, coincide con la dirección del nodo mismo antes de la mezcla. Haciendo el mismo análisis para los nodos subordinado del nodo B, vemos que podemos aplicar el mismo criterio. Solo el nodo E no cumpliría con la estrategia mencionada, ya que el resultado de la dirección R luego de la mezcla sería 01000 que coincide con al dirección R del nodo A. Entonces el criterio mencionado funciona para los nodos subordinado del nodo que forma parte del par que se establece la mezcla. En este caso el B, y sus subordinados C y D. Esta solución nos permite que las direcciones por los menos de los nodos

subordinados permanezcan con la misma relación de distancias y jerarquía entre sí.

Para el caso del nodo E, hay otro tratamiento de renombramiento de direcciones que se verá en el mecanismo 2.

Volviendo al esquema propuesto de la figura 4.15, las máscaras resultantes en los nodo B, C y D, se pueden obtener sumando su máscara de cada uno antes de la mezcla, con la máscara del nodo A antes de la mezcla. Por ejemplo, inicialmente el nodo C tiene máscara 2. Luego de la mezcla, suma la máscara 2 del nodo A antes de la mezcla, obteniendo como resultado la máscara 4. Si se observa la máscara del nodo A después de la mezcla, resulta de ser 3 ya que en la mezcla toda la red quedará en el espacio de dirección dependiente del nodo C, por ende éste debe de ceder el espacio aumentando en uno su máscara.

Segundo caso En el primer caso, el nodo E tiene la primera dirección del espacio de dirección por hipercubo, es decir, todos los bits significativos en cero. En el segundo caso el nodo E tiene un valor 10000/3 como se muestra en la figura 4.16.

Si aplicamos el mismo criterio del primer caso al nodo B, luego de la mezcla como resultado se obtiene la dirección R 01110. Calculando la distancia entre el nodo A y la dirección resultante del nodo B, resulta de distancia 2. La idea en la mezcla es que en el par de nodos que entran en juego y desencadenan la mezcla, estén a distancia 1, en donde no ocurre con el criterio del caso anterior. Entonces en este caso, se centra primero en realizar alguna operación antes de aplicar el criterio del primer caso.

El mecanismo 1 en el esquema de la figura 4.16, solo será aplicado a los nodos B, C y D. Para que el nodo B sea un subordinado en dirección R del nodo A, tiene que cumplir con dos características: Que la resultante de la dirección R tenga el prefijo de la dirección del nodo A hasta el valor de la máscara de este último, y que esté a distancia uno de dicho nodo A.
Una de la maneras es que cada nodo realice un *shift* de cero a la izquierda la cantidad de veces según el valor de la *máscara inicial (mi)* del nodo B.

El concepto de *mi* fue visto en la sección de fragmentación 4.2, igualmente se repasa el significado. La *mi* de cada nodo, se refiere a la máscara que el nodo obtuvo junto con la dirección primaria R cuando se conectó a la red por medio del proceso de conexión de nodos con el intercambio de los paquetes PAR, PAP, PAN y PANC. Por ejemplo, la *mi* del nodo B es la máscara que le cedió el nodo

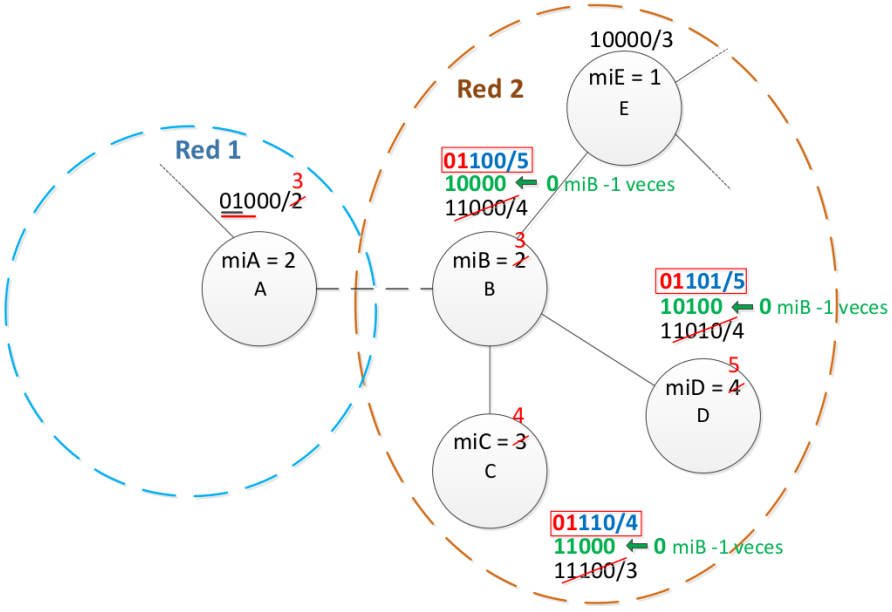

Figura 4.16: Segundo caso de tratamiento de dirección del mecanismo 1 de mezcla

E cuando se conectó a la Red 2. Es decir, el valor de máscara 2.

Volviendo a la idea del *shift* de cero a la izquierda de la dirección R de cada nodo, la cantidad exacta de esta operación sería del valor de *miB-1* del nodo B. Por ejemplo, el nodo C, luego de la cantidad de $(2-1)$ operaciones *shift* de cero, se obtiene 11000, que luego aplicando el criterio del primer caso visto, obtenemos la dirección R resultante 01110.
Si se realiza el mismo procedimiento para el nodo B y D, se obtienen los mismos resultado que en el primer caso.

La cantidad de *miB-1* ceros a izquierda, no se obtuvo por casualidad, sino que tiene una explicación lógica. Cuando a una dirección se realiza *shift* de cero a izquierda, a su vez está subiendo en la jerarquía de direcciones de la estructura por hipercubo. Es decir, que la dirección cada vez tiene menor distancia a la dirección de todos los bits en cero. Por ejemplo, el nodo B en este caso, tiene dirección R 11000 y cuyo nodo que le antecede es el 10000. Luego de *shift* de cero de la

cantidad de 1, el nodo B coincide con la dirección de su antecesor, y se podría trasladar cuyo resultado al primer caso mostrado en la figura 4.15. Por ende, la cantidad de *miB-1* operaciones *shift* de cero a la izquierda, en realidad se está trasladando el caso presente al primer caso visto anteriormente.

Recapitulando, las direcciones R de cada nodo consistirá primero de realizar *shift* de cero a la izquierda la cantidad de *miB-1*. Esta operación nos permite eliminar el prefijo que no es relevante para nuestra futura nueva dirección, o también se puede pensar que es la forma de trasladar el caso presente al primer caso. Luego se realiza el proceso visto en el caso anterior, en donde se vuelve a realizar *shift* de cero pero esta vez a la derecha y la cantidad del valor de la máscara del nodo A antes de le mezcla. Por último se realiza una operación OR con la dirección del nodo A.

Aún no se analizó el valor resultante de las máscara y la *máscara inicial (mi)* de los nodos luego en la mezcla. La máscara es indudable que debemos intervenir para saber cual sería su resultante después de la mezcla. Lo mismo ocurre con la *mi*, ya que como mencionamos antes, la red resultante se podría pensarse como si los nodos se fueran conectando uno a uno, empezando primero el B conectándose con el A, luego el C con el B, y por último el D con el B. Entonces las *máscara inicial* se deben setear nuevamente, para que si en un futuro vuelva a ocurrir una mezcla o fragmentación, puedan ejecutarse correctamente los algoritmos desarrollados en este capítulo.

Primero se analiza la máscara resultante de cada nodo. En el primer caso de la figura 4.15, la máscara luego de la mezcla se podía obtener con la suma de la máscara antes de la mezcla, más la máscara del nodo A también antes de la mezcla. Pero ahora hay que considerar el concepto de la operación extra que trató en este caso.
Cuando se menciono que para obtener la dirección R resultante, en primer medida es necesario realizar el *shift* de cero a izquierda la cantidad de *miB-1* veces. En realidad lo que se está realizando es que se aumenta el espacio de direcciones la cantidad de *miB-1*, es decir, que la máscara resultante ahora hay que restarle *miB-1* a la suma de la máscara local y máscara del nodo A antes de la mezcla. La máscara después de la mezcla se puede obtener aplicando la siguiente ecuación 4.1.

$$m_{mezcla} = mA + m - (miB - 1) \tag{4.1}$$

Por ejemplo, en la figura 4.16 y aplicando la ecuación 4.1, la máscara del nodo D luego de la mezcla resulta del siguiente desarrollo.

$$mD_{mezcla} = 2 + 4 - (2 - 1)$$
$$mD_{mezcla} = 5$$

El concepto de *mi* tiene un resultado interesante relacionadas con las diferencias de máscara respectivas de un nodo con otro. En la formación de la Red 2, el nodo E con *miE* de 1, termina con máscara 3 luego de asignar una de sus direcciones al nodo B. Si realizamos la cuenta de *m* - *mi*, se puede obtener la cantidades de nodos que éste le cedió su espacio de direcciones. Entonces el nodo E cedió espacio de direcciones a $(3 - 1)$ nodos, o sea 2. Y ahora si realizamos la diferencia de las *mi* del nodo E con el nodo B, obtenemos el número 1, es decir, que el B fue el primer nodo que el E le cedió su espacio de direcciones. Este último resultado también se puede interpretar como que el espacio direcciones que tiene el B para ceder, difiere solo de un bit de espacio de direcciones con respecto al espacio de direcciones en total que tenia el nodo E para ceder.
Por ejemplo, el nodo B con *miB* de 2, al primero en ceder el espacio de direcciones es al nodo C. Es decir que este tiene para ceder el espacio de direcciones con máscara igual a la *miB* del nodo B pero con un bit más de máscara. Con el nodo D pasa lo mismo, solo que el espacio que le cede el B es de la cantidad de máscara igual *miB* pero ahora con dos bits más de máscara ya que fue el segundo nodo que el nodo B cedió espacio de direcciones. Entonces si se realiza la diferencia entre las *mi* se puede saber los espacio relativos que el nodo de mayor jerarquía le cede a sus subordinados. Entonces en este caso, habiendo calculado la *miB* después de la mezcla, luego podemos obtener las *mi* de C y D resultantes de la mezcla.
El *miB* luego de la mezcla, se puede calcular fácilmente mediante la máscara del nodo A. Simplemente consistirá de incrementar en 1 el valor de la máscara del A, entonces el *miB* resulta del valor de 3 luego del proceso de mezcla. Para lo nodos subordinado a B, se calcula el *mi* sumándole la diferencia del *mi* del nodo con *miB* antes de la mezcla. Se ilustra con más detalle en la ecuación 4.3.

$$mi_{mezcla} = mA + 1 + (mi - miB) \tag{4.3}$$

Por ejemplo, en la figura 4.16 y aplicando la ecuación 4.3 la *miC* del nodo C luego de la mezcla se obtiene el siguiente resultado.

$$miC_{mezcla} = 2 + 1 + (3 - 2)$$
$$miC_{mezcla} = 4$$

Tercer caso Falta un caso más por analizar, y es cuando el nodo B tiene la dirección R con todos los bits en cero como se muestra en la figura 4.17. Todos los nodos de la Red 2 sin contar el B, serán subordinados de este último, por ende el tratamiento de direcciones que veremos será valida para todos los nodos de dicha red.

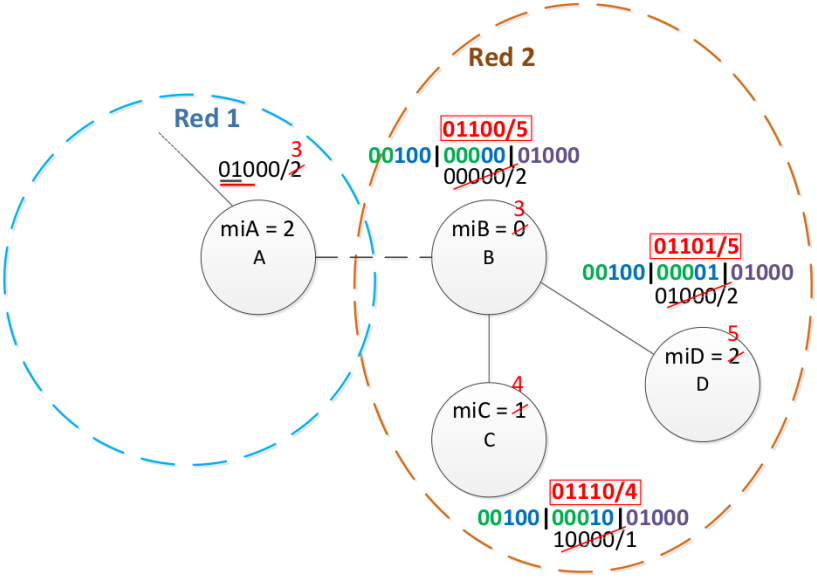

Figura 4.17: Tercer caso de tratamiento de dirección del mecanismo 1 de mezcla

El criterio del primer caso no se podrá aplicar a este esquema, ya que el nodo B al realizar el *shift* de cero a derecha de la cantidad de máscara 2 del nodo A, obtendremos la misma dirección 00000. Y cuando se agregue el prefijo 01 del nodo A, se tendrá como resultado la misma dirección del nodo A.

Como primera medida de debe crear una dimensión por medio del vector 10000. Dicho vector tiene la característica de poseer el bit más significativo en 1, y luego se podrá utilizar en cada nodo, desplazándolo con la operación *shift* a

derecha y crear la dimensión faltante.

La estrategia consistirá primero de tomar el vector 10000 y realizarle al operación de *shift* de cero a derecha la cantidad de veces la máscara del nodo A. Luego se vuelve a realizar la operación de *shift* de cero a derecha a la dirección R actual de cada nodo, la cantidad de veces la máscara del nodo A más uno. Por último, se realiza la operación OR en entre las dos direcciones que obtuvimos más la dirección R del nodo A.
Por ejemplo, el nodo D se obtiene su dirección R luego de la mezcla de la siguiente operación.

$$\textit{00100} \mid \textit{00001} \mid \textit{01000} \rightarrow \boldsymbol{01101}$$

1. *00100*: El vector 10000, se realiza la operación de *shift* de cero hacia derecha la cantidad de veces la máscara 2 del nodo A.
2. *00001*: la dirección R actual del nodo D 01000, se realiza la operación de *shift* de cero la cantidad de veces la máscara 2 del nodo A más 1, osea 3.
3. *01000*: Dirección R del nodo A.

El último ejemplo se puede aplicar para todos los nodos de la Red 2, y se podrá observar que cumple con lo mostrado en la figura 4.17.

Los valores de las máscara y *mi* de cada nodo, se podrá aplicar las mismas ecuaciones 4.1 y 4.3 del segundo caso.

Tratamiento de dirección relativa y máscara En resumen, el mecanismo 1 consiste en informar sobre la mezcla a todos los nodos subordinados al nodo asociado al par que genera la mezcla. Este último lo llamaremos *par receptor* y pertenece a la red que será renombrada. El otro nodo del par se denomina como *par emisor*. En los casos planteados, el *par emisor* es el nodo A, y el *par receptor* es el nodo B.
El mecanismo 1 a parte de notificar que ocurrió una mezcla, también transmite información necesaria a los nodos para que calculen y se reasignen la nueva dirección R, máscara y *mi*. La información trasmitida es la dirección R, máscara y *número de red* del *par emisor*, y también la máscara inicial del *par receptor*.

El tratamiento de la dirección R y máscara en cada nodo en el mecanismo 1, se diferencia según la dirección R del *par receptor*. Es decir, que si el *par receptor*

tiene la dirección R distinta a la de los bits todos en cero, se hará el tratamiento de direcciones en todos los nodos según el criterio del segundo caso. En cambio si la dirección R del *par receptor* tiene la dirección R con todos los bits en cero, se aplica el criterio del tercer caso.

***par receptor* con dirección R distinta a todos los bits en cero**

1. De la dirección R se realiza la operación de *shift* de cero a izquierda la cantidad de veces de $(mi_{\ par\ receptor} - 1)$
2. El resultado del punto 1, realizar la operación de *shift* de ceros a derecha la cantidad de veces igual a la máscara del nodo *par emisor*.
3. Realizar la operación OR del resultado del punto 2, con la dirección R del nodo *par emisor*. Se obtiene la dirección R final del proceso de mezcla.
4. La máscara resultante se puede calcular con la ecuación 4.5. Siendo todos los parámetros antes de la mezcla.

$$m_{mezcla} = m_{\ par\ emisor} + m - (mi_{\ par\ receptor} - 1) \tag{4.5}$$

5. La máscara inicial del nodo en cuestión, se obtiene con la ecuación 4.6. Todos los parámetros presente son antes del proceso de mezcla.

$$mi_{mezcla} = m_{\ par\ emisor} + 1 + (mi - mi_{\ par\ recpetor}) \tag{4.6}$$

***par receptor* con dirección R igual a todos los bits en cero**

1. Se toma la dirección con todos los bits en cero y se setea en 1 el bit más significativo en el espacio de direcciones.
2. La dirección obtenida en el punto 1, se realiza la operación de shift de cero a derecha la cantidad de veces igual a la máscara del *par emisor*.
3. Se toma la dirección R del nodo en cuestión y se realiza la operación de *shift* de cero a derecha la cantidad de veces igual a la máscara del *par emisor* más 1.
4. Se realiza la operación OR entre las direcciones del resultado del punto 2 y 3, y la dirección R del *par emisor*. Se obtiene la dirección R resultante de la mezcla.
5. La máscara del nodo, se obtiene aplicando la ecuación 4.5.
6. La máscara inicial se calcula aplicando la ecuación 4.6.

Paquetes de control En los trabajos de [2] y [4], se diseñaron paquetes de control del protocolo, que gestionan y controlan los distintos servicios con los que ANTop cuenta.

Anteriormente en la figura 4.4, se asigno un nuevo campo *número de red* que es utilizado en la detección de mezcla y fragmentación como se discutió en el presente capítulo. Sin embargo en la implementación desarrollada en el trabajo [4], el encabezado opcional tienen otro campos opcionales que no están habilitados, como ser el campo *Length* que fue pensado para dar información de la dimensión del mismo encabezado opcional. En el trabajo presente, el campo *Length* es habilitado dado que sería útil para transmitir cierta información en el mecanismo 1 planteado. El paquete de control genérico del protocolo ANTop quedaría definido por la estructura de la figura 4.18.

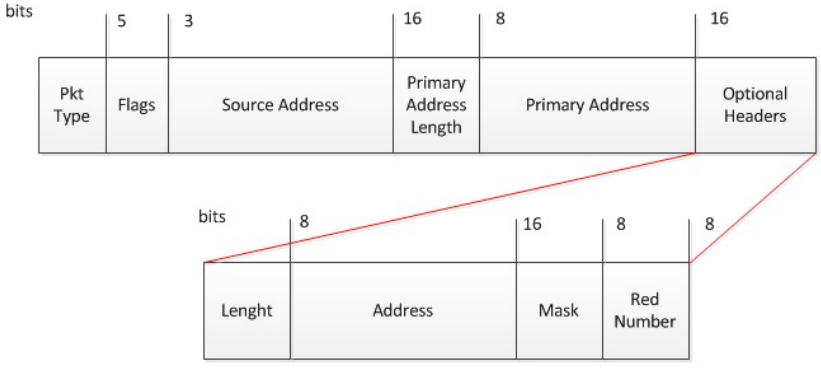

Figura 4.18: Paquete de control genérico con encabezado opcional con el campo "Lengh" habilitado.

Para el mecanismo 1, se diseñan dos paquetes de control que permitirá transmitir los mensajes necesarios para notificar y dar directivas a los nodos que deben cambiar sus direcciones y máscara en un proceso de mezcla. Se definen los campos para los paquetes MAR1 *(Mix Address Request 1)* y MAN1 *(Mix Address Notification 1)*.

1. **Paquete MAR1** ***(Mix Address Request 1)***

   packet type Este campo toma el valor de 14.

source address La dirección primaria del nodo *par emisor*.

primary address La dirección primaria del nodo que envía el paquete.

optional headers Se incluye un encabezado opcional de tipo *Additional Address*.

El encabezado opcional contiene los siguientes valores,

length El valor de la máscara inicial (mi) del nodo *par receptor*.

address La dirección del nodo que recibirá el paquete.

mask La máscara del nodo *par emisor*.

red number *Número de red* del *par emisor*.

2. **Paquete MAN1** ***(Mix Address Notification 1)***

packet type Este campo toma el valor de 16.

source address La dirección primaria del nodo emisor del paquete, antes de la mezcla.

optional headers Se envía un encabezado opcional de tipo *Additional Address*.

El encabezado opcional contiene los siguientes valores,

mask La máscara del nodo *par emisor*.

red number *Número de red* del *par emisor*

Mecanismo 2

En la figura 4.15 del primer caso del mecanismo 1, el nodo E es antecesor del nodo B en donde se produce la mezcla. Como se comento, el mecanismo 1 contempla a aquellos nodos que son subordinados del nodo B, por ende, el nodo E queda fuera del alcance de dicho mecanismo. Por tal motivo se desarrolla el mecanismo 2, que se enfoca en resolver las direcciones R de todos aquellos nodos que no son subordinados al *par receptor* en una mezcla de dos redes ANTop.

En la figura 4.19, tenemos el caso de una mezcla entre el nodo A y B. Tanto el nodo B y C, corren el mecanismo 1 y se asignan las direcciones y máscara según en el primer caso. Para los nodos que pertenecen a la rama del nodo E desde el nodo B, tendrán un tratamiento particular en la dirección y máscara. En este caso el mecanismo 2 tendrá efecto sobre los nodos E y F.

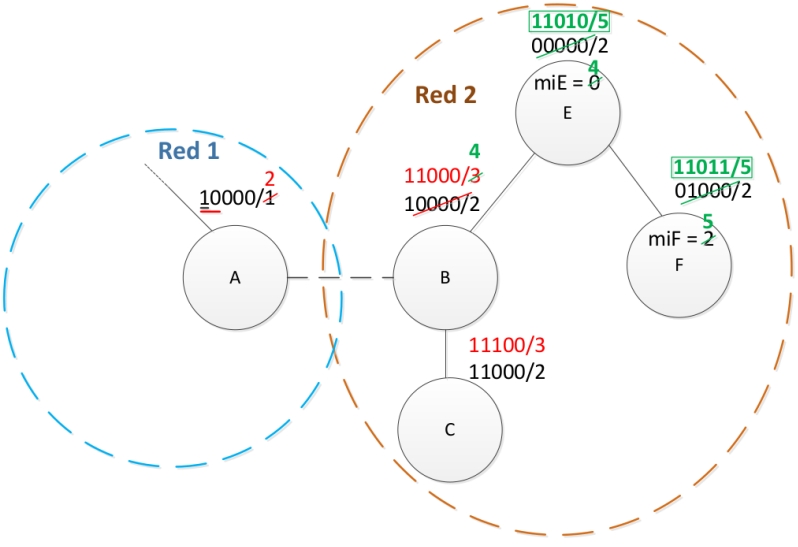

Figura 4.19: Tratamiento de direcciones mecanismo 2

Cuando el nodo B notifica a su vecinos sobre la existencia de la mezcla, los mensajes que le transmitirá a C será distinto el mensaje que le transmitirá a E. En primer lugar el nodo C necesita cierta información propia del mecanismo 1, en cambio el nodo E recibirá otro tipo de información acorde al mecanismo 2.

En primer lugar vamos a describir el tratamiento de las direcciones sobre los nodos E y F, que tiene como resultado en la figura 4.19. El mecanismo 2, propone básicamente que los nodos de la rama se vayan asignando las direcciones del espacio del hipercubo, según el valor de la dirección R y máscara del nodo predecesor de la rama. Por ejemplo, el nodo E recibe el mensaje del mecanismo 2 del nodo B, en donde le avisa que se asigne el próximo espacio de direcciones según la dirección y máscara actual del propio nodo B, es decir, dirección R 11000 y máscara 3. Esto último, se considera que el nodo B ya se asigno su nueva dirección por medio del mecanismo 1 antes que le envié el mensaje al nodo E por el mecanismo 2, en donde el resultado se ven en color rojo en el esquema, propios de los cambios del producidos por el mecanismo 1, y los cambios del mecanismo 2 se muestran en color verde. Entonces el nodo E, se asignará el próximo espacio dando como resultado la dirección R 11010 y máscara 4. Luego el nodo B incrementa en 1 el valor de su máscara, dando como resultado 4. El mismo proceso ocurre entre el nodo E y F, en donde el nodo E envía un mensaje del mecanismo 2 al nodo F con la dirección 11010 y máscara 4. Este último se asigna la siguiente dirección del espacio de direcciones, o sea, dirección R 11011 y máscara 5. Por último el nodo E incrementa en uno su máscara resultando el valor de 5.

El valor de *mi* de cada nodo, se puede obtener con el valor de la máscara del nodo del que recibimos el mensaje por mecanismo 2, sumándole el valor de 1. Por ejemplo, el nodo E, toma el valor de la máscara 3 del nodo B y le suma 1, obteniendo una *mi* de 4.

Tratamiento de dirección relativa y máscara El mecanismo 2 se aplica al nodo predecesor del *par receptor* y a todos aquellos nodos que están sujetos en la misma rama. En nodo que recibe un mensaje del mecanismo 2, debe de tener información de la dirección R, máscara y *número de red* del nodo que le envío dicho mensaje y que le será asignado luego de concluido el proceso de mezcla. Se aclara este último punto, ya que los nodos en cualquiera de los mecanismo, para transmitir mensaje del mecanismo 1 o 2, se debe conservar la dirección antes de la mezcla para utilizarla a la hora de enviar, ya que si se envía con la direcciones R resultante, se corre el riego que en la mezcla que se está renombrando, haya algún vecino que coincida con dicha dirección. Esto se tratará más adelante en el capítulo.

El tratamiento de dirección R y máscara es el siguiente.

1. De la dirección R y máscara resultante del mecanismo del nodo predecesor de la rama, se asigna la siguiente dirección R y máscara del espacio del

direcciones por hipercubo.

2. La máscara inicial se calcula sumando la máscara resultante del mecanismo del nodo predecesor de la rama, más uno.

Paquetes de control Para el mecanismo 2, se diseñan dos paquetes de control que permitirá transmitir los mensajes necesarios para notificar y transmitir información a lo nodos que están involucrado en dicho mecanismo. Se definen los campos para los paquetes MAR2 (*Mix Address Request 2*) y MAN2 (*Mix Address Notification 2*).

1. **Paquete MAR2** ***(Mix Address Request 2)***

   packet type Este campo toma el valor de 15.

   source address La dirección primaria resultante del mecanismo del nodo que envía el paquete.

   primary address La dirección primaria del nodo que envía el paquete.

   optional headers Se incluye un encabezado opcional de tipo *Additional Address*.

   El encabezado opcional contiene los siguientes valores,

   address La dirección del nodo que recibirá el paquete.

   mask La máscara resultante del mecanismo del nodo que envía el paquete.

   red number *Número de red* del nodo que envía el paquete.

2. **Paquete MAN2** ***(Mix Address Notification 2)***

   packet type Este campo toma el valor de 17.

   source address La dirección primaria resultante del mecanismo del nodo que envía el paquete.

primary address La dirección primaria del nodo que envía el paquete.

optional headers Se envía un encabezado opcional de tipo *Additional Address*.

El encabezado opcional contiene los siguientes valores,

address La dirección del nodo que recibirá el paquete.

mask La máscara resultante del mecanismo del nodo que envío el paquete MAR2.

red number *Número de red* del nodo que envía el paquete.

Algoritmos de envío de mensajes de control

Mezcla En un proceso de mezcla, como se describió anteriormente, en principio entran en acción dos nodos en el que llamamos *par emisor* y *par receptor*. Cuando el *par emisor* detecta la mezcla por medio del nodo *par receptor*, este comienza el mecanismo de mezcla en el cual se encargará de enviar el mensaje MAR1 al *nodo receptor* como se muestra en la figura 4.20.

Si el envío del mensaje MAR1 lo hace con su dirección R, puede ocurrir que dicha dirección exista en la red con la que se mezcla, por ende, cuando el *par receptor* le envíe la confirmación MAN1, puede ser recibida por otro nodo. Entonces, antes de enviar el paquete MAR1, debe cambiar la dirección R a una que llamaremos *dirección default*. Esta última dirección tendrá como característica una dirección que no esté dentro del espacio de direcciones del hipercubo. Por otro lado, *par emisor* debe suspender los servicio de envío HB y los paquetes de registro y resolución a los servidores *Rendez Vous*, en el cual serán restablecido en el momento que reciba el mensaje MAN1 del *nodo receptor*, como así también volverá a asignarse la dirección R que tenia en un principio. El algoritmo mezcla 20, se muestran los puntos descripto anteriormente.

Procesamiento MAR1 En la figura 4.20, muestra al *nodo receptor* recibiendo el MAR1 del *nodo emisor* en el cual procesará dicho paquete y dispara el mecanismo de envío de paquetes MAR1 a los nodos subordinados y MAR2 al *nodo padre* como se observa en la figura 4.21. En este caso los vecinos V2 y Vn recibirán el mensaje MAR1, y el V1 recibirá el mensaje MAR2. Antes se debe

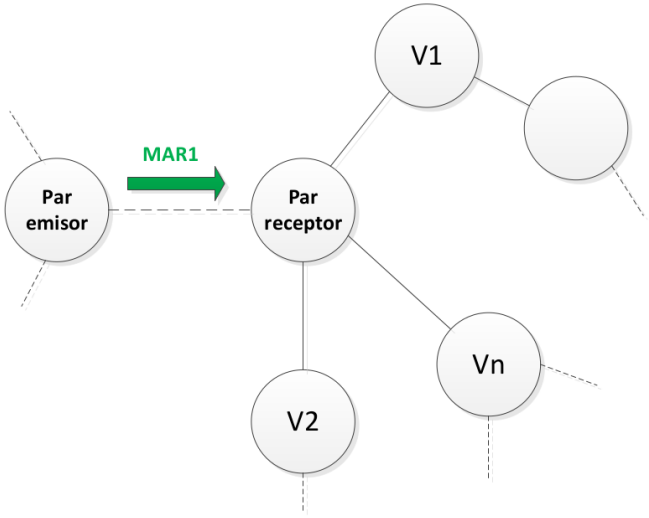

Figura 4.20: Envío del paquete MAR1 del *par emisor* al *par receptor*.

suspender el servicio de envío HB y el servicio de resolución y registración a los servidores *Rendes Vouz*. Luego del envío de los paquetes MAR1 y MAR2, se debe enviar el mensaje de confirmación MAN1 al *par emisor*.

El algoritmo 21, describe el procesamiento de un paquete MAR1. Si un nodo contiene dirección secundaria y no envió ningún mensaje MAR1 o MAR2, se deberá esperar que ambas direcciones, la primaria y la secundaria, se asignen las direcciones correspondiente en el proceso de mezcla para restablecer el servicio de HB y registración y resolución de nombres *Rendez Vous*. También se muestra el calculo de máscara y *mi* según el caso de que se trate, si es *par receptor* o un nodo

Algorithm 20 Mezcla

Require: None
Ensure: None
1: Suspender el envío y procesamiento de paquetes HB;
2: Suspender el servicio de registración y resolución de nombres *Rendez Vous*;
3: Asignar como dirección R, la *dirección default*;
4: Enviar MAR1 al *nodo receptor*;
5: return;

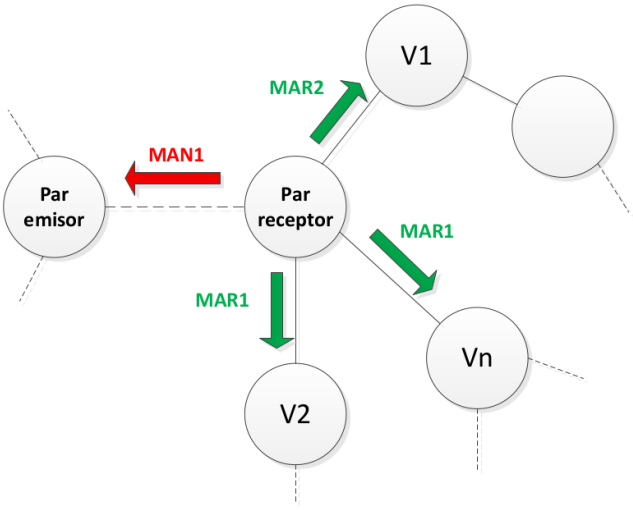

Figura 4.21: Envíos de MAR1 y MAR2 del par recepto, y confirmación MAN1 al nodo emisor.

intermedio como lo son V2 y Vn en la figura 4.21. Esto último respeta al teoría desarrollada en la sub sección 4.3.2.

El nodo al correr dicho algoritmo distingue si tiene el perfil de *par receptor* o no. La distinción se debe por un lado a los diferencia del calculo de máscara y *mi*, y por el otro la necesidad de enviar un paquete MAR2 en el caso del *par receptor*, que en los demás nodos no es necesario.

Procesamiento MAR2 En la figura 4.21, se muestra a V1 que recibe un paquete MAR2 del *par receptor*.

En la figura 4.22 al procesar dicho paquete, V1 debe de seguir propagando el mecanismo 2 vista en la sub sección 4.3.2 a todo los nodos de su rama. Para ello como se describió tanto en el algoritmo 20 y 21, primero se suspende el servicio de HB y registración y resolución de nombres *Rendez Vous*. También se debe distinguir si el destino del paquete MAR2 recibido corresponde a la dirección primaria o secundaria si es que tiene. En el caso de que el destino del mensaje MAR2 es la dirección primaria, luego se envía paquete MAR2 a todos los vecino que están distancia uno de la dirección primaria excepto al nodo del quien recibió el MAR2. Por último, para reanudar los servicios de HB y registración y resolución de nombres *Rendez Vous*, se debe esperar hasta que tanto la dirección primaria

Algorithm 21 Procesamiento MAR1

Require: Paquete MAR1
Ensure: None

Suspender el envío y procesamiento de paquetes HB;
Suspender el servicio de registración y resolución de nombres *Rendez Vous*;
if la dirección destino del MAR1 es la dirección secundaria **then**
    $m_{secundaria} = m_{secundaria} + máscara\ MAR1 - (length\ MAR1 - 1)$;
    Enviar confirmación MAN1 al nodo que envío MAR1;
    Tratamiento de mecanismo 1 a la dirección secundaria;
    if el tratamiento de dirección primaria ocurrió **then**
        Reanudar servicio de HB y *Rendez Vous*;
    end if
    return;
end if
Asignar el *número de red* del campo del paquete MAR1;
if es el nodo receptor **then**
    $m = m + máscara\ MAR1 - (mi - 1)$;
    $mi = máscara\ MAR1 + 1$;
else
    $m = m + máscara\ MAR1 - (length\ MAR1 - 1)$;
    $mi = máscara\ MAR1 + 1 + (mi - length\ MAR1)$;
end if
Borrar tabla de nombres *Rendez Vous*;
if tiene vecinos a distancia 1 de la dirección primaria **then**
    Enviar MAR1 a los vecinos subordinados que están a distancia 1 de la dirección primaria;
    Si es el *par receptor* y tiene nodo padre, enviarle MAR2;
end if
Borrar tabla de vecinos;
if no se envío paquete MAR1 o MAR2 **then**
    Enviar confirmación MAN1 al nodo que envío MAR1;
    Tratamiento de mecanismo 1 a la dirección primaria, y a la tabla de direcciones recuperadas;
    if no tiene dirección secundaria o el tratamiento de la dirección secundaria ocurrió **then**
        Reanudar servicio de HB y *Rendez Vous*;
    end if
else
    Enviar confirmación MAN1 al nodo que envío MAR1;
end if
return;

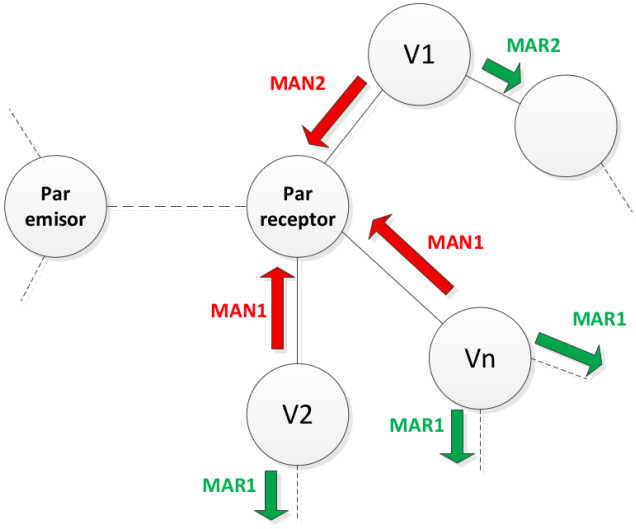

Figura 4.22: Propagación de mensaje MAR1, MAR2 y sus confirmaciones.

como la secundaria se hayan tratados por los mecanismo correspondientes. En el caso de que no se haya enviado ningún maquete MAR2, ya que el mismo es el último de la rama el mecanismo 2, debe de asignarse la dirección R correspondiente al mecanismo 2 4.3.2. El algoritmo 22, se muestra en detalle el el mecanismo del procesamiento de paquetes MAR2.

Procesamiento MAN1 En la figura 4.21, se muestra al *nodo receptor* enviando el mensaje de confirmación MAN1 al *par emisor*. Este al recibirlo y procesarlo, habilita a reanudar el servicio de HB y registración y resolución de nombres *Rendez Vous* y también da la orden de volver a asignarse la dirección que tenia inicialmente antes de la mezcla. El mensaje de confirmación, avisa al emisor de paquetes MAR1, que recibió dicho paquete y por ende puede restablecer sus servicios y asignarse la dirección R y *mi* con la que contará de ahora en más luego del proceso de mezcla. Por ejemplo, volviendo a la figura 4.22 el *par receptor* recibe las confirmaciones MAN1 del V2 y Vn. Entonces a partir de procesar la primer confirmación, habilita los servicios de HB y registración y resolución de nombres *Rendez Vous*, y se asigna la dirección R correspondiente al mecanismo 1. El algoritmo 23 describe el el procesamiento de MAN1.

Algorithm 22 Procesamiento MAR2

Require: Paquete MAR2
Ensure: None

Suspender el envío y procesamiento de paquetes HB;
Suspender el servicio de registración y resolución de nombres *Rendez Vous*;
if la dirección destino del MAR2 es la dirección secundaria **then**
    $m_{secundaria} = máscara\ MAR2$;
    Enviar confirmación MAN2 al nodo que envío MAR2;
    Tratamiento de mecanismo 2 a la dirección secundaria;
    if el tratamiento de dirección primaria ocurrió **then**
        Reanudar servicio de HB y *Rendez Vous*;
    end if
    return;
end if
Asignar el *número de red* del campo del paquete MAR2;
$m = máscara\ MAR2$;
$mi = m$;
Borrar tabla de nombres *Rendez Vous*;
if tiene vecinos a distancia 1 de la dirección primaria **then**
    Enviar MAR2 a los vecinos que están a distancia 1 de la dirección primaria, excepto el nodo que envío MAR2;
    Por cada MAR2 enviado, sumar 1 a m;
end if
Borrar tabla de vecinos;
if no se envío paquete MAR2 **then**
    Enviar confirmación MAN2 al nodo que envío MAR2;
    Tratamiento de mecanismo 2 a la dirección primaria;
    Borrar tabla de direcciones recuperadas;
    if no tiene dirección secundaria o el tratamiento de la dirección secundaria ocurrió **then**
        Reanudar servicio de HB y *Rendez Vous*;
    end if
else
    Enviar confirmación MAN2 al nodo que envío MAR2;
end if
return;

Algorithm 23 Procesamiento MAN1

Require: Paquete MAN1
Ensure: None
1: **if** si aún no se restableció el servicio de HB y *Rendez Vous* **then**
2: **if** la dirección primaria es igual a la *dirección default* **then**
3: Asignar la dirección R primaria inicial;
4: $m = m + 1$;
5: Reanudar servicio de HB y *Rendez Vous*;
6: **else**
7: Tratamiento de mecanismo 1 a la dirección primaria y a la tabla de direcciones recuperadas;
8: **if** no tiene dirección secundaria o el tratamiento de la dirección primaria y secundaria ocurrió **then**
9: Reanudar servicio de HB y *Rendez Vous*;
10: **end if**
11: **end if**
12: **end if**
13: return;

Procesamiento MAN2 El algoritmo 24 de procesamiento MAN2 es muy similar al del MAN1. Solo se diferencia que al restablecer los servicios y asignar la dirección R, utiliza los campos de datos y procedimientos del mecanismo 2 4.3.2. Por ende, el algoritmo cuando reconstruye la dirección R final, debe tener el mismo efecto que si se hubiese reconstruido la dirección R mediante el algoritmo 23 procesamiento MAN1.

Algorithm 24 Procesamiento MAN2

Require: Paquete MAN2
Ensure: None

```
1: if si aún no se restableció el servicio de HB y Rendez Vous then
2:     Tratamiento de mecanismo 2 a la dirección primaria;
3:     Borrar tabla de direcciones recuperadas;
4:     if no tiene dirección secundaria o el tratamiento de la dirección primaria
   y secundaria ocurrió then
5:         Reanudar servicio de HB y Rendez Vous;
6:     end if
7: end if
8: return;
```

Capítulo 5

Simulación de redes

En este capítulo se realiza simulaciones sobre el protocolo ANTop, en donde se proponen distintos esquemas y se obtienen diversos datos de funcionamiento. Luego se analizan los resultados y se dictan conclusiones al respecto.

En primer lugar, en la sección 5.1 se presenta la plataforma de simulación Mininet, explicando su esquema de simulación, arquitectura y componentes del mismo.

En la sección 5.2 se presentan las simulaciones realizadas en este trabajo. En dicha sección se obtienen datos en base diferentes esquema de simulación, empezando primero en la representación de redes aleatorias con un foco en nodos cercanos y luego en nodos más alejados entre sí. También se realizan diferentes topologías que se mueven en el tiempo para representar la fragmentación y mezcla de redes ANTop y dar un fundamento práctico a la teoría realizada en el capítulo 4.

5.1. Mininet

Mininet [17] es un emulador de red que ejecuta una colección de host finales, conmutadores, enrutadores y enlaces en un único kernel de Linux. Utiliza una visualización liviana para hacer que un solo sistema se vea como una red completa, ejecutando el mismo kernel, sistema y código usuario. Un host mininet se comporta como una maquina real; se puede entrar en él y ejecutar programas arbitrarios (incluido todo lo que esté instalado en el sistema Linux subyacente). Los programas que se ejecutan pueden enviar paquetes través de lo que parece una interfaz ethernet real. En esta plataforma, no está definidos los elemento de una red Wire-

less, como ser un punto de acceso o estación que emule interfaz inalámbrica. Para ello existe una extensión de Mininet, llamada Mininet Wifi 5.1.1.

5.1.1. Mininet Wifi

Mininet Wifi [18] es un *fork* del emulador de red Mininet cuyas funcionalidades fueron ampliadas al agregar estaciones Wifi virtualizadas y puntos de accesos basados en los controladores inalámbricos estándar de Linux y el controlador de simulación inalámbrica `80211_hwsim`. Se agregaron clases para admitir la incorporación de estos dispositivos inalámbricos en un escenario de red de Mininet y para simular los atributos de una estación móvil, como la posición y el movimiento en relación con otras estaciones o puntos de acceso.

Arquitectura y componentes Los principales componentes que forman parte del desarrollo de Mininet Wifi se ilustran en la figura 5.1.

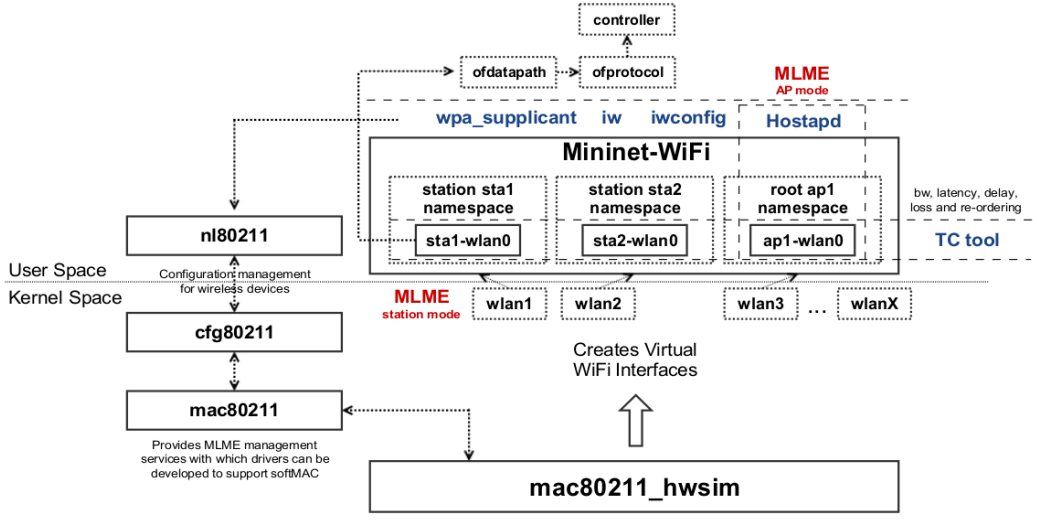

Figura 5.1: Componentes Mininet Wifi, Fuente Figura 1.1 de [18]

En el espacio kernel, el módulo `80211_hwsim` es responsable de crear interfaces Wifi virtuales, importantes para estaciones y puntos de acceso. Continuando en el

espacio del kernel, MLME [1] (*Media Access Control Sublayer Management Entity*) se realiza en el lado de las estaciones, mientras que en el espacio de usuario el *hostapd (host access point daemon)* es responsable de esta tarea en el lado del punto de acceso. Mininet Wifi también utiliza un par de utilidades como *iw*, *iwconfig* o *wpa_supplicant*. Los dos primeros se usan para la configuración de la interfaz y para obtener información de las interfaces inalámbricas y la última se utiliza con *hostapd*, para admitir WPA (acceso protegido Wifi), entre otras cosas. Además de ellos, otra utilidad fundamental es TC (*Traffic Control*). El TC es un programa de utilidad de espacio de usuario utilizado para configurar el programador de paquetes kernel de Linux, responsable de controlar la velocidad, la demora, la latencia y la pérdida, aplicando estos atributos en interfaces virtuales de estaciones y puntos de acceso.

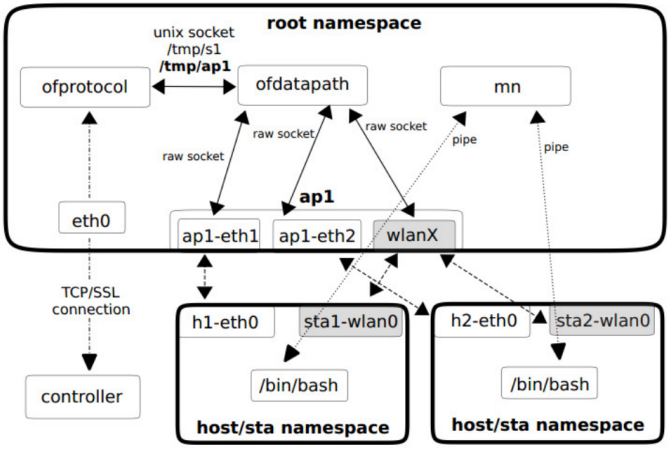

Figura 5.2: Componentes y conexiones en dos host creados con Mininet Wifi, Fuente Figura 1.2 de [18]

La figura 5.2 muestra los componentes y las conexiones en una topología simple con dos estaciones (o hosts) creados con Mininet Wifi, donde los componentes recién implementados (resaltados en gris) se presentan a lo largo de los bloques de construcción originales de Mininet. Aunque las estaciones están equipadas con interfaz inalámbrica de forma predeterminada, pueden conectarse con puntos de acceso a través de enlaces cableados también. Más específicamente, agregamos in-

[1] algunas de las funciones realizadas por MLME son autenticación, asociación, envío y recepción de beacons, etc.

terfaces Wifi en estaciones que ahora pueden conectarse a través de su interfaz `wlanX` que está puenteada a un conmutador *OpenFlow* con capacidades de punto de acceso representado por `ap1`. Similar a Mininet, la red virtual se crea colocando procesos de host en espacios de nombres de red del sistema operativo Linux interconectados a través de pares de Ethernet virtuales. Los dispositivos definidos en Mininet Wifi son los siguientes.

Estaciones son dispositivos que se conectan a un punto de acceso mediante autenticación y asociación, o mismo inician una conexión con otra estación en si. En nuestra implementación, cada estación tiene una tarjeta inalámbrica `wlan0`. Dado que los hosts Mininet tradicionales están conectados a un punto de acceso, las estaciones pueden comunicarse con esos hosts.

Puntos de acceso son dispositivos que administran estaciones asociadas.

Las estaciones y puntos de acceso usan `cfg80211` para comunicarse con el controlador del dispositivo inalámbrico, que resulta ser una API de configuración Linux 802.11 que proporciona comunicación entre estaciones y `mac80211`. Este a su vez, se comunica directamente con el controlador del dispositivo Wifi a través de un conector Netlink (o más específicamente `nl80211`) que se usa para configurar el dispositivo `cfg80211`.

En este trabajo de tesis la intención es simular una red *ad hoc*, y por tal motivo no será necesario el uso de puntos de acceso como lo es común en una red Wifi. Sin embargo, se utilizaran estaciones para representar cada nodo de la red.

Scripts y API en Python Mininet proporciona una API de Python para que los usuarios puedan crear scripts de Python simples que configurarán topologías personalizadas. Mininet Wifi amplía esta API para admitir un entorno inalámbrico. Los desarrolladores de Mininet Wifi agregaron nuevas clases a Mininet para admitir la emulación de nodos en un entorno inalámbrico. Mininet Wifi agrega los métodos `addStation` y `addAccessPoint`, y un método `addLink` modificado para definir el entorno inalámbrico. En un escenario simple, se puede agregar una estación y un punto de acceso con los siguientes métodos en una secuencia de comandos Python de Mininet Wifi.

Para agregar una nueva estación llamada `sta1`, con todos los parámetros establecidos en valores predeterminados.

```
net.addStation('sta1')
```

Para agregar un nuevo punto de acceso llamado `ap1`.

```
net.addAccesPoint('ap1',ssid='new_ssid')
```

Para definir el vinculo inalámbrico entre la estación y el punto de acceso, con valores predeterminados para los atributos del enlace.

```
net.addLink(ap1,sta1)
```

Las estaciones se pueden asociar a una red *ad hoc* con el siguiente comando.

```
net.addHoc(sta1, ssid='adhocNet')
```

Para escenarios más complejos, hay más parámetros disponibles para cada método. Se puede especificar la dirección MAC, la dirección IP, la ubicación en el espacio tridimensional, el alcance de la radio y más. Por ejemplo, la siguiente linea define la posición física de una estación, la MAC y el rango.

```
sta1 = net.addStation('sta1', position='115,100,0', mac='00:00:00:00:00:02',
range=10)
```

Comandos Dado una topología definida en un script Python como se menciono antes, también existe comandos Mininet que nos permite cambiar parámetros de cada elemento durante la simulación, como ser el cambio de posición geográfica de una estación. Algunos de los comandos que se utilizan en este trabajo son los siguientes.

Para definir una nueva posición de una estación es el siguiente comando.

```
py sta1.setPosition('150,100,0')
```

Para abrir un terminal de una estación en particular.

```
xterm sta1
```

Se pueden usar comando del sistema Linux en cada estación o punto de acceso, solo mencionando en primer lugar el elemento de interés y luego colocando el comando correspondiente del sistema. Por ejemplo para consultar la configuración de interfaz de red por medio el comando Linux `ifconfig` de la estación `sta1`, es

con el siguiente comando.

```
sta1 ifconfig
```

5.2. Simulaciones

Utilizando Mininet Wifi descripto en la sección anterior, se diseñaron tres tipos de simulaciones: Una consiste en redes generadas aleatoriamente, y las otras dos utilizando topologías adecuada para representar la fragmentación y mezcla de redes. Las topologías de Mininet Wifi se define en un script Python para cada caso de simulación.

Para efectuar las simulaciones, la aplicación ANTop que corre en cada nodo de la topología irá informando en un archivo `.txt` los procesos y registros como ser las tablas de vecinos, entradas *Rendez Vous*, tabla de ruteo y las direcciones IPv6 asignadas en la interfaz de red. Luego estos datos serán fuentes para el análisis de las simulaciones.

Las simulaciones que se muestran en esta sección, se realizaron creando redes de tamaños limitado sumando 16 nodos en total.

5.2.1. Redes aleatorias

Esta simulación a su vez consiste de dos casos de simulaciones: Una con una topología de nodos con distancia entre sí menor al radio de cobertura fijo (nodos cercanos), y la otra con una topología de nodos con distancia entre sí mayor al radio de cobertura (nodos lejanos).

Para disminuir el error, en cada caso se efectuaron 10 series de distribución aleatoria de posiciones de los nodos, generando así 10 simulaciones con topología diferentes. En total se realizaron 20 simulaciones.

Desarrollo de la Simulación

La simulación corre a partir de un script en Python que define el plano físico, la posición de los nodos utilizando una funciona aleatoria de coordenadas y el llamado a un *Shell script* que se encarga de ejecutar la aplicación ANTop en cada nodo. El código del apéndice A.1 describe el script Python utilizado en el caso de

nodos cercanos.

Para el caso de nodos lejanos, difiere en los parámetros de la función `randint()` y `net.plotGraph()`, en donde el primero define la distribución de la posición de los nodos y el segundo el plano físico en el que se montan. Los valores que se utilizo en este trabajo para dicho caso, fue de `randint(0,140)` y `net.plotGraph(max_x=150, max_y=150)`.

La función `makeTerm` hace un llamado a un *Shell scrip* `.sh` particular en cada nodo. Las siguientes lineas definen el archivo `.sh` para un nodo `x`.

```
sleep x_Time
sudo rmmod kantop
./antopd wlan0 nodox 64 >./log/logx.txt
sudo ./antopd wlan0 nodox 64
```

El valor *x_Time* es el tiempo que debe esperar el nodo en correr ANTop, para lograr la conexión en forma secuencial entre los nodos. Es decir, que el primer nodo en correr ANTop será el nodo1, luego el nodo2, luego el nodo3, y así sucesivamente. En el directorio `./log/` se encuentran los archivos `.txt` de todos los nodos luego de haber concretado la simulación.

Análisis de datos obtenidos

Para el caso de nodos cercanos se realiza la posición aleatoria de los nodos acotado en una región el que el radio de alcance sea mayor a la distancias entre los nodos. En la figura 5.6 se muestran el plano de posicionamiento de los nodos en las simulaciones.

Para los nodos lejanos, la posición aleatoria de los nodos están acotados en una región más amplia con respecto al radio de alcance, dando como resultado más de una red formada en el plano. En la gráfica 5.7 se muestran los planos de posicionamiento de cada simulación.

De los datos obtenidos en cada simulación, se construyen una serie de gráfica para ilustrar el comportamiento del protocolo desde el punto de vista de la cantidad de vecinos, la cantidad de entradas máximas en la tabla de ruteo y la cantidad de entradas en la tabla *Rendez Vous*. Como se menciono anteriormente, el comportamiento difiere si la topología consiste de nodos cercanos o nodos lejanos.

Cantidad de vecinos De los datos obtenidos en las simulaciones, se gráfica la distribución de la cantidad de vecinos en cada simulación como se muestra en la figura 5.3.

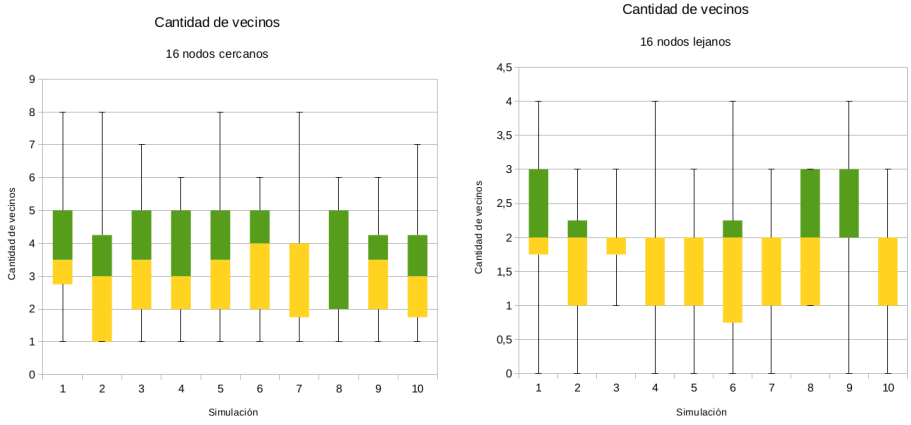

Figura 5.3: Cantidad de vecinos

En el protocolo ANTop, un nodo es vecino de otro si cumple con la condición de que respectivamente cada uno esté dentro del rango de alcance de otro, y que a su vez sus direcciones difieran en solo un bit, es decir, que estén a distancia uno. Esto último se puede repasar en la sección 2.3.3.

En el caso de los nodos cercanos, la mayor concentración de la cantidad de vecinos registrados, están dentro del rango entre 3 y 5. En cambio para el caso de los nodos lejanos se registraron mayormente entre 1 a 3 vecinos. Cuando los nodos son cercanos, tienen más nodos en su rango de alcance de la interfaz wireless, aumentando el número de posibles vecinos. Por otro lado, en los nodos lejanos, al estar más distanciados, se arman redes aisladas de menor tamaño disminuyendo la cantidad de posibles vecinos.

El mínimo de 0 alcanzado en la gráfica de nodos lejanos, se debe a que se registraron nodos solitarios, fuera del alcance de algún otro nodo, formando una red de solo un nodo. En cambio en la gráfica de nodos cercanos, se registra un mínimo de 1 vecino, debido a que cada nodo tiene por lo menos como vecino al nodo que le cedió parte de su espacio de direcciones.

Cantidad de entradas máximas en la tabla de ruteo En la figura 5.4 se muestra la gráfica de la entradas máximas en la tablas de ruteo registradas.

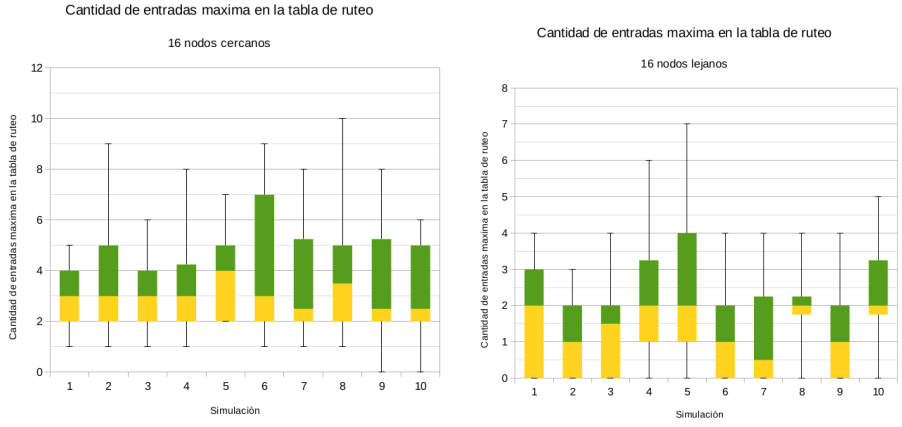

Figura 5.4: Cantidad de entradas máximas en la tabla de ruteo

Cuando los paquetes son ruteados se van generando entradas en la tabla de ruteo. En las gráficas muestran el tamaño registrado en la simulaciones. Se debe tener en cuenta que la simulaciones se realizaron sin la generación de paquetes exentos al protocolo ANTop, es decir, que solo los paquete del servicio de registración *Rendez Vous* fueron los que generaron entradas en la tabla de ruteo. Los demás paquetes de control del protocolo ANTop, no necesitan ser ruteados y por ende no generan entradas en la tabla de ruteo.

Se registrón la cantidad de entradas máximas en la tabla de ruteo, ya que las entradas se construyen y se destruyen luego de un *time out* configurable del protocolo. Esto hace que sea muy volátil la cantidad de entradas en la tabla de rutas en el tiempo.

En el caso de nodos cercanos se registraron mayormente entre 2 a 5 entradas máximas en la tabla de ruteo, en cambio en el caso de nodos lejanos, se registraron entre 0 a 3 entradas. Se puede explicar este resultado a partir de que en las redes de gran tamaño tienden a tener más entradas en la tablas de ruteo debido que tienen que registrarse en los nodos *Rendez Vous* que generalmente estén a distancia mayor a 1 de uno mismo, teniendo la necesidad de enrutar por medio de nodos intermedios. A mayor cantidad de nodos en una red, habrá más paquetes *Register*,

y por ende más rutas en la tabla de ruteo. En una red pequeña, un nodo tiene mayor probabilidad de recibir entradas *Rendez Vous*, por ende no es necesario enrutarlo hacia otro nodo, obteniendo así menos entradas en la tablas de rutas. Esto último se explica en detalle en el siguiente análisis 5.1.1.

En la gráfica de nodos cercanos, se llego a un mínimo de 0 en la simulación 9 y 10, debido a que se registraron nodos que a su vez tienen el papel de servidor *Rendez Vous* de sí mismo, ya sea que las direcciones de hipercubo obtenidas de la función *Hash* están dentro espacio de direcciones que administra, y por tal motivo no tienen la necesidad de enrutar los paquetes *Register*. En la gráfica de nodos lejanos, también se llega a un mínimo de 0, pero esta vez se registraron en mayor número de simulaciones que en el caso de nodos cercanos. Esto último se debe a que en redes más pequeñas, los nodos tienden a tener mayor espacio de direcciones que en una red grande, y como se comento antes, tienen mayor probabilidad de obtener una entrada *Rendez Vous*. Por ende, en el caso de nodos lejanos se dieron muchos casos en el que un nodo es el servidor *Rendez Vous* de sí mismo sin tener la necesidad de crear una entrada en la tabla de ruteo para rutear los paquetes *Register*.

Cantidad de entradas en la tabla *Rendez Vous* En la figura 5.5 se muestra la cantidad de entradas en la tabla *Rendez Vous*.

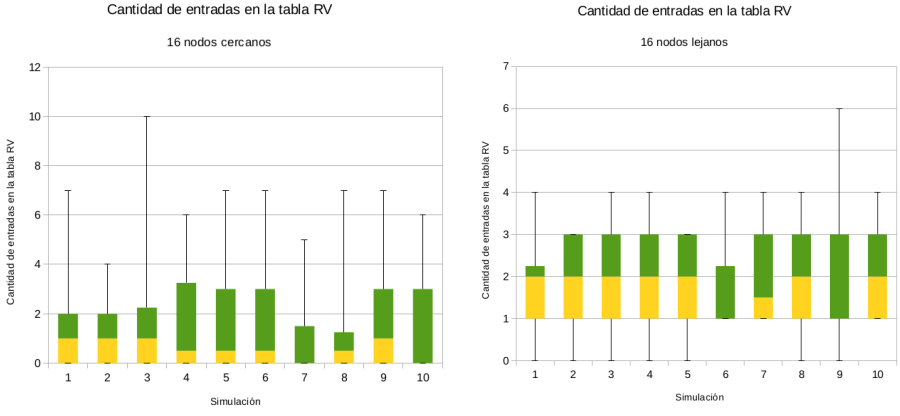

Figura 5.5: Cantidad de entradas en la tabla *Rendez Vous*

El *Rendez Vous* para el nodo se elije aplicando una función *Hash* a la dirección *U* de forma tal que el resultado sea una dirección del hipercubo. Luego el nodo que

tiene esa dirección en su espacio es el que recibe la entrada. Entonces, aquellos que tienen un espacio de direcciones más grande (máscara más corta) tienen mayor probabilidad de recibir entradas.

En la gráfica de nodos cercano, se observa que la distribución es más dispersa, observando una mayor concentración entre 0 y 2, y un pico de 10 entradas. En la gráfica de nodos lejanos, en cambio tiene una distribución más normal en donde se centra su mayor distribución entre el rango de 1 y 3, con mínimo en 0 y máximo en 4, excepto en la simulación 9 que se tuvo un máximo de 6. En el caso de nodos cercanos, al ser una red más grande en comparación a las redes de los nodos lejanos, los servidores *Rendez Vous* se concentran en pocos nodos en el cual se llevan la mayor cantidades de entradas, dejando otros con pocas entradas del orden no mayor a 2. En cambio para el caso de nodos lejanos, los nodos que componen una red son menor en cantidad pero a su vez hay más redes formadas. Esto último, explica la distribución simétrica de entradas *Rendez Vous* obtenida, ya que cada red tiene un nodo que se lleva la mayor cantidad de entradas y que se asemeja a la cantidad de entradas del nodo par *Rendez Vous* de la red vecina. En conclusión, en nodos lejanos, hay tantos nodos *Rendez Vous* con cantidad de entradas similares según la cantidad de redes formadas.

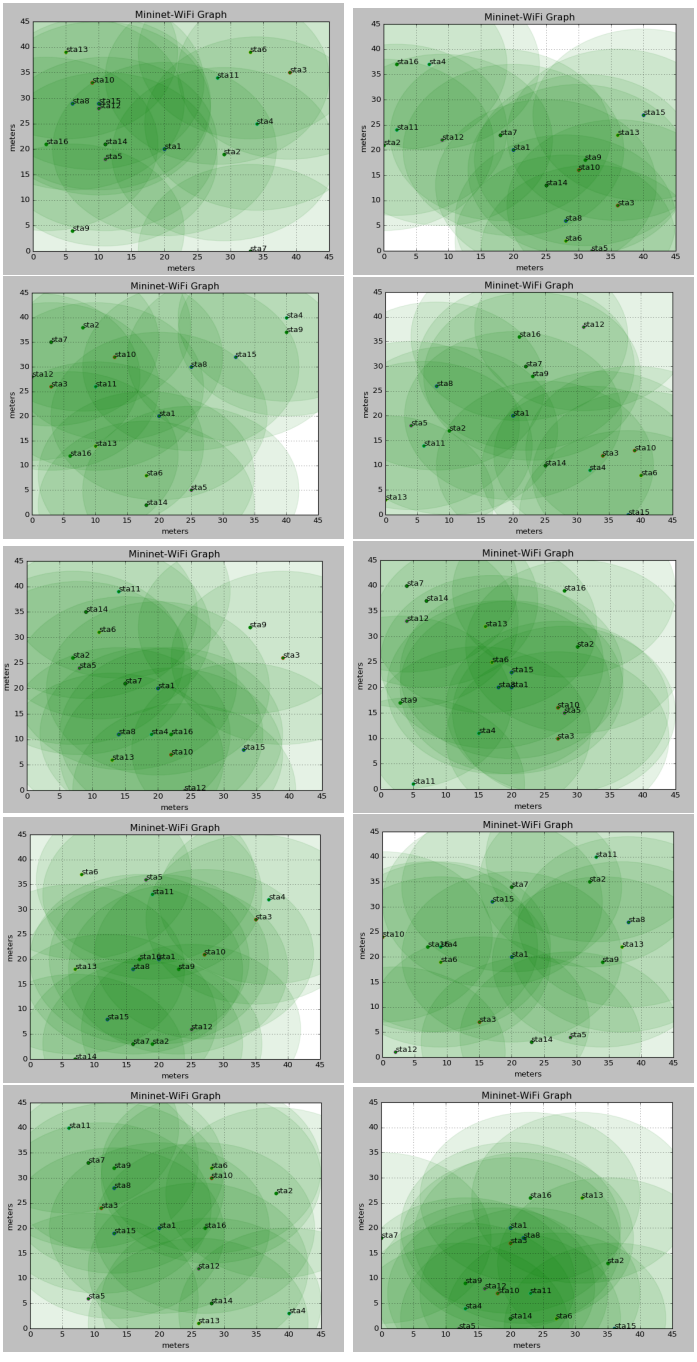

Figura 5.6: Nodos cercanos: Posición aleatoria de los nodos en cada simulación.

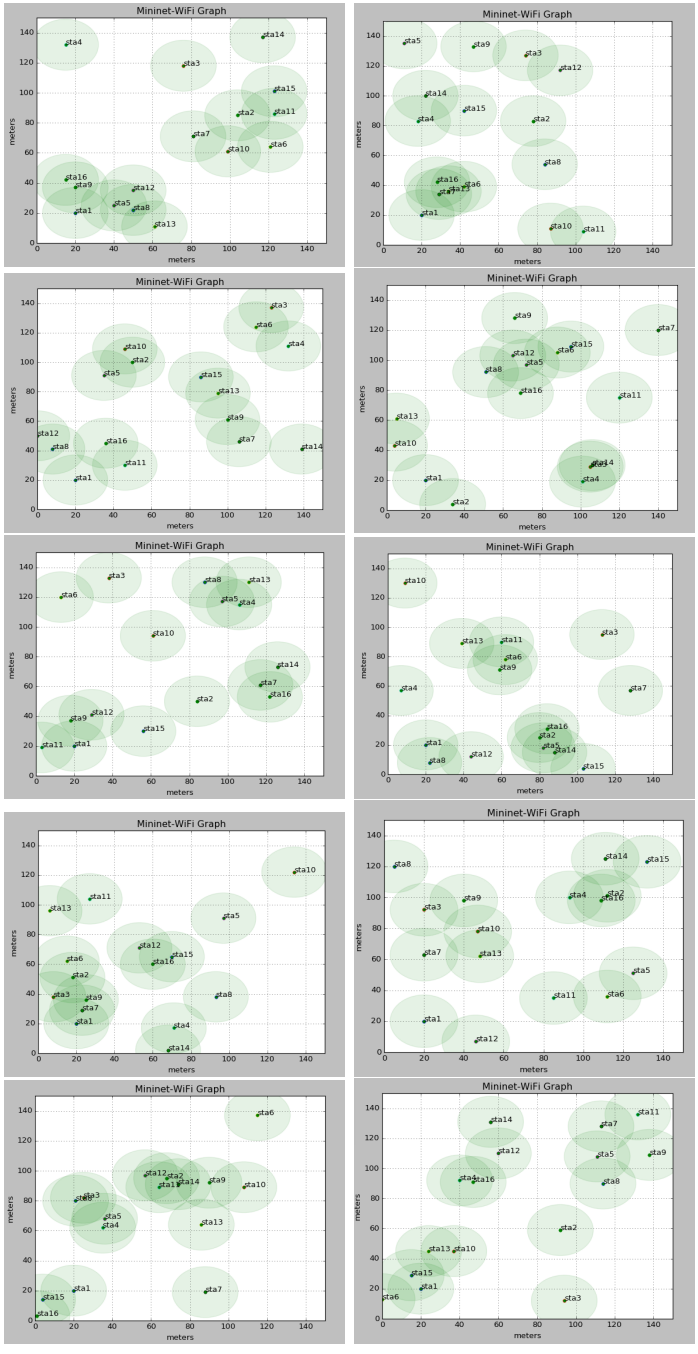

Figura 5.7: Nodos lejanos: Posición aleatoria de los nodos en cada simulación.

5.2.2. Fragmentación de una red

Aquí se presentan las simulaciones de fragmentación de una red ANTop para constatar la teoría desarrollada en la sección 4.2. Las simulaciones consisten en la construcción de una red que luego se particiona en dos en un determinado tiempo de simulación. Se realizaron 2 simulaciones con topologías diferentes.

Desarrollo de la Simulación

Para las simulaciones se corre un script Python que define la posiciones de cada nodo y también una nueva funcionalidad de Mininet Wifi que permite ejecutar la movilidad de ciertos nodos en determinado tiempo de simulación.

El código del apéndice A.2 muestra el script Python utilizado en la primer topología simulada. Las lineas `net.startMobility()` y `net.stoptMobility()` definen el dominio de la función `net.mobility()` en donde se configura la posición y el instante del tiempo inicial y final de un nodo en la simulación. Para este caso, se definió que los nodos `sta8`, `sta11`, `sta12`, `sta13`, `sta14` y `sta15`, se muevan en linea recta hacia la derecha alejándose del alcance de los demás nodos para simular la fragmentación.

La función `makeTerm` utiliza el mismo *Shell script* mencionado en las simulaciones anteriores.

Se utilizó la herramienta *Wireshark* para la captura de paquetes IPv6 en determinados nodos. Esto nos permitirá saber que tipos de paquete transmite y recibe un nodo en particular para luego analizarlo. Si bien las direcciones IPv6 de cada nodo se asigna en base al protocolo ANTop, las direcciones físicas (*MAC Address*) se utilizan por defecto las que impone Mininet Wifi. En el cuadro 5.1 se detallan las direcciones físicas de cada nodo.

Al ejecutar el script Python, la simulación evoluciona de tal forma que un conjunto de nodos se alejan uniformemente de los otros, hasta que ocurre la fragmentación. Uno de los nodos que se mueve se configura para que ejecute un *ping* a una o dos direcciones del hipercubo, que serán elegidas en cada simulación según la topología configurada. Si bien se puede utilizar el servicio de *Rendez Vous* y solo referenciar el *ping* al nombre de un nodo en particular, cuando se realice la fragmentación puede que dicho nodo quede en el otro fragmento de la red y por ende no se pueda resolver la dirección *U*. Entonces si nos referimos directamente a una dirección *R* se tendrá más información antes y después de la fragmentación de una red.

Nodo	MAC Address
sta1	02:00:00:00:00:00
sta2	02:00:00:00:01:00
sta3	02:00:00:00:02:00
sta4	02:00:00:00:03:00
sta5	02:00:00:00:04:00
sta6	02:00:00:00:05:00
sta7	02:00:00:00:06:00
sta8	02:00:00:00:07:00
sta9	02:00:00:00:08:00
sta10	02:00:00:00:09:00
sta11	02:00:00:00:0a:00
sta12	02:00:00:00:0b:00
sta13	02:00:00:00:0c:00
sta14	02:00:00:00:0d:00
sta15	02:00:00:00:0e:00
sta16	02:00:00:00:0f:00

Cuadro 5.1: Tabla de direcciones MAC por Mininet Wifi

Análisis de datos obtenidos

Topología 1 Antes de comenzar con la movilidad de los nodos, la red comienza como muestra la figura 5.8, en el cual las direcciones R de cada nodo fueron obtenidas en función de la conexiones de los mismos empezando primero el nodo `sta1`, luego `sta2`, y `sta16` el último en conectarse.

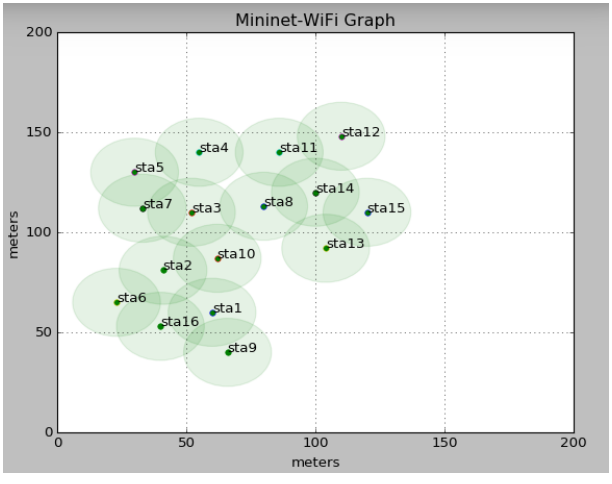

Figura 5.8: Topología 1, t=0.

Las direcciones asignadas, tanto primaria como secundaria, se detalla en el

cuadro 5.2.

Nodo	Dirección primaria	Dirección secundaria
sta1	2001:db8::1	-
sta2	2001:db8:0:0:8000:0:0:1	-
sta3	2001:db8:0:0:c000:0:0:1	-
sta4	2001:db8:0:0:e000:0:0:1	2001:db8:0:0:f000:0:0:1
sta5	2001:db8:0:0:d000:0:0:1	-
sta6	2001:db8:0:0:a000:0:0:1	-
sta7	2001:db8:0:0:9000:0:0:1	-
sta8	2001:db8:0:0:c800:0:0:1	-
sta9	2001:db8:0:0:4000:0:0:1	-
sta10	2001:db8:0:0:2000:0:0:1	
sta11	2001:db8:0:0:cc00:0:0:1	2001:db8:0:0:cd00:0:0:1
sta12	2001:db8:0:0:ce00:0:0:1	-
sta13	2001:db8:0:0:ca00:0:0:1	2001:db8:0:0:cb00:0:0:1
sta14	2001:db8:0:0:c900:0:0:1	-
sta15	2001:db8:0:0:c980:0:0:1	-
sta16	2001:db8:0:0:6000:0:0:1	-

Cuadro 5.2: Topología 1, tabla de direcciones relativas antes de la fragmentación.

Al correr el tiempo de simulación, una porción de la red se desplaza hacia la derecha hasta llegado a un punto que unos de los nodos pierde conexión con otro al estar fuera del alcance. En la figura 5.9 se muestra que el nodo `sta11` pierde conexión con el nodo `sta4`.

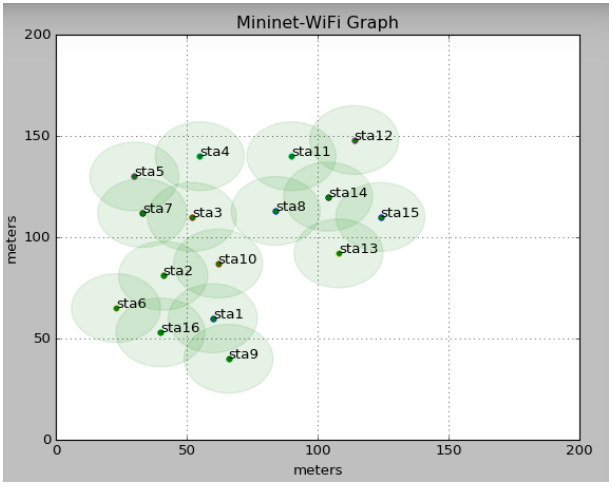

Figura 5.9: Topología 1, t=100s.

Si bien no hay una fragmentación debido a que el nodo `sta8` aún tiene conexión con `sta3`, se realizó desde el nodo `sta11` un *ping* a `sta3` con `ping6 -I wlan0 2001:db8::c000:0:0:1` para corroborar dicha conectividad. La captura de los paquetes de ICMPv6 *Echo Request* y *Echo Reply* se puede ver en la figura 5.10.

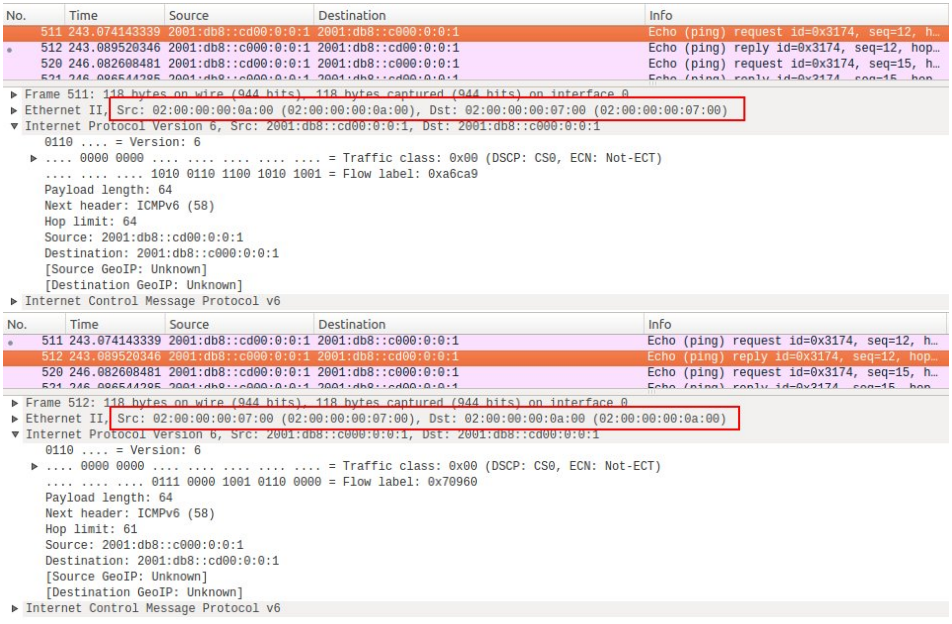

Figura 5.10: Topología 1, ping desde `sta11` a la dirección *2001:db8::c000:0:0:1* antes de la fragmentación. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*.

Se observa que el *header* de la trama ethernet, las direcciones física del que origino el paquete ICMPv6 *Echo Request* corresponde al nodo `sta11`, y el destino `02:00:00:00:07:00` a la dirección física del nodo `sta8`. Lo mismo sucede con el paquete ICMPv6 *Echo Reply*, pero el que construye la trama ethernet es el nodo `sta8` y lo envía al nodo `sta11`. Esto ocurre ya que el siguiente salto del `sta11` es el `sta8` para la comunicación con el `sta3`. Se demuestra que la red no se fragmento y por el momento se puede rutear paquetes entre cualquiera de los nodos. En las figuras 5.11 y 5.12, se muestra las capturas ICMPv6 *Echo Request* y *Reply* a partir de un *ping* desde `sta8` a las direcciones *2001:db8::1* y *2001:db8::8000:0:0:1*, que corresponde a los nodos `sta1` y `sta2` respectivamente. Estos paquetes se enrutan

por medio del `sta3`.

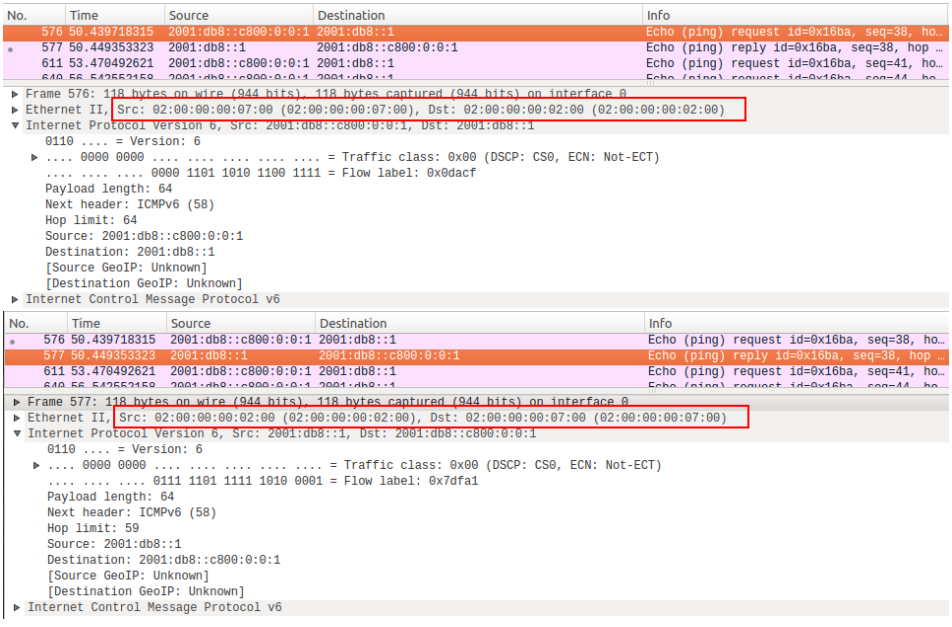

Figura 5.11: Topología 1, ping desde `sta8` a la dirección *2001:db8::1* antes de la fragmentación. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*.

Avanzando en el tiempo de simulación y en el momento que `sta8` pierde conexión con `sta3` se produce la fragmentación como muestra la figura 5.13.

Los cuadros 5.3 y 5.4, describen las direcciones resultante de la fragmentación de la red. El nodo `sta8` es el que dispara el mecanismo de fragmentación, y por ende se auto asigna la primer dirección del hipercubo para conservar todo el espacio de direcciones en la nueva red formada. También envía el mensaje FAR a su vecinos para que ajusten su direcciones R, y luego estos hacen lo mismo con sus subordinados hasta cubrir toda la red fragmentada. En las capturas 5.14, 5.15 y 5.16 de `sta8` en el momento de fragmentación, se muestran los paquetes FAR generados y los paquetes de confirmación FAN recibidos. El recuadro en negro especifica el tipo de paquete de control ANTop, ya sea que el valor de `0x0c` (12 en decimal) corresponde al mensaje FAR, y `0x0d` (13 decimal) a un mensaje FAN.

Para verificar la fragmentación, se vuelve a repetir el ping de `sta11` a la direc-

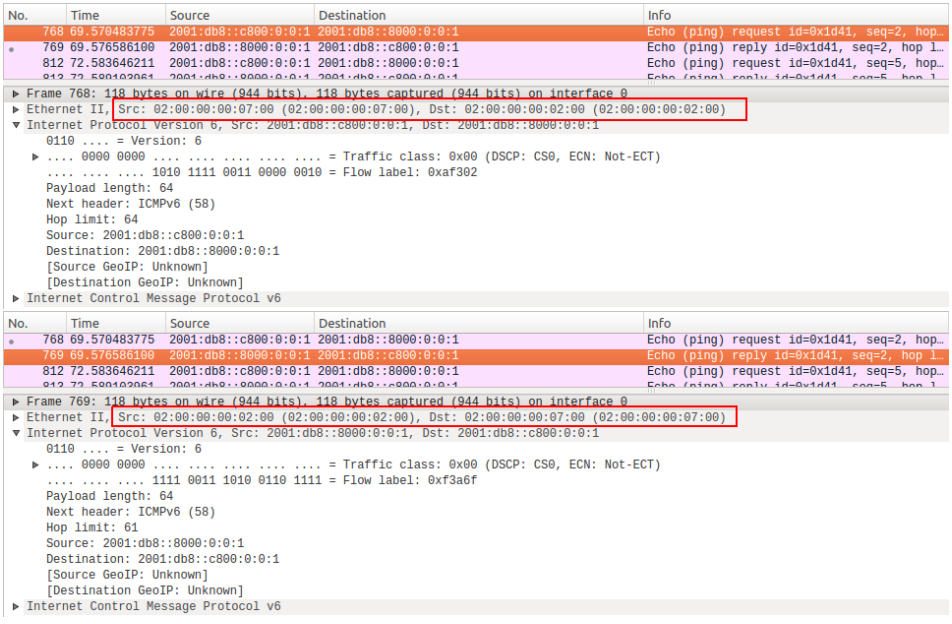

Figura 5.12: Topología 1, ping desde `sta8` a la dirección *2001:db8::8000:0:0:1* antes de la fragmentación. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*.

Red 1		
Nodo	**Dirección primaria**	**Dirección secundaria**
sta1	2001:db8::1	-
sta2	2001:db8:0:0:8000:0:0:1	-
sta3	2001:db8:0:0:c000:0:0:1	-
sta4	2001:db8:0:0:e000:0:0:1	2001:db8:0:0:f000:0:0:1
sta5	2001:db8:0:0:d000:0:0:1	-
sta6	2001:db8:0:0:a000:0:0:1	-
sta7	2001:db8:0:0:9000:0:0:1	-
sta9	2001:db8:0:0:4000:0:0:1	-
sta10	2001:db8:0:0:2000:0:0:1	
sta16	2001:db8:0:0:6000:0:0:1	-

Cuadro 5.3: Topología 1, tabla de direcciones relativas de la Red 1 resultante de la fragmentación.

ción *2001:db8::c000:0:0:1* y de `sta8` a la dirección *2001:db8::8000:0:0:1* mostradas en las capturas de la figura 5.17 y 5.18.

Se observa que en el caso del nodo `sta11` la transmisión de ICMPv6 *Echo Re-*

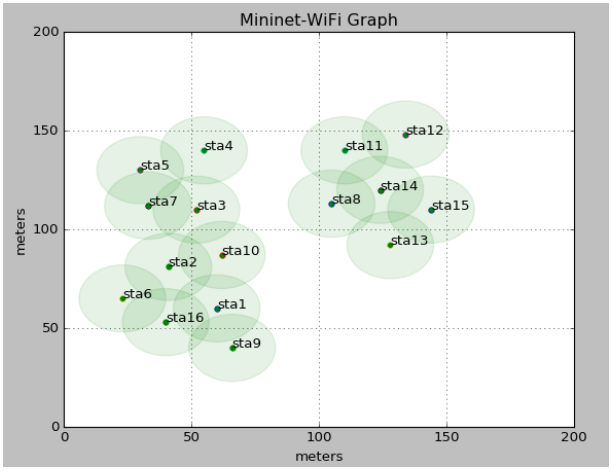

Figura 5.13: Topología 1, t=300s. Fragmentación en Red 1 a la Izquierda y Red 2 a la derecha.

Red 2		
Nodo	**Dirección primaria**	**Dirección secundaria**
sta8	2001:db8::1	-
sta11	2001:db8:0:0:8000:0:0:1	2001:db8:0:0:a000:0:0:1
sta12	2001:db8:0:0:c000:0:0:1	-
sta13	2001:db8:0:0:4000:0:0:1	2001:db8:0:0:6000:0:0:1
sta14	2001:db8:0:0:2000:0:0:1	-
sta15	2001:db8:0:0:3000:0:0:1	-

Cuadro 5.4: Topología 1, tabla de direcciones relativas de la Red 2 resultante de la fragmentación.

quest y la recepción de *Echo Reply* lo hace por medio de `sta12` con dirección física `02:00:00:00:0b:00`. Si se repasa las direcciones asignadas en la Red 2 en el cuadro 5.4, la dirección *R 2001:db8::c000:0:0:1* corresponde al nodo `sta12` producto de la fragmentación. Entonces no se está enrutando por medio de `sta12`, si no que este último es el destinatario del ping. Por otro lado, la dirección IPv6 origen del paquete *Echo Request* o la destino del *Echo Reply* (*2001:db8::a000:0:0:1*), sería la dirección *R* secundaria del `sta11` luego del mecanismo de renombramiento de la red fragmentada que coincide con los datos obtenidos en la cuadro 5.4.

Ocurre lo mismo con las capturas de ICMPv6 *Echo Request* y *Echo Reply* en `sta8`, en donde la transmisión y recepción de los respectivos mensajes hace referencia a la dirección física `02:00:00:00:0b:00` y dirección *R* correspondiente

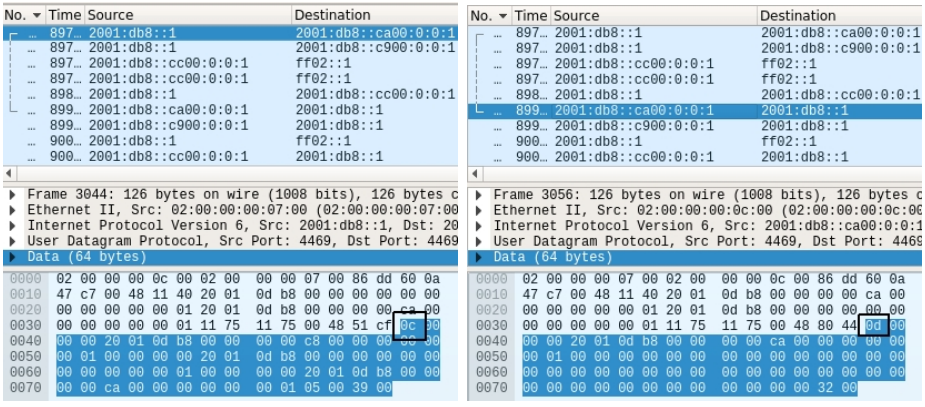

Figura 5.14: Topología 1, captura de paquetes de control FAR y FAN en `sta8` al momento de la fragmentación. En la figura de la izquierda se muestra a `sta8` enviando un paquete FAR a `sta13`, y la figura de la derecha la confirmación FAN de `sta13` a `sta8`

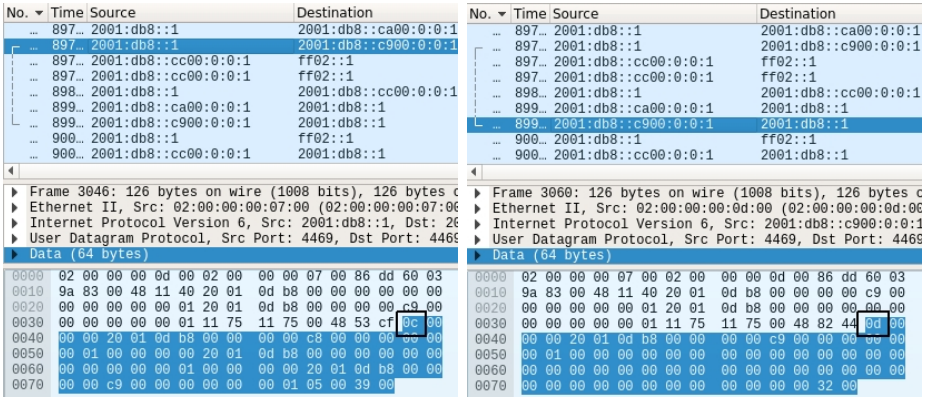

Figura 5.15: Topología 1, captura de paquetes de control FAR y FAN en `sta8` al momento de la fragmentación. En la figura de la izquierda se muestra a `sta8` enviando un paquete FAR a `sta14`, y la figura de la derecha la confirmación FAN de `sta14` a `sta8`

a `sta11`. De la dirección origen del *Echo Request*, *2001:db8::1*, se demuestra la dirección *R* del `sta8` luego de la fragmentación. Este último resultado es esperado,

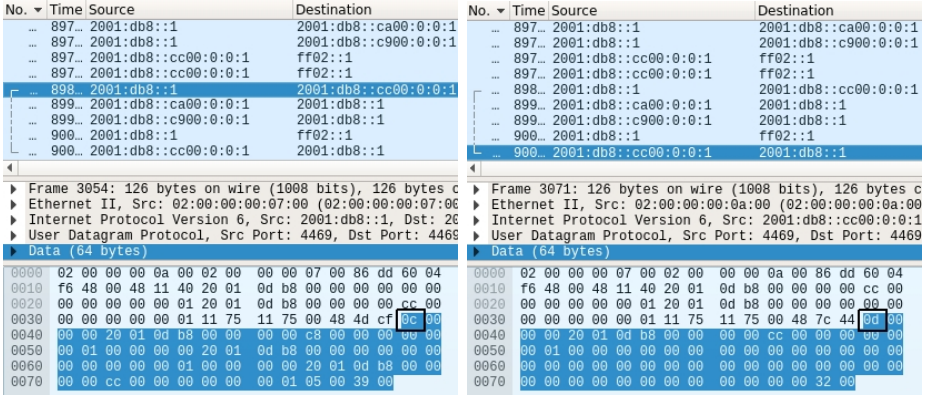

Figura 5.16: Topología 1, captura de paquetes de control FAR y FAN en `sta8` al momento de la fragmentación. En la figura de la izquierda se muestra a `sta8` enviando un paquete FAR a `sta11`, y la figura de la derecha la confirmación FAN de `sta11` a `sta8`

ya que fue el `sta8` quien disparo el mecanismo de fragmentación.

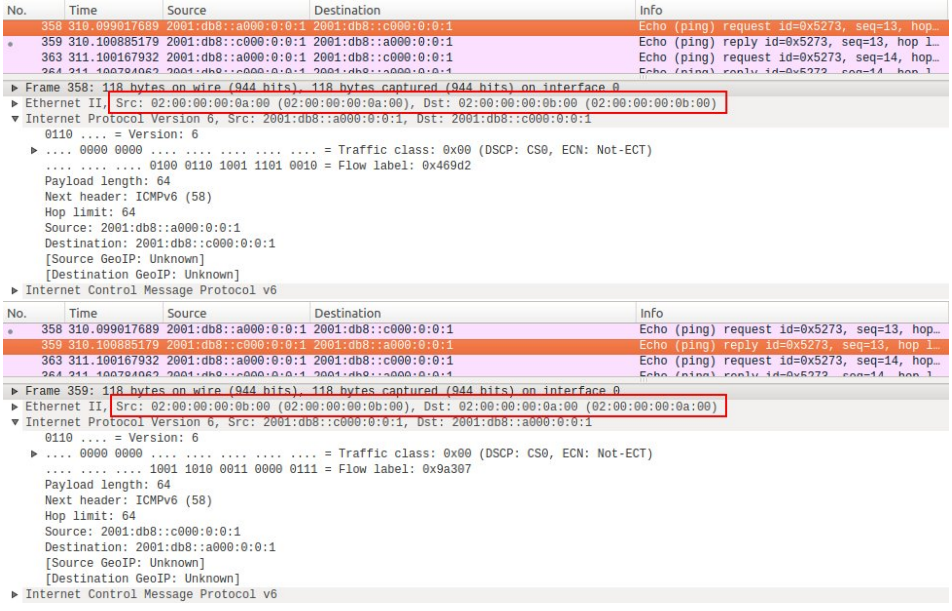

Figura 5.17: Topología 1, ping desde `sta11` a la dirección *2001:db8::c000:0:0:1* luego de la fragmentación. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*.

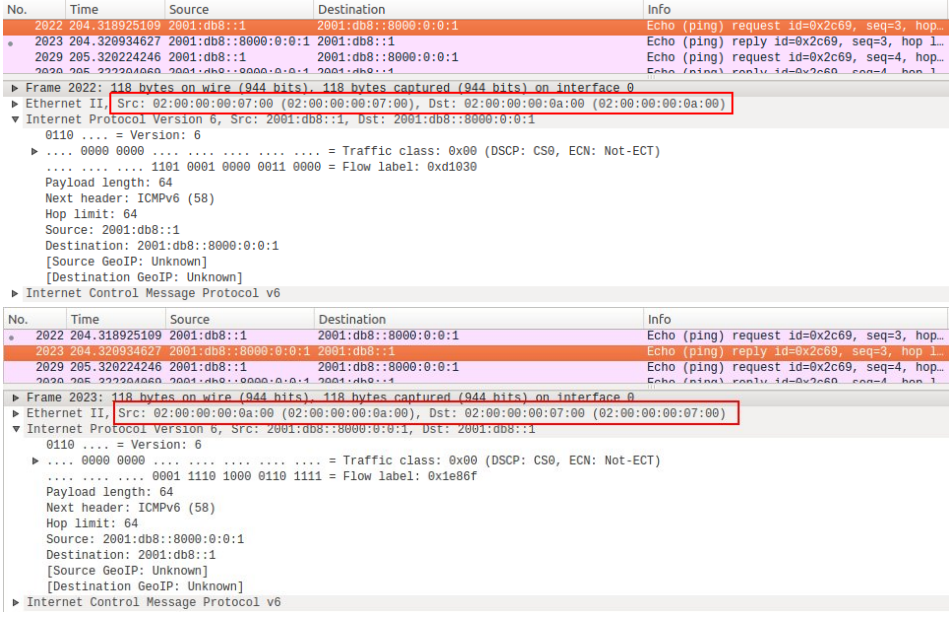

Figura 5.18: Topología 1, ping desde `sta8` a la dirección *2001:db8::8000:0:0:1* luego de la fragmentación. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*.

Topología 2 En la figura 5.19 se muestra el estado de la red al principio de la simulación. La direcciones R se asignan en función de las conexiones de los nodos en forma secuencial empezando primero con `sta1` y por último `sta16`. La direcciones obtenidas se pueden consultar en el cuadro 5.5

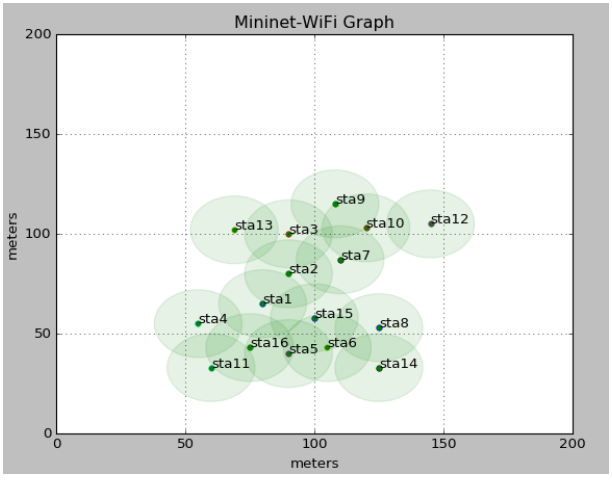

Figura 5.19: Topología 2, t=0.

Nodo	Dirección primaria	Dirección secundaria
sta1	2001:db8::1	2001:db8:0:0:1000:0:0:1
sta2	2001:db8:0:0:8000:0:0:1	-
sta3	2001:db8:0:0:c000:0:0:1	2001:db8:0:0:e000:0:0:1
sta4	2001:db8:0:0:4000:0:0:1	2001:db8:0:0:6000:0:0:1
sta5	2001:db8:0:0:2000:0:0:1	-
sta6	2001:db8:0:0:3000:0:0:1	-
sta7	2001:db8:0:0:a000:0:0:1	-
sta8	2001:db8:0:0:5000:0:0:1	-
sta9	2001:db8:0:0:d000:0:0:1	2001:db8:0:0:d800:0:0:1
sta10	2001:db8:0:0:c800:0:0:1	
sta11	2001:db8:0:0:2800:0:0:1	-
sta12	2001:db8:0:0:cc00:0:0:1	-
sta13	2001:db8:0:0:8800:0:0:1	-
sta14	2001:db8:0:0:5800:0:0:1	-
sta15	2001:db8:0:0:400:0:0:1	-
sta16	2001:db8:0:0:3800:0:0:1	-

Cuadro 5.5: Topología 2, tabla de direcciones relativas antes de la fragmentación.

La simulación evoluciona desplazando una porción de la red hacia arriba hasta que el nodo `sta7` pierde conexión con `sta15` como se ve en la figura 5.20.

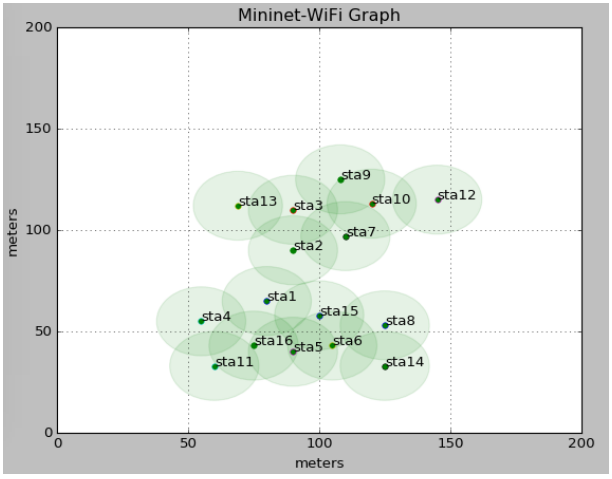

Figura 5.20: Topología 2, t=150s.

Todavía no se observa fragmentación ya que `sta2` aún tiene conexión con `sta1` y `sta15`. Se realizó un *ping* de `sta7` a la primer dirección del hipercubo (*2001:db8::1*) que está asignada al `sta1`, y en las capturas 5.21 muestran que los paquetes ICMPv6 *Echo Request* y *Echo Reply* son ruteados por el nodo `sta2` como es esperado. Las direcciones física de las tramas ethernet se pueden consultar en el cuadro 5.1.

Avanzando en el tiempo de simulación, el `sta2` pierde conexión con `sta1` y `sta15`, y se activa el mecanismo de fragmentación asignándose la dirección *2001:db8::1*. Luego envía los mensajes FAR a su vecinos repitiendo lo descripto en la simulación de la topología 1 en base la teoría desarrollada en la sección 4.2. La figura .5.22 muestra las redes fragmentada, y los cuadros 5.6 y 5.7 la direcciones resultantes de ambas.

Desde `sta7` se vuelve a capturar paquetes en base al *ping* a la dirección *2001:db8::1*, como se muestra en la figura 5.23. En dicha captura el nodo que responde con *Echo Reply* sigue siendo el `sta2`, pero con la diferencia que la dirección IPv6 del `sta7` (dirección R) cambia producto del renombramiento de los nodos que pertenecen a la Red 2 fragmentada. La dirección R resultante de *sta7* es *2001:db8::4000:0:0:1*.

Por otro lado, como `sta2` ya no tiene conexión con `sta1`, y siendo este pri-

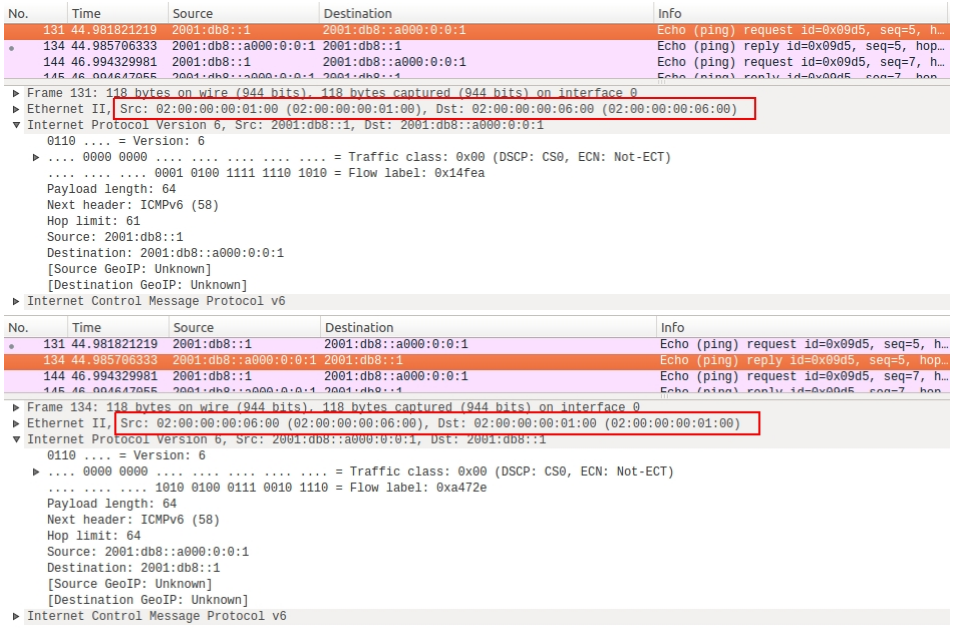

Figura 5.21: Topología 2, ping desde `sta7` a la dirección *2001:db8::1* antes de la fragmentación. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*.

Red 1		
Nodo	**Dirección primaria**	**Dirección secundaria**
sta1	2001:db8::1	2001:db8:0:0:1000:0:0:1
sta4	2001:db8:0:0:4000:0:0:1	2001:db8:0:0:6000:0:0:1
sta5	2001:db8:0:0:2000:0:0:1	-
sta6	2001:db8:0:0:3000:0:0:1	-
sta8	2001:db8:0:0:5000:0:0:1	-
sta11	2001:db8:0:0:2800:0:0:1	-
sta14	2001:db8:0:0:5800:0:0:1	-
sta15	2001:db8:0:0:400:0:0:1	-
sta16	2001:db8:0:0:3800:0:0:1	-

Cuadro 5.6: Topología 2, tabla de direcciones relativas de la Red 1 resultante de la fragmentación.

mero el que disparo el mecanismo de fragmentación, la única posibilidad es que la dirección IPv6 destino del *Echo Request* sea la asignada a `sta2` luego de la fragmentación, o sea, la dirección *2001:db8::1*.

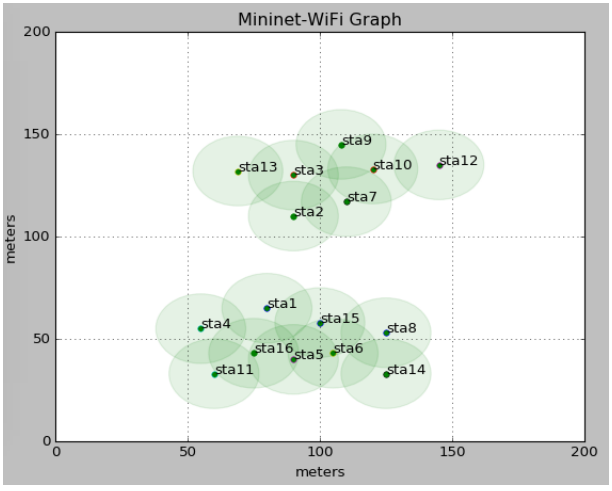

Figura 5.22: Topología 2, t=400s. Fragmentación en Red 1 abajo y Red 2 arriba.

Red 2		
Nodo	**Dirección primaria**	**Dirección secundaria**
sta2	2001:db8::1	-
sta3	2001:db8:0:0:8000:0:0:1	2001:db8:0:0:c000:0:0:1
sta7	2001:db8:0:0:4000:0:0:1	-
sta9	2001:db8:0:0:a000:0:0:1	2001:db8:0:0:b000:0:0:1
sta10	2001:db8:0:0:9000:0:0:1	-
sta12	2001:db8:0:0:9800:0:0:1	-
sta13	2001:db8:0:0:1000:0:0:1	-

Cuadro 5.7: Topología 2, tabla de direcciones relativas de la Red 2 resultante de la fragmentación.

Desde nodo `sta3` se realizó un *ping* hacia el nodo `sta12`, con el comando `pin6 -I wlan0 nodo12`. Los paquetes ICMPv6 *Echo Request* y *Echo Reply* son ruteados mediante el `sta10`. Se realizó capturas en `sta10` mostradas en las figura 5.24 y 5.25 para confirmar el ruteo de los paquetes en el *ping*.

El `sta3` al resolver la dirección *U* del `nodo12` por medio del servicio *Rendez Vous*, obtiene la dirección IPv6 *2001:db8::9800:0:0:1* correspondiente a la dirección *R* de `sta12`. Como este último no es vecino, `sta3` instala una ruta en donde el siguiente salto es el nodo `sta10` enviando el paquete ICMPv6 *Echo Request* con dirección MAC destino `02:00:00:00:09:00` en el *header* de la trama ethernet como muestra la captura 5.24. El `sta10` al recibir dicho paquete y verificar que la

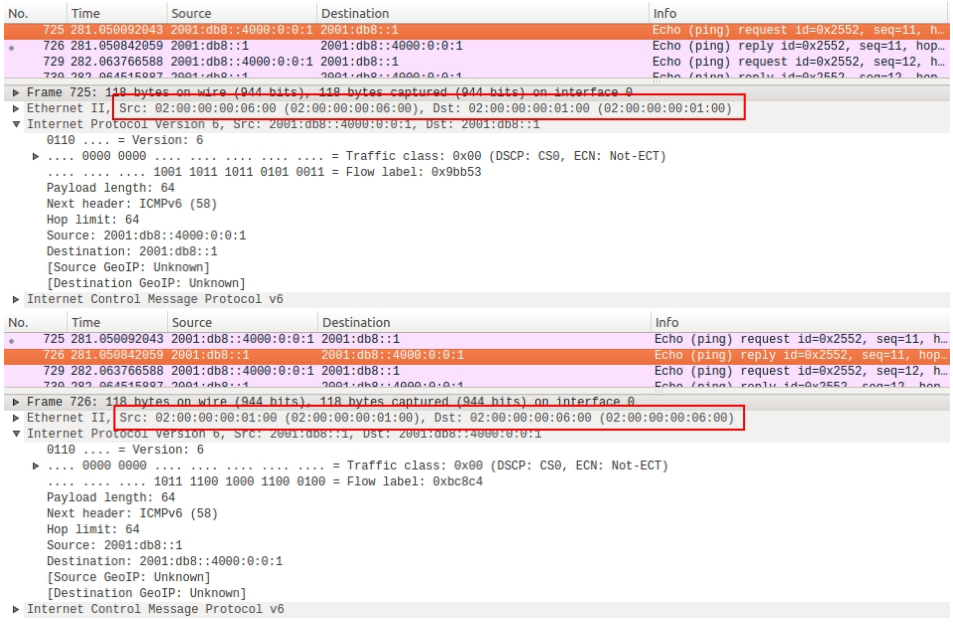

Figura 5.23: Topología 2, ping desde `sta7` a la dirección *2001:db8::1* después de la fragmentación. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*.

dirección IPv6 destino no es la que tiene asignada, ANTop instala la ruta hacia el destino `sta12` que resulta ser vecino y reenvía el paquete *Echo Request* generando la trama ethernet con dirección MAC destino igual a `02:00:00:00:0b:00`. El `sta12` al recibir el mensaje *Echo Request*, responde con una paquete ICMPv6 *Echo Reply* repitiendo el mismo proceso entre los mismo nodos mencionados pero en dirección contraria como se muestra en las capturas de la figura 5.25.
Se puede observar de las capturas que las direcciones R de cada nodo corresponde a las asignadas en la red fragmentada (Red 2) de la tabla 5.7. También queda comprobado la consistencia de dicha red quedando intacto los servicios que presta el protocolo ANTop.

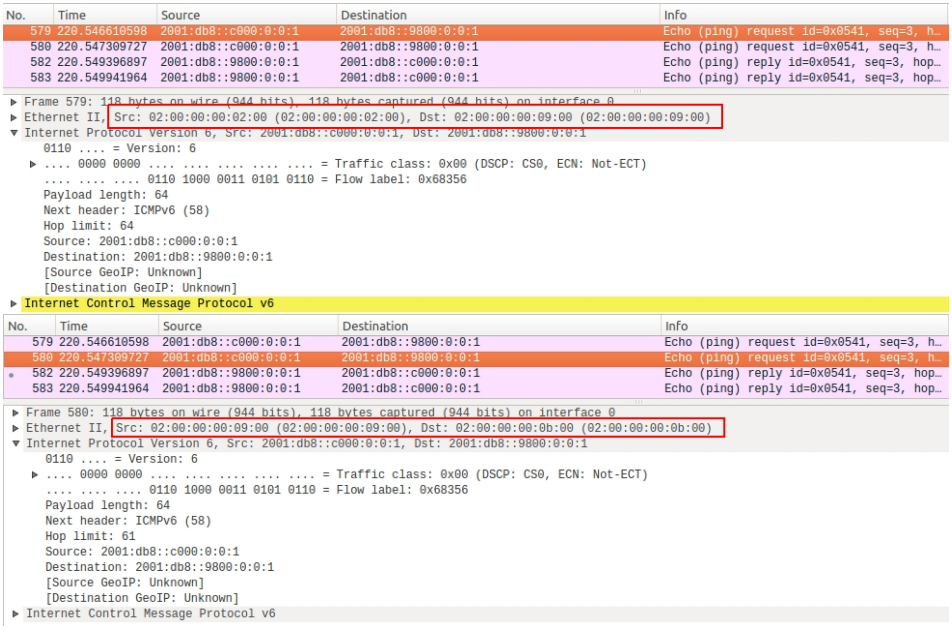

Figura 5.24: Topología 2, ruteo de paquetes ICMPv6 *Echo Request* en `sta10` producto del *ping* de `sta3` a `sta12`, luego de la fragmentación.

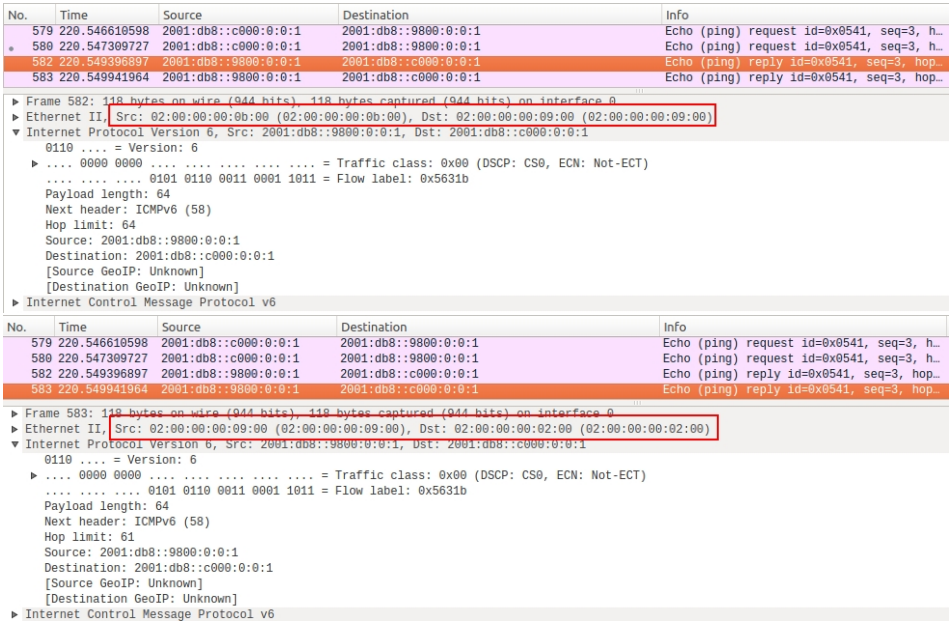

Figura 5.25: Topología 2, ruteo de paquetes ICMPv6 *Echo Reply* en `sta10` producto del *ping* de `sta3` a `sta12`, luego de la fragmentación.

5.2.3. Mezcla de dos redes

En este apartado se presentan las simulaciones de la mezcla de dos redes ANTop para comprobar la teoría desarrollada en la sección 4.3. Las simulaciones consisten en la construcción de dos redes que luego de determinado tiempo de simulación se mezclan formando así una sola red. Se realizaron 2 simulaciones con topologías diferentes.

Desarrollo de la Simulación

Para el desarrollo de las simulaciones se utiliza la misma vista en la sub sección anterior 5.2.2. En este caso la movilidad en el script Python se configura para que una de las redes se acerque hacia la otra en el tiempo y se mezcla en algún momento. En el código del apéndice A.3 se describe el script Python utilizado en la primer topología simulada. También se utiliza la configuración por defecto de la direcciones físicas de cada nodos descriptas en la tabla 5.1.

Uno de los nodos que pertenece a la red que se mueve, se configura para que ejecute un *ping* a una dirección U de un nodo en particular antes y después de la mezcla de ambas redes.

Análisis de datos obtenidos

Topología 1 En la figura 5.26 se muestra las redes al principio de la simulación. Como se menciono en las simulaciones anteriores, la conexiones de los nodos ocurre en forma secuencial, conectándose primero el nodo `sta1` y por último el nodo `sta16`. La red de la derecha se nombra como Red 1 y la de la izquierda como Red 2.

En este momento las redes se encuentran totalmente separadas. Sin embargo se verifico la conexión desde `sta8` con el nodo `sta15` mediante el comando `ping6 -I wlan0 nodo15`. Como es esperado, el `sta8` no logra resolver el nombre `nodo12` y por ende deja de ejecutar el *ping*.

Los datos de direcciones R de cada red que se obtuvieron hasta el momento se muestran en los cuadros 5.8 y 5.9.

Siguiendo el tiempo de simulación, una de la redes se moviliza hasta que el nodo `sta6` entra dentro del alcance del `sta15` como se muestra en la figura 5.27. En este punto, se produce la mezcla entre ambas redes. Repasando la sección 4.3, en la mezcla de dos redes, una de ellas tiene que ser renombrada por la otra para conservar la estructura direccionamiento por hipercubo que propone ANTop. Esto

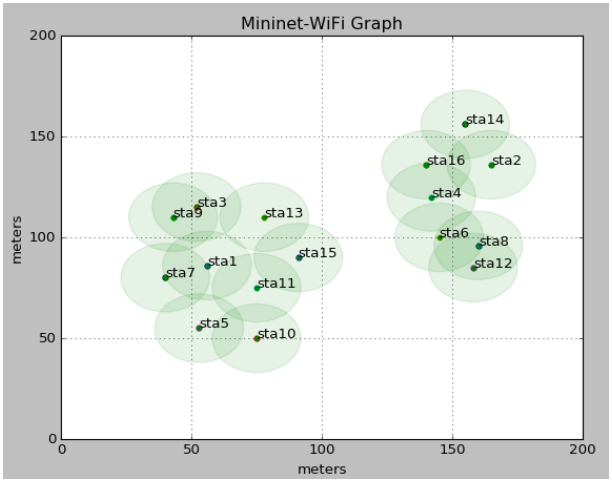

Figura 5.26: Topología 1, t=0.

Red 1		
Nodo	**Dirección primaria**	**Dirección secundaria**
sta1	2001:db8::1	2001:db8:0:0:2000:0:0:1
sta3	2001:db8:0:0:8000:0:0:1	-
sta5	2001:db8:0:0:4000:0:0:1	2001:db8:0:0:4800:0:0:1
sta7	2001:db8:0:0:6000:0:0:1	-
sta9	2001:db8:0:0:c000:0:0:1	2001:db8:0:0:e000:0:0:1
sta10	2001:db8:0:0:5000:0:0:1	-
sta11	2001:db8:0:0:800:0:0:1	-
sta13	2001:db8:0:0:a000:0:0:1	-
sta15	2001:db8:0:0:b000:0:0:1	-

Cuadro 5.8: Topología 1, tabla de direcciones relativas de la Red 1 antes de la mezcla.

Red 2		
Nodo	**Dirección primaria**	**Dirección secundaria**
sta2	2001:db8::1	-
sta4	2001:db8:0:0:8000:0:0:1	-
sta6	2001:db8:0:0:c000:0:0:1	2001:db8:0:0:e000:0:0:1
sta8	2001:db8:0:0:a000:0:0:1	-
sta12	2001:db8:0:0:b000:0:0:1	-
sta14	2001:db8:0:0:4000:0:0:1	2001:db8:0:0:6000:0:0:1
sta16	2001:db8:0:0:2000:0:0:1	-

Cuadro 5.9: Topología 1, tabla de direcciones relativas de la Red 2 antes de la mezcla.

último, la decisión de quien renombra las direcciones es a partir de que si el valor

de *número de red* del mismo es mayor a la de la otra red. En este caso, el *número de red* de la Red 1 tiene el valor de 0, mientras que la Red 2 tiene valor *número de red* de 1. Cabe recordar que en este trabajo se definió el *número de red* como el cuarto byte de la *MAC address* del nodo que inicio la red. De la tablas de direcciones física que figura en el cuadro 5.1, el cuarto byte vale cero para todos los nodos, y el único byte que difiere de un nodo a otro es el quinto. Si bien no se menciono anteriormente, pero para las simulaciones de este capítulo se definió en la aplicación ANTop que el *número de red* es el quinto byte de la *MAC address* del primer nodo de la red. Aclarado esto último, el valor de 0 del *número de red* de la Red 1 es el quinto byte de la dirección física del nodo `sta1`, y por otro lado el valor de 1 de la Red 2 corresponde al quinto byte de la dirección física del `sta2`. Entonces la Red 2 será quien renombre las direcciones R a los nodos participantes.

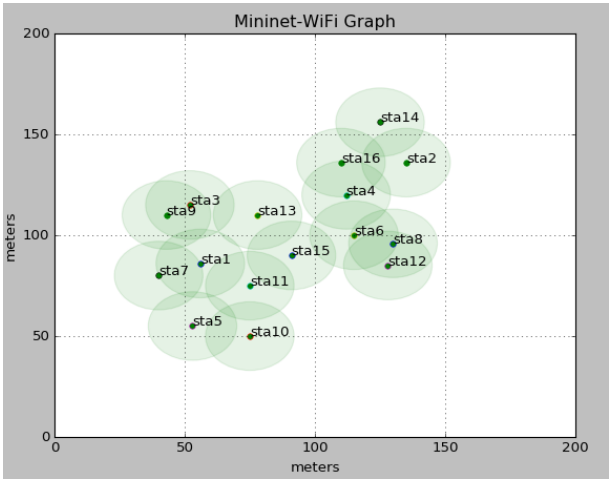

Figura 5.27: Topología 1, t=200s. Mezcla de redes, Red 2 es absorbida por la Red 1.

Cuando `sta15` recibe un HB del nodo `sta6`, chequea el campo *número de red* y al verificar que tiene un valor mayor al de él mismo, activa el mecanismo de mezcla enviando un paquete MAR1 a `sta6` para que comience el mecanismo de ajuste de las direcciones R de todos los nodos de la red que pertenece. Desde `sta6` se realizaron las capturas 5.28, 5.29 y 5.30 de los paquetes de control en el momento de la mezcla.

En las capturas, se marca en recuadro negro el tipo de paquete de control

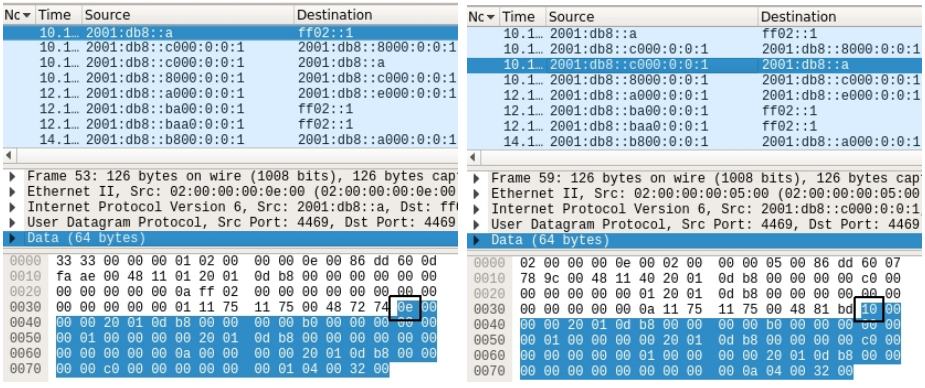

Figura 5.28: Topología 1, captura de paquetes de control MAR1, MAR2, MAN1 y MAN2 en `sta6` al momento de la mezcla. En la figura de la izquierda se muestra a `sta15` enviando un paquete MAR1 en modo multicast, y la figura de la derecha la confirmación MAN1 de `sta6` a `sta15`

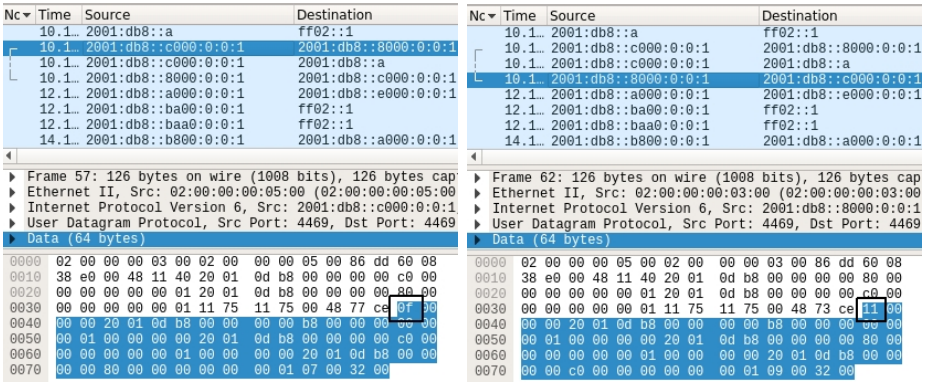

Figura 5.29: Topología 1, captura de paquetes de control MAR1, MAR2, MAN1 y MAN2 en `sta6` al momento de la mezcla. En la figura de la izquierda se muestra a `sta6` enviando un paquete MAR2 a `sta4`, y la figura de la derecha la confirmación MAN2 de `sta4` a `sta6`

ANTop. El valor de $0x0e$ (14 en decimal) corresponde a un mensaje MAR1, el valor de $0x0f$ (15 en decimal) a un mensaje MAR2, el de $0x10$ (16 en decimal) al mensaje MAN1, y por último el valor $0x11$ (17 en decimal) al MAN2.

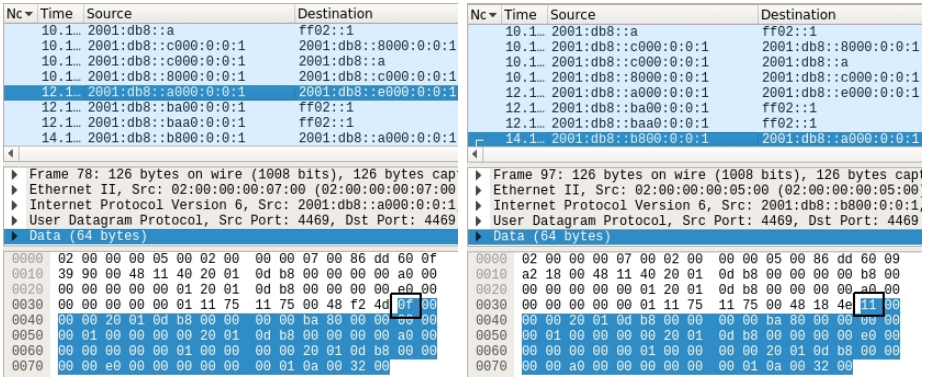

Figura 5.30: Topología 1, captura de paquetes de control MAR1, MAR2, MAN1 y MAN2 en `sta6` al momento de la mezcla. En la figura de la izquierda se muestra a `sta8` enviando un paquete MAR2 a `sta6`, y la figura de la derecha la confirmación MAN2 de `sta6` a `sta8`

En la captura 5.28 el mensaje MAR1 enviado por `sta15` no tiene destino a la dirección IPv6 del `sta6`, si no que lo hace en modo multicast con dirección destino *ff02::1*. Entonces cualquier nodo que esté en el entorno de este último, puede procesar dicho paquete. Sin embargo, la dirección que utiliza para enviar el mensaje no lo hace con su dirección primaria IPv6, si no que usa una dirección *2001:db8::a* que en la sección 4.3 la llamamos *default*, y es utilizada justamente para para evitar posibles conflictos de direcciones en ambas redes en el momento que empiezan a mezclarse. Los nodos que están dentro del entorno del `sta15` y que pertenecen a la misma red que este último, filtran el mensaje MAR1 y no lo procesan. En cambio el `sta6` toma el mensaje MAR1 y lo procesa activando los mecanismo de mezcla. En la captura se muestra un mensaje de confirmación MAN1 que es el enviado por `sta6` hacia `sta15` como confirmación del mensaje MAR1 que inicio la mezcla.

En el cuadro 5.9 se observa que `sta6` no tiene ningún nodo subordinado en la jerarquía de árbol de direccionamiento por hipercubo, solo al vecino `sta4` que resulta ser su *padre*. Entonces, activa el mecanismo 2 4.3.2 y envía un mensaje MAR2 a este último como se muestra en la captura 5.29. Luego el `sta4` hace lo mismo con sus vecinos hasta barrer todos los nodos de la Red 2. Por cada mensaje MAR2 que envía cada nodo, les llega el mensaje de confirmación MAN2 de los nodos vecinos correspondientes, como pasa en dicha captura en donde `sta6` recibe

el mensaje de confirmación MAN2 del nodo `sta4`.

Por otro lado, en la captura 5.30 se observa un mensaje MAR2 desde el nodo `sta8` al `sta6` con dirección destino a la secundaria de este último. Si se recuerda, el *flooding* de MAR1 para el mecanismo 1 4.3.2 o MAR2 para el mecanismo 2 4.3.2, se lo hace mediante aquellos vecinos que tienen distancia 1 con respecto a la dirección R primaria. Es por eso que `sta8` siendo vecino del `sta6`, este último no le haya enviado MAR2 debido a que desde su punto de vista son vecinos mediante su dirección secundaria. En cambio `sta8` tiene como vecino al `sta6` y llegado el momento de activar el proceso de mezcla, le transmite el mensaje MAR2 como en este caso. El nodo `sta6` al estar al tanto de la cuestión le confirma con un mensaje MAN2 cerrando el ciclo del mecanismo de mezcla.

Luego de haber concluido el mecanismo de mezcla, la Red 2 desaparece del esquema y queda consumada solo la Red 1 con nuevos miembros. La direcciones R de cada nodo está detallado en el cuadro 5.10 en función de los datos obtenidos en la simulación.

Red 1		
Nodo	**Dirección primaria**	**Dirección secundaria**
sta1	2001:db8::1	2001:db8:0:0:2000:0:0:1
sta2	2001:db8:0:0:b980:0:0:1	-
sta3	2001:db8:0:0:8000:0:0:1	-
sta4	2001:db8:0:0:b900:0:0:1	2001:db8:0:0:b920:0:0:1
sta5	2001:db8:0:0:4000:0:0:1	2001:db8:0:0:4800:0:0:1
sta6	2001:db8:0:0:b800:0:0:1	-
sta7	2001:db8:0:0:6000:0:0:1	-
sta8	2001:db8:0:0:b940:0:0:1	-
sta9	2001:db8:0:0:c000:0:0:1	2001:db8:0:0:e000:0:0:1
sta10	2001:db8:0:0:5000:0:0:1	-
sta11	2001:db8:0:0:800:0:0:1	-
sta12	2001:db8:0:0:b950:0:0:1	-
sta13	2001:db8:0:0:a000:0:0:1	-
sta14	2001:db8:0:0:b9c0:0:0:1	-
sta15	2001:db8:0:0:b000:0:0:1	-
sta16	2001:db8:0:0:b9a0:0:0:1	-

Cuadro 5.10: Topología 1, tabla de direcciones relativas de la Red 1 después de la mezcla.

Para comprobar que el mecanismo de mezcla haya sido exitoso, se realizó un *ping* desde el nodo `sta8` (ex integrante de la Red 2) hacia el `sta15`. Las capturas se realizaron en el `sta8` 5.31, y en el nodo `sta6` (5.32 y 5.33).

En la captura 5.31, se muestra que el paquete ICMPv6 *Echo Request* y *Echo Reply* son ruteados mediante el nodo vecino con dirección física `02:00:00:00:03:00`, que en el cuadro 5.1 hace referencia al nodo `sta4`. Este resultado es esperable, ya

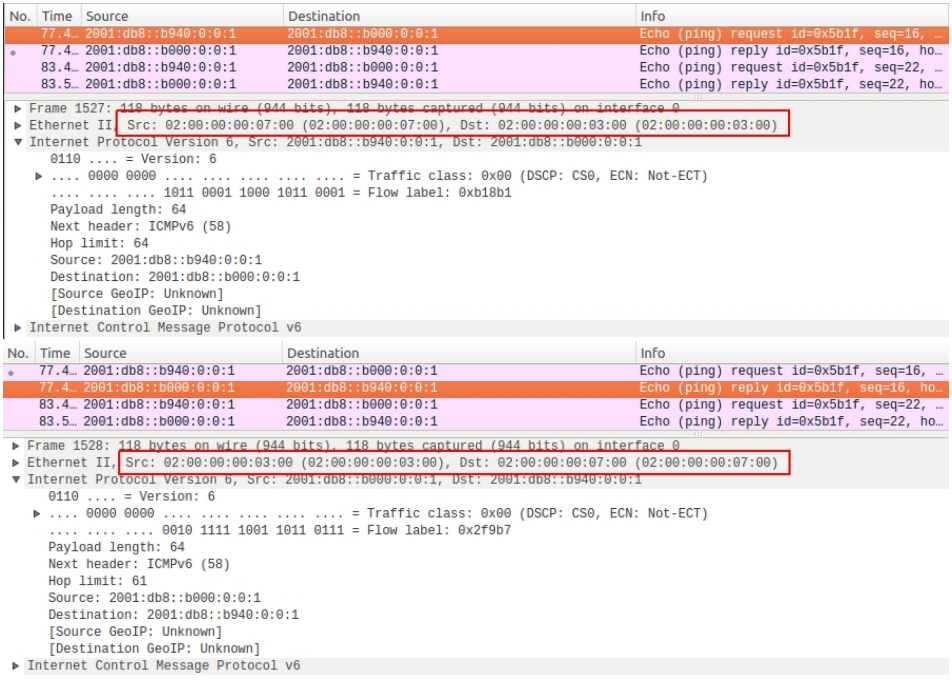

Figura 5.31: Topología 1, ping desde `sta8` al nodo `sta15` después de la mezcla. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*.

que si se observa el cuadro 5.10, la dirección *R* del `sta4` es *2001:db8::b900:0:0:1* que resulte ser el nodo *padre* del `sta8` (*2001:db8::b940:0:0:1*) que tiene solo a éste como vecino.

Las captura 5.32 muestra que los paquetes *Echo Request* son recibidos del `sta4` y luego reenviados al nodo `sta5` con dirección física `02:00:00:00:0e:00`. Sucede lo mismo para la captura 5.33 que muestra el ruteo de los paquetes *Echo Request*, en donde el nodo `sta6` lo recibe del `sta15` y luego es reenviado al `sta4`.

En conclusión la ruta que lleva a cabo el *ping* desde el `sta8` al `sta15`, lo hace por medio de los nodos `sta4` y `sta6`. El ruteo resulta el esperado con respecto al ruteo reactivo ANTop.

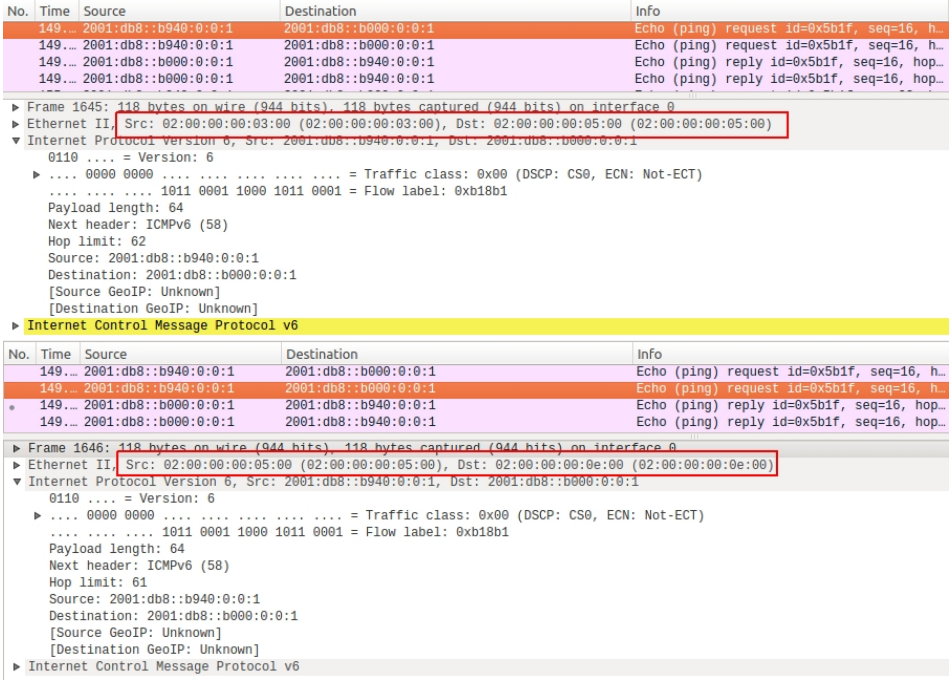

Figura 5.32: Topología 1, ruteo de paquetes ICMPv6 *Echo Request* en `sta6` producto del *ping* de `sta8` a `sta15`, luego de mezcla.

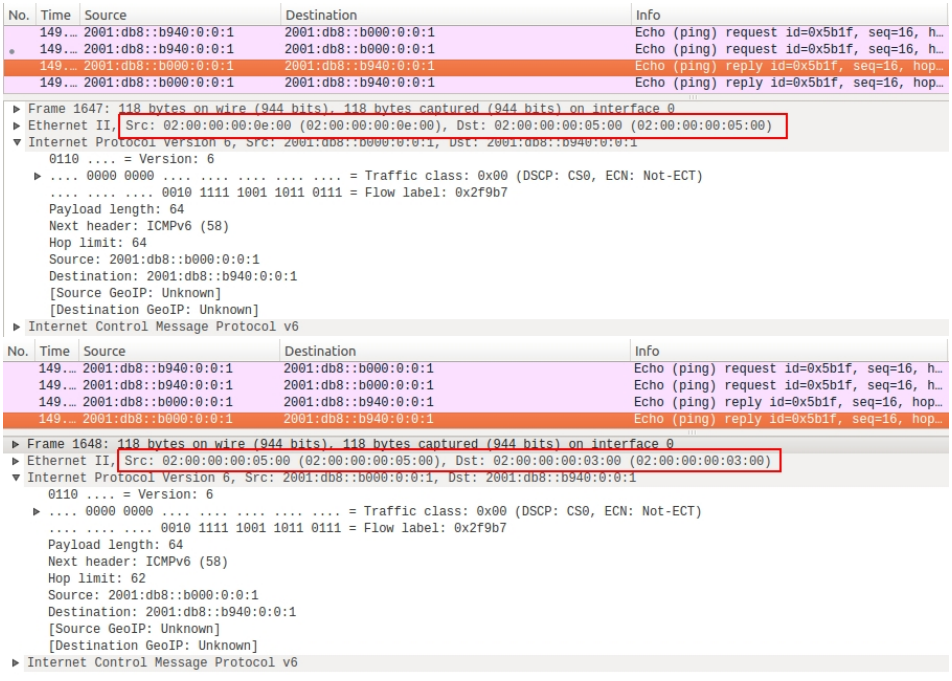

Figura 5.33: Topología 1, ruteo de paquetes ICMPv6 *Echo Reply* en `sta6` producto del *ping* de `sta8` a `sta15`, luego de mezcla.

Topología 2 En la figura 5.34 se observa las redes al comienzo de la simulación. Como en las simulaciones anteriores, la conexion de nodos ocurre en forma secuencial, siendo el primero el conectarse el nodo `sta1` y por último el `sta16`. La red de abajo la llamaremos Red 1, mientras que la red de arriba se nombra como Red 2.

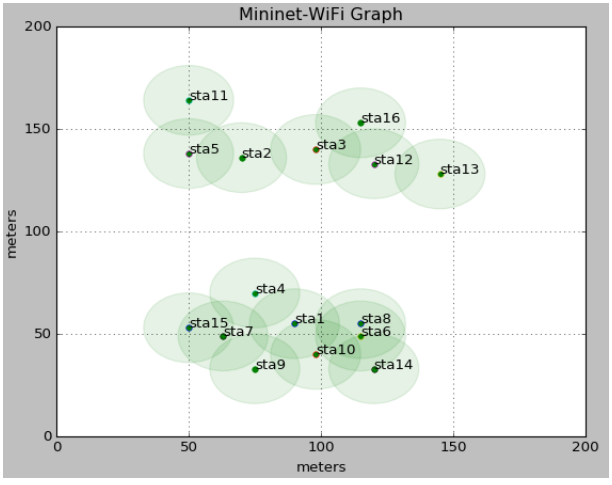

Figura 5.34: Topología 2, t=0.

Los datos de direcciones R obtenidos en la simulación, se muestran en los cuadros 5.11 y 5.12.

Red 1		
Nodo	**Dirección primaria**	**Dirección secundaria**
sta1	2001:db8::1	-
sta4	2001:db8:0:0:8000:0:0:1	-
sta6	2001:db8:0:0:4000:0:0:1	2001:db8:0:0:6000:0:0:1
sta7	2001:db8:0:0:c000:0:0:1	-
sta8	2001:db8:0:0:2000:0:0:1	2001:db8:0:0:3000:0:0:1
sta9	2001:db8:0:0:e000:0:0:1	-
sta10	2001:db8:0:0:1000:0:0:1	
sta14	2001:db8:0:0:4800:0:0:1	-
sta15	2001:db8:0:0:a000:0:0:1	-

Cuadro 5.11: Topología 2, tabla de direcciones relativas de la Red 1 antes de la mezcla.

Al correr la simulación, la Red 2 se irá desplazando hacia abajo, hasta llegado un momento que el nodo `sta2` entra en conexión con el nodo `sta4` como se muestra

Red 2		
Nodo	**Dirección primaria**	**Dirección secundaria**
sta2	2001:db8::1	-
sta3	2001:db8:0:0:8000:0:0:1	-
sta5	2001:db8:0:0:4000:0:0:1	-
sta11	2001:db8:0:0:6000:0:0:1	-
sta12	2001:db8:0:0:c000:0:0:1	-
sta13	2001:db8:0:0:e000:0:0:1	-
sta16	2001:db8:0:0:a00:0:0:1	

Cuadro 5.12: Topología 2, tabla de direcciones relativas de la Red 2 antes de la mezcla.

en la figura 5.35, produciendo así la mezcla.

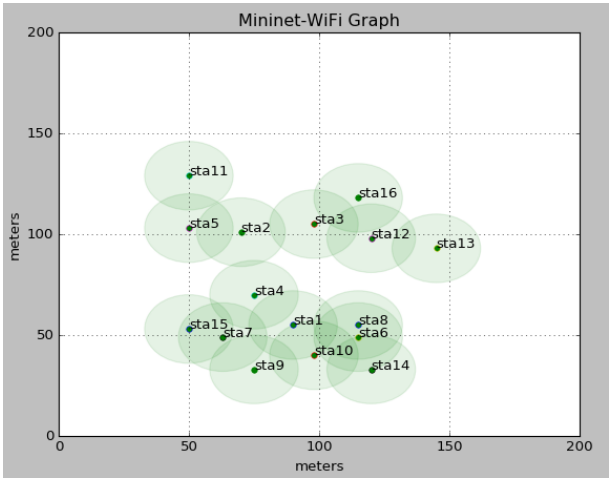

Figura 5.35: Topología 2, t=350s. Mezcla de redes, Red 2 es absorbida por la Red 1.

Como se menciono en la simulación anterior, la red que tenga mayor *número de red* es la que deberá ajustar las direcciones R de todos los nodos en función de la otra red. En este caso, la Red 1 tiene un *número de red* 0 y la Red 2 el valor de 1. Entonces la Red 2 es el que dispara el mecanismo de mezcla.

El nodo `sta4` es el encargado de enviar en modo multicast el mensaje MAR1 en el cual solo el nodo `sta2` lo procesa y comienza el mecanismo mezcla. El `sta1` al tener la primer dirección del hipercubo en la Red 2, no tiene nodo *padre* y por ende todos sus vecinos son subordinados. El mecanismo que se dispara, en este caso, es

el mecanismo 1 4.3.2 en donde `sta1` le envía a sus vecinos el mensaje MAR1, y a su vez recibe las confirmaciones MAN1 de los mismos. Luego dichos nodos vecinos hacen los mismos pero con sus propios vecinos, y así hasta barrer toda la red 2.

Al terminar la movilidad de los nodos de la Red 2, como se muestra en la figura 5.36, se recolectaron los datos de la direcciones R de la red resultante luego del proceso de mezcla y se los puede consultar en el cuadro 5.13.

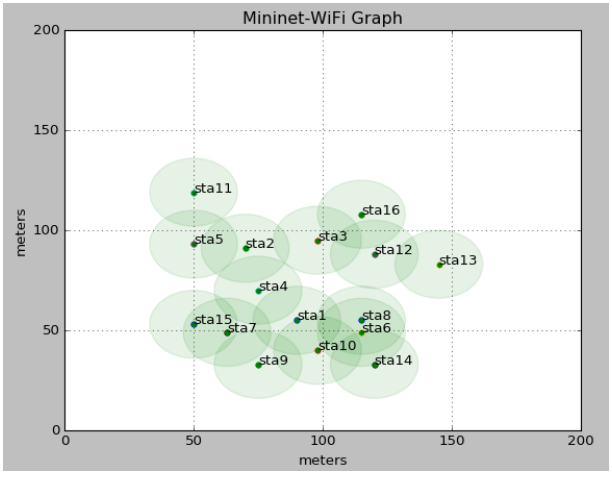

Figura 5.36: Topología 2, t=500s.

En la captura 5.37 del nodo `sta1` se observa los paquetes ICMPv6 *Echo Request* y *Echo Reply* producto de un *ping* al nodo `sta2`. La ruta que toma los paquetes es por medio del nodo `sta4` con dirección física `02:00:00:00:03:00`.

Luego se realizó desde el nodo `sta7` un *ping* a los nodos `sta2` y `sta3`. En las capturas 5.38 y 5.39 se muestran los paquetes ICMPv6 *Echo Request* y *Echo Reply* que son ruteados en ambos casos también por el nodo `sta4`.

En las capturas realizadas, se puede concluir que las direcciones R de los nodos que se sumaron a la Red 1, coinciden con los datos obtenidos del cuadro 5.13. También se puede observar que luego del proceso de mezcla, el ruteo de los paquetes es acorde al ruteo reactivo ANTop, y las direcciones R asignadas a los nodos corresponde a la estructura de direccionamiento por hipercubo. Se puede concluir que la mezcla fue exitosa.

Red 1		
Nodo	**Dirección primaria**	**Dirección secundaria**
sta1	2001:db8::1	-
sta2	2001:db8:0:0:9000:0:0:1	-
sta3	2001:db8:0:0:9800:0:0:1	-
sta4	2001:db8:0:0:8000:0:0:1	-
sta5	2001:db8:0:0:9400:0:0:1	-
sta6	2001:db8:0:0:4000:0:0:1	2001:db8:0:0:6000:0:0:1
sta7	2001:db8:0:0:c000:0:0:1	-
sta8	2001:db8:0:0:2000:0:0:1	2001:db8:0:0:3000:0:0:1
sta9	2001:db8:0:0:e000:0:0:1	-
sta10	2001:db8:0:0:1000:0:0:1	
sta11	2001:db8:0:0:9600:0:0:1	-
sta12	2001:db8:0:0:9c00:0:0:1	-
sta13	2001:db8:0:0:9e00:0:0:1	-
sta14	2001:db8:0:0:4800:0:0:1	-
sta15	2001:db8:0:0:a000:0:0:1	-
sta16	2001:db8:0:0:9a00:0:0:1	-

Cuadro 5.13: Topología 2, tabla de direcciones relativas de la Red 1 después de la mezcla.

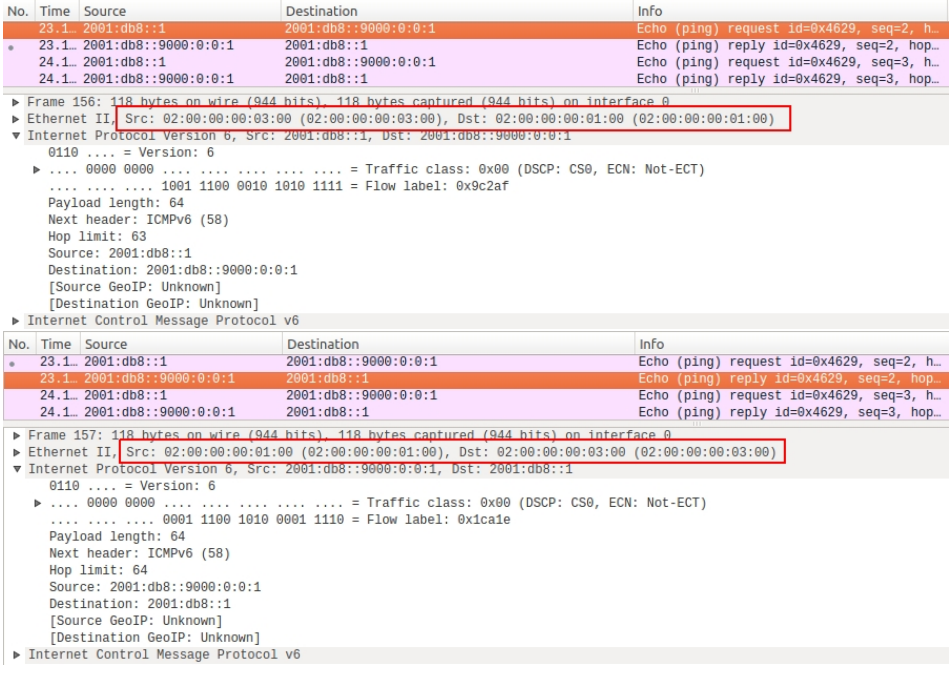

Figura 5.37: Topología 2, ping desde `sta1` al nodo `sta2` después de la mezcla. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*.

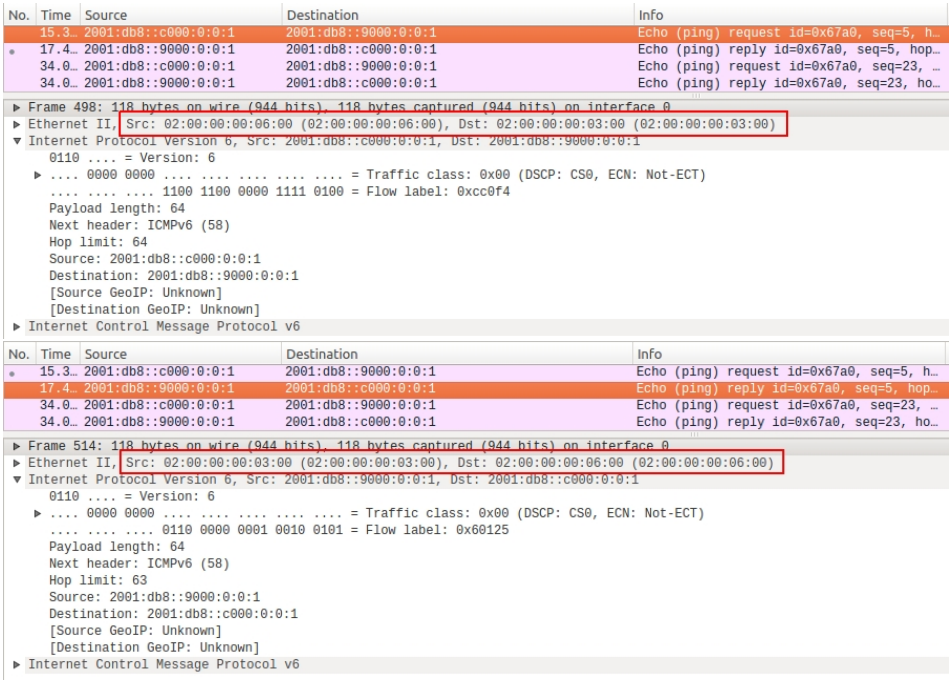

Figura 5.38: Topología 2, ping desde `sta7` al nodo `sta2` después de la mezcla. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*.

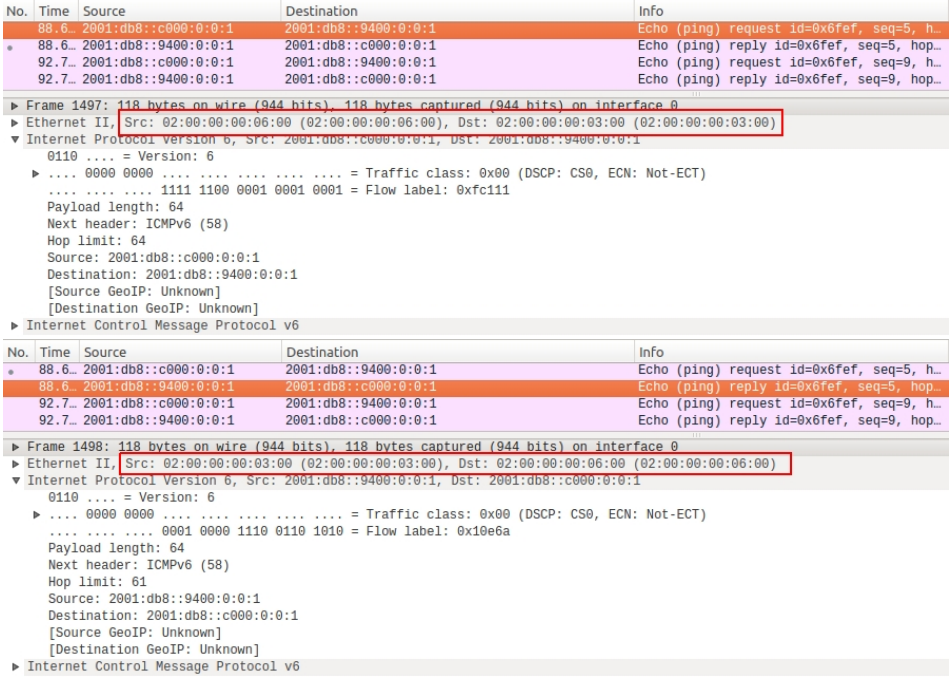

Figura 5.39: Topología 2, ping desde `sta7` al nodo `sta3` después de la mezcla. En la figura de arriba se muestra el paquete *Echo Request* y la de abajo la respuesta *Echo Reply*.

Capítulo 6

Conclusiones

En la presente tesis se partió de los trabajos [1], [2] y [4], donde se desarrolla distintos aspectos del protocolo ANTop (*Adjacent Network Topologies*) de redes *ad hoc* basados en hipercubos.

Se logró correr la implementación ANTop del trabajo [4] sobre IPv6 en un sistema operativo Ubuntu versión 16.04, en el cual se realizaron pruebas del funcionamiento y correcciones en el campo del direccionamiento IPv6, gestión de direcciones secundarias, gestión de direcciones recuperadas, ruteo reactivo y servicio de resolución de nombres *Rendez Vous*. Se cambió el *makefile* del modulo kernel del código, para que pueda ser compilado en una versión kernel 4.10.0.
El lenguaje utilizado para programar el demonio fue *C*, ya que tanto el código ANTop como las interfaces del sistema operativo están escrita con dicho lenguaje.

Una vez obtenido una versión de software actualizado de la implementación ANTop, se estudió la herramienta de simulación Mininet Wifi para representar distintas topología de redes *ad hoc*. Dicha plataforma está orientada a redes inalámbricas, en donde cada estación de Mininet Wifi se comporta como una maquina real logrando ejecutar el programa ANTop en cada una de ellas. También fue posible la construcción de ciertas topologías con posiciones preestablecidas de las estaciones en un plano y se definió la movilidad de algunas de ellas. Esto permitió realizar pruebas del protocolo ANTop hasta 16 nodos inalámbricos en simultaneo en una simulación.

Se realizaron ensayos en Mininet Wifi recreando la fragmentación de una red y luego la mezcla de dos redes. Se tomaron en cuenta los problemas encontrados en ambos casos desde punto de vista del funcionamiento del protocolo ANTop.

Se logró desarrollar una solución con respecto a la fragmentación de una red ANTop, en donde se diseñó algoritmos de detección y un mecanismo de renombramiento de direcciones R y otros parámetros para los nodos afectados. Se diseñaron los mensajes FAR y FAN que fueron integrados en la cartera de paquetes de control del protocolo ANTop. Luego se crearon algoritmos que gestionan el procesamiento y el envío de los mensaje FAR y FAN, en base al mecanismo de fragmentación desarrollado teóricamente.

Fue posible encontrar un solución con respecto a la mezcla de redes ANTop, para ello primero se implementó e incorporó el concepto de numeración de redes, que permitió resolver la detección cuando se juntan dos o más redes. El *número de red* se agregó como un parámetro más del protocolo, en el cual se diseñó el algoritmo de asignación del valor de dicho parámetro en el momento que un dispositivo corre ANTop.

En base a este último concepto, se diseñó los algoritmos de detección de mezcla de redes ANTop y se logró desarrollar una serie de mecanismos que renombran la dirección R y otros parámetros de los nodos afectados. Para ello se diseñaron los mensajes MAR1, MAR2, MAN1 y MAN2 que fueron adheridos como paquetes de control de ANTop. También se diseñó una serie de algoritmos que controlan el procesamiento y el envío de los paquetes de control recién mencionados, y que responde a los mecanismos de mezcla desarrollados teóricamente.

Se logró programar los algoritmos de fragmentación y mezcla en la implementación ANTop, y se realizaron pruebas de funcionamiento mediante topologías simuladas y diseñadas en Mininet Wifi por medio de scripts construidos en lenguaje Python. La versión del software está disponible a la comunidad en paquete Debian [19] y se podrá instalarlo en un sistema operativo Linux 4.10.0.

Se obtuvieron resultados de funcionamiento de redes ANTop en un esquema de nodos cercanos y otro más alejados entre sí, obteniendo información y un posterior analisis de la cantidad de vecinos, cantidad de entradas máximas en la tabla de rutas y cantidad de entradas en la tabla *Rendez Vous*. Luego se logró recrear esquemas de simulación para representar la fragmentación, comprobando la conexión de ciertos nodos antes y después de dicha fragmentación, y por último se corroboró la generación de paquetes de control FAR y FAN en un nodo particular. Fue posible crear un esquema de simulación para comprobar la mezcla de dos redes, en donde se obtuvieron capturas de los paquetes de control MAR1, MAR2, MAN1 y MAN2 en ciertos nodos. También se probó el ruteo de datos y la conectividad entre diferentes nodos para demostrar la consistencia del protocolo ANTop

en la red mezclada.

6.1. Trabajos futuros

- *Prueba de funcionamiento*: La nueva versión de software ANTop soporta la fragmentación y mezcla de redes. Sin embargo, se deberá seguir con las pruebas en concepto de evaluación y detección de problemas, para luego encontrar una solución.
- *Función Hash*: Debido a la forma de obtención de los servidores *Rendez Vous*, aquellos nodos que tienen espacios de direcciones más grande tienen grandes chances de que sea un servidor *Rendez Vous* obteniendo a su vez tabla más grande provocando que sean consultados frecuentemente, aumentando el tráfico en ellos. Es deseable encontrar una forma de distribuir los servidores *Rendez Vous* de tal forma que los tamaños de las tablas sean parejas. Se podría armar una Hash que sea dinámica en función de la topología presente.
- *Algoritmo de decisión en mezcla*: En el presente trabajo la decisión de cuales de las redes ANTop dispara el mecanismo de renombramiento en la mezcla, se optó por hacerla en forma aleatoria. A futuro es deseable estimar la cantidad de nodos de una red y otra, para luego tener la certeza de que la red que será renombrada sea la de menor número de integrantes.

Apéndice A

Scripts de simulación

En esta sección se presentan los scripts en Python utilizado en las simulaciones 5 de este trabajo.

A.1. Script de simulación de redes aleatorias

Aquí se describe el script en Python utilizado en la simulaciones 5.2.1 para el caso de nodos cercanos.

```
#!/usr/bin/python

import os
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.term import makeTerm
from random import randint
from array import array

#FUNCION QUE ENTREGA UNA POSICION RANDOM
def position_random():
    x1=randint(0,40)
    x2=randint(0,40)
    x3=0
    vec=[x1,x2,x3]
    vec=map(str, vec)
    vec= ",".join(vec)
    return vec

def topology():
    print "*** Creating network"
    net = Mininet(enable_wmediumd=True, enable_interference=True)

    print "*** Creating nodes"
    sta1 = net.addStation('sta1',range=17, position=position_random())
    sta2 = net.addStation('sta2',range=17, position=position_random())
    sta3 = net.addStation('sta3',range=17, position=position_random())
```

```
sta4 = net.addStation('sta4',range=17, position=position_random())
sta5 = net.addStation('sta5',range=17, position=position_random())
sta6 = net.addStation('sta6',range=17, position=position_random())
sta7 = net.addStation('sta7',range=17, position=position_random())
sta8 = net.addStation('sta8',range=17, position=position_random())
sta9 = net.addStation('sta9',range=17, position=position_random())
sta10 = net.addStation('sta10',range=17, position=position_random())
sta11 = net.addStation('sta11',range=17, position=position_random())
sta12 = net.addStation('sta12',range=17, position=position_random())
sta13 = net.addStation('sta13',range=17, position=position_random())
sta14 = net.addStation('sta14',range=17, position=position_random())
sta15 = net.addStation('sta15',range=17, position=position_random())
sta16 = net.addStation('sta16',range=17, position=position_random())

print "*** Configuring propagation model"
net.propagationModel(model="logDistance", exp=6)

print "*** Configuring wifi nodes"
net.configureWifiNodes()

print "*** Creating links"
net.addHoc(sta1, ssid='adhocNet')
net.addHoc(sta2, ssid='adhocNet')
net.addHoc(sta3, ssid='adhocNet')
net.addHoc(sta4, ssid='adhocNet')
net.addHoc(sta5, ssid='adhocNet')
net.addHoc(sta6, ssid='adhocNet')
net.addHoc(sta7, ssid='adhocNet')
net.addHoc(sta8, ssid='adhocNet')
net.addHoc(sta9, ssid='adhocNet')
net.addHoc(sta10, ssid='adhocNet')
net.addHoc(sta11, ssid='adhocNet')
net.addHoc(sta12, ssid='adhocNet')
net.addHoc(sta13, ssid='adhocNet')
net.addHoc(sta14, ssid='adhocNet')
net.addHoc(sta15, ssid='adhocNet')
net.addHoc(sta16, ssid='adhocNet')

print "*** Graficando"
net.plotGraph(max_x=45, max_y=45)

print "*** Starting network"
net.build()

print "*** Run ANTop in nodes"
makeTerm(sta1, cmd="bash --init-file ./script1.sh")
makeTerm(sta2, cmd="bash --init-file ./script2.sh")
makeTerm(sta3, cmd="bash --init-file ./script3.sh")
makeTerm(sta4, cmd="bash --init-file ./script4.sh")
makeTerm(sta5, cmd="bash --init-file ./script5.sh")
makeTerm(sta6, cmd="bash --init-file ./script6.sh")
makeTerm(sta7, cmd="bash --init-file ./script7.sh")
makeTerm(sta8, cmd="bash --init-file ./script8.sh")
makeTerm(sta9, cmd="bash --init-file ./script9.sh")
makeTerm(sta10, cmd="bash --init-file ./script10.sh")
makeTerm(sta11, cmd="bash --init-file ./script11.sh")
makeTerm(sta12, cmd="bash --init-file ./script12.sh")
makeTerm(sta13, cmd="bash --init-file ./script13.sh")
makeTerm(sta14, cmd="bash --init-file ./script14.sh")
makeTerm(sta15, cmd="bash --init-file ./script15.sh")
makeTerm(sta16, cmd="bash --init-file ./script16.sh")

print "*** Running CLI"
CLI(net)

print "*** Stopping network"
net.stop()
os.system("mn -c")
```

```
    os.system("sudo killall xterm")

if __name__ == '__main__':
    setLogLevel('info')
    topology()
```

A.2. Script de simulación de fragmentación topología 1

Aquí se describe el script utilizado en la simulación de la topología 1 de fragmentación 5.2.2.

```
#!/usr/bin/python

import os
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.term import makeTerm

 def topology():
    print "*** Creating network"
    net = Mininet(enable_wmediumd=True, enable_interference=True)

    print "*** Creating nodes"
    sta1 = net.addStation('sta1',range=17, position='60,60,0')
    sta2 = net.addStation('sta2',range=17, position='41,81,0')
    sta3 = net.addStation('sta3',range=17, position='52,110,0')
    sta4 = net.addStation('sta4',range=17, position='55,140,0')
    sta5 = net.addStation('sta5',range=17, position='30,130,0')
    sta6 = net.addStation('sta6',range=17, position='23,65,0')
    sta7 = net.addStation('sta7',range=17, position='33,112,0')
    sta8 = net.addStation('sta8',range=17, position='80,113,0')
    sta9 = net.addStation('sta9',range=17, position='66,40,0')
    sta10 = net.addStation('sta10',range=17, position='62,87,0')
    sta11 = net.addStation('sta11',range=17, position='86,140,0')
    sta12 = net.addStation('sta12',range=17, position='110,148,0')
    sta13 = net.addStation('sta13',range=17, position='104,92,0')
    sta14 = net.addStation('sta14',range=17, position='100,120,0')
    sta15 = net.addStation('sta15',range=17, position='120,110,0')
    sta16 = net.addStation('sta16',range=17, position='40,53,0')

    print "*** Configuring propagation model"
    net.propagationModel('logDistancePropagationLossModel', exp=4.5)

    print "*** Configuring wifi nodes"
    net.configureWifiNodes()

    print "*** Creating links"
    net.addHoc(sta1, ssid='adhocNet')
    net.addHoc(sta2, ssid='adhocNet')
    net.addHoc(sta3, ssid='adhocNet')
    net.addHoc(sta4, ssid='adhocNet')
    net.addHoc(sta5, ssid='adhocNet')
    net.addHoc(sta6, ssid='adhocNet')
    net.addHoc(sta7, ssid='adhocNet')
    net.addHoc(sta8, ssid='adhocNet')
    net.addHoc(sta9, ssid='adhocNet')
    net.addHoc(sta10, ssid='adhocNet')
    net.addHoc(sta11, ssid='adhocNet')
```

```
    net.addHoc(sta12, ssid='adhocNet')
    net.addHoc(sta13, ssid='adhocNet')
    net.addHoc(sta14, ssid='adhocNet')
    net.addHoc(sta15, ssid='adhocNet')
    net.addHoc(sta16, ssid='adhocNet')

    print "*** Graficando"
    net.plotGraph(max_x=200, max_y=200)

    print "*** Starting network"
    net.build()

    print "*** Starting Mobility"
    net.startMobility(time=0)
    net.mobility(sta8,'start', time=0, position'80,113,0')
    net.mobility(sta8,'stop', time=500, position='110,113,0')
    net.mobility(sta11,'start', time=0, position='86,140,0')
    net.mobility(sta11,'stop', time=500, position='116,140,0')
    net.mobility(sta12,'start', time=0, position='110,148,0')
    net.mobility(sta12,'stop', time=500, position='140,148,0')
    net.mobility(sta13,'start', time=0, position='104,92,0')
    net.mobility(sta13,'stop', time=500, position='134,92,0')
    net.mobility(sta14,'start', time=1, position='100,120,0')
    net.mobility(sta14,'stop', time=500, position='130,120,0')
    net.mobility(sta15,'start', time=1, position='120,110,0')
    net.mobility(sta15,'stop', time=500, position='150,110,0')
    net.stopMobility(time=500)

    print "*** Run ANTop in nodes"
    makeTerm(sta1, cmd="bash --init-file ./script1.sh")
    makeTerm(sta2, cmd="bash --init-file ./script2.sh")
    makeTerm(sta3, cmd="bash --init-file ./script3.sh")
    makeTerm(sta4, cmd="bash --init-file ./script4.sh")
    makeTerm(sta5, cmd="bash --init-file ./script5.sh")
    makeTerm(sta6, cmd="bash --init-file ./script6.sh")
    makeTerm(sta7, cmd="bash --init-file ./script7.sh")
    makeTerm(sta8, cmd="bash --init-file ./script8.sh")
    makeTerm(sta9, cmd="bash --init-file ./script9.sh")
    makeTerm(sta10, cmd="bash --init-file ./script10.sh")
    makeTerm(sta11, cmd="bash --init-file ./script11.sh")
    makeTerm(sta12, cmd="bash --init-file ./script12.sh")
    makeTerm(sta13, cmd="bash --init-file ./script13.sh")
    makeTerm(sta14, cmd="bash --init-file ./script14.sh")
    makeTerm(sta15, cmd="bash --init-file ./script15.sh")
    makeTerm(sta16, cmd="bash --init-file ./script16.sh")

    print "*** Running CLI"
    CLI(net)

    print "*** Stopping network"
    net.stop()
    os.system("mn -c")
    os.system("sudo killall xterm")

if __name__ == '__main__':
    setLogLevel('info')
    topology()
```

A.3. Script de simulación de mezcla topología 1

Aquí se describe el script utilizado en la simulación de la topología 1 de mezcla 5.2.3.

```
#!/usr/bin/python

import os
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.term import makeTerm

 def topology():
    print "*** Creating network"
    net = Mininet(enable_wmediumd=True, enable_interference=True)

    print "*** Creating nodes"
    sta1 = net.addStation('sta1',range=17, position='56,86,0')
    sta2 = net.addStation('sta2',range=17, position='165,136,0')
    sta3 = net.addStation('sta3',range=17, position='52,115,0')
    sta4 = net.addStation('sta4',range=17, position='142,120,0')
    sta5 = net.addStation('sta5',range=17, position='53,55,0')
    sta6 = net.addStation('sta6',range=17, position='145,100,0')
    sta7 = net.addStation('sta7',range=17, position='40,80,0')
    sta8 = net.addStation('sta8',range=17, position='160,96,0')
    sta9 = net.addStation('sta9',range=17, position='43,110,0')
    sta10 = net.addStation('sta10',range=17, position='75,50,0')
    sta11 = net.addStation('sta11',range=17, position='75,75,0')
    sta12 = net.addStation('sta12',range=17, position='158,85,0')
    sta13 = net.addStation('sta13',range=17, position='78,110,0')
    sta14 = net.addStation('sta14',range=17, position='155,156,0')
    sta15 = net.addStation('sta15',range=17, position='91,90,0')
    sta16 = net.addStation('sta16',range=17, position='140,136,0')

    print "*** Configuring propagation model"
    net.propagationModel('logDistancePropagationLossModel', exp=4.5)

    print "*** Configuring wifi nodes"
    net.configureWifiNodes()

    print "*** Creating links"
    net.addHoc(sta1, ssid='adhocNet')
    net.addHoc(sta2, ssid='adhocNet')
    net.addHoc(sta3, ssid='adhocNet')
    net.addHoc(sta4, ssid='adhocNet')
    net.addHoc(sta5, ssid='adhocNet')
    net.addHoc(sta6, ssid='adhocNet')
    net.addHoc(sta7, ssid='adhocNet')
    net.addHoc(sta8, ssid='adhocNet')
    net.addHoc(sta9, ssid='adhocNet')
    net.addHoc(sta10, ssid='adhocNet')
    net.addHoc(sta11, ssid='adhocNet')
    net.addHoc(sta12, ssid='adhocNet')
    net.addHoc(sta13, ssid='adhocNet')
    net.addHoc(sta14, ssid='adhocNet')
    net.addHoc(sta15, ssid='adhocNet')
    net.addHoc(sta16, ssid='adhocNet')

    print "*** Graficando"
    net.plotGraph(max_x=200, max_y=200)

    print "*** Starting network"
    net.build()
```

```
    print "*** Starting Mobility"
    net.startMobility(time=0)
    net.mobility(sta2,'start', time=0, position'165,136,0')
    net.mobility(sta2,'stop', time=500, position='125,136,0')
    net.mobility(sta4,'start', time=0, position='142,120,0')
    net.mobility(sta4,'stop', time=500, position='102,120,0')
    net.mobility(sta6,'start', time=0, position='145,100,0')
    net.mobility(sta6,'stop', time=500, position='105,100,0')
    net.mobility(sta8,'start', time=0, position='160,96,0')
    net.mobility(sta8,'stop', time=500, position='120,96,0')
    net.mobility(sta12,'start', time=1, position='158,85,0')
    net.mobility(sta12,'stop', time=500, position='118,85,0')
    net.mobility(sta14,'start', time=1, position='155,156,0')
    net.mobility(sta14,'stop', time=500, position='115,156,0')
    net.mobility(sta16,'start', time=1, position='140,136,0')
    net.mobility(sta16,'stop', time=500, position='100,136,0')
    net.stopMobility(time=500)

    print "*** Run ANTop in nodes"
    makeTerm(sta1, cmd="bash --init-file ./script1.sh")
    makeTerm(sta2, cmd="bash --init-file ./script2.sh")
    makeTerm(sta3, cmd="bash --init-file ./script3.sh")
    makeTerm(sta4, cmd="bash --init-file ./script4.sh")
    makeTerm(sta5, cmd="bash --init-file ./script5.sh")
    makeTerm(sta6, cmd="bash --init-file ./script6.sh")
    makeTerm(sta7, cmd="bash --init-file ./script7.sh")
    makeTerm(sta8, cmd="bash --init-file ./script8.sh")
    makeTerm(sta9, cmd="bash --init-file ./script9.sh")
    makeTerm(sta10, cmd="bash --init-file ./script10.sh")
    makeTerm(sta11, cmd="bash --init-file ./script11.sh")
    makeTerm(sta12, cmd="bash --init-file ./script12.sh")
    makeTerm(sta13, cmd="bash --init-file ./script13.sh")
    makeTerm(sta14, cmd="bash --init-file ./script14.sh")
    makeTerm(sta15, cmd="bash --init-file ./script15.sh")
    makeTerm(sta16, cmd="bash --init-file ./script16.sh")

    print "*** Running CLI"
    CLI(net)

    print "*** Stopping network"
    net.stop()
    os.system("mn -c")
    os.system("sudo killall xterm")

if __name__ == '__main__':
    setLogLevel('info')
    topology()
```

Bibliografía

[1] J. I. Alvarez-Hamelin, A.C. Viana, and M.D. de Amorín, DHT-based functionalities using hypercubes, in *Ad-hoc Networking*, vol. 212/2006, pp. 157-176, IFIP International Federation for Information Processing, Springer Boston, ISSN 1571-5736 (Print) 1861-2288 (Online).

[2] A. Marcu, *Desarrollo y simulación de un protoclo para redes ad-hoc*. Universidad de Buenos Ares, Facultad de Ingeniería, July 2007. http://cnet.fi.uba.ar/alejandro.marcu/Tesis_Alejandro_Marcu.pdf.

[3] G. Tejia, *Estudio y análisis de mecanismos orientados al robustecimiento de ANTop, utilizando ruteo proactivo.* Universidad de Buenos Ares, Facultad de Ingeniería, Nov. 2009. http://cnet.fi.uba.ar/alejandro.marcu/Tesis_Gaston_Tejia.pdf.

[4] M. Campolo, *Estudio y análisis del funcionamiento de ANTop sobre IPv6*. Universidad de Buenos Ares, Facultad de Ingeniería, April 2012. http://cnet.fi.uba.ar/alejandro.marcu/Tesis_Matias_Campolo.pdf.

[5] Better Approach To Mobile Ad-hoc Networking (B.A.T.M.A.N) draft-wunderlich-openmesh-manet-routing-00, October 9, 2008.

[6] RFC 6126: The Babel Routing Protocol. University of Paris, ISSN: 2070-1721, April 2011.

[7] RFC 3561: Ad hoc On-Demand Distance Vector (AODV) Routing. University of Cincinnati, July 2003.

[8] Charles E. Perkins, P. Bhagwat, Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers, 1994.

[9] RFC 7868: Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP). May 2016.

[10] C. Thomas, C. Leiberson and R. Rivest, Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001.

[11] J. Salim, H. Khosravi, A. Kleen, and A. Kuznetsov, RFC 3549: Linux Netlink as an IP Services Protocol,"July 2003.

[12] J. Corbet, A. Rubini and G. Kroah-Hartman, "Linux Device Drivers", third edition.

[13] IEEE standard for information tehcnology telecommunications and information exchange between system local and metropolitan are networks specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications,*IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pp. 1 2193, 29 2012.

[14] Iso/iec standard for information technology- telecommunications and information exchange between system- local and metropolitan area networks- specific requeriments- part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 4: Further higher data rate extension in the 2.4 ghz band, *ISO/IEC 802-11:2005/Amd.4:2006(E) IEEE Std 802.11g-2003 (Amendment to IEEE Std 802.11-1999)*, pp.cl 68, 2006.

[15] S. Deering and R. Heiden, RFC 4291: IP Version 6 Addressing Architecture, Feb. 2006.

[16] G. Huston, A. Lord and P. Smith, RFC: IPv6 Address Prefix Reserved for Documentation, July 2004.

[17] Mininet: http://mininet.org/

[18] R. Fontes and C. Rothenberg, The User Manual Mininet Wifi, April 2017.

[19] ANTop (Adjacent Networks Topologies) ad-hoc routing. https://github.com/CoNexDat/ANTop.