Implementación y Análisis de CBTv2 en el medioambiente Ns

Guillermo Rigotti

Maestría en Redes de Datos
Facultad de Ciencias Exactas
Universidad Nacional de La Plata

Director: Dr. Jean-Pierre Deschamps
Codirector: Lic. Javier Díaz

Resumen

Desde la definición del modelo de servicio multicast en IP, el uso de estas facilidades ha ido adquiriendo cada vez mayor importancia en el ámbito de la Internet. El número de aplicaciones que demandan transmisión multicast, tales como conferencias de audio y video, simulación interactiva distribuida, descubrimiento de recursos, etc, ha aumentado en forma continua, viéndose reflejado en el crecimiento experimentado por el MBONE, experimental en sus inicios en 1992 y actualmente considerado indispensable para decenas de miles de usuarios.

Los protocolos de ruteo multicast son los encargados de construir y mantener los árboles de distribución por los que fluye la información multicast.

Estos protocolos deben ofrecer una performance adecuada a los requerimientos de una diversidad de aplicaciones (demora máxima acotada, diferencia entre demoras acotada, etc.), y a su vez realizar un uso eficiente de los recursos de la red. Es además de suma importancia su escalabilidad, ya sea en cuanto a la cantidad de grupos, número de emisores por grupo, dispersidad de los grupos y extensión de la red sobre la cual operan.

Mientras que los primeros protocolos fueron concebidos para grupos densos y áreas reducidas (DVMRP, MOSPF), basándose en el paradigma source-group trees y en mecanismos de integración a los grupos por defecto y podas explícitas, otros como SM-PIM y fundamentalmente CBT versión 2, han sido diseñados para lograr escalabilidad, basándose en el paradigma shared trees para la construcción del árbol de distribución y en la solicitud explícita de integración a un grupo por parte de los routers interesados.

Para poder estudiar las características de estos protocolos, es necesario recurrir a la simulación, debido a que en algunas ocasiones un modelo analítico resulta en demasiadas simplificaciones que lo alejan de la realidad. Por otra parte, en la mayoría de los casos es imposible disponer de redes reales de las dimensiones necesarias para llevar a cabo la experimentación.

En este trabajo se presenta una implementación del protocolo CBT versión 2 en el medioambiente de simulación provisto por Ns, y un análisis de los costos de recuperación (PDUs transmitidas) y demoras que insume la reconstrucción de subárboles de distribución que quedan aislados como consecuencia de fallas en los vínculos de transmisión.

Indice

Lista de figuras

Lista de Tablas

Capítulo I

Introducción

Si bien en la actualidad la mayoría de las aplicaciones en la Internet requieren un soporte de transmisión punto a punto, en los últimos años se ha notado un incremento considerable en el número de nuevas aplicaciones que demandan un soporte de transmisión multipunto a multipunto[1].

Aunque los servicios de transmisión requeridos por estos tipos de aplicaciones podrían ser satisfechos con un soporte de transmisión unicast o broadcast, el consumo de recursos en la red sería desmedido e imposible de sostener.

En el primero de los casos, sería necesario que cada emisor almacenara las direcciones unicast de cada uno de los demás miembros del grupo produciéndose a su vez una gran cantidad de transmisiones, una a cada miembro del grupo.

En el caso de contar con facilidades broadcast, la sobrecarga generada se produciría como consecuencia de que los paquetes serían diseminados por toda la red (la Internet) en lugar de distribuirlos selectivamente a las subredes con miembros del grupo. Esto produciría carga innecesaria en los vínculos y proceso adicional en los routers y hosts para reaccionar ante la llegada de cada paquete, que, en la mayoría de los casos, sería luego descartado.

Es posible distinguir dos ámbitos en los cuales se diferencian las funciones necesarias para soporte de transmisión multicast: el ámbito restringido a una red local (LAN) en el cual se distingue un router y un conjunto de hosts locales, y el ámbito que se extiende más allá de la red local, involucrando comunicación entre routers.

El primero, al que tradicionalmente se ha visto limitada la transmisión multicast se describe en [1], donde se define IGMP versión 1, y en el que se especifica la funcionalidad a este nivel, consistente en el mantenimiento de información por parte del router respecto de membresía a grupos de los hosts locales.

Una versión posterior de IGMP, IGMPv2[2], agrega mecanismos a nivel local para la elección del "querier" router en una LAN (en la versión 1 estaba determinado por el protocolo de ruteo) y nuevas PDUs (Group-Specific-Query y Leave-Group); la primera permite a un router la transmisión de un query para un grupo específico en lugar de hacerlo para todos los grupos, mientras que la segunda permite a un host enviar un a indicación específica de abandono de un grupo, lo cual reduce la demora producida por IGMP para indicar al protocolo de ruteo que no desea recibir más información para un determinado grupo.[2]

La última versión del protocolo, IGMPv3[3], en estado aún de borrador (Internet Draft), agrega funcionalidad que se ve reflejada en el tipo de servicio ofrecido y en el consumo de recursos en la red[3].

La funcionalidad necesaria para la comunicación entre routers, se encuentra definida en los protocolos de ruteo multicast, a través de los cuales intercambian información los routers en base a los requerimientos de sus hosts locales, realizados vía IGMP.

---

[1] Esta modalidad de comunicación (multipunto a multipunto) engloba al caso punto a multipunto (un única fuente o emisor en el grupo multicast y varios miembros o receptores).

[2] En el caso en que dicho host sea el último miembro del grupo.

[3] A través de PDUs específicas, un host puede solicitar recibir información de sólo algunos de los emisores de un grupo determinado, o excluír emisores específicos del grupo; esta característica permite ofrecer un servicio más eficiente y a su vez lograr un mejor uso de los recursos de la red (ancho de banda) a través de la construcción de árboles de distribución más eficientes.

Los protocolos de ruteo multicast tienen la función de construir y mantener árboles de distribución para los distintos grupos[4]. Dichos árboles están constituidos por información almacenada en las tablas de ruteo multicast de los nodos involucrados, y son utilizados para la diseminación en la red de los paquetes multicast.

Se puede distinguir dos aspectos de importancia en este tipo de protocolos: 1-la manera en que son satisfechos los requerimientos de las diversas aplicaciones y 2-los recursos de red consumidos.

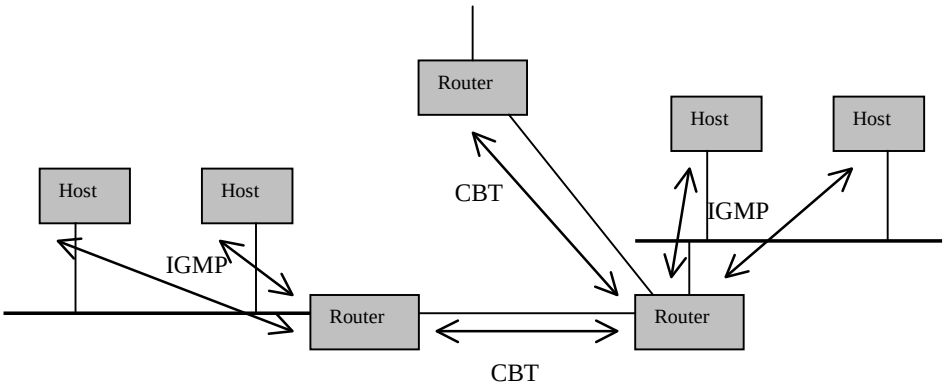

Fig 1.1. Ambitos de IGMP y de protocolos de ruteo multicast *(en este caso ejemplificados por CBT)*

En cuanto al primero de ellos, puede citarse por ejemplo requerimientos de aplicaciones tales como simulación interactiva distribuida, que demanda rapidez en el proceso de integrarción a un grupo (join), conferencias de audio, que requieren demoras máximas acotadas, etc.

El segundo aspecto se refiere por un lado a los recursos que se consumen como consecuencia de la actividad de control (tamaño de las tablas de ruteo y volumen de la información de control intercambiada entre los routers) y por otro a la cantidad de tráfico inyectado en la red como consecuencia de la distribución de los datos (paquetes multicast) generados por las aplicaciones.

Los dos aspectos mencionados se ven influenciados por las características que presentan los grupos multicast, que están determinadas por el tipo de aplicación y la distribución de los usuarios en la red. Es posible encontrar grupos con miembros conocidos y estáticos o grupos en los que los miembros cambian dinámicamente; grupos densos, en los cuales la mayoría de los nodos conforman el grupo, o grupos dispersos (sparse groups), conformados por una pequeña cantidad de nodos respecto de la totalidad de los que integran la red; grupos en los cuales uno o muy pocos de los participantes son emisores, o grupos en los que gran cantidad (o la totalidad) de los participantes son emisores.

Un característica importante a considerar es el grado de escalabilidad que posee un protocolo de ruteo multicast, es decir, como afecta a su comportamiento el tamaño de la red sobre la cual opera y el número de routers involucrados.

Un grupo de protocolos multicast, entre los que se encuentran DVMRP[4] y DM-PIM[5], han sido concebidos para grupos densos en áreas reducidas con uno o pocos emisores por grupo.

[4] De acuerdo al paradigma source-based trees, los árboles de distribución se construyen por par <emisor, grupo>.

De acuerdo a estas características, se basan en el paradigma source-based trees para la construcción de los árboles de distribución (un árbol de distribución por cada emisor del grupo) y en un esquema de solicitud de poda explícita del árbol de distribución por parte de aquellos routers que no estén interesados en pertenecer a un grupo.

Estas características afectan la escalabilidad del protocolo en dos aspectos. El primero de ellos se refiere a los recursos consumidos en los routers, ya que el tamaño de las tablas de ruteo aumenta con el orden del producto de la cantidad de grupos por la cantidad de emisores de cada grupo (O (|S| X |G|). Como contrapartida se ofrece a las aplicaciones un árbol de costo mínimo entre cada origen y cada destino, lo que en ciertos casos es de importancia, por ejemplo, aplicaciones que requieren demoras mínimas.

El segundo aspecto se refiere al uso ineficiente de los recursos de la red, ya que en caso de grupos dispersos que abarquen áreas considerables de la Internet, el esquema de poda explícita produce una inundación periódica de la red con paquetes multicast que no serán utilizados.

Otro grupo de protocolos ha sido concebido para adaptarse a grupos dispersos con un número considerable (o la totalidad) de los miembros del grupo actuando a su vez como emisores; entre ellos se encuentran CBT versión 2[6][7][8] y SM-PIM[9].

En particular, CBT versión 2, ha sido concebido en base al paradigma shared trees, que implica la creación de un único árbol de distribución por grupo, reduciendo de esta manera los recursos necesarios en los routers a un orden relacionado con la cantidad de grupos (O(|G|). Debe considerarse el posible efecto negativo de este paradigma, que consiste en el uso de árboles no óptimos para cada par <fuente,destino>, lo que resulta en mayores demoras. CBT utiliza además un mecanismo de join explícito que evita que la información multicast sea diseminada en la totalidad de la red, característica que permite un uso eficiente de los recursos de la red en el caso de grupos dispersos que cubren áreas considerables de la Internet.

A diferencia de SM-PIM, CBT versión 2 no provee la capacidad de conmutar del árbol compartido a árboles específicos para cada fuente, lo que lo hace más simple. Por otro lado, presenta la característica de considerar, a efectos de la distribución de los datos, al árbol compartido como bidireccional, lo que evita que los paquetes multicast deban ser enviados al core para luego ser inyectados en el árbol.

El trabajo que se presenta a continuación se encuentra organizado de la siguiente manera:

- **Perspectiva de los protocolos de ruteo multicast** : En este capítulo se describen las características y paradigmas en los que se basan los protocolos multicast en desarrollo y cómo responden a los requerimientos de las aplicaciones.
- **Medioambiente de simulación**: En este capítulo se describe el medioambiente de simulación sobre el que se realizó la implementación y el análisis de algunos aspectos del protocolo. Se describe someramente el funcionamiento del simulador, resaltando las características utilizadas en el presente trabajo.
- **Descripción de CBT versión 2**: En este capítulo se describe en particular la arquitectura y funcionamiento de CBT versión 2, dando una idea de su motivación, objetivos y operación.
- **Especificación del protocolo**: En este capítulo se especifica la operación de CBT versión 2 a partir de la descripción en lenguaje natural realizada en [7]. Dicha especificación se realiza de acuerdo a lo establecido por la ISO respecto a la definición de la operación de protocolos.
- **Implementación**: En este capítulo se descibe la estructura de la implementación, las clases definidas y su funcionalidad, su relación con el soporte provisto por el medioambiente de simulación y las facilidades provistas.
- **Análisis**: En este capítulo se realiza un análisis simple de algunas características de CBT versión 2 a través de simulaciones basadas en la implementación.

- **Conclusiones**: En el último capítulo se realizan conclusiones acerca del desarrollo realizado y se describen los próximos pasos a seguir en la línea del trabajo.

Capítulo 2

Perspectiva de los protocolos de ruteo multicast

Como se mencionó en el capítulo anterior, se puede clasficar a los protocolos de ruteo multicast en dos clases, basándose en su diseño orientado a la operación con grupos multicast densos o dispersos.

Los grupos densos pueden ser caracterizados como aquellos integrados por miembros que se encuentran localizados en una o más subredes en las cuales los recursos (ancho de banda de los vínculos, memoria en los routers) son abundantes. Es normal que en el área de influencia de este tipo de grupos multicast, un alto porcentaje de los routers esté interesado en recibir datagrams destinados al grupo debido a requerimientos de los hosts ubicados en alguna de sus subredes locales, realizados a través de IGMP.

Los grupos dispersos, en cambio, son aquellos en los cuales los miembros se hallan distribuidos en grandes áreas (por ejemplo, en toda la Internet). En estos casos, no es válido suponer abundancia de recursos, dado que algunos vínculos pueden ser de baja velocidad y los routers pueden tener capacidades variadas. Otro aspecto que debe ser considerado en estos casos, es la presencia de una gran cantidad de otros grupos multicast en el área de influencia, lo cual agrava la escasez de recursos. Debe notarse que el hecho de que el grupo sea disperso, no significa que tenga pocos miembros, sino que constituyen un pequeño porcentaje de los nodos que integran la red en la que opera el protocolo.

2.1 Relación entre las funciones de ruteo y reenvío

De igual manera que en ruteo unicast, en multicast debe distinguirse entre la función de ruteo y la función de reenvío.

En direccionamiento unicast, la función de ruteo es la encargada de contruir las tablas de rutas, y es llevada a cabo por los diferentes protocolos de ruteo (RIP, OSPF, etc).

En direccionamiento multicast, los protocolos de ruteo (CBT, PIM-DM, PIM-SM, etc) son los encargados de construir los árboles de distribución, en base a los cuales se realizará la distribución (reenvío) de los paquetes multicast.

El reenvío unicast se realiza en función de la dirección de destino del paquete. Dicha dirección se busca en las tablas de reenvío (derivadas de las tablas de ruteo) y se accede así a la interfaz a través de la cual debe enviarse dicho datagram.

El reenvío multicast no puede basarse sólo en la dirección de destino de la misma manera que en ruteo unicast, ya que por un lado una dirección multicast no hace referencia a una ubicación en particular, sino que es una dirección lógica la cual involucra varias ubicaciones físicas; además, la composición de un grupo (el mapeo de la dirección lógica en las diferentes ubicaciones) puede variar dinámicamente. Por otro lado, si se trata de árboles específicos por grupo, el reenvío se basa también en la dirección de origen.

Podría decirse que en ruteo unicast, la función de reenvío trata de enviar el paquete cada vez más cerca de su destino, mientras que el reenvío multicast, trata de enviar los paquetes de manera que se alejen del emisor, en base al árbol de distribución correspondiente.

De acuerdo al árbol de distribución, habrá una interfaz del router a través de la cual se espera recibir un paquete multicast emitido por un cierto emisor y dirigido a un determinado grupo. Esta interfaz se conoce como "parent" o "upstream". A diferencia del reenvío unicast, en el que el paquete se envía por una única interfaz hacia el siguiente router camino al destino, en el caso multicast el paquete debe distribuirse a todos los routers interesados en recibirlo (de acuerdo al árbol de distribución); las interfaces por las cuales el paquete debe ser enviado, se conocen como "downstream" o "child"[5].

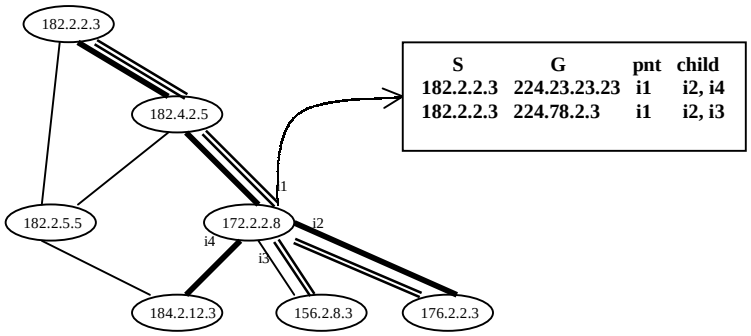

***Fig. 2.1. Arbol de distribución multicast**: interfaces hacia la raíz y hacia las hojas.*
Grupo 224.23.23.23 *Grupo 224.78.2.3*

2.2 Source based trees

De acuerdo con el paradigma source-based trees, se construye un árbol de distribución diferente por cada emisor del grupo, que tiene por raíz a este nodo. La difusión de la información de ruteo, se basa en asumir un grupo denso, por lo cual se propaga a todos los routers de la red, disparada por el envío de datos por parte del emisor (data driven). Esta inundación de la red con información de ruteo, es seguida por truncamientos y podas del árbol por parte de los routers no interesados, evitando así la difusión de información multicast a subredes que no participan del grupo.

Para la construcción de los árboles de distribución, se utiliza el mecanismo conocido como Reverse Path Multicasting (RPM), una mejora a Reverse Path Broadcasting, técnica para transmisión broadcast, y Truncated Reverse Path Broadcasting, una mejora a esta última que incluye truncamiento del árbol por parte de los routers hojas del árbol –leaf routers-. A continuación se describe brevemente el procedimiento.

Un router que recibe un paquete con un origen y destino multicast determinados, decide, en función de la información de ruteo disponible, si la interfaz de arribo del paquete constituye el mejor camino al nodo fuente (Reverse Path). En caso de no serlo, el paquete es descartado. En cambio, si dicho paquete ha sido recibido por el mejor camino al origen, es procesado generándose así información que determinará la construcción del árbol.

El proceso realizado consiste en el almacenamiento de información de ruteo si se trata del primer paquete recibido y en el reenvío del mismo. En su versión más simple, el paquete recibido es enviado por el resto de las interfaces del router (Reverse Path Broadcast). La difusión indiscriminada de paquetes a través de todas las interfaces del nodo (excepto la de recepción) puede evitarse si el router tiene información que le permite conocer cuáles de los routers adyacentes a los que se les enviará el paquete lo consideran el mejor camino respecto al

[5] La descripción es general, ya que cada protocolo en particular opera de una manera específica.

nodo origen del paquete[6]; en caso de que no lo consideren de esta manera, el router no envía el paquete por la correspondiente interfaz, dado que será descartado por el receptor.

Como mejora al método anterior, se incorpora la capacidad de los routers hoja (leaf) para truncar el árbol de distribución (Truncated Reverse Path Broadcasting). El truncamiento consiste en que un router que no tiene hosts miembros del grupo (información conocida a través de IGMP), no difunde los datagrams multicast en la subred.

Por último, la mejora introducida por Reverse Path Multicasting[7], consiste en la poda del árbol de distribución, de manera que éste cubra sólo las subredes con hosts miembros del grupo, y aquellas subredes de tránsito que se encuentran en el camino más corto entre el emisor y las primeras.

La poda del árbol consiste en que un router que no tiene miembros para la entrada multicast envíe a su parent un mensaje explícito indicando que no desea recibir dicha información. Esta procedimiento se aplica recursivamente desde los routers hoja hacia la raíz del árbol.

Una vez construído el árbol de distribución, representado por las entradas en cada router involucrado, es necesario definir procedimientos que permitan su actualización periódica, debido a posibles cambios en la topología de la red y la composición (miembros) del grupo.

Para lograr la actualización dinámica del árbol, se da un tiempo de vida limitado a las entradas en las tablas de ruteo: a intervalos regulares, la información respecto de interfaces podadas expira, haciendo que el router realice nuevamente el flooding al recibir el siguiente datagram para el par <fuente, destino>. Este procedimiento produce, por un lado, que aquella información de podas mantenida en el router sea eliminada, lo que es necesario para no mantener información de árboles no existentes (en el caso de que el emisor haya dejado de enviar), mientras que, por otro lado, se posibilita que si en alguna subred podada ha aparecido un nuevo miembro del grupo, pueda extenderse el árbol, ya que el primer datagram multicast después de la eliminación de la información de poda se difundirá a través de las demás interfaces del router.

Debe tenerse en cuenta que estas características implican que en todos los routers se almacene información respecto a los árboles de distribución existentes (estén o no podados) y a su vez se produzca un flooding periódico de paquetes multicast (posiblemente no deseados) en toda la red.

2.2.1 DVMRP (Distance Vector Multicast Routing Protocol)

Fue el primer protocolo multicast, definido en [4] y derivado de RIP (Routing Information Protocol). Desde su primera especificación ha ido evolucionando, y en la actualidad su versión 3 se encuentra en estado de borrador [10]. Actualmente es el protocolo más difundido en el MBONE.

Construye un árbol de distribución de costo mínimo (saltos) por emisor y depende de un ruteo propio. Utiliza Reverse Path Multicasting, y agrega características tales como transmisión confiable de mensajes de poda del árbol y provisión de un mecanismo rápido para recomponer (graft) el árbol podado en caso de la aparición de nuevos miembros del grupo.

Es un protocolo adaptado para funcionar en áreas restringidas con grupos densos, debido a la carga producida como consecuencia de su esquema de membresía por defecto y podas explícitas.

2.2.2 PIM-DM (Protocol Independent Multicast, Dense Mode)

[6] El router puede disponer de esta información de diferentes maneras, por ejemplo a través del conocimiento de la topología de la red si el ruteo difunde información topológica (link state) o a través de información enviada específicamente por los routers adyacentes

[7] Recién aquí es se cambia de un envío broadcast a un envío multicast (envío selectivo a partes de la red).

Bajo el nombre PIM, encontramos dos tipos de protocolos (DM y SM, diseñados para grupos densos y dispersos respectivamente). Pese a la similitud de nombres, estos protocolos no son compatibles en el sentido de que no pueden operar en la misma área. Su similitud se reduce a las PDUs intercambiadas entre las entidades que conforman el protocolo. Como su nombre lo indica, PIM-DM es un protocolo independiente del ruteo unicast, lo que lo hace más simple. Es similar a DVMRP, ya que utiliza RPM y construye árboles de distribución por emisor. A raíz de una decisión de diseño, en la que se sacrifica eficiencia en función de obtener mayor simplicidad, un router no selecciona interfaces para el reenvío de los datos, sino que indiscriminadamente los disemina sin tener en cuenta si el router downstream lo considera el mejor camino al emisor. Estas ramas innecesarias serán podadas posteriormente.

2.2.3 MOSPF (Multicast Open Shortest Path First)

MOSPF[11] es totalmente dependiente del protocolo unicast de ruteo, OSPF. Está concebido para su funcionamiento en un dominio, compuesto por una o más áreas OSPF. Basado en la información topológica provista por OSPF, e información propia respecto a la composición de grupos multicast propagada por el protocolo, cada router construye independientemente el árbol de costo mínimo con raiz en el emisor del grupo y teniendo como hojas los demás miembros. Debido a que cada router debe calcular el árbol de camino más corto para cada par <fuente, destino>, se decide hacerlo siguiendo un esquema data driven, al recibir el primero de los datagrams multicast, evitando de esta manera picos de carga[8].

El router almacena esta información para luego utilizarla en el reenvío de los demás paquetes. No existe eliminación por tiempo de las entradas sino que por razones de performance, son conservadas mientras el router tenga recursos.

Debe tenerse en cuenta que si existen varias áreas OSPF, en cada una de ellas se calcula el árbol de distribución con información propia (del área); esto implica que si el emisor se encuentra en otra área, forzosamente debe utilizarse RPF debido a que OSPF sólo anuncia costos en dirección reversa a través de los LSA y LSA externos.

A pesar de ser un protocolo modo denso, OSPF no se basa en un esquema de pertenencia implícito seguido por podas, ya que un router anuncia explícitamente su deseo de integrar un grupo; esta información, diseminada junto con la información topológica OSPF, hace que los demás routers lo incluyan en el correspondiente árbol de distribución.

2.3 Shared trees

En el paradigma shared trees, se construye un único árbol de distribución (común a todos los emisores) por cada grupo multicast. El nodo raíz de este árbol (Core en términos CBT, Rendevouz Point en términos SM-PIM) es un router predeterminado, no necesariamente miembro del grupo, cuya dirección unicast debe ser conocida por todos aquellos routers que deseen incorporarse al grupo. Concebido para grupos dispersos, no se produce difusión de información de ruteo salvo a aquellos nodos interesados, que lo indican explícitamente a través de un procedimiento especial. Este mecanismo consiste en el envío de una solicitud de integrar el grupo (join) por parte del router interesado, que es enviada[9], utilizando información de ruteo unicast, en dirección al core del grupo. El nodo core, o uno intermedio ya perteneciente al árbol de distribución, responden con un asentimiento. Este se propaga por el camino inverso al que han seguido las solicitudes, determinando así la nueva rama del árbol que tendrá como hoja al router que originó la solicitud.

[8] De todas maneras, se producirán picos de carga debido al recálculo de los árboles cuando la topología de la red se modifica.

[9] Este envío se realiza en función de la capacidad multicast o no de los vínculos, y será analizado en la descripción de CBT.

La característica de utilizar un árbol compartido entre los emisores de un grupo, con raíz en el core, plantea dos problemas adicionales: la elección de un árbol de costo mínimo y la difusión de la ubicación del (los) core(s) a los routers. El primer problema no es abordado en las especificaciones de este tipo de protocolos, mientras que el segundo está siendo tratado en el ámbito de PIM-SM.

El primero de los problemas ha sido tratado ampliamente. Se refiere a la construcción de un árbol de costo mínimo que abarque a todos los miembros del grupo (Steiner Tree). Sin embargo, este tipo de construcción no satisface los requerimientos del ruteo multicast, ya que trata el problema restringiéndolo a un único emisor y un conjunto fijo de miembros del grupo. Debido a la característica dinámica que presenta la composición de los grupos multicast, estos árboles deben ser contruídos por demanda, lo que implica un elevado costo. Se han propuesto varias heurísticas que tratan de construir árboles con una performance cercana al costo mínimo, pero en forma dinámica y distribuida, algunas de ellas integrando la fase de selección del core con la de la construcción del árbol.

Para la difusión de información acerca de la ubicación de cores a los routers, existen dos alternativas. Una de ellas es la configuración manual, que demanda cierta carga administrativa, mientras que la otra se basa en un procedimiento de arranque que deben seguir los routers del dominio. El mecanismo propuesto se basa en un mapeo algorítmico de los posibles cores (o Rendevouz points) de un dominio a los diferentes grupos multicast, siendo su objetivo la minimización de los cambios de cores, la transparencia del mecanismo respecto de los hosts, la minimización de parámetros tales como tiempos de convergencia y carga producida por la distribución de mensajes, y la obtención de un buen balanceo de carga.

2.3.1 PIM-SM

PIM-SM es un protocolo concebido para grupos dispersos sobre grandes áreas (p.ej. la Internet). Utiliza árboles compartidos (shared trees) y un mecanismo explícito de solicitud de pertenencia a un grupo por parte de los routers interesados.

Los routers que ofician de “punto de encuentro” se denominan Rendevouz Points (RP). Un RP no necesita ser configurado especialmente. Cuando un router desea pertenecer a un grupo, envía un requerimiento en dirección al RP correspondiente, que responderá como fue descripto antes. El árbol de distribución formado, con raíz en el RP, es unidireccional, y debido a esto un emisor de paquetes multicast debe encapsularlos y enviarlos al RP; éste los desencapsula e inyecta en el árbol de distribución.

En casos en que el tráfico generado por un emisor supere un cierto nivel, un router receptor puede conmutar del árbol compartido a un árbol específico <fuente, destino>, enviando un requerimiento especial al router emisor. Este comenzará a enviar los datos multicast sobre el nuevo árbol construido, enviando a su vez sobre el árbol compartido. Cuando el router receptor comienza a recibir la información por el árbol específico, envía mensajes al RP para podarse del árbol compartido. Esta conmutación permite reducir las demoras producidas entre el emisor y el receptor que solicitó el cambio. Una consecuencia adicional es disminuir la carga en el árbol de distribución compartido.

El RP puede decidir en ciertos casos que un emisor cambie su modo de envío: en lugar de encapsular los paquetes multicast y enviarlos a la dirección unicast del RP, los enviará sobre un árbol de costo mínimo especialmente contruido. Para ello el RP se vale de mensajes específicos hacia el emisor.

2.3.2 CBT

CBT es un protocolo concebido al igual que PIM-SM, para grupos dispersos sobre grandes áreas. Está basado en árboles compartidos y en mecanismos explícitos de solicitud de pertenencia a un grupo. Al diseñar CBT, se dió gran importancia a la escalabilidad del protocolo, teniendo en cuenta escenarios compuestos de miles de routers con capacidad multicast. Debido a esto, se sacrificaron algunas características que se encuentran en PIM-SM, como la opción de conmutar de un árbol compartido a un árbol específico. Otra característica que los diferencia, es que en CBT el árbol de distribución es considerado bidireccional, motivo por el cual un router conectado a él puede inyectar la información directamente en las interfaces correspondientes en lugar de tener que realizar la encapsulación unicast para enviar al core.
Actualmente CBT se encuentra especificado en su versión 2 como RFC[7] y en su versión 3 en estado borrador [12]. En esta última versión se incluyen algunas mejoras a la versión 2, y fundamentalmente se agregan características para posibilitar que un dominio CBT actúe eficientemente como dominio de tránsito (soporte para entradas de reenvío de diferentes granularidades, soporte de prefijos, etc.)

2.4 Resumen

En este capítulo se han presentado los elementos que caracterizan a los protocolos de ruteo multicast. Ellos son el tipo de árbol utilizado para la distribución de los paquetes multicast (árboles específicos o compartidos) y pertenencia a los grupos implícita o explícita. Para grupos densos, se adaptan los árboles específicos por emisor y la pertenencia implícita; para grupos dispersos, los árboles compartidos y la pertenencia explícita.

Las características más importantes se pueden resumir en los siguientes puntos:

- El paradigma source based trees produce menores demoras entre emisor y receptor que el paradigma shared trees, ya que construye para cada emisor el camino de menor costo.
- El paradigma source based trees produce una mejor distribución de la carga en los diferentes vínculos; en un árbol compartido la totalidad del tráfico generado por los emisores se envía por cada uno de los vínculos que componen el árbol.
- El paradigma shared trees ofrece una escalabilidad mayor que el paradigma source based trees, en una magnitud relacionada con la cantidad de emisores por grupo. Esta característica se refiere a la memoria necesaria en los routers para el almacenamiento de las tablas y a la carga producida en los vínculos debido al mantenimiento de una mayor cantidad de árboles de distribución.
- El esquema de pertenencia por defecto y podas explícitas produce una carga innecesaria en la red debido al flooding periódico y a la necesidad de mantener información respecto de interfaces podadas en todos los routers. Esta característica es inadmisible en redes de gran extensión.
- Respecto de los protocolos diseñados para redes de área extensa (en particular la Internet), PIM-SM ofrece las característcas de escalabilidad del paradigma shared trees, permitiendo en ciertos casos la conmutación a árboles específicos para satisfacer los requerimientos (demoras) de las aplicaciones. CBT, en cambio, ha sido diseñado considerando la escalabilidad y la simplicidad como el aspecto más crítico, a consecuencia de lo cual no ofrece la posibilidad de conmutación a árboles específicos por emisor.

Capítulo 3

Medioambiente de Simulación

El medioambiente utilizado para implementar la simulación fue Ns[13] – Network Simulator – versión 2.0, un simulador orientado a objetos basado en eventos discretos cuyo objetivo es la simulación de redes de comunicación. Ns está actualmente en desarrollo en el marco del proyecto VINT (Virtual InterNetwork Testbed, USC/ISI/, UC Berkeley y Xerox Parc).

Ns se encuentra implementado en C++ y en Otcl [14][15], extensión de Tcl[16] orientada a objetos. Soporta de esta manera dos jerarquías de clases relacionadas, una de ellas en C++ (jerarquía compilada) y otra en Otcl (jerarquía interpretada). Desde el punto de vista del usuario, existe una relación uno a uno entre una clase de la jerarquía interpretada y una de la compilada. Una característica importante es que puede ampliarse la funcionalidad de las clases compiladas creando dinámicamente nuevos métodos en Otcl.

La clase *Simulator* es la que provee al usuario con elementos para configurar una determinada simulación y seleccionar el tipo de planificador de eventos (scheduler) a utilizar. En la actualidad existen cuatro tipos de planificadores: “linked-list”, utilizado en el presente desarrollo, “heap”, “calendar queue” y “real-time”. Cada uno de los tres primeros tipos de scheduler utiliza una estructura de datos diferente para el manejo de los eventos, cada uno de ellos presentando mejor performance en determinados casos. El último, aún en estado experimental, provee la sincronización entre los eventos manejados por el simulador y el tiempo real. Su objetivo es posibilitar la emulación, es decir, la capacidad para introducir el simulador en una red real.

A continuación se describen brevemente las clases que son utilizadas para la configuración de las simulaciones. Esta descripción sólo abarca los aspectos más importantes desde el punto de vista de un usuario[10], no pretendiendo profundizar en otros detalles.

Las clases que permiten configurar una topología sobre la cual se producirá la simulación, son principalmente *node*, cuyas instancias son nodos con la funcionalidad de routers y *link*, cuyas instancias son vínculos de comunicación..

3.1 Nodos

Un objeto de la clase *node*, representa la funcionalidad de un router (capacidad de reenvío de paquetes). Un nodo está conectado por objetos de la clase *link* a otros nodos, pudiendo a su vez soportar procesos (protocolos tales como tcp, udp, procesos de usuarios, etc.). Estos procesos, son derivados de la clase *Agent*, que se describe más adelante.

La dinámica de un nodo es simple: recibe un paquete por uno de sus vínculos (*link*) o un paquete generado por alguno de los agentes que en él residen, y lo procesa de acuerdo a su dirección de destino[11], entregándolo al vínculo o al agente local que corresponda.[12]

[10] Por usuario debe entendrse el programador de un script de simulación en Otcl.

Si bien existen métodos de la clase *node* relativos a su configuración, no se describen aquí por razones de brevedad.

En la figura 3.1 se muestra la estructura de un nodo sin capacidad multicast[13]. Un paquete recibido por el nodo es procesado en orden por un conjunto determinado de objetos[14]. El primero de ellos está indicado por la variable de instancia *_entry* y en el caso unicast, corresponde a un objeto de la clase *AddressClassifier*, derivada de la clase *Classifier*[15]. El objeto *AddressClassifier* del nodo (indicado por el contenido de la variable *_classifier*) analiza su "dirección", es decir, la identificación del nodo de destino que lleva dicho paquete y en base a ella lo envía a un vínculo (si la dirección de destino no coincide con la del nodo) o al objeto encargado de determinar el agente local a quien va destinado el paquete, en función del campo port. Este objeto, el *Port Classifier* del nodo, derivado de la clase *Classiffier,* está indicado por la variable de instancia *_dmux*.

Un vínculo será el encargado de entregar el paquete al próximo nodo, mientras que el agente representa un proceso que tendrá a su cargo el tratamiento del contenido del paquete (por ejemplo, una aplicación).

Se ha descripto en los párrafos anteriores el recorrido de un paquete dentro del nodo. Debe notarse que las transferencias del paquete no implican la generación de eventos para el

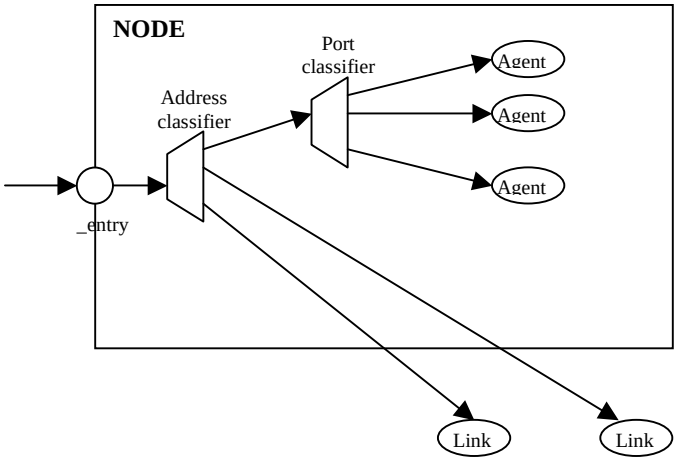

Figura 3.1. Estructura de un nodo sin capacidad multicast

planificador de eventos; Ns asume que los tiempos de proceso son nulos. Más adelante se verá la generación de eventos a cargo de un componente de los objetos *link*, que implica demoras.

---

[11] En Ns las direcciones se componen, al igual que en la jerarquía TCP/IP, de una dirección de nodo (equivalente a una dirección IP) y una identificación del agente (similar al port).

[12] En el caso que nos ocupa, multicast, el nodo podrá enviar el paquete por más de un link de salida, de acuerdo al árbol de distribución.

[13] Un nodo no posee por defecto capacidad multicast en Ns. Es posible configurar la totalidad de los nodos (no algunos) de la simulación con dicha capacidad, indicándolo explícitamente al Simulador.

[14] Este proceso es instantáneo desde el punto de vista de la simulación, mientras no se generen eventos.

[15] Un classifier, en general, es un objeto que analiza una cierta parte del paquete recibido, y en base a su contenido, envía el paquete a un objeto determinado, el siguiente en la secuencia que procesará dicho paquete.

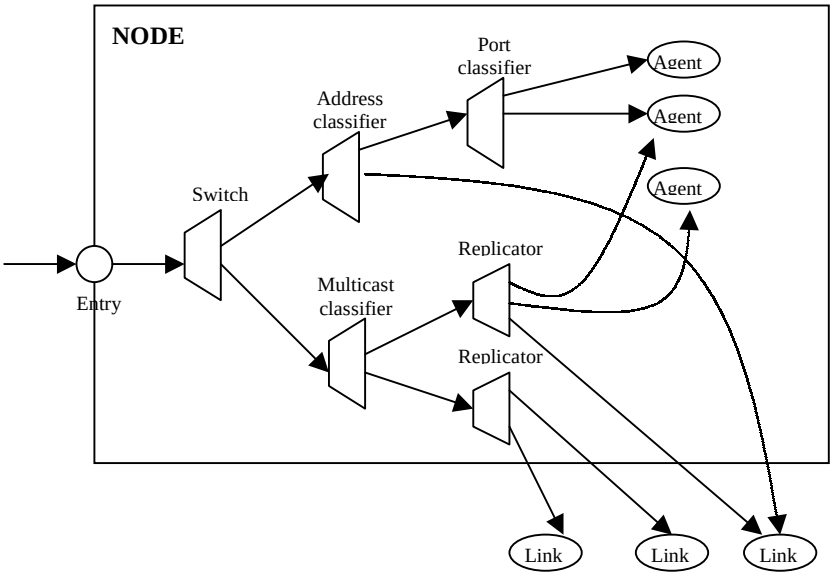

Figura 3.2. Estructura de un nodo con capacidad multicast

En el caso de nodos con capacidad multicast, cuya estructura se presenta en la figura 3.2, se agregan elementos adicionales.

El objeto que constituye la entrada al nodo, es en este caso un C*lassifier* que sólo distingue los paquetes en función del primer bit de su dirección de destino (*Switch*). En el caso de un paquete con dirección unicast (primer bit en 0) es pasado al objeto *AddressClassifier*, produciendo en el nodo un comportamiento como el descripto para el caso unicast. Si en cambio la dirección de destino del paquete es multicast (primer bit en 1), el *Switch* entrega el paquete a un C*lassiffier* (*Multiclassifier*) que clasifica los paquetes según su dirección de origen y de destino[16]. De acuerdo a estas direcciones, el paquete es entregado a un objeto *replicator*[17], cuya función es crear copias del paquete para distribuirlas entre los posibles receptores, ya sea vínculos para su reenvío o agentes locales miembros del grupo multicast.

3.2 Vínculos

Así como los nodos están compuestos de objetos *classifier*, que realizan una selección del paquete en base al contenido de algunos de sus campos, los vínculos están compuestos de objetos derivados de la clase *connector*. Básicamente, un objeto de esta clase recibe un paquete, realiza una alguna función sobre el mismo, y luego lo entrega al próximo objeto en la secuencia (*target_*) o lo descarta (*drop_target_*).

[16] La clasificación del paquete por dirección origen y destino corresponde a un paradigma source based trees. Como se explica en el capítulo correspondiente a la implementación, esta característica obligó a tener en cuenta consideraciones adicionales para el caso que nos ocupa, shared trees. Además de la clasificación por dirección y destino, el soporte provisto por el simulador provee un chequeo de interfaz de entrada (upstream) que se detalla en el capítulo correspondiente a la implementación.

[17] Un *replicator*, si bien se deriva de la clase *classifier*, no realiza una clasificación del paquete, sino que sólo lo replica. Se encuentra implementado de esta manera sólo por la conveniencia de utilizar las tablas de slots de la clase *classifier*.

De acuerdo a las características de un vínculo, éste estará compuesto de varios conectores. Las variables de instancia del objeto, contienen referencias a su entrada (*head_*), al elemento principal de su cola de paquetes (*queue_*), al elemento que realmente modela las características de la transmisión: demora de propagación y velocidad de transmisión (*link_*), al elemento que maneja el time to live (ttl) en cada paquete, y al primer elemento de la cola de elementos que procesan el descarte de paquetes (*drophead_*).

Los conectores que pueden conformar un link, son los siguientes:

- *Networkinterface*: su función es rotular los paquetes con una identificación de interfaz de arribo. Esta característica es utilizada por los protocolos de ruteo multicast. Debe habilitarse dando valor uno a la variable de clase *NumberInterfaces_* del objeto Simulator.
- *DelayLink*: Tiene por objeto producir las demoras propias del vínculo de transmisión, dependiendo de la demora de propagación de la señal, la velocidad de transmisión y la longitud de los paquetes. Normalmente, recibe los paquetes y genera un evento que el planificador de eventos entregará, en el tiempo adecuado, al siguiente conector. En casos de vínculos dinámicos, forma una cola interna con los paquetes previendo casos de caída del vínculo, en los cuales debe descartarse los paquetes en la cola.
- *DynamicLink*: Es un conector cuyo objeto es permitir o no el paso de los paquetes según el estado del vínculo (funcionando o caído). Para configurar el estado de un vínculo, debe utilizarse la capacidad de caída dinámica de vínculos provista por el

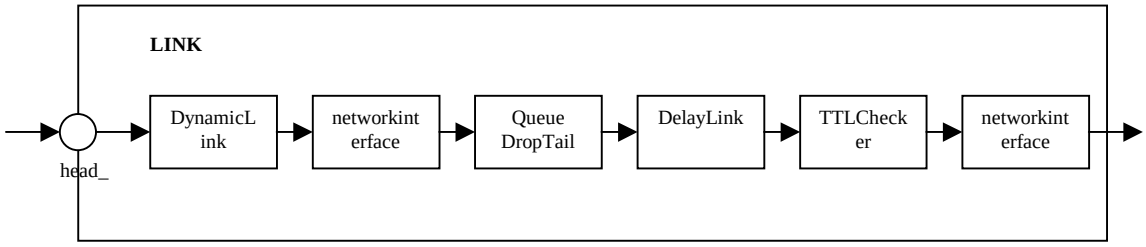

Fig. 3.3. Componentes de un link dinámico sin capacidad de tracing.

simulador.

- *Queues*: Son conectores utilizados para modelar las colas asociadas a los vínculos en los routers. Existen varios tipos, cada uno de ellos con una política de descarte de paquetes diferente.
- *TTLChecker*: Es un conector que decrementa el campo ttl de cada paquete, descartándolo en caso de que llegue a cero.
- *Trace*: Se proveen conectores destinados a realizar el seguimiento (trace) de los paquetes. Estos objetos se insertan antes y después de la cola (*Queue*) que representa los buffers del router para poder realizar el seguimiento de los paquetes que llegan y son enviados o descartados.

En la figura 3.3 se muestra la estructura de un vínculo dinámico que no incluye los objetos para realizar tracing. El primer conector, *Dynamiclink*, determina si el vínculo está o no operativo, descartando paquetes en este último caso. Los objetos *networkinterface* se ocupan de colocar en el encabezamiento del paquete, la identificación de la interfaz. El objeto *Queue*

DropTail, representa los buffers del router, implementándose aquí la política de descarte de paquetes. *DelayLink* produce las demoras de transmisión e interactúa con el planificador de eventos, registrando el evento correspondiente a la llegada del paquete a destino y a su vez el que corresponde a la liberación del buffer que ocupaba en el router de origen (una demora de propagación antes). Por último, *TTLChecker* controla el campo ttl del paquete para descartarlo en caso de que llegue a cero.

Las características vistas corresponden a la clase *Link*; de ella se deriva la clase *simplex-link*, que une dos nodos a través de un vínculo unidireccional, y la clase *duplex-link*, que determina la unión de dos nodos a través de un vínculo bidireccional.

3.3 Agentes

Finalizada la descripción de los elementos básicos que permiten definir la topología, nodos y vínculos, resta describir la clase base (*Agent*) de la que se derivan los elementos activos en la simulación: los procesos residentes en los nodos. Estos pueden ser procesos de soporte, tales como protocolos, o simplemente aplicaciones.

Un agente incluye estado acerca de su dirección (identificación del nodo y del port dentro del nodo) y de la dirección del agente al cual enviará los paquetes que genere. La dirección propia la adquiere al asociarlo a un nodo a través del método *attach-agent* provisto por la clase *Simulator*, mientras que la dirección de destino es configurada a través del método *connect* [18]o asignando un valor específico a la variable *dst_* del agente.

Se define además el valor ttl que el agente asignará a cada paquete generado, la longitud en bytes del paquete (utilizada sólo para el cálculo del tiempo de transmisión por los objetos DelayLink), la prioridad IP, un identificador de flujo también a nivel IP, y el tipo de paquete.

Los métodos provistos por la clase permiten generar y enviar un paquete y recibir paquetes destinados al agente.

Entre los agentes provistos se encuentran varias clases de agentes TCP, agentes CBR, UDP, un agente para envío de mensajes, y el agente nulo cuya función es descartar paquetes.

3.4 Ejemplo

En el siguiente ejemplo se muestra una simulación de una topología simple, compuesta de tres nodos unidos por dos vínculos bidireccionales de 1Mbps con una demora de propagación de 5 milisegundos y una política de descarte de paquetes específica (DropTail). En los nodos n1 y n2 hay dos procesos que envían información a una tasa constante a los agentes correspondientes en n0.

```
#Instanciacion del simulador
set ns [new Simulator]

#Facilidad de trace
set f  [open salida.tr w]
$ns trace-all $f

#Nodos
set n0 [$ns node]
```

[18] No debe asociarse la invocación del método *connect* entre dos agentes con el establecimiento de una conexión entre ellos. Sólo es una asociación de direcciones. Es así por ejemplo que dos agentes UDP deben ser relacionados por el método connect para que puedan comunicarse.

```
set n1 [$ns node]
set n2 [$ns node]

#Vinculos
$ns duplex-link $n0 $n1 1Mb 5ms DropTail
$ns duplex-link $n0 $n2 1Mb 5ms DropTail

#Agentes
set emisor1 [new Agent/CBR]
set emisor2 [new Agent/CBR]
set receptor1 [new Agent/Null]
set receptor2 [new Agent/Null]

#Asociación de agentes y nodos
$ns attach-agent $n0 $receptor1
$ns attach-agent $n0 $receptor2
$ns attach-agent $n1 $emisor1
$ns attach-agent $n2 $emisor2

#Conexiones entre los agentes
$ns connect $emisor1 $receptor1
$ns connect $emisor2 $receptor2

#Eventos al simulador (arranque de agentes)
$ns at 1.0 “$emisor1 start”
$ns at 1.1 “$emisor2 start”
$ns at 5.0 “$ns halt”

$ns run
```

Los agentes CBR generan paquetes a una tasa constante desde que son activados (*start*). Ambos toman valores por defecto (longitud de paquetes y tasa de envío). A través del método *attach-agent* se asocian los agentes (procesos) a los nodos.

Como fue mencionado, las “conexiones” entre los pares de agentes, no tienen el significado de una comunicación orientada a conexión, como es por ejemplo TCP. El método *connect* provisto por el simulador, sólo tiene como consecuencia dar valor a las variables de instancia *dst_* en ambos agentes, para que envíen sus paquetes a la dirección indicada.

El comienzo de la actividad de los agentes (*start*) y la indicación al simulador para que detenga la simulación, se especifican a través de la generación de eventos específicos para el planificador de eventos utilizando el método “*at*” provisto por el simulador. Notar que en el transcurso de la simulación, serán generados otros eventos, por medios diferentes a “*at*” por los diferentes componentes (por ejemplo, objetos *DelayLink*).

Por último, se indica al simulador que comience la corrida, a través del método “*run*”. La salida de resultados se especifica a través de sentencias trace al simulador (en este caso *trace-all*) indicando que la salida se dirija al archivo “salida.tr”.

3.5 Otras facilidades provistas por el simulador

En las secciones anteriores fueron presentados los componentes básicos del simulador, de manera de dar una idea general respecto a su estructura y funcionamiento. Por razones de

extensión, se describe brevemente en los siguientes párrafos otras características de importancia de Ns.

- **Paquetes**: Los paquetes constituyen eventos. Contienen un encabezamiento común (longitud, tiempo, uid, rótulo de interfaz), un encabezamiento definido según cada protocolo y opcionalmente datos.
- **Modelos de error**: Simula errores a nivel data-link, marcando el paquete como erróneo a través de flags del encabezamiento o descartando el paquete (envío al drop-target). La generación de errores puede hacerse a través de modelos simples (tasa de errores en paquetes) o modelos más complejos. La granularidad puede ir desde paquetes hasta bits.
- **Generación de tráfico**:Se provee la facilidad de generación de tráfico, determinando tasas de envío y longitud de paquetes. La generación y envío de paquetes está a cargo de agentes UDP. Como clases derivadas de *TrafficGenerator*, actualmente se provee *EXPOO_Source* que genera tráfico de acuerdo períodos on/off de acuerdo a una distribución exponencial y *POO_Source*, igual que la anterior pero utilizando una distribución Pareto.
- **Administración de buffers**: Se refiere a la política que se utiliza para regular qué paquetes ocuparán las colas que representan buffers en los routers. Puede ser FIFO (DropTail) o RED (Random Early Detection).
- **Planificación (scheduling) de paquetes**: Se refiere al mecanismo para seleccionar la política de atención de los paquetes en los buffers (en las colas que representan los buffers). Puede ser FIFO, variantes de Fair Queueing, incluyendo Wiighted Fair Queueing, etc.
- **Facilidades de trace**: Las facilidades de tracing provistas por el simulador, permiten almacenar datos en archivos seleccionados por el usuario, para realizar posteriormente a la simulación, el análisis de los mismos. Ns incluye objetos de la clase *Trace*, que permiten registrar cada paquete individualmente, cuando arriba a un nodo, cuando deja el nodo o es descartado, y objetos de la clase *Monitor*, que permiten registrar contadores respecto a la totalidad o según el tipo de paquetes (por ejemplo contador de paquetes que arriban a un nodo). Tanto los objetos *Trace* como los *Monitor* utilizan el encabezamiento común que tienen los paquetes.
- **Redes locales**: Permite especificar características de redes locales, capturando el comportamiento de los tres niveles inferiores: nivel físico, subnivel de acceso al medio y subnivel LLC. Se provee el soporte para la definición de diferentes protocolos a nivel MAC y LLC.
- **Ruteo unicast**: Las facilidades provistas para ruteo unicast permiten calcular las tablas de ruteo en los nodos definidos.
  El ruteo centralizado consiste en un cálculo de rutas realizado externamente a la simulación. Se calcula los caminos mínimos (algoritmo SPF). Se proveen dos alternativas: ruteo estático, calculado sólo al comienzo de la simulación y Ruteo por Sesión, en la cual se calculan las rutas al comienzo de la simulación y cada vez que se produce un cambio en la topología.
  El ruteo detallado consiste de un cálculo de rutas realizado como parte de la simulación. En la actualidad, está implementado el ruteo Distance Vector, pudiendo configurarse el tiempo de intercambio de tablas, el uso de split horizon with poison reverse y triggered updates. El uso de ruteo detallado en topologías con cantidad considerable de nodos puede dar lugar a tiempos más elevados en las corridas.
- **Ruteo multicast**: La estructura de un nodo con capacidad multicast fue descripta antes. Las clases que soportan el desarrollo de protocolos multicast serán descriptas en el capítulo correspondiente a la implementación. Las facilidades multicast que actualmente ofrece Ns consisten de protocolos centralizados (CtrMcast: Sparse mode y Dense mode) y protocolos detallados, (Dynamic Dense Mode y Pim modo denso). En la actualidad se encuentra en desarrollo PIM modo denso y sparse. En el presente trabajo se implementó CBTv2 detallado.

3.6 Resumen

En este capítulo fueron presentadas las características más relevantes del medioambiente de simulación utilizado. Este medioambiente permite la rápida y sencilla generación de topologías y agentes utilizando las facilidades provistas como así también la incorporación de nuevos elementos a través de la creación o extensión de clases compiladas o interpretadas. En el presente desarrollo se extendieron algunas clases ya provistas y se incorporaron nuevas clases íntegramente en Otcl a efectos de obtener un código fácilmente portable al no ser necesaria la recompilación del simulador.

Capítulo 4

CBT versión 2

Como se mencionó antes, CBT es un protocolo diseñado para operar en redes de área extensa, que en el futuro deberán soportar facilidades multicast[19] en sus routers y una diversidad y gran número de aplicaciones que harán uso de este tipo de transmisión.

Los principios de su diseño son escalabilidad y simplicidad, características que se ven reflejadas en la magnitud de la información de estado que debe mantenerse en los routers[20] y en la carga introducida en la red como consecuencia de la actividad del protocolo.

Como consecuencia de sus principios de diseño, presenta mayores demoras que los protocolos basados en la construcción de SPTs (Shortest Path Trees) y una mayor vulnerabilidad a fallas debido a la existencia de un único core por grupo[21].

4.1 Operación del protocolo

Inicialmente, cada router con capacidad CBT, conoce la dirección unicast de un nodo seleccionado como "core" o centro del árbol compartido a través del cual la información originada por los emisores será distribuida a los miembros del grupo multicast.

El mecanismo para la elección del nodo core, no está especificado en el protocolo, mientras que las alternativas para la distribución de la información respectiva a cores a los routers interesados puede realizarse manualmente (configuración estática) o a través de un procedimiento de bootstrapping, actualmente en desarrollo en el contexto de SM-PIM. Cualquiera de los routers puede ser seleccionado como core, no requiriendo capacidad especial. Un nodo core, puede serlo para uno o varios grupos, no siendo necesario que pertenezca a dichos grupos.

4.1.1 Proceso de integración a un grupo

Inicialmente, un router que aún no integra el árbol de distribución de un grupo, recibe una solicitud de un host de su red local para integrarse al mismo. Esta solicitud será recibida a través de IGMP[22]. Como consecuencia de la solicitud recibida, la entidad CBT en el router determina, en base a las tablas de ruteo unicast, la interfaz de salida hacia el router seleccionado como core para el grupo. En el caso en que la interfaz no soporte transmisión multicast[23], deberá tenerse en cuenta además la dirección unicast del próximo router camino al core.

Esta característica se debe a que en vínculos soportando transmisión multicast, las PDUs CBT se transmiten con la dirección multicast "all_cbt_routers" y con ttl igual a 1. Si en

[19] En la actualidad, el MBONE funciona encapsulando paquetes entre routers predeterminados que soportan multicast, pero un router en la Internet no necesariamente soporta actualmente transmisión multicast.

[20] Los términos "router" y "nodo" se utilizan en este capítulo de manera indistinta.

[21] Un nodo puede actuar como core para varios grupos, pero un grupo puede tener sólo un core.

[22] La solicitud podrá ser recibida también a través de una aplicación local en un nodo que actuando como router soporte funcionalidad a niveles superiores al de red.

[23] Cuando se menciona que la interfaz o el vínculo no soportan transmisión multicast,en realidad se hace referencia a una red que no provee tal capacidad, por ejemplo NBMA.

cambio, el vínculo no soporta transmisión multicast, las PDUs son enviadas a la dirección unicast del próximo router camino al core. Debe notarse aquí la independencia de CBT respecto del protocolo de ruteo unicast ya que sólo utiliza las tablas construidas por éste.

Una vez determinado el próximo router o interfaz hacia el nodo core, la entidad CBT envía una solicitud (JOIN-REQUEST) para integrarse al árbol de distribución.

Se almacena a su vez información transitoria respecto a la interfaz desde la que se originó el pedido y a la interfaz a la cual se solicitó el requerimiento:

<grupo, interfaz de llegada, interfaz de salida>

En la 3-upla generada, interfaz de llegada (downstream) o interfaz de salida (upstream) pueden ser direcciones unicast del próximo nodo camino al core en caso de que el vínculo no soporte multicast o referencias a interfaces locales en caso de que lo soporte [24].

Interfaz de llegada, puede ser (y lo será en el caso que se está describiendo) una referencia a interfaz hoja (leaf), en caso en que la solicitud haya sido recibida por una entidad local o por IGMP en representación de un host de una red local al router. Esto significa que el router constituirá, una vez efectivamente integrado al árbol, una hoja del mismo y actuará de acuerdo a ello en ciertos procedimientos especiales.

La información generada y almacenada de la manera descripta se denomina transitoria ya que hasta este momento el nodo no pertenece al árbol de distribución, sino que debe esperar la respuesta al JOIN-REQUEST, un JOIN-ACK. El arribo de esta PDU confirma que se ha contactado al core o a un nodo ya perteneciente al árbol de distribución y que efectivamente se integra el árbol del grupo. Al recibir el JOIN-ACK, el nodo crea información permanente respecto de su pertenencia al árbol , eliminando la información transitoria.

La información transitoria tiene un tiempo máximo de vida, al cabo del cual, si no es recibido el JOIN-ACK, es eliminada. Los procedimientos de recuperación en este caso, dependen si el router es leaf o no. Sólo un router leaf es responsable por la recuperación, realizando una serie de reintentos de envío del JOIN-REQUEST, luego de los cuales desiste. Los próximos reintentos se realizarán en función de solicitudes recibidas de IGMP o de aplicaciones locales.

El procedimiento descripto, a través del cual se recibe una solicitud y se intenta la conexión al árbol a través del próximo router camino al core, se irá produciendo en los routers que reciban un JOIN-REQUEST (en este caso, no serán routers leaf).

Cuando la solicitud arriba a un nodo que ya integra el árbol de distribución o cumple funciones de core para el grupo, éste responde con un JOIN-ACK al nodo anterior, incorporando la interfaz a través de la cual fue recibida la solicitud (JOIN-REQUEST) a la lista de interfaces “child” para el grupo. Este procedimiento se repetirá router a router en camino inverso al recorrido por el JOIN-REQUEST, hasta que el JOIN-ACK llegue hasta el que originó la solicitud inicial.

Como resultado del procedimiento descripto, se crean las entradas permanentes en los nodos, con la siguiente información:

< grupo, interfaz parent, {interfaz child, . . ., interfaz child} >

Debe tenerse en cuenta que la característica de parent o child de una interfaz sólo es utilizada por CBT (a diferencia de SM-PIM) para el mantenimiento del árbol de distribución y no para la distribución de los datos multicast, ya que el árbol creado por CBT es bidireccional.

[24] En una implementación, se almacenará una dirección IP: la de la placa local hará referencia a la interfaz, mientras que la dirección IP del próximo nodo hará referencia al siguiente nodo camino al core.

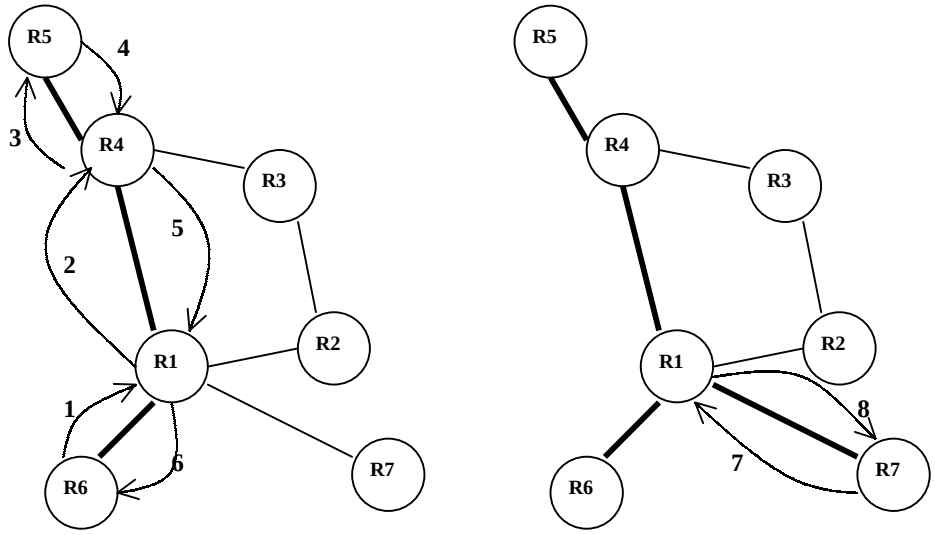

Fig 4.1. Construcción del árbol de distribución CBT*: En la parte izquierda se muestra la secuencia de PDUs intercambiada como consecuencia de un requerimiento de integración al grupo por parte del router R6. A la derecha se muestra la actividad que se produce como consecuencia de un requerimiento posterior de R7. El nodo core para el grupo es R5. En ambos casos los vínculos que integran el árbol de distribución están trazados con línea gruesa.(1,2,3,7:Join-request; 4,5,6,8:Join-Ack).*

En la figura 4.1 se muestra el proceso de integración de un router a un grupo y el correspondiente conjunto de PDUs intercambiadas.

4.1.2 Envío y distribución de datos

Una vez integrado el árbol de distribución, un router es capaz de recibir información multicast direccionada al grupo y entregarla a los hosts locales o aplicaciones interesadas.

Un router origen de los datos multicast (emisor) puede o no pertenecer al grupo. En caso de pertenecer, enviará el paquete por la totalidad de las interfaces correspondientes al grupo. De esta manera, la información fluirá por el árbol hacia todos los receptores. En el caso en que un router no miembro del grupo desee enviar información, sólo debe conocer la dirección unicast del core y encapsular el paquete multicast de manera de enviarlo unicast al core. Este desencapsulará el paquete y lo inyectará luego en todas las interfaces asociadas al grupo.

Un paquete que llega a un router, es simplemente enviado por todas las interfaces asociadas al grupo, excepto por la de arribo.

El la figura 4.2 se muestra la manera en que se distribuyen los datos.

4.1.3 Poda del árbol

La poda del árbol de distribución se realiza en el sentido de las hojas hacia el core, y es iniciada por un router cuando la lista de interfaces child para un grupo se vuelve vacía. Esto puede deberse, en el caso de un router leaf, a que los hosts locales no responden a los queries IGMP, o en caso de routers no leaf, a requerimientos de poda de sus routers downstream.

El requerimento de poda se realiza a través del envío de un QUIT-NOTIFICATION al nodo parent. Este envío no es confirmado por el parent, debido a lo cual la PDU se reenvía a intervalos regulares una cierta cantidad de veces. Al realizarse el primer envío, el router elimina la información del grupo almacenada localmente. Un router, al recibir un QUIT-NOTIFICATION por una interfaz child válida, eliminará la misma de su lista de child para el grupo[25], lo que puede dar lugar a que dicha lista se convierta en vacía, lo que hará que se continúe el proceso de poda en dirección al core del grupo.

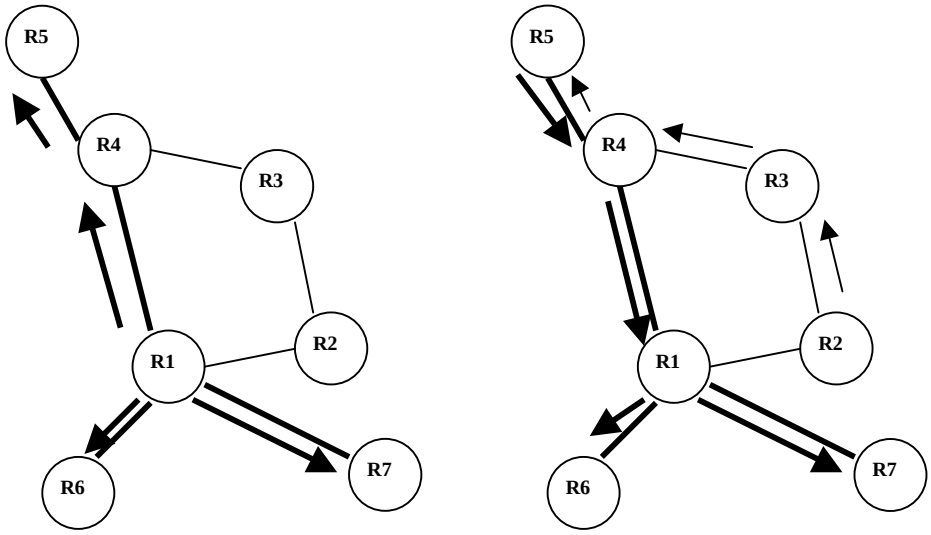

***Fig 4.2. Envío y distribución de paquetes multicast en CBT**: En la parte izquierda se muestra cómo fluye la información multicast originada por R1, quien la inyecta en todas las interfaces asociadas al grupo. A la derecha un router no miembro del grupo, R2, encapsula el paquete multicast y lo envía al core; éste lo inyecta en el árbol, en todas las interfaces. Los routers pertenecientes al árbol, copian un paquete recibido en todas las interfaces del grupo excepto en la de arribo. Las transmisiones unicast se muestran con línea fina, y las multicast con línea gruesa.*

4.1.4 Mantenimiento del árbol de distribución

En CBT versión 2, cada nodo child es el responsable de monitorear los vínculos con sus nodos parent[26], para determinar si existe conectividad. En caso de detectarse pérdida de la conectividad, el nodo se desconecta de los árboles de distribución correspondientes[27], iniciando un proceso similar hacia las hojas de los subárboles involucrados. De esta manera, un subárbol que resulte desconectado, se eliminará totalmente, y luego, los nodos leaf (interesados en la pertenencia al árbol) tratarán, independientemente, de restaurar la conexión. En ningún caso, un nodo mantiene el subárbol del cual es raíz e intenta reconectarse, debido a que este procedimiento podría dar lugar a la producción de ciclos en el árbol de distribución.

[25] La descripción es simplificada, ya que en casos de vínculos de acceso múltiple se toman otras acciones que serán descriptas más adelante.
[26] En CBTv3 se incorpora un mecanismo a través del cual un nodo parent monitorea también el vínculo hacia sus nodos child.
[27] Aquellos árboles correspondientes a grupos cuya interfaz parent está asociada al vínculo.

El monitoreo de cada vínculo se realiza a través de PDUs ECHO-REQUEST, enviadas periódicamente por un nodo child, que son respondidas por el parent con un ECHO-REPLY. La recepción de un ECHO-REPLY por parte de un nodo, provoca el reinicio de los timers asociados a las entradas permanentes para cada grupo involucrado en el mensaje. Si el ECHO-REPLY no es recibido en término, se produce el timeout de la entrada que de esta manera es eliminada, provocando a su vez la eliminación del subárbol a través del envío de FLUSH-TREE a los nodos child para los grupos involucrados.

Un ECHO-REQUEST no contiene información respecto a grupos; un ECHO-REPLY, en cambio, contiene información explícita respecto de los grupos para los que el nodo emisor actúa como parent en el vínculo sobre el que se envía la PDU. Lo mismo ocurre con un FLUSH-TREE.

4.2 Operación en vínculos multiacceso

La operación del protocolo se vé modificada en vínculos de acceso múltiple. Por un lado aparece el concepto de "designated router" (DR), que implica que uno de los routers conectados al vínculo será seleccionado para realizar las conexiones al árbol para todos los grupos, y por otro lado se modifica en algunos casos la operación descripta arriba para vínculos punto a punto.

La necesidad de un DR en un vínculo multiacceso está dada en el caso de que dicho vínculo conecte dos o más dominios de ruteo debido a que cada uno de ellos puede tener una visión diferente del ruteo unicast, lo cual podría dar lugar a la formación de ciclos en el árbol de distribución.

4.2.1 Elección del DR

Cualquiera de los routers tiene la capacidad de actuar como DR, estando esta condición indicada por una variable (flag DR) asociada a cada una de las interfaces multiacceso de un router; por defecto su valor es FALSE, indicando que el router no es DR.

Cada router tiene almacenado un valor de preferencia por interfaz multiacceso, que puede variar entre 1 y 255, siendo 1 el valor que representa una mayor prioridad para la elección del router como DR, y 255 el que representa la menor prioridad. Por defecto, los routers asumen un valor de 255; pueden ser configurados con valores entre 1 y 254. El valor 0 se reserva para uso del router elegido como DR, independientemente de cual sea su valor de preferencia configurado.

La elección y mantenimiento como tal de un router para cumplir el rol de DR, se realiza de acuerdo al protocolo HELLO, como se describe a continuación.

Cada router tiene un timer para enviar periódicamente una PDU HELLO a la dirección multicast "all-cbt-routers". Un router que recibe un HELLO con un valor de preferencia más bajo que el propio (o igual pero proveniente de un router con dirección más baja), reinicia el timer y de esta manera se abstiene de enviar el HELLO propio.

El router que ha anunciado un valor de preferencia más bajo, es el que se convierte en DR luego de un cierto tiempo, al no recibir HELLOs de los demás routers. Independientemente del valor de preferencia configurado, el DR anunciará un valor de preferencia 0.

Un router que recibe un HELLO con un valor de preferencia menor que el propio, o igual pero de un router con una dirección mayor, debe contestar enviando su valor de preferencia.

4.2.2 Consideraciones respecto a la operación del protocolo

Como consecuencia de la existencia de un DR, un router podrá tener dos interfaces (desde el punto de vista del mantenimiento del árbol de distribución) sobre el mismo vínculo multiacceso[28]. De acuerdo a esto, un DR es el único router que procesa JOIN-REQUESTs multicast recibidos por el vínculo multiacceso.

Un router que recibe un QUIT-NOTIFICATION multicast por una interfaz correspondiente a un vínculo multiacceso, no debe eliminar inmediatamente la interfaz child, dado que puede haber otros routers child en la misma interfaz[29]. En cambio, activa un timer (cache_deletion_timer) para eliminar la información referente a la interfaz luego de cumplido el timeout. Esta espera da tiempo a otros routers child sobre el mismo vínculo, para producir el envío de un JOIN-REQUEST para el grupo y lograr de esta manera que el parent no elimine la interfaz.

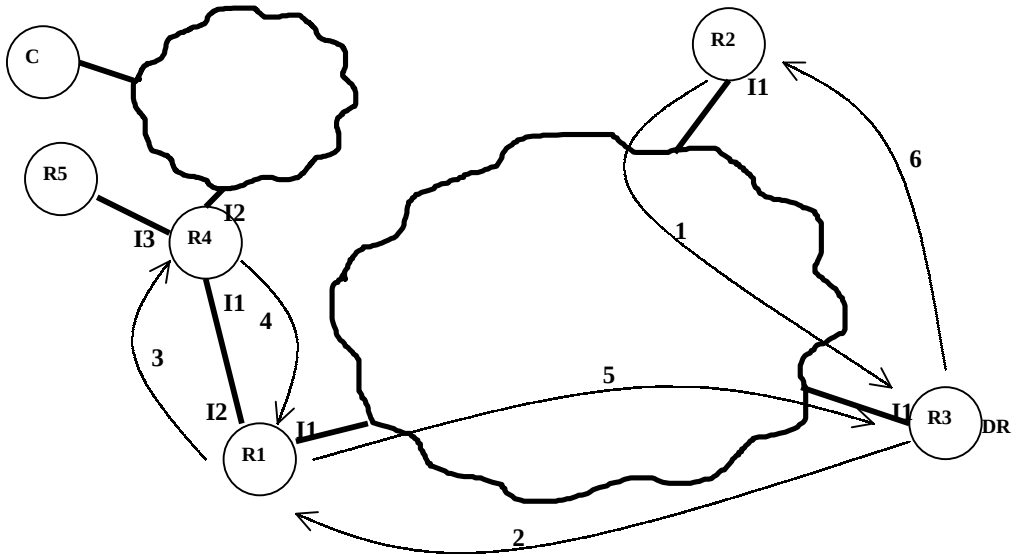

Fig 4.3. Secuencia de PDUs intercambiadas en un vínculo multiacceso*: actuando R3 como designated router para el establecimiento del árbol de distribución del grupo G teniendo como core al nodo C*

Para posibilitar un uso eficiente de la red respecto al envío del JOIN-REQUEST, un router que recibe un QUIT-NOTIFICATION por una interfaz parent multiacceso, activa un timer con un tiempo al azar entre 0 y un tiempo máximo prefijado, para realizar el envío. Si antes del vencimiento, el router detecta un JOIN-REQUEST para el mismo grupo, proveniente de un router que ha elegido un tiempo menor, cancela el envío pendiente.

En la figura 4.3 se muestra el intercambio de PDUs en el caso de un vínculo multiacceso con un DR (R3) que no es el mejor camino al core (C).

La notación utilizada para indicar las tablas en cada nodo se muestra más abajo. El primer elemento (G) indica el grupo, el segundo elemento la interfaz parent, compuesta de una

[28] Un DR tendrá una interfaz child multicast sobre el vínculo multiacceso y una interfaz parent unicast, con la dirección del próximo nodo camino al core, en caso de que el nodo (DR) no constituya el mejor camino al core (para la métrica de ruteo unicast) entre los routers del vínculo multiacceso.

[29] El router parent sólo tiene información respecto de la interfaz pero no respecto a los routers en particular.

referencia a la interfaz física y entre paréntesis una indicación del nodo adyacente (* indica que no existe referencia a un nodo específico –envío multicast-) y el tercer elemento el conjunto de interfaces child, para las cuales se utiliza la misma notación que para la interfaz parent.

Inicialmente, R4 pertenece al árbol de distribución del grupo (G), como consecuencia de un requerimiento de R5.

Su tabla es: <G, I2 (*), { I3 (*)}>

Se produce a continuación la solicitud del router R2 para integrarse al grupo G. R2 obtiene como mejor interfaz hacia el core C, su interfaz I1 correspondiente al vínculo multiacceso. Como consecuencia se produce el intercambio de PDUs en el orden especificado en la figura:

1 JOIN-REQUEST multicast
2 JOIN-REQUEST unicast a R1
3 JOIN-REQUEST multicast
4 JOIN-ACK multicast
5 JOIN-ACK unicast a R3
6 JOIN-ACK multicast

Como consecuencia de no ser el DR el mejor router al core en el vínculo multiacceso, se producen dos PDUs adicionales. Debe tenerse en cuenta que el overhead es mínimo, ya que no se produce en la distribución de los datos.

Las tablas en cada router contendrán:

R1: <G, I2 (*), I1 (R3)>
R2: <G, I1 (*), {LEAF}>
R3: <G, I1 (R1),{ I1 (*)}>
R4: <G, I2 (*), { I1 (*), I3 (*) }>

4.3 Resumen

En este capítulo se presentó una descripción simple de las características más importantes de CBTv2: integración a un grupo y construcción del árbol de distribución, mantenimiento y poda del árbol y envío de datos. Posteriormente se describieron características adicionales que surgen al considerar su operación en vínculos multiacceso. Esta descripción junto con la documentación del protocolo[6][7][8] constituyen la base para la especificación realizada en el capítulo 5.

Capítulo 5

Especificación del protocolo

Es necesario definir de manera precisa la operación del protocolo CBT a efectos de lograr una implementación correcta. Para realizar esta definición se toma como base la descripción en lenguaje natural provista en [7], y se la expresa en forma de máquina de estados finitos utilizando una notación basada en tablas evento/estado. Esta notación corresponde a la propuesta y utilizada por la ISO/OSI[30] para especificar la operación de las entidades que conforman un protocolo.

Los documentos de especificación de protocolos producidos por ISO/OSI consideran dos aspectos, la definición formal de la unidades de datos del protocolo (PDUs), y la especificación de su operación.

La definición de las unidades de datos del protocolo, o sintaxis del protocolo, se refiere al aspecto estático del mismo y se realiza en ASN.1 (Abstract Syntax Notation One), una notación estandarizada para descripción de datos que soporta sintaxis variadas.

La especificación de la operación de un protocolo describe las reglas que caracterizan el comportamiento del mismo. Este es el aspecto más complejo debido a que contempla interacciones entre diferentes entidades del mismo nivel o niveles adyacentes de la arquitectura.

En este caso particular, no es de importancia la definición de las PDUs debido a la simplicidad de su formato y su realización sólo sería utilizable como una manera no ambigua de especificación de las mismas[31]. Sí es de importancia la especificación de la operación del protocolo, la cual se realiza con el objetivo de disponer de una base que constituya una guía en la fase de implementación. Debe aclararse que la especificación realizada no tiene por objeto producir derivaciones automáticas de código ni posibilitar el análisis del protocolo respecto a su correctitud u otros aspectos a los que apuntan las técnicas formales de descripción (FDT).

5.1 Método utilizado para la especificación

La definición precisa de la operación del protocolo se realiza través de un máquina de estados finitos (FSM) que representa a la entidad CBT. Se describen sus estados, los eventos a los cuales reacciona y sus acciones, internas y externas (envío de PDUs). Para materializar esta representación se utiliza una tabla que contiene todas las combinaciones posibles de estados y eventos, indicando para cada una de ellas, posibles 3-uplas cuyas componentes representan un conjunto de acciones internas (IA) (por ejemplo cambio en el valor de una variable), un conjunto de eventos de salida (OE) (por ejemplo envío de una PDU) y el nuevo estado en que quedará el autómata (NS).

Los eventos, como se muestra en la figura 5.1, se agrupan en eventos provenientes del nivel superior (1), eventos provenientes de entidades pares remotas (2) y eventos internos provenientes de entidades locales (3).

[30] International Organization for Standardization, Open Systems Interconnection.

[31] El caso de el desarrollo de aplicaciones (Nivel Aplicación) en el contexto ISO/OSI es diferente, debido a la existencia de compiladores ASN.1 a diferentes lenguajes y al uso de una sintaxis común de transferencia (sintaxis concreta).

Dado un estado y un evento de entrada, podrán tomarse distintos conjuntos de acciones, según el estado de ciertas variables y la evaluación de ciertas condiciones. El conjunto

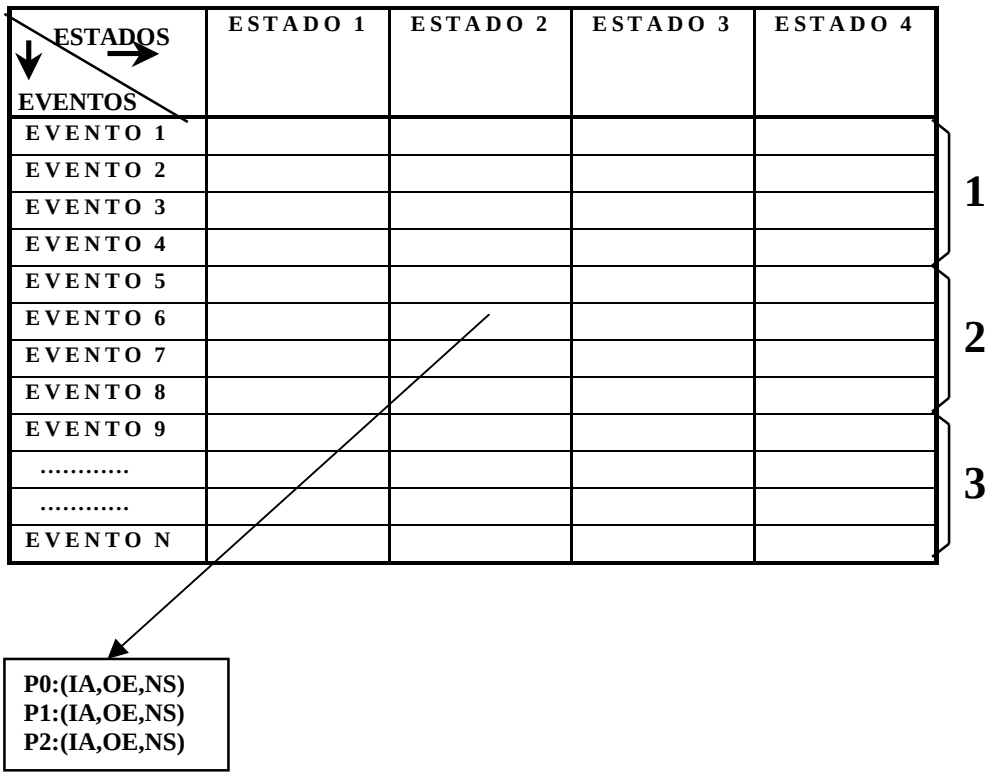

Fig 5.1. Formato de las tablas estado/evento. Utilizadas para describir la operación del protocolo

específico de acciones a tomar (definido por la 3-upla correspondiente), se basa en el valor de verdad de un predicado asociado al mismo. El conjunto de predicados de una entrada, debe cubrir todos los casos posibles.

5.2 Operación del protocolo

Previo a la descripción de los elementos de la FSM (estados, eventos, acciones internas y predicados), se describen los objetos internos con los cuales interactúa la entidad CBT: interfaces, timers y tablas.

5.2.1 Interfaces y vínculos

Una interfaz física está compuesta de un dispositivo de hardware (por ejemplo una placa de red) y el software asociado (driver del sistema operativo).

Una interfaz de este tipo está asociada con un vínculo físico, que puede ser por ejemplo un vínculo RS-232, una red Ethernet, etc. A continuación se describen las propiedades de interés para esta descripción que presentan dichas interfaces.

Desde el punto de vista de la cantidad de nodos que acceden al vínculo, tal como se lo percibe desde la interfaz física[32]:

- Interfaces a vínculos con capacidad multiacceso: desde la misma interfaz física, es posible acceder a diferentes equipos.
- Interfaces a vínculos sin capacidad multiacceso (punto a punto): desde la interfaz, sólo es posible acceder a un solo equipo.

Con relación al soporte de direcciones grupales[33]:

- Interfaces a vínculos con capacidad multicast: es posible utilizar direcciones grupales.
- Interfaces a vínculos sin capacidad multicast: sólo se permite el uso de direcciones de destino individuales.

Debe tenerse en cuenta que son válidas las cuatro combinaciones posibles:

- Interfaces a vínculos sin capacidad multiacceso y sin capacidad multicast.
- Interfaces a vínculos sin capacidad multiacceso con capacidad multicast
- Interfaces a vínculos con capacidad multiacceso sin capacidad multicast (redes NBMA).
- Interfaces a vínculos con capacidad multiacceso con capacidad multicast (por ejemplo redes Ethernet).

CBT puede operar sobre cualquier tipo de vínculo, y distingue el concepto de interfaz (CBT), diferente del concepto de interfaz física. Una interfaz CBT, se refiere a la interfaz física pero agrega una referencia al tipo de dirección (unicast o multicast) del router accedido a través de ella.

Una interfaz CBT se identifica, dentro del nodo, por la dirección IP de la interfaz local[34] (transmisión multicast), o bien por la dirección IP del nodo (upstream o downstream) (transmisión unicast). Debe considerarse que al identificar una interfaz CBT mediante una dirección de red, ya sea local o remota, se está haciendo referencia a una interfaz física local, la asociada a dicha dirección. De esta manera, pueden corresponder varias interfaces CBT a una interfaz física.

Sobre un vínculo punto a punto, se tendrá una interfaz CBT identificada por la dirección local si el vínculo soporta multicast, o una interfaz CBT identificada por la dirección del equipo remoto en caso contrario.

Sobre un vínculo multiacceso sin soporte multicast, se tendrá un conjunto de interfaces CBT cada una de ellas identificada por la dirección del equipo remoto.

Sobre un vínculo multiacceso con soporte multicast, se debe distinguir dos casos, existirá una interfaz (multicast) en la que el parent será el DR y los posibles child los demás nodos; esta interfaz se identificará en cada nodo, por su propia dirección. Puede existir una interfaz (unicast), en la cual el nodo DR es child y uno de los nodos no DR es parent; este caso se dará si el mejor camino al core del grupo no es el DR. Esta interfaz unicast, será identificada en el nodo DR a través de la dirección del "best hop" y en el nodo "best hop" a través de la dirección del nodo DR. De esta manera, el tráfico de PDUs unicast es ignorado por el resto de los nodos en el vínculo.

---

[32] Esta característica está dada por el tipo de subred asociada a la interfaz.

[33] Esta característica estará dada por las alternativas que ofrezca el protocolo de nivel de red (por ejemplo routers IP pueden o no soportar direccionamiento multicast) o por las facilidades ofrecidas por la subred.

[34] En el presente trabajo no se considera la utilización de varias redes lógicas -direcciones de redes IP- sobre un vínculo.

Para evitar ciclos en el árbol de distribución, un nodo debe comprobar al agregar una interfaz child que ella no coincida con la parent del grupo. Una comprobación adicional que debe realizar un nodo no DR es que la red asociada a cualquiera de sus interfaces child para el grupo sea diferente de la asociada a su parent para evitar que un nodo al cual el DR elige como parent produzca un ciclo en el caso de seleccionar el vínculo multiacceso como mejor camino al core. Un nodo actuando como DR, en cambio, puede tener una interfaz parent (unicast) sobre la misma red que algunas de sus child (multicast), en el caso de que el mejor camino al core esté constituido por otro router del vínculo multiacceso.

Una interfaz CBT almacena, además de la dirección y tipo de envío (unicast o multicast), otro tipo de información, tal como estado de DR o no, valor de preferencia (no

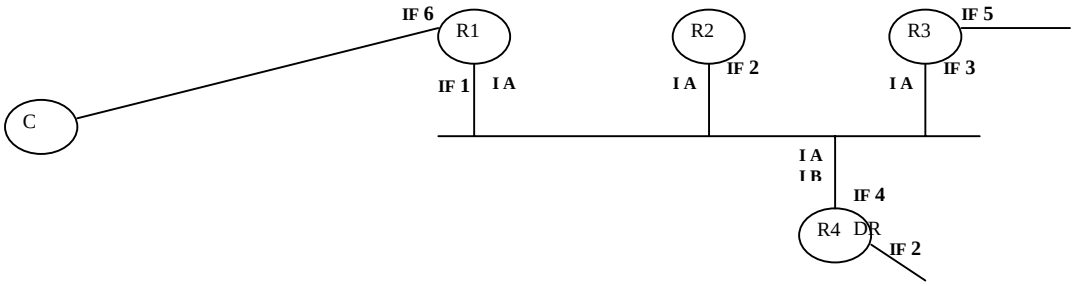

Figura 5.2. Relación entre interfaces físicas e interfaces CBT.

utilizado en la especificación debido a que no se implementa el protocolo HELLO), etc.

En la figura 5.2 se muestra una red multiacceso con capacidad multicast a la cual están conectados 4 routers, R1, R2, R3 y R4. R4 actúa como DR mientras que R1 constituye el mejor nodo desde el punto de vista del vínculo multiacceso para enviar al core de un grupo determinado, C. Las interfaces físicas se indican como IF x, siendo esta identificación la dirección de red local en el vínculo, mientras que las interfaces CBT se indican como I x.

Se muestran a continuación las interfaces CBT en cada uno de los nodos y cómo son identificadas a partir de las correspondientes interfaces físicas (o direcciones de las placas de red):

Router R4:
- IA: interfaz child sobre el vínculo multiacceso. IF4,M
- IB: interfaz parent al grupo (a R1): IF4, IF1

Router R3:
- IA:interfaz parent sobre el vínculo multiacceso. IF3,M

Router R2:
- IA:interfaz parent sobre el vínculo multiacceso. IF2,M

Router R1:
- IA:interfaz child sobre el vínculo multiacceso. IF1,IF4

El carácter unicast o multicast de una interfaz CBT está indicado por el segundo componente de su identificación, M (multicast) o la dirección del nodo adyacente (IFx).

Un error que produciría ciclos en el árbol de distribución sería que R1 constituyera una interfaz child sobre el vínculo multiacceso, de la misma manera que R2 o R3.

En adelante se hará referencia a una interfaz CBT utilizando el término "interfaz", y a una interfaz física utilizando el término "vínculo".

5.2.2 Tablas

Las tablas sobre las cuales opera la entidad CBT, representan la información multicast correspondiente al árbol de distribución (MFC: Multicast Forwarding Cache), que será utilizada para el mantenimiento del árbol y la distribución de los paquetes multicast. La información que contiene cada entrada, como fue visto en capítulos anteriores, consiste del grupo multicast, que constituye el índice de la tabla, la interfaz (CBT) parent (en dirección a la raíz del árbol) y un conjunto de interfaces child, hacia las hojas del árbol.

Existen dos tablas con información similar: la tabla de entradas transitorias, que contiene información respecto de grupos para los cuales se ha solicitado la pertenencia al árbol de distribución pero aún no ha sido confirmada, y la tabla permanente, que contiene aquellas entradas que indican interfaces que ya pertenecen al árbol de distribución (estado on-tree del nodo respecto del grupo).

Una información adicional contenida en cada interfaz child de cada entrada, se refiere a la condición de hoja del árbol. Dicha condición influye en la operación del protocolo.

La entidad CBT creará y eliminará entradas en las tablas de acuerdo a requerimientos recibidos, sus correspondientes respuestas y eventos tales como timeouts.

5.2.3 Timers

Los timers son utilizados por la entidad CBT para realizar reintentos ante operaciones no exitosas (p.ej. retransmisión de JOIN_REQUESTs), para la eliminación de entradas en las tablas transitorias y permanentes y para la repetición de procedimientos que deben efectuarse periódicamente, tales como el monitoreo de las interfaces parent por parte de un nodo.

Las operaciones realizadas sobre los timers tenidas en cuenta en la especificación son las siguientes:

- **Activación**: El timer es cargado con un valor y comienza a decrementarlo. Cuando llegue a cero producirá el evento asociado con su expiración (timeout).
- **Desactivación**: El conteo del timer es interrumpido, evitando el timeout.
- **Reactivación**: El timer es desactivado y vuelto a activar con un nuevo valor. Se utiliza, por defecto el valor original.

Las operaciones anteriores son las que tienen relevancia en la especificación del protocolo. Es importante destacar que desde el punto de vista de una implementación, existen dos operaciones adicionales, la creación y la eliminación de un timer, que se refieren a la asignación y liberación de los recursos insumidos por el objeto. Es importante la liberación de recursos, ya que en este caso la cantidad de timers (y por lo tanto los recursos que se consumen) varía de acuerdo a la actividad en la red; por ejemplo, un nodo que esté incorporado al árbol de distribución de un cierto número de grupos, deberá tener una entrada permanente por cada uno de ellos, junto con su timer asociado; al dejar de pertenecer al árbol, tanto la entrada como el timer serán eliminados.

Otro aspecto a tener en cuenta, es la utilización de timers comunes a más de un grupo; es el caso del timer para producir las PDUs ECHO_REQUEST. Bastará con uno por interfaz

CBT, pudiendo éste servir a todos los grupos que compartan en el nodo dicha interfaz parent. En la descripción, por razones de simplicidad, se considerará un timer por grupo multicast.

A continuación se describen los timers utilizados en la especificación, indicando su tiempo asociado, tal como se especifica en [7], una breve descripción de la operatroria asociada con cada uno de ellos, su objetivo y las causas que producen que la entidad CBT invoque sus métodos de activación, reactivación o desactivación.

T01: Retransmit_Join_Request_Timer (RTX_INTERVAL): Una entidad CBT que crea una entrada transitoria leaf como consecuencia de recibir un requerimiento de agregarse a un grupo por parte de un proceso local o de una entidad IGMP tiene la responsabilidad de generar reintentos de conexión al grupo a través de la retransmisión de JOIN_REQUESTs en forma periódica.

- **Objetivo**: Provocar la retransmisión del JOIN_REQUEST si no se obtiene respuesta (JOIN_ACK) en un determinado tiempo, RTX_INTERVAL.
- **Activación**: Al crear la entrada transitoria y luego cada vez que se reenvía el JOIN_REQUEST.
- **Reactivación**: No.
- **Desactivación**: Cuando se elimina la entrada transitoria, ya sea porque se recibe respuesta (JOIN_ACK) o transcurre el tiempo máximo prefijado.

T02: Delete_Leaf_Transient_Entry_Timer (JOIN_TIMEOUT): Una entidad CBT que ha creado una entrada leaf transitoria establece un tiempo máximo de espera por la respuesta del parent (JOIN_ACK). Transcurrido este tiempo, la entrada es eliminada.

- **Objetivo**: Provocar la eliminación de la entrada leaf transitoria si no se recibe respuesta (JOIN_ACK) en el tiempo preestablecido (JOIN_TIMEOUT).
- **Activación**: Al crear la entrada transitoria.
- **Reactivación**: No.
- **Desactivación**: Cuando se elimina la entrada transitoria, ya sea porque se recibe respuesta (JOIN_ACK) o transcurre el tiempo máximo prefijado.

T03: Delete_Transient_Entry_Timer (TRANSIENT_TIMEOUT): Una entidad CBT que ha creado una entrada transitoria (no leaf) establece un tiempo máximo de espera por la respuesta del parent (JOIN_ACK). Transcurrido este tiempo, la entrada es eliminada.

- **Objetivo**: Provocar la eliminación de la entrada transitoria si no se recibe respuesta (JOIN_ACK) en el tiempo preestablecido (TRANSIENT_TIMEOUT).
- **Activación**: Al crear la entrada transitoria
- **Reactivación**: No.
- **Desactivación**: Cuando se elimina la entrada transitoria, ya sea porque se recibe respuesta (JOIN_ACK) o transcurre el tiempo máximo prefijado.

T04: Cache_Deletion_Timer (CACHE_DEL_TIMER): Una entidad CBT que recibe un QUIT_NOTIFICATION por una interfaz child multicast, no la elimina inmediatamente, ya que pueden existir otros nodos child que deseen permanecer en el grupo. Para dar tiempo a que estos nodos lo soliciten (a través de un JOIN_REQUEST), espera un tiempo predeterminado, dado por este timer.

- **Objetivo**: Provocar la eliminación de la interfaz child multicast asociada si luego de un período de tiempo (CACHE_DEL_TIMER) no es recibido un JOIN_REQUEST de algún nodo child.
- **Activación**: Al recibir el primer QUIT_NOTIFICATION por una interfaz child multicast.
- **Reactivación**: No.

- **Desactivación**: Cuando se elimina la entrada child o cuando se recibe un JOIN_REQUEST por la interfaz asociada.

T05: Transmit_Join_Request_Timer (Random [0, HOLDTIME]): Una entidad CBT que recibe un QUIT_NOTIFICATION por una interfaz parent multicast y desea seguir perteneciendo al grupo, debe enviar un JOIN_REQUEST al parent para que éste no elimine la interfaz.

- **Objetivo**: Provocar el envío de un JOIN_REQUEST por la interfaz correspondiente, en un tiempo generado al azar entre 0 y HOLDTIME.
- **Activación**: Al recibir un QUIT_NOTIFICATION por la interfaz parent multicast.
- **Reactivación**: No.
- **Desactivación:** Cuando se cumple el tiempo y se envía el JOIN_REQUEST o cuando se recibe un JOIN_REQUEST (de otro nodo child) por la interfaz parent multicast.

T06: Quit_Notification_Timer (HOLDTIME): Una entidad que desea dejar de integrar el grupo, envía un QUIT_NOTIFICATION al parent. Debido a que esta PDU no tiene asentimiento, es enviada un cierto número de veces a intervalos regulares.

- **Objetivo**: Provocar la retransmisión del QUIT_NOTIFICATION a intervalos regulares (HOLDTIME).
- **Activación**: Al eliminar una entrada permanente y enviar el primer QUIT_NOTIFICATION al parent y luego al realizar cada una de las retransmisiones.
- **Reactivación**: No.
- **Desactivación**: Al concluir la cuenta de reenvíos.

T07: Echo_Request_Timer (ECHO_INTERVAL): Una entidad CBT debe comprobar periódicamente el funcionamiento normal de su parent (para cada grupo). Lo hace enviando ECHO_REQUEST a intervalos regulares (ECHO_INTERVAL); a cada ECHO REQUEST debe responder el parent con un ECHO_REPLY.

- **Objetivo**: Producir el envío de un ECHO_REQUEST.
- **Activación**: Al crear una entrada permanente y luego al enviar cada ECHO_REQUEST.
- **Reactivación**: Cuando se recibe un ECHO_REQUEST por la interfaz parent (caso multicast); de esta manera se evita saturar el vínculo con envíos innecesarios.
- **Desactivación**: Al eliminar la entrada permanente.

T08: Delete_On-tree_Entry_Timer (GROUP_EXPIRE_TIME): Una entidad CBT que ha creado una entrada permanente (on-tree) establece un tiempo máximo de espera para ECHO_REPLYs enviados por el parent (GROUP_EXPIRE_TIME). Transcurrido este tiempo, si no se ha recibido el ECHO_REPLY, la entrada es eliminada.

- **Objetivo**: Provocar la eliminación de la entrada permanente si no se registra actividad del parent en el tiempo preestablecido.
- **Activación**: Al crear la entrada permanente.
- **Reactivación**: Al recibir un ECHO_REPLY por la interfaz parent.
- **Desactivación**: Cuando se elimina la entrada permanente.

T09: Echo_Reply_Timer (random[0,HOLDTIME]): Una entidad CBT que recibe un ECHO_REQUEST por una interfaz child, debe responder en un plazo máximo (HOLDTIME) con un ECHO_REPLY para indicar a su(s) child que está activa.

- **Objetivo**: Provocar el envío de un ECHO_REPLY en una interfaz child en la que se recibió un ECHO_REQUEST, en un tiempo al azar, entre 0 y HOLDTIME después de haber recibido el ECHO_REQUEST.

- **Activación**: Al recibir un ECHO_REQUEST por una interfaz child (si está inactivo).
- **Reactivación**: No.
- **Desactivación**: Cuando se elimina la entrada child.

5.2.4 Descripción de los elementos de la FSM

A continuación se describen los estados, eventos de entrada, condiciones que conforman los predicados, acciones internas y eventos externos generados por la entidad CBT. La descripción se realiza en base a un grupo multicast.

5.2.4.1 Estados

Se considera que la entidad CBT puede estar en los siguientes estados respecto a un grupo multicast:

ES01: Estado on-tree: El nodo pertenece al árbol de distribución CBT. La entidad CBT debe responder a requerimientos de integrar el árbol provenientes de nodos adyacentes y mantener la conectividad con su parent y sus child.

ES02: Estado transitorio: La entidad CBT ha recibido un requerimiento, ya sea de una entidad local (proceso de aplicación), de una entidad IGMP o de una entidad CBT adyacente para integrar el árbol de distribución de un grupo, y como consecuencia ha iniciado y aún no completado el procedimiento de integración a un árbol de distribución. Es responsable de completar este procedimiento o abortarlo después de un cierto tiempo.

ES03: Estado sin entrada: El nodo no pertenece al árbol de distribución del grupo ni ha iniciado procedimientos para integrarlo.

5.2.4.2 Eventos de entrada

Los eventos de entrada se clasifican de la siguiente manera, según su origen:

- Eventos externos, provenientes de entidades CBT remotas, que interatcúan con la entidad local a través de PDUs CBT.
- Eventos provenientes de otras entidades en el nodo: aplicaciones en el nodo que se unen o abandonan los grupos multicast y entidades IGMP que solicitan unirse o abandonar grupos en representación de hosts de subredes locales; la interacción se produce a través de los métodos join-group y leave-group.
- Eventos internos del protocolo, producidos por los timers asociados al mismo.

5.2.4.2.1 Eventos externos provenientes de entidades CBT remotas

Este tipo de eventos ocurre como consecuencia del arribo de PDUs. Su efecto sobre la entidad que los recibe depende de condiciones tales como el estado de la entidad receptora, tipo de vínculo por el que se los recibe (capacidad multicast o no) y tipo de envío (unicast o multicast).

EV01: Recepción de JOIN_REQUEST: La ocurrencia de este evento puede tener dos significados, según se detalla a continuación:

- El nodo receptor recibe la PDU como consecuencia de que otro nodo que aún no integra el árbol de distribución del grupo desea integrarlo, siendo el receptor el próximo nodo en el camino al core (según el emisor). El nodo que lo recibe, podrá pertenecer al árbol de distribución del grupo, caso en el que responderá con un JOIN_ACK, o no, caso en el que repetirá el procedimiento realizado por el nodo anterior.
- El nodo recibe la PDU sobre una interfaz child multiacceso para el grupo, como consecuencia de que otro nodo ha detectado que un tercero desea podar el árbol, mientras que aquél desea mantenerlo.

EV02: Recepción de JOIN_ACK: Es recibido en respuesta a un JOIN_REQUEST enviado previamente. Debe ser recibido por la interfaz (CBT) por la cual se ha enviado el JOIN-REQUEST correspondiente. Pone al nodo en estado on-tree, haciendo que éste integre el árbol de distribución para el grupo.

EV03: Recepción de QUIT_NOTIFICATION: Es recibido como consecuencia de que un nodo child notifica al nodo (su parent) que desea dejar de pertenecer al árbol de distribución de un grupo.

EV04: Recepción de FLUSH_TREE: Se recibe como consecuencia de que el nodo parent desea eliminar el subárbol de distribución que de él depende. Puede ser consecuencia de que ha perdido contacto con su parent o ha recibido a su vez un FLUSH_TREE de éste.

EV05: Recepción de ECHO_REQUEST: Se recibe como consecuencia de la comprobación periódica llevada a cabo por un nodo child para confirmar la conectividad con el parent.

EV06: Recepción de ECHO_REPLY: Es la respuesta a un ECHO_REQUEST enviado previamente; el nodo parent confirma que está operativo.

5.2.4.2.2 Eventos producidos por otras entidades residentes en el nodo

Este tipo de eventos se produce como consecuencia de requerimientos realizados por dos tipos de entidades: procesos de aplicación y entidades IGMP, ambas residentes en el nodo.

Un proceso de aplicación realiza la solicitud de integrar un grupo multicast en virtud de su funcionalidad, mientras que una entidad local IGMP lo hace en función de haber recibido un requerimiento de un host en la subred local.

A efectos de ignorar detalles de implementación, se considera que la indicación a la entidad CBT se produce a través de los eventos genéricos join-group y leave-group.[35], es decir, no se hace diferencia en cuanto a los mecanismos para la interacción entre entidades representando a aplicaciones locales y entidades IGMP con el agente CBT, utilizando las primitivas join-group y leave-group para solicitar integrar o dejar de pertenecer a un grupo respectivamente.

[35] En una implementación concreta, si este evento proviene de IGMP, éste lo anunciará al protocolo de ruteo a través de las acciones "notify routing +" y "notify routing –" definidas en IGMPv2[2]. En el medioambinte de simulación, pese a que no existen aún implementaciones de IGMP, es razonable que tomen la forma de agentes y que invoquen a métodos CBT que se correspondan con los eventos join-group y leave-group, de igual manera que se implementa para las aplicaciones multicast en el nodo.

EV11: Recepción de JOIN_GROUP: Se recibe como consecuencia de que una entidad residente en el nodo (IGMP o proceso de aplicación) solicita integrar un grupo multicast. Como posible consecuencia, si el nodo no integra el árbol de distribución del grupo, se iniciará el procedimiento correspondiente, y en este caso se creará una entrada transitoria leaf.

EV12: Recepción de LEAVE-GROUP: Se recibe como consecuencia de que una entidad residente en el nodo (IGMP o proceso de aplicación) desea dejar de pertenecer a un grupo multicast. Como posible consecuencia, la entidad CBT podrá requerir que se pode el árbol de distribución del grupo.

5.2.4.2.3 Eventos locales

Estos eventos se producen como consecuencia de la expiración (timeout) de los diferentes timers controlados por la entidad CBT. Para una explicación más detallada, referirse a la sección timers.

EV21: Vencimiento de Retransmit_Join_Request_Timer (T01)

EV22: Vencimiento de Delete_Leaf_Transient_Entry_Timer (T02)

EV23: Vencimiento de Delete_Transient_Entry_Timer (T03)

EV24: Vencimiento de Cache_Deletion_Timer (T04)

EV25: Vencimiento de Transmit_Join_Request_Timer (T05)

EV26: Quit_Notification_Timer (T06)

EV27: Vencimiento de Echo_Request_Timer (T07)

EV28: Vencimiento de Delete_On-tree_Entry_Timer (T08)

EV24: Vencimiento de Echo_Reply_Timer (T09)

5.2.4.3 Acciones internas

Las acciones internas se seleccionan de manera tal que describan la operación con una granularidad que permita una descripción simple y no ambigua.

Para el caso de CBT, las acciones internas afectan a las tablas que representan el árbol de distribución, (ya sea información transitoria o permanente), el estado de los timers (activo, inactivo) y los valores de variables internas del protocolo (por ejemplo, contadores de reintentos).

Se trata, a nivel de la implementación, que los métodos de las clases que componen el protocolo CBT en el medioambiente de la simulación estén estrechamente relacionadas con estas acciones.

Las acciones internas, son las siguientes:

IA000: Crear entrada transitoria para un grupo.
IA001: Eliminar entrada transitoria para un grupo.
IA002: Agregar child en entrada transitoria.
IA003: Eliminar child en entrada transitoria.

IA010: Crear entrada permanente (on-tree) para un grupo.
IA011: Eliminar entrada permanente (on-tree) para un grupo.
IA012: Agregar child en entrada permanente (on-tree).
IA013: Eliminar child en entrada permanente (on-tree).

IA020: Activar Retransmit_Join_Request_Timer (T01)
IA021: Desactivar Retransmit_Join_Request_Timer (T01)
IA022: Reactivar Retransmit_Join_Request_Timer (T01)

IA030: Activar Delete_Leaf_Transient_Entry_Timer (T02)
IA031: Desactivar Delete_Leaf_Transient_Entry_Timer (T02)
IA032: Reactivar Delete_Leaf_Transient_Entry_Timer (T02)

IA040: Activar Delete_Transient_Entry_Timer (T03)
IA041: Desactivar Delete_Transient_Entry_Timer (T03)
IA042: Reactivar Delete_Transient_Entry_Timer (T03)

IA050: Activar Cache_Deletion_Timer (T04).
IA051: Desactivar Cache_Deletion_Timer (T04).
IA052: Reactivar Cache_Deletion_Timer (T04).

IA060: Activar Transmit_Join_Request_Timer (T05).
IA061: Desactivar Transmit_Join_Request_Timer (T05)
IA062: Reactivar Transmit_Join_Request_Timer (T05)

IA070: Activar Quit_Notification_Timer (T06)
IA071: Desactivar Quit_Notification_Timer (T06).
IA072: Reactivar Quit_Notification_Timer (T06).

IA080: Activar Echo_Request_Timer (T07).
IA081: Desactivar Echo_Request_Timer (T07).
IA082: Reactivar Echo_Request_Timer (T07)

IA090: Activar Delete_On-tree_Entry_Timer (T08)
IA091: Desactivar Delete_On-tree_Timer (T08).
IA092: Reactivar Delete_On-tree_Timer (T08).

IA100: Activar Echo_Reply_Timer (T09).
IA101: Desactivar Echo_Reply_Timer (T09).
IA102: Reactivar Echo_Reply_Timer (T09).

IA200: Variable RTX = RTX + 1
IA201: Variable RTX = 0

5.2.4.4 Eventos de salida (OE)

Los eventos de salida consisten en el envío de PDUs CBT a entidades pares CBT residentes en otros nodos.

Los eventos de salida, son los siguientes:

OE01: Enviar JOIN_ACK
OE02: Enviar JOIN_REQUEST

OE03: Enviar QUIT_NOTIFICATION
OE04: Enviar FLUSH_TREE
OE05: Enviar ECHO_REQUEST
OE06: Enviar ECHO_REPLY

5.2.4.5 Predicados

Un predicado representa la evaluación de ciertas condiciones acerca del estado de variables internas y otras tales como el tipo de interfaz a través de la cual arriba una PDU.

Según sea el valor del predicado verdadero o falso, se ejecuta o no el conjunto de acciones (3-upla) asociado a él.

El conjunto de predicados especificado para un par (estado, evento) debe cumplir con las condiciones siguientes:

- Sólo uno de los predicados puede tener valor verdadero.
- El conjunto de los predicados debe cubrir todos los posibles casos.

En la presente especificación se asume que si ninguno de los predicados que figuran explícitamente tiene valor verdadero, el evento es igonrado, es decir IA = OE = ∅ (conjunto vacío) y NS = estado actual.

El objeto de los predicados es evitar la explosión de estados del autómata que tendría lugar si no se considerara el uso de variables internas. Las condiciones que componen los predicados, se han seleccionado de manera que la especificación resulte clara y no dependa de detalles de implementación.

5.2.4.6 Condiciones

Las condiciones que conforman un predicado se refieren a sentencias con valores verdadero o falso que representan por ejemplo estados de contadores, tipo de un vínculo, estados de una interfaz o del nodo (DR, no DR), etc. En una implementación, no es necesaria la existencia de variables específicas que representen dichos valores.

C01: Interfaz de arribo de la PDU es parent (transitoria o permanente) para el grupo.
C02: Interfaz de arribo de la PDU es child (transitoria o permanente) para el grupo.
C03: Entrada transitoria es leaf.
C04: Vínculo de arribo de la PDU es multiacceso.
C05: Vínculo de arribo de la PDU es multicast.
C06: Interfaz de arribo es la única child para el grupo.
C07: Existe interfaz correcta a nodo upstream en dirección al core del grupo.
C08: Vínculo asociado a la interfaz parent del grupo (permanente o transitoria) es multiacceso.
C09: Vínculo asociado a la interfaz parent del grupo (permanente o transitoria) es multicast.
C10: Nodo es DR en el vínculo asociado a la interfaz.
C11: Red asociada a la interfaz parent es diferente de red asociada a la interfaz de arribo.
C12: Interfaz a nodo upstream es diferente de interfaz de arribo.
C13: La PDU recibida lleva dirección multicast.
C14: Contador de reintentos no llegó al límite (RTX < MAX_RTX).

	ON TREE (ES01)	TRANSITORIO (ES02)	SIN ENTRADA (ES03)
(EV01) Recepción JOIN_REQUEST	P01 P06 P02 P07 P03 P08 P04 P09 P05 P10	P11 P12 P13 P14 P15	P16 P17 P18 P19 P20
(EV02) Recepción JOIN_ACK	IGNORAR	P21 P22	IGNORAR
(EV03) Recepción QUIT_NOTIFICATION	P23 P24 P25 P26	IGNORAR	IGNORAR
(EV04) Recepción FLUSH_TREE	P27 P28	IGNORAR	IGNORAR
(EV05) Recepción ECHO_REQUEST	P29 P30	IGNORAR	IGNORAR
(EV06) Recepción ECHO_REPLY	P31	IGNORAR	IGNORAR
(EV10) Recepción JOIN-GROUP	P32	P33	P34
(EV11) Recepción LEAVE-GROUP	P35 P36 P37	IGNORAR	IGNORAR

Tabla 5.1. Tabla de estados/eventos para eventos externos *(provenientes de otros nodos o de entidades en el nodo) a la entidad CBT*

	ON TREE (ES01)	TRANSITORIO (ES02)	SIN ENTRADA (ES03)
(EV21) Retransmit_Join_Request_Timer (T01) timeout	IGNORAR	P38	IGNORAR
(EV22) Delete_Leaf_Transient_Entry_Timer (T02) timeout	IGNORAR	P39	IGNORAR
(EV23) Delete_Transient_Entry_Timer (T03) timeout	IGNORAR	P40	IGNORAR
(EV24) Cache_Deletion_Timer (T04) timeout	P41 P42	IGNORAR	IGNORAR
(EV25) Transmit_Join_Request_Timer (T05) timeout	P43	IGNORAR	IGNORAR
(EV26) Quit_Notification_Timer (T06) timeout	IGNORAR	IGNORAR	P44 P45
(EV27) Echo_Request_Timer (T07) timeout	P46	IGNORAR	IGNORAR
(EV28) Delete_Ontree_Timer (T08) timeout	P47 P48 P49	IGNORAR	IGNORAR
(EV29) Echo_Reply_Timer (T09) timeout	P50	IGNORAR	IGNORAR

Tabla 5.2. Tabla de estados/eventos para eventos internos *(provenientes de timers) de la entidad CBT*

5.2.4.7 Tablas evento/estado

Las tablas 5.1 y 5.2 muestran los posibles estados de la entidad CBT y los posibles eventos de entrada. De acuerdo con lo descripto al principio del capítulo, en cada caso se indica una referencia a los posibles predicados (compuestos por operaciones lógicas entre las condiciones ya enunciadas) y las acciones internas, externas y nuevo estado de la entidad.

La notación utilizada está compuesta de una referencia al casillero de la tabla, operación lógica entre los valores verdadero o falso de determinadas condiciones, seguida de la 3-upla que contiene referencias a acciones internas, eventos externos y nuevo estado. En caso de realizarse alguna acción para la totalidad de las entradas child, se lo indica con *C: seguido por el conjunto de acciones internas a realizar o eventos a producir.

Por ejemplo:

P50: C20 & C21 {*C:(IA050, IA051), ,(ES02)}

Indica referencia a P50; su predicado está dado por el valor de C20 y C21, y en caso de que esta operación produzca un resultado verdadero, se realiza, para cada una de las interfaces child del grupo, las acciones IA050 y IA051; no se producen eventos externos, y el nuevo estado es ES02. En el caso de una 3-upla que sea válida para cualquier conjunto de valores, se indica su predicado con el valor verdadero, V.

A continuación se indican las condiciones a evaluar, acciones internas y eventos de salida para cada una de las referencias de las tablas 5.1 y 5.2, junto con una explicación breve relativa al evento producido.

Recepción de JOIN_REQUEST en estado on-tree

P01: !C01 & !C02 & !C04 {(IA012), (OE01), (ES01)}
La PDU es recibida por una interfaz que aún no es parent ni child para el grupo (!C01 & !C02); el vínculo asociado no tiene capacidad multiacceso (!C04). En este caso se agrega la interfaz de arribo como child para el grupo (IA012). El evento a generar es un JOIN_ACK (OE01) direccionado según la interfaz de arribo. No hay cambio de estado.

P02: !C01 & !C02 & C04 & !C05 & C10 {(IA012), (OE01), (ES01)}
La PDU es recibida por una interfaz que aún no es parent ni child para el grupo (!C01 & !C02); el vínculo asociado tiene capacidad multiacceso (C04) no multicast (!C05) y el nodo es DR en el vínculo (C10). En este caso se agrega la interfaz de arribo como child para el grupo (IA012). El evento a generar consiste en la emisión de un JOIN_ACK direccionado según la interfaz de arribo (OE01). No se produce cambio de estado.

P03: !C01 & !C02 & C04 & !C05 & !C10 & C11 {(IA012), (OE01), (ES01)}
La PDU es recibida por una interfaz que aún no es parent ni child para el grupo (!C01 & !C02); el vínculo asociado es multiacceso (C04) no multicast (!C05) , el nodo no es DR en el vínculo (!C10), y la red a la que pertenece el nodo parent es diferente a la red de arribo (C11) (esta comprobación es necesaria para evitar ciclos en el árbol de distribución). En este caso se agrega la interfaz de arribo como child para el grupo (IA012). El evento externo consiste en el envío de un JOIN_ACK direccionado según la interfaz de arribo (OE01). No se produce cambio de estado.

P04: !C01 & !C02 & C04 & C05 & C10 & C13 {(IA102), (OE01), (ES01)}
La PDU es recibida por una interfaz que aún no es parent ni child para el grupo (!C01 & !C02); el vínculo asociado tiene capacidad multiacceso (C04) multicast (C05); y el nodo es DR en el vínculo (C10). La PDU recibida lleva dirección multicast (C13). En este caso se agrega la interfaz de arribo como child para el grupo (IA012). El evento a generar es el envío de un JOIN_ACK direccionado según la interfaz de arribo (OE01). No se produce cambio de estado.

P05: !C01 & !C02 & C04 & C05 & !C10 & !C13 & C11 {(IA102), (OE01), (ES01)}
La PDU es recibida por una interfaz que aún no es parent ni child para el grupo (!C01 & !C02); el vínculo asociado tiene capacidad multiacceso (C04) multicast (C05); y el nodo no es DR en el vínculo (!C10). La PDU recibida lleva dirección unicast (!C13). La red a la que pertenece el nodo parent es diferente a la red de arribo (C11). En este caso se agrega la interfaz de arribo como child para el grupo (IA012). El evento a generar es el envío de un JOIN_ACK direccionado según la interfaz de arribo (OE01). No se produce cambio de estado.

P06: C01 & C04 & C05 & !C10 {(IA061), , (ES01)}
La PDU es recibida por la interfaz parent para el grupo (C01), el vínculo asociado es multiacceso (C04) multicast (C05) y el nodo no es DR en el vínculo (!C10). En este caso, se debe desactivar, si está activo, el timer para envío de JOIN_REQUEST (IA061) (T05). No se generan eventos externos. No hay cambio de estado.

P07: C02 & !C04 { , (OE01), (ES01)}
La PDU es recibida por una interfaz child para el grupo (C02); el vínculo asociado no posee característica multiacceso (!C04). En este caso no se producen acciones internas. Se genera un evento externo consistente en el envío de un JOIN_ACK direccionado según la interfaz de arribo (OE01). No hay cambio de estado.

P08: C02 & C04 & !C05 { , (OE01), (ES01)}
La PDU es recibida por una interfaz child para el grupo (C02); el vínculo asociado es multiacceso (C04) no multicast (!C05). En este caso no se realizan acciones internas. Se genera un evento externo consistente en el envío de un JOIN_ACK según la interfaz de arribo (OE01). No se cambia de estado.

P09: C02 & C04 & C05 & C10 {(IA051), (OE01), (ES01)}
La PDU es recibida por una interfaz child para el grupo (C02); el vínculo asociado es multiacceso (C04) multicast (C05) y el nodo es DR en el vínculo (C10). En este caso se desactiva el timer para borrado de la interfaz child (IA051) (T04) si es que estaba activo. Como evento externo, se envía un JOIN_ACK direccionado según la interfaz de arribo (OE01). No se produce cambio de estado.

P10: C02 & C04 & C05 & !C10 { , (OE01), (ES01)}
La PDU es recibida por una interfaz child para el grupo (C02); el vínculo asociado es multiacceso (C04) multicast (C05) y el nodo no es DR en el vínculo (!C10). En este caso no se realizan acciones internas. Como evento externo, se envía un JOIN_ACK direccionado según la interfaz de arribo (OE01). No se produce cambio de estado.

Recepción de JOIN_REQUEST en estado transitorio

P11: !C01 & !C02 & !C04 {(IA002), , (ES02)}
La PDU es recibida por una interfaz que aún no es parent (!C01) ni child (!C02) para el grupo y el vínculo asociado no tiene capacidad multiacceso (!C04). En este caso se agrega la interfaz de arribo como child para el grupo transitorio (IA002). No se producen eventos externos. No se cambia de estado.

P12: !C01 & !C02 & C04 & !C05 & C10 {(IA002), , (ES02)}
La PDU es recibida por una interfaz que aún no es parent (!C01) ni child (!C02) para el grupo, el vínculo asociado es multiacceso (C04) no multicast (!C05) y el nodo es DR en el vínculo (C10). En este caso se agrega la interfaz de arribo como child para el grupo transitorio (IA002). No se producen eventos externos. No se cambia de estado.

P13: !C01 & !C02 & C04 & !C05 & !C10 & C11 {(IA002), ,(ES02)}
La PDU es recibida por una interfaz que aún no es parent (!C01) ni child (!C02) para el grupo, el vínculo asociado es multiacceso (C04) no multicast (!C05), el nodo no es DR en el vínculo (!C10) y la red a la que pertenece el nodo parent es diferente a la red de arribo de la PDU (C11). En este caso se agrega la interfaz de arribo como child para el grupo transitorio (IA002). No se producen eventos externos. No se cambia de estado.

P14: !C01 & !C02 & C04 & C05 & C10 & C13 {(IA002), , (ES02)}
La PDU es recibida por una interfaz que aún no es parent (!C01) ni child (!C02) para el grupo, el vínculo asociado es multiacceso (C04) multicast (C05); el nodo es DR en dicho vínculo (C10) y la PDU ha sido enviada con dirección multicast (C13). En este caso se agrega la interfaz de arribo como child para el grupo transitorio (IA002). No se producen eventos externos. No se cambia de estado.

P15: !C01 & !C02 & C04 & C05 & !C10 & !C13 & C11 {(IA002), , (ES02)}
La PDU es recibida por una interfaz que aún no es parent (!C01) ni child (!C02) para el grupo, el vínculo asociado es multiacceso (C04) multicast (C05); el nodo no es DR en dicho vínculo (!C10) y la PDU ha sido enviada con dirección unicast (!C13). La red asociada a la interfaz parent es diferente de la red asociada a la interfaz de arribo de la PDU (C11). En este caso se agrega la interfaz de arribo como child para el grupo transitorio (IA002). No se producen eventos externos. No se cambia de estado.

Recepción de JOIN_REQUEST en estado sin entrada

P16: !C04 & C07 & C12 {(IA000, IA040, IA002, IA071), (OE02), (ES02)}
El vínculo por el que se recibe la PDU no tiene capacidad multiacceso (!C04). Existe una interfaz en dirección al core del grupo (C07), y la interfaz al nodo upstream es diferente a la interfaz de arribo (C12). En este caso se crea una entrada transitoria para el grupo, no leaf, (IA000), se activa el timer para eliminación de la entrada (IA040) (T03), y se agrega la interfaz de arribo como child para el grupo transitorio (IA002). Se desactiva el timer de envío de QUIT_NOTIFICATION asociado a la interfaz parent y al grupo, si está activo (IA071) (T06). Como evento externo, se genera un JOIN_REQUEST direccionado a la interfaz parent transitoria para el grupo. Se pasa a estado transitorio (ES02).

P17: C04 & !C05 & C10 & C07 & C12 {(IA000, IA040, IA002, IA071), (OE02), (ES02)}
El vínculo por el que se recibe la PDU tiene capacidad multiacceso (C04) no multicast (!C05). El nodo es DR en el vínculo (C10). Existe una interfaz en dirección al core del grupo (C07), y la interfaz al nodo upstream es diferente de la interfaz de arribo (C12). En este caso se crea una entrada transitoria para el grupo, no leaf, (IA000), se activa el timer para borrado de la entrada (IA040) (T03), y se agrega la interfaz de arribo como child para el grupo transitorio (IA002). Se desactiva el timer de envío de QUIT_NOTIFICATION asociado a la interfaz parent y al grupo, si está activo (IA071) (T06). Como evento externo, se genera un JOIN_REQUEST direccionado a la interfaz parent transitoria para el grupo. Se pasa a estado transitorio (ES02).

P18: C04 & !C05 & !C10 & C07 & C11 {(IA000, IA040, IA002, IA071), (OE02), (ES02)}
El vínculo por el que se recibe la PDU tiene capacidad multiacceso (C04) no multicast (!C05). El nodo no es DR en el vínculo (!C10). Existe una interfaz en dirección al core del grupo (C07), y la red asociada a la interfaz hacia el nodo upstream es diferente de la red asociada a la interfaz de arribo (C11). En este caso se crea una entrada transitoria para el grupo, no leaf, (IA000), se activa el timer para borrado de la entrada (IA040) (T03), y se agrega la interfaz de arribo como child para el grupo transitorio (IA002). Se desactiva el timer de envío de QUIT_NOTIFICATION asociado a la interfaz parent y al grupo, si está activo (IA071) (T06). Como evento externo, se genera un JOIN_REQUEST direccionado a la interfaz parent transitoria para el grupo. Se pasa a estado transitorio (ES02).

P19: C04 & C05 & C10 & C13 & C07 {(IA000, IA040, IA002, IA071), (OE02), (ES02)}
El vínculo por el que se recibe la PDU tiene capacidad multiacceso (C04) multicast (C05). El nodo es DR en el vínculo (C10) y la PDU ha sido enviada multicast (C13). Existe una interfaz en dirección al core del grupo (C07). En este caso se crea una entrada transitoria para el grupo, no leaf, (IA000), se activa el timer para borrado de la entrada (IA040) (T03), y se agrega la interfaz de arribo como child para el grupo transitorio (IA002). Se desactiva el timer de envío de QUIT_NOTIFICATION asociado a la interfaz parent y al grupo, si está activo (IA071) (T06). Como evento externo, se genera un JOIN_REQUEST direccionado a la interfaz parent transitoria para el grupo. Se pasa a estado transitorio (ES02).

P20: C04 & C05 & !C10 & !C13 & C07 & C12 {(IA000, IA040, IA002, IA071), (OE02), (ES02)}
El vínculo por el que se recibe la PDU tiene capacidad multiacceso (C04) multicast (C05). El nodo no es DR en el vínculo (!C10) y la PDU ha sido enviada unicast (!C13). Existe una interfaz en dirección al core del grupo (C07) y la red asociada a ella (parent) es diferente a la red asociada al vínculo de arribo (C12). En este caso se crea una entrada transitoria para el grupo, no leaf, (IA000), se activa el timer para borrado de la entrada (IA040) (T03), y se agrega la interfaz de arribo como child para el grupo transitorio (IA002). Se desactiva el timer de envío de QUIT_NOTIFICATION asociado a la interfaz parent y al grupo, si está activo (IA071) (T06). Como evento externo, se genera un JOIN_REQUEST direccionado a la interfaz parent transitoria para el grupo. Se pasa a estado transitorio (ES02).

Recepción de JOIN_ACK en estado transitorio

P21: C01 & C03 {(IA010, IA090, IA080, IA031, IA021, *C:(IA001, IA012)),(*C: OE01), (ES01)}

La PDU es recibida por la interfaz parent (transitoria) para el grupo (C01), y la entrada es leaf (C03); se crea la entrada on-tree para el grupo (IA010) activando el timer de borrado de la entrada (IA090) (T08) y el timer para realizar el echo request al parent (IA080) (T07). Cada interfaz child transitoria, es eliminada de la entrada transitoria (IA001) y agregada en la entrada on-tree (IA012). Finalmente, se desactivan los timers para la entrada transitoria leaf: el de expiración de la entrada (IA031) (T02) y el utilizado para la retransmisión de JOIN_REQUESTs (IA021) (T01). Como evento externo, se genera un JOIN_ACK por cada interfaz child (no local). Se pasa a estado on-tree (ES01).

P22: C01 & !C03 {(IA010, IA090, IA080, IA041, *C: (IA001, IA012)), (*C:OE01), (ES01)}
La PDU es recibida por la interfaz parent (transitoria) para el grupo (C01), y la entrada transitoria no es leaf (!C03). Se crea la entrada on-tree para el grupo (IA010) activando el timer de borrado de la entrada (IA090) (T08) y el timer para realizar el echo request al parent (IA080) (T07). Cada interfaz child transitoria, es eliminada de la entrada transitoria (IA001) y agregada en la entrada on-tree (IA012). Finalmente se desactiva el timer para borrado de la entrada no leaf (IA041) (T03). Como evento externo, se genera un JOIN_ACK por cada interfaz child (no local). Se pasa a estado on-tree (ES01).

Recepción de QUIT_NOTIFICATION en estado on-tree

P23: C02 & C04 & C05 & C06 {(IA050), ,(ES01)}
La PDU es recibida a través de una interfaz child para el grupo (C02) cuyo vínculo asociado es multiacceso (C04) con capacidad multicast (C05). La interfaz de arribo es la única child (C06). Se activa el timer para borrado de la entrada (IA050) (T04) (si ningún otro nodo child solicita permanecer en el grupo enviando un JOIN_REQUEST, la entrada será eliminada). No se generan eventos externos ni se cambia de estado.

P24: C02 & !(C04 & C05) & !C06 {(IA013), ,(ES01)}
La PDU es recibida por una interfaz child para el grupo (C02) cuyo vínculo asociado no cumple con la condición de multiacceso con capacidad multicast (!(C04 & C05)). La interfaz no es la única child para el grupo (!C06). En este caso, la interfaz child es eliminada (IA013). No se generan eventos externos ni se cambia de estado.

P25: C02 & !(C04 & C05) & C06 {(IA013, IA011, IA091, IA081, IA070, IA201), (OE03), (ES03)}
La PDU es recibida a través de una interfaz child (C02), cuyo vínculo asociado no es multiacceso con capacidad multicast (!(C04 & C05)); esta interfaz es la única child para el grupo (C06). En este caso, debe eliminarse la entrada child (IA013), la entrada (on-tree) para el grupo (IA011), desactivar el timer de borrado de la entrada on-tree (IA091) (T08), y el timer para solicitar eco (IA081) (T07). Además se llevan a cabo acciones relativas al envío del QUIT_NOTIFICATION al nodo parent: activación del timer para reenvío del QUIT_NOTIFICATION (IA070) (T06) e inicialización del contador de reintentos (IA201). Como evento externo, se genera el primer QUIT_NOTIFICATION direccionado al parent del grupo (OE03). Se pasa a estado sin entrada (ES03).

P26: C01 & C04 & C05 {(IA060), ,(ES01)}
La PDU es recibida a través de la interfaz parent para el grupo (C01). Su vínculo asociado tiene capacidad multiacceso (C04) multicast (C05). Esto significa que algún otro nodo child solicita

la desconexión del grupo; en este caso, se activa el timer de envío de JOIN_REQUEST (IA060) (T05) para evitar el podado del árbol. No se generan eventos externos ni se cambia de estado.

Recepción de FLUSH_TREE en estado on-tree

P27: C01 & !(C04 & C05) {((*C: IA013), IA011, IA081, IA091), ((*C: OE04)), (ES03)}
La PDU es recibida por la interfaz parent para el grupo (C01); el vínculo asociado no presenta la característica multiacceso multicast (!(C04 & C05)). En este caso, se eliminan las entradas child para el grupo (IA013), se elimina la entrada para el grupo (IA011), se desactiva el timer para realizar el ECHO_REQUEST (IA081) (T07) y el timer para eliminación de la entrada on-tree (IA091) (T08). Como evento externo se genera un FLUSH_TREE por cada interfaz child (OE04). Se pasa a estado sin entrada (ES03).

P28: C01 & C04 & C05 {((*C: IA013), IA011, IA081, IA091, IA061), ((*C:OE04)), (ES03)}
La PDU es recibida por la interfaz parent para el grupo (C01); el vínculo asociado posee capacidad multiacceso (C04) multicast (C05). En este caso, se eliminan las entradas child para el grupo (IA013), se elimina la entrada para el grupo (IA011), se desactiva el timer para realizar el ECHO_REQUEST (IA081) (T07), el timer para eliminación de la entrada on-tree (IA091) (T08), y el timer para envío de JOIN_REQUEST, en el caso de que estuviera activo (IA061) (T05). Como evento externo se genera un FLUSH_TREE por cada interfaz child. Se pasa a estado sin entrada (ES03).

Recepción de ECHO_REQUEST en estado on-tree

P29: C02 {(IA100), , (ES01)}
La PDU es recibida por una interfaz child (C02). En caso de que el timer para envío del ECHO_REPLY correspondiente esté inactivo, es activado (IA100) (T09). No se generan eventos externos ni se cambia de estado.

P30: C01 & C04 & C05 {(IA081), , (ES01)}
La PDU es recibida por la interfaz parent para el grupo (C01) y su vínculo asociado presenta la característica multiacceso (C04) multicast (C05). En este caso, se desactiva el timer para envío de ECHO_REQUEST al parent (IA081) (T07) si estaba activo. No se generan eventos externos ni se cambia de estado.

Recepción de ECHO_REPLY en estado on-tree

P31: C01 {(IA092), , (ES01)}
La PDU es recibida por la interfaz parent para el grupo (C01). En este caso se reactiva el timer de eliminación de la entrada (IA092) (T08), cargándolo con su valor original. No se generan eventos externos ni se cambia de estado.

Recepción de una solicitud JOIN-GROUP en estado on-tree

P32: V {(IA012), , (ES01)}
En este caso se agrega la interfaz (local) como child para el grupo (IA012). No se generan eventos externos ni se cambia de estado.

Recepción de una solicitud JOIN-GROUP en estado transitorio

P33: V {(IA002), , (ES01)}
En este caso se agrega la interfaz (local) como child en la entrada transitoria para el grupo (IA002). No se generan eventos externos ni se cambia de estado.

Recepción de una solicitud JOIN-GROUP en estado sin entrada

P34: C07 {(IA000, IA020, IA030, IA002), (OE02), (ES02)}
Si existe una interfaz en dirección al core del grupo (posiblemente a través de nodos intermedios) (C07), se crea una entrada transitoria, leaf (IA000), y se activan los timers correspondientes: para reenvío de JOIN_REQUESTs (IA020) (T01) y para eliminación de la entrada transitoria (IA030) (T02). Luego se agrega la interfaz local como child para el grupo (IA002). El evento externo consiste en generar un JOIN_REQUEST direccionado al parent (transitorio) del grupo. Se pasa a estado transitorio.

Recepción de una solicitud LEAVE-GROUP en estado on-tree

P35: !C06 {(IA013), , (ES01)}
Si la interfaz local no es la única interfaz child para el grupo (!C06), se elimina la interfaz child (IA013). No se generan eventos externos ni se cambia de estado.

P36: C06 & !(C08 & C09) {(IA013, IA011, IA081, IA091, IA070), (OE03), (ES03)}
Si la interfaz local es la única child para el grupo (C06) y el vínculo asociado a la interfaz parent para el grupo no tiene la característica multiacceso multicast (!(C08 & C09)), se elimina la interfaz child (IA013), se elimina la entrada para el grupo (IA011), se desactiva el timer para envío de ECHO_REQUEST al parent (IA081) (T07), se desactiva el timer para eliminación de la entrada permanente (IA091) (T08) y se activa el timer para reenvío de QUIT_NOTIFICATION al parent (IA070) (T06). Como evento externo, se genera un QUIT_NOTIFICATION al parent (OE03). Se pasa a estado sin entrada.

P37: C06 & C08 & C09 {(IA013, IA011, IA081, IA091, IA070, IA061), (OE03), (ES03)}
Si la interfaz local es la única child para el grupo (C06) y la interfaz parent es multiacceso (C08) multicast (C09), se elimina la interfaz child (IA013), se elimina la entrada para el grupo (IA011), se desactiva el timer para envío de ECHO_REQUEST al parent (IA081) (T07), se desactiva el timer para la eliminación de la entrada permanente (IA091) (T08) y se activa el timer para reenvío de QUIT_NOTIFICATION al parent (IA070) (T06). En caso de que el timer para envío de JOIN_REQUEST (T05) esté activo, se lo desactiva (IA061). Como evento externo, se genera el primer QUIT_NOTIFICATION al parent (OE03). Se pasa a estado sin entrada.

Retransmit_Join_Request_Timer (T01): timeout en estado transitorio

P38: V {(IA020), (OE02), (ES02)}
Este evento producirá el reenvío del JOIN_REQUEST sobre la interfaz parent transitoria para el grupo (OE02) y la activación del timer (IA020) (T01). Sólo se produce en entradas leaf.

Delete_Leaf_Transient_Entry_Timer (T02): timeout en estado transitorio

P39: V { ((*C:IA003), IA001, IA021) , , (ES03)}
Este evento, producido si después de un cierto tiempo de enviado el primer JOIN_REQUEST no se obtiene respuesta del supuesto parent, producirá la eliminación de las entradas child transitorias (IA003), la eliminación de la entrada transitoria leaf (IA001) y la desactivación del timer para retransmisiones de JOIN_REQUEST (IA021) (T01). No se generan eventos externos. El nuevo estado es sin entrada.

Delete_Transient_Entry_Timer (T03): timeout en estado transitorio

P40: V { ((*C:IA003), IA001), , (ES03)}
Este evento, producido si luego de un cierto tiempo de enviado el JOIN_REQUEST no se obtiene respuesta del supuesto parent, producirá la eliminación de las entradas child transitorias (IA003), de la entrada transitoria no leaf (IA001). No se generan eventos externos. El nuevo estado es sin entrada.

Cache_Deletion_Timer (T04): timeout en estado on-tree

P41: !C06 {(IA013), (OE04), (ES01)}
Este evento sólo puede producirse relativo a una interfaz child asociada a un vínculo multiacceso multicast. Si la interfaz child correspondiente al timer que produce este evento no es la única child para el grupo (!C06), se elimina dicha entrada child (IA013). Se genera un FLUSH_TREE (OE04) sobre la interfaz child. No se cambia de estado.

P42: C06 {(IA013, IA081, IA091, IA011, IA070), (OE04, OE04), (ES03)
Este evento sólo puede producirse relativo a una interfaz child asociada a un vínculo multiacceso multicast Si la interfaz child correspondiente al timer que produce el evento es la única child para el grupo (C06), se elimina dicha interfaz child (IA013), y luego se desactiva el timer para solicitud de eco correspondiente a la entrada para el grupo (IA081) (T07), y el timer para eliminación de la entrada (IA091) (T08); se elimina la entrada on-tree (IA011), y se activa el timer para enviar QUIT_NOTIFICATION al parent (IA070) (T06). Como eventos externos, se envía un QUIT_NOTIFICATION (el primero) al parent (OE03), y un FLUSH_TREE (OE04) direccionado a la interfaz child. Se pasa a estado sin entrada (ES03).

Transmit_Join_Request_Timer (T05): timeout en estado on-tree

P43: V { , (OE02), (ES01)}

Este evento indica que deben tomarse acciones para evitar que el parent elimine la entrada multicast. No hay acción interna. Se genera un evento externo, consistente en el envío de un JOIN_REQUEST (OE02) al parent. No se cambia de estado.

Quit_Notification_Timer (T06): timeout en estado sin entrada

P44: C14 {(IA200, IA070), (OE03), (ES03)}
Si la variable de conteo de reenvíos es menor que el límite (C14), se aumenta en uno dicha variable (IA200) y se activa el timer para producir el próximo envío (T06) (IA070). Se genera un evento externo consistente en el envío de un QUIT_NOTIFICATION (OE03) al parent.

P45: !C14 { , (OE03), (ES03)}
Si la variable de conteo de reenvíos es igual o mayor que el límite (!C14), no se realizan acciones internas. Se envía un QUIT_NOTIFICATION (OE03) (el último) al parent. No se cambia de estado.

Echo_Request_Timer (T07): timeout en estado on-tree

P46: V {(IA080), (OE05), (ES01)}
Al producirse este evento, se activa nuevamente el timer (T07) (IA080). Se genera un ECHO_REQUEST (OE05) y se envía al parent. No se produce cambio de estado.

Delete_on-tree_Timer (T08): timeout en estado on-tree

P47: !(C04 & C05) {(*C:(IA013, IA081), IA011, IA070), (*C:(OE04), OE03), (ES03)}
Si el vínculo asociado a la interfaz parent del grupo no tiene capacidad multiacceso multicast (!(C04 & C05), se eliminan todas las entradas child (IA013), y se desactiva su timer para ECHO_REQUEST (T07) (IA081). Por último se elimina la entrada para el grupo (IA011) y se activa el timer para enviar QUIT_NOTIFICATION al parent (IA070) (T06). Como eventos externos, se envía un QUIT_NOTIFICATION (el primero) al parent (OE03), y un FLUSH_TREE (OE04) direccionado a cada interfaz child. Se pasa a estado sin entrada (ES03).

P48: C04 & C05 & C10 {(*C:(IA013, IA081), IA011, IA070), (*C:(OE04), OE03), (ES03)}
Si el vínculo asociado a la interfaz parent del grupo tiene capacidad multiacceso (C04) multicast (C05) y el nodo es DR en el vínculo (C10) se eliminan todas las entradas child (IA013), y se desactiva su timer para ECHO_REQUEST (T07) (IA081). Por último se elimina la entrada para el grupo (IA011) y se activa el timer para enviar QUIT_NOTIFICATION al parent (IA070) (T06). Como eventos externos, se envía un QUIT_NOTIFICATION (el primero) al parent (OE03), y un FLUSH_TREE (OE04) direccionado a cada interfaz child. Se pasa a estado sin entrada (ES03).

P49: C04 & C05 & !C10 {(*C:(IA013, IA081, IA051), IA011, IA070), (*C:(OE04), OE03), (ES03)}
Si el vínculo asociado a la interfaz parent del grupo tiene capacidad multiacceso (C04) multicast (C05) y el nodo no es DR en el vínculo (!C10), se eliminan las entradas child para el grupo (IA013), se desactiva el timer para ECHO_REQUEST (T07) (IA081) y el timer para reenvío de

JOIN_REQUEST (T05) (IA051) si estaba activo. Por último se elimina la entrada para el grupo (IA011) y se activa el timer para enviar QUIT_NOTIFICATION al parent (IA070) (T06). Como eventos externos, se envía un QUIT_NOTIFICATION (el primero) al parent (OE03), y un FLUSH_TREE (OE04) direccionado a la interfaz child. Se pasa a estado sin entrada (ES03).

Echo_Reply_Timer (T09): timeout en estado on-tree

P50: V {, (OE06), (ES01)}
Este evento ocurrirá asociado a la interfaz parent para el grupo. No se producen acciones internas. Como evento de salida, se envía un ECHO_REPLY (OE06) por la interfaz parent. No se cambia de estado.

5.3 Resumen

En este capítulo se presentó una descripción formal de la operación del protocolo CBT versión 2 en forma de máquina de estados finitos. La especificación realizada tiene por objeto ser utilizada como base para desarrollar la implementación, en este caso en el ambiente de simulación Ns. No se intenta en modo alguno la derivación del código en forma automática a partir de la especificación realizada, ni su utilización en aspectos tales como comprobación de la correctitud del protocolo, para lo cual sería necesario su especificación en un lenguaje apropiado, como por ejemplo SDL.

Capítulo 6

Implementación

La implementación del protocolo comprende la totalidad de su funcionalidad excepto el envío desde un nodo no miembro del grupo y el mecanismo para la elección del "Designated Router"; dicha configuración se realiza explícitamente desde el script de simulación del usuario. Se agregan facilidades de configuración de cores y de monitoreo de las acciones de la entidad CBT, que se detallan más adelante.

Un aspecto al que se dió importancia fué el de la portabilidad del código. Por ello se decidió implementar la totalidad del protocolo en forma de objetos interpretados, en Otcl, sin utilizar C++. Esta característica permite incorporar el código implementado al simulador sin necesidad de recompilación.

Debido a su total implementación en Otcl, se descartó la creación de nuevas clases compiladas (C++), adaptando en cambio el código Otcl desarrollado, en los dos casos siguientes:

- la creación de una clase derivada de *Agent/Message* (C++) que hiciera accesible al agente CBT el rótulo (label) de la interfaz de llegada de una PDU. Para obtener el label de la interfaz de llegada, se incluyó la identificación del nodo que envía la PDU como un campo adicional de la misma.
- se decidió utilizar el soporte provisto actualmente por Ns para el reenvío de paquetes multicast, orientado al paradigma de árboles de distribución con raíz en el origen (Source based trees). Si bien esto introduce en el caso de CBT una carga innecesaria en la cantidad de objetos *replicator* necesarios en los nodos, sus consecuencias no son significativas debido a que los dichos objetos están implementados eficientemente en C++.

6.1 Clases implementadas y su relación con las clases provistas por Ns

A continuación se describe brevemente las clases implementadas, su funcionalidad y su relación con el soporte Ns. La información más detallada se encuentra en comentarios incluidos en el código fuente, en el Apéndice A.

6.1.1 Clase CBT

Es la clase que implementa la funcionalidad del protocolo tal como fue descripta en el Capítulo 5. Su objetivo principal es la creación y mantenimiento de las tablas de reenvío CBT, de los elementos de reenvío multicast en el nodo y la interacción con entidades pares remotas.

La reacción de la entidad a eventos externos (PDUs) está implementada en métodos asociados a la recepción de cada PDU (*recv_join-request*, etc.) que son invocados por el agente de recepción y emisión de PDUs de la entidad CBT.

Cuenta con métodos definidos en la clase *McastProtocol* para reaccionar ante eventos locales producidos por IGMP o aplicaciones locales (métodos *join-group, leave-group*) y para interactuar con los elementos de reenvío provistos por el nodo (métodos *upcall, handle-cache-miss, handle-wrong-iif, drop*). Estos métodos son redefinidos para adaptarlos a las acciones específicas del protocolo CBT.

Se agregan además métodos invocados desde el script de simulación, para posibilitar la configuración de la entidad CBT respecto de cores para los grupos, Designated Routers en vínculos multiacceso, y la manera de operación sobre algunos vínculos, modos unicast o multicast, permitiendo así la posibilidad de operar CBT en ambas modalidades en el ambiente Ns.

Otro grupo de métodos cuyo objetivo es posibilitar el seguimiento de las acciones del protocolo permite la configuración de la entidad respecto a la emisión de mensajes por salida standard al producirse determinados eventos: emisión o recepción de PDUs, vencimiento de timers, etc.

A continuación se describe en forma general la clase CBT.

6.1.1.1 Inicialización

Las funciones de inicialización se encuentran en los procedimientos de instancia *init* e *initialize*. En estos procedimientos, invocados cuando se crea un objeto de la clase, se crean e inicializan los objetos asociados a la entidad CBT, descriptos más abajo.

Se crea un agente de transmisión, *messager* (clase *Agent/Message/CBT*), que posibilitará la comunicación del objeto CBT con pares residentes en nodos remotos. Se asocia este agente al nodo (*Node::attach*) para que le sea asignada una dirección, compuesta por la identificación del nodo y un port específico.

Se crean e inicializan, sin contenido, las tablas de reenvío CBT, permanente y transitoria, respectivamente instancias de las clases *MFC_Table* y *Transient_Table*.

Se crea el objeto administrador de timers para envío de ECHO_REQUESTs, clase *Echoreq_adm*, que posibilita compartir un timer para echo request entre varios grupos con una interfaz parent común.

Se crea un objeto de tipo *CBTinterface* por cada interfaz del nodo, inicializando sus valores de acuerdo a las características del vínculo asociado. En la presente implementación, no se tiene en cuenta el agregado y la eliminación de interfaces CBT en forma dinámica.

Por último se crea un agente de desencapsulación de paquetes multicast para poder recibir datos a través de vínculos que no soporten direccionamiento multicast.

6.1.1.2 Activación

La activación del objeto CBT se produce a través del procedimiento de instancia *start*, y consiste en las acciones para dejar operativo el protocolo en el nodo: se invoca al método *join-group* del objeto *Node*, para incorporar al agente de transmisión CBT (*messager*) al grupo multicast “all_cbt_routers”, de manera que pueda recibir PDUs direccionadas al grupo. En este caso, el procedimiento *join-group* específico de CBT (invocado por el *join-group* del objeto *Node*) no tomará ninguna acción, ya que no es necesaria la construcción del árbol de distribución para este grupo.

6.1.1.3 Agregado y abandono de aplicaciones a grupos multicast

Se definen los procedimientos para agregado y abandono de aplicaciones locales (o agentes IGMP) del nodo a los diferentes grupos, *join-group* y *leave-group* respectivamente.

Estos procedimientos actualizan variables del objeto CBT y por último invocan a los procedimientos de instancia de los objetos que representan las tablas de reenvío multicast para crear entradas transitorias o permanentes.

Debe tenerse en cuenta que el agregado de un agente a un grupo no produce una actualización inmediata de los elementos que componen la función de reenvío. Dicha actualización se producirá al recibirse un paquete multicast para el cual no existe aún el elemento de reenvío correspondiente, luego de la consulta a las tablas de reenvío mantenidas por CBT.

El abandono de un grupo por parte de un agente puede provocar la eliminación de la entrada para el grupo en la tabla de reenvío multicast, si se trata del último agente local en el grupo y no existen otras entradas child. Esto traerá como consecuencia la desactivación de algunos de los elementos de reenvío.

Los agentes invocan a los procedimientos *join-group* y *leave-group* del objeto *Node*, el que luego de registrar la información correspondiente a los agentes y al grupo, invoca a su vez al procedimiento respectivo del objeto CBT.

6.1.1.4 Configuración

Como fue mencionado, se implementaron procedimientos de configuración, los cuales permiten configurar al agente CBT para que opere de una manera determinada. Los tipos de configuración que es posible realizar abarcan los siguientes aspectos:

- **Configuración de vínculos**: el ambiente Ns ofrece las opciones de operación con direcciones unicast y/o multicast, pero sobre la totalidad de los nodos, es decir, no es posible que algunos vínculos sólo tengan capacidad unicast mientras que otros tengan además capacidad multicast. Para posibilitar simulaciones que incluyan ambos tipos de vínculo, se ofrece la opción de configurar las interfaces CBT para que operen de una u otra manera, es decir, aunque la opción multicast esté habilitada, el protocolo CBT enviará sus PDUs con direcciones unicast. Respecto al envío de los paquetes de datos multicast, se crea para este tipo de vínculos, un agente de encapsulación y otro de desencapsulación (a ver más adelante).
  Una interfaz CBT (y por lo tanto el vínculo asociado) puede ser configurada para que trabaje sólo con direcciones unicast, invocando el procedimiento de instancia *set-unicast*. El parámetro que debe pasarse para identificar la interfaz física, por razones de facilidad de uso, es la identificación (*id_*) del nodo al cual se accede por dicho vínculo (en vínculos multiacceso, cualquiera de ellos). Todos los nodos conectados al vínculo deben configurar sus interfaces relacionadas con él como unicast.
- **Configuración de designated router**: en vínculos multiacceso, como primer paso de la operación del protocolo, debe acordarse entre los nodos cual de ellos será el designated router. Esta elección es realizada a través del protocolo Hello, actualmente no implementado. Para posibilitar la operación del protocolo, se provee un procedimiento de instancia, invocado normalmente desde el script de simulación del usuario, *set_des_router*, que permite que todos los nodos conectados al vínculo conozcan la dirección del designated router. Los parámetros que deben entregarse al procedimiento son un nodo adyacente, que servirá para identificar el vínculo, y la identificación del nodo que actuará como designated router en dicho vínculo. Debe invocarse este procedimiento en la totalidad de las entidades CBT residentes en los nodos conectados por el vínculo multiacceso.
- **Configuración de cores para los diferentes grupos**: La configuración de cores admite ser realizada a través de procedimientos de bootstrapping (plug and play), comunes a varios protocolos multicast, o bien a través de configuración manual. Debido a que los procedimientos de descubrimiento automático de cores escapan al objetivo del presente trabajo, se proveen procedimientos para realizar la configuración manual de los cores. Se provee un procedimiento de instancia en la clase CBT, *set-core*, que provoca acciones internas en el objeto CBT (la creación de una entrada de tipo core en la tabla de ruteo

permanente) y luego invoca a un procedimiento de instancia agregado a la clase *Simulator*, *set-core*, para hacer accesible globalmente la información acerca de los nodos core. Cuando se solicita a una entidad CBT integrarse al árbol de distribución de un grupo determinado, dicha entidad accede a la dirección del core invocando al método *get-core*, también agregado a la clase *Simulator*.

6.1.1.5 Emisión de PDUs

Esta función es llevada a cabo por el agente de transmisión asociado al objeto CBT. Su finalidad es posibilitar la comunicación del agente CBT con entidades pares remotas.

Para realizar la transmisión, el agente CBT provee un procedimiento de instancia, *send_msg*, cuyo objetivo es configurar de manera apropiada al agente de transmisión y luego invocar su procedimiento para emisión (*transmit*).

La configuración que debe realizarse consiste en colocar la dirección de destino apropiada, "all_cbt_routers" si el vínculo soporta multicast o la dirección unicast del nodo CBT destino en caso contrario (vínculo sólo unicast o túneles). Otro aspecto es la configuración del campo time-to-live, para indicar la cantidad de nodos máxima a atravesar; en caso de utilizar dirección multicast, el valor de ttl debe ser 1 según la especificación del protocolo [7].

Por último, debe tenerse en cuenta que la PDU debe ser enviada por una determinada interfaz (si bien se dispone de la dirección de destino del paquete, compuesta por la identificación del nodo y el port asociado al agente, en el nodo emisor debe seleccionarse la interfaz a través de la cual se enviará el paquete); para ello es necesario colocar un valor determinado en la variable *target_*[36] del agente de transmisión. Como se verá más adelante, el *target_* podrá ser un objeto *networkinterface* o un objeto *DynamicLink* en el caso de utilizar la capacidad dinámica de caída y restablecimiento de vínculos provista por Ns.

6.1.1.6 Recepción de PDUs

Cuando se recibe una PDU, el agente de transmisión interpreta su tipo y en base a él invoca al procedimiento de instancia apropiado del objeto CBT.

Estos métodos de instancia constituyen gran parte de la funcionalidad descripta en la especificación. Existe un procedimiento por tipo de PDU posible (*recv-join_request*, *recv-join_ack*, etc). Estos procedimientos son invocados por el agente de transmisión, recibiendo el objeto CBT como parámetro una lista con los diferentes campos de la PDU recibida, excepto su tipo, ya eliminado por el agente mencionado. Además de los campos definidos en la norma, se utilizan campos adicionales necesarios en el medioambiente Ns, que se describen más adelante.

6.1.1.7 Interacción con la función de reenvío

Como fue mencionado, la interacción entre la entidad CBT y el soporte de reenvío multicast provisto por Ns se realiza a través de los métodos provistos por la clase *McastProtocol*, redefinidos por cada protocolo de acuerdo a su funcionalidad específica.

En las secciones siguientes, se describe brevemente la relación entre ruteo y reenvío y el soporte multicast provisto por Ns para luego detallar las acciones específicas implementadas en el código CBT.

6.1.1.7.1 Ruteo y reenvío de paquetes

[36] La variable *target_* indica el siguiente objeto que procesará el paquete.

Tanto en el contexto unicast como multicast, las funciones de ruteo y de reenvío (forwarding) están claramente diferenciadas.

La primera de ellas trata con los problemas relativos a la adquisición de la información necesaria para un envío correcto y eficiente de los paquetes, y para ello implementa diferentes estrategias que se ven reflejadas en los protocolos de ruteo, tales como RIP y OSPF en casos unicast y DVMRP, PIM, CBT en multicast.

Estos protocolos incluyen dos aspectos, mecanismos para la adquisición de la información necesaria y especificación de un algoritmo que en función de dicha información determine caminos eficientes a través de los cuales serán enviados los paquetes.

La segunda de las funciones mencionadas, reenvío, se encarga de tomar la desición de reenvío de los paquetes, en tiempo real con el arribo de los mismos al nodo. Para ello utiliza tablas (generalmente residentes en el kernel) que contienen para cada posible destino, el próximo nodo hacia donde se debe reenviar el paquete.

Estas tablas son actualizadas por la función de ruteo, cada vez que el algoritmo determina que es necesario.

Para el caso que nos ocupa, multicast, la información de reenvío se halla en tablas del protocolo, que reflejan el árbol de distribución multicast. El objeto del mismo es cubrir la totalidad de los nodos miembros de un determinado grupo multicast, de manera que un paquete enviado al grupo llegue a todos los nodos interesados.

Básicamente un protocolo multicast se encargará de formar el árbol de distribución, para hacerlo, podrá tener su noción propia del estado de la red, o bien interactuar con el ruteo unicast, como ocurre en el caso de CBT.

6.1.1.7.2 Soporte multicast provisto por Ns

Como fue mencionado en el capítulo 3, la función de reenvío multicast está provista por Ns y reside en algunos de los objetos que conforman el nodo, que se encargan de discriminar entre paquetes con destinos unicast o multicast (*switch*), y en el caso de estos últimos, reenviarlos con la posible producción de nuevas copias según el estado del árbol de distribución.

El protocolo multicast, debe interactuar con esos objetos (*replicator* y *multiclassifier_*) en el nodo, para construir y actualizar el equivalente a las tablas de reenvío en una implementación real.

Debe tenerse en cuenta que los elementos de reenvío encargados de producir las copias de los paquetes multicast (objetos *replicator*) presentes en el nodo, son creados recién cuando se recibe un paquete multicast asociado al par <origen, destino> [37](data driven) y no cuando, debido a recibir una PDU (JOIN-ACK por ejemplo), se actualizan las tablas multicast.

El soporte provisto por Ns se describió brevemente en el capítulo 3, mostrándose sus elementos en la figura 3.2. A continuación se describe la función de cada uno de ellos.

- *Classifier*: Es una clase que permite, en base a ciertos campos de un paquete (por ejemplo su dirección), establecer el destino del paquete[38] (*target_*), ya sea éste un objeto de los que conforma el nodo o un objeto que forma parte del vínculo. De esta clase heredan varias de las que se describen a continuación.
- *Switch*: es un *classifier*, encargado de entregar el paquete, según sea su dirección de destino multicast o unicast, al siguiente componente; en el caso que nos ocupa, paquetes multicast, es entregado al *multiclassfier*; mientras que en el caso de paquetes unicast, es entregado al

[37] Orientado al paradigma Source based trees.

[38] En este caso, el destino del paquete se refiere al siguiente elemento que lo procesará, y no a la aplicación a la cual es destinado (por ejemplo un agente en un nodo).

objeto *classifier* del nodo. Un paquete se reconoce como unicast o multicast, según el primer bit de su dirección sea cero o uno, de acuerdo al esquema de direccionamiento Ns.

- *Multiclassifier*: es derivado de *classifier*. Clasifica los paquetes de acuerdo a su dirección de origen y su dirección de destino (multicast). De acuerdo al contenido de este par <origen, destino>, entrega el paquete al objeto *replicator* correspondiente. Además realiza la comprobación de que el paquete haya arribado al nodo por la interfaz correcta (a ver más abajo).
- *Replicator*: Deriva de *classifier* aunque sólo a efectos de utilizar su código. En realidad no realiza ninguna clasificación sobre el paquete que recibe, sino que lo replica en cada uno de los objetos que tiene asociados (con destino a agentes en el nodo o vínculos hacia otros nodos).

Es importante resaltar que este soporte para reenvío provisto por Ns está orientado a protocolos multicast basados en el paradigma source based trees; en el caso de CBT, que precisamente se basa en evitar este tipo de árboles de distribución para reducir costos en los nodos, se plantearán alternativas de implementación que luego serán analizadas.

6.1.1.7.3 Tratamiento de un paquete multicast en un nodo

Se describe a continuación el camino seguido por un paquete multicast que arriba a un nodo.

En primer término es entregado al objeto *switch*, quien en base al primer bit de la dirección de destino determina si se trata de un paquete unicast o multicast. En este último caso, el paquete es entregado al objeto *multiclassifier*, el que decide, en base al par de direcciones <fuente, destino> y a la interfaz de arribo, el objeto *replicator* a quien entregar el paquete. En el caso en que el *replicator* no exista o la interfaz sea errónea, el objeto *multiclassifier* interactúa con el protocolo multicast en el nodo para que trate el problema (creación de los elementos de reenvío si corresponde de acuerdo a las entradas en la tabla, etc.). Por último, el paquete es procesado por el objeto *replicator* correspondiente quien se encarga de producir las copias necesarias y distribuirlas a los agentes locales miembros del grupo multicast y a los vínculos que conectan al nodo con los nodos adyacentes pertenecientes al árbol de distribución[39]. En caso de que no exista el objeto *replicator*, y no haya podido ser creado luego de la interacción con el protocolo multicast, el paquete es descartado, invocándose al método *drop* del protocolo.

El objeto *multiclassifier* interactúa con el protocolo de ruteo de la siguiente manera:

- Llega un paquete para el cual existe la entrada <origen, destino> y cuya interfaz de arribo coincide con la almacenada para esta entrada; en este caso el paquete se pasa directamente al objeto *replicator* correspondiente, quien produce tantas copias del paquete como destinos tenga especificados.
- Llega un paquete para el cual no existe la entrada <origen, destino>; en este caso, se invoca a un procedimiento del nodo (*new-group*) a través del cual se invoca a un método del protocolo multicast: *handle_cache_miss*. El protocolo tomará las medidas que correspondan: si existe una entrada en el árbol de distribución, creará el elemento de reenvío (*replicator*).
- Llega un paquete para el cual existe la entrada <origen, destino>, pero la interfaz de arribo asociada a esta entrada no corresponde con la interfaz de arribo del paquete. Esto implica una situación anormal en el ruteo multicast; se notifica al protocolo multicast de igual manera a la descripta en el caso anterior, pero a través de su método *handle_wrong_iif*.

[39] A través de su envío a los targets_ locales correspondientes, objetos *networkinterface* o *DynamicLink*.

El agregado o supresión de *replicators* se realiza invocando a los métodos *add-mfc* y *del-mfc* del nodo.

Como puede deducirse de lo explicado anteriormente, el soporte multicast provisto por Ns está orientado a protocolos multicast del tipo source based trees, que tienen un costo O (|S| X |G|). Esto se aprecia en el tamaño de la tabla y la cantidad de objetos *replicator* que debe ser mantenida por cada nodo. El protocolo CBT, por otra parte, está diseñado para evitar este orden de consumo de recursos, no discriminando el origen de cada paquete, sino construyendo el árbol de distribución en función del destino, y realizando el reenvío en función del grupo y de la interfaz de arribo[40].

6.1.1.7.4 Alternativas de implementación

Se plantean aquí dos posibilidades de implementación, una de ellas modificando el soporte multicast de Ns, a través del reemplazo del *multiclassifier* por otro objeto que ignore los direcciones de origen de los paquetes, logrando un costo de simulación O(|G|) en cada nodo.

Esta alternativa, si bien es la más eficiente desde el punto de vista del simulador, implica pérdida de compatibilidad y pérdida de portabilidad, ya que se generaría un nodo multicast diferente a los soportados por el simulador, siendo necesario además recompilar el código para la instalación del protocolo CBT (*Multiclassifier* es un objeto perteneciente a la jerarquía compilada, en contraste con todo el código CBT, implementado en OTcl y por lo tanto instalable sin necesidad de compilación).

La segunda de las alternativas, que fue la elegida debido a presentar compatibilidad y no requerir recompilación, adapta el manejo del reenvío llevado a cabo por la implementación a las características que presenta Ns. Es decir, desde el punto de vista de CBT, se manejan las entradas sin tener en cuenta el origen de los paquetes, pero para crear los elementos de reenvío sí se lo hace. Se utilizan funciones provistas por Ns como si se tratara de un protocolo source based trees: *new-group*, *add-mfc*, *del-mfc*, *handle-cache-miss* y *handle-wrong-iff*. La elección de esta alternativa no afecta de manera alguna al comportamiento del protocolo CBT.

El procedimiento es el siguiente:

- Cuando el protocolo CBT recibe una invocación a *handle-cache-miss*, significa que se ha recibido un paquete multicast para un par <origen, destino> para el cual no existen elementos de reenvío. El agente CBT analiza entonces si se trata de una entrada válida comprobando sólo (paradigma shared trees) que el nodo sea on-tree para el destino del paquete; si lo es, crea el *replicator* correspondiente utilizando *add-mfc*. El *replicator* creado estará asociado a la interfaz de arribo del paquete, al origen y a la dirección de destino, y su función será replicar el paquete en todas las demás interfaces asociadas al grupo (notar que a efectos del reenvío, no existe la noción parent/child).
- Cuando llega un paquete destinado al mismo grupo pero proveniente de otro origen, el reenvío provocará otro *handle-cache-miss*, y como consecuencia se creará otro *replicator*. Si ambos paquetes arribaron por la misma interfaz, se estará creando un replicator que no sería necesario para un reenvío diseñado para CBT. Si en cambio el paquete arriba por otra interfaz, sería necesaria la creación de otro *replicator*, ya que el conjunto de interfaces de salida de un objeto de este tipo, es el conjunto de interfaces asociadas al grupo excepto la interfaz de arribo.
- Cuando se agrega una nueva entrada child para un grupo, se debe actualizar a todos los *replicators* del grupo agregando a cada uno de ellos el objeto asociado con esta nueva interfaz. Esta acción debe realizarse en forma explícita en el momento de la creación de la

[40] Los costos mencionados se refieren a costos en los routers. En el caso de la simulación, sólo se tendrán costos mayores en cuanto a memoria ocupada.

entrada en la tabla de reenvío, ya que por las características del soporte (árboles de distribución unidireccionales) el código Ns no las creará cuando se invoque (data driven) al método *add-mfc*.

Cuando se elimina una entrada child en un nodo (por ejemplo al recibir la entidad CBT un QUIT-NOTIFICATION), debe inhabilitarse los *replicators* de la siguiente manera:

- Para todos aquellos *replicators* para los cuales coincide el grupo (sin importar el origen) y su interfaz de entrada es la child a eliminar, debe inhabilitarse todas las salidas.
- Para aquellos *replicators* en los cuales coincide el grupo pero la interfaz de entrada difiere de la child, debe inhabilitrarse sólo la interfaz child de salida.

Debe tenerse en cuenta que la última acción indicada en ambos casos (agregado y eliminación de entradas) es consecuencia de la bidirecionalidad del árbol de distribución CBT. Dichas acciones, consistentes en la actualización de *replicators* asociados a otras entradas[41], complican el código implementado. Esta característica se encuentra reflejada en los procedimientos *borrar_child* y *actualizar_replicators* (invocado al agregar una entrada child) de la clase CBT.

Para posibilitar la realización de estos procesos y soportar las posibles clases a las que puede pertenecer un objeto destino de un *replicator* (a ver más adelante), el objeto CBT lleva un registro de los *replicators* creados, asociándoles la 3-upla <origen, destino , interfaz de arribo> y una lista con las interfaces y los objetos *targets* asociados de salida.

Cuando el agente CBT recibe una invocación a *handle-wrong-iif*, significa que se ha recibido un paquete para un grupo on-tree pero que éste ha arribado por una interfaz diferente proviniendo del mismo origen. Esta es una situación anormal que puede deberse a errores o estados de no convergencia del protocolo de ruteo unicast y debe ser tenida en cuenta ya que el reenvío de los paquetes multicast en estas condiciones podría provocar ciclos.

6.1.1.7.5 Organización de los objetos replicator

En el siguiente ejemplo se muestra una topología compuesta de seis nodos pertenecientes al grupo multicast G (figura 6.1), de los cuales se supone que los nodos N0, N2, N3, N4 y N5 son también emisores. Se supone además que en N1 hay dos aplicaciones (agentes) ,ag1 y ag2, que reciben información multicast para el grupo G.

En N1 se muestran las interfaces a b y c, a través de las cuales se realiza el reenvío de paquetes multicast. Estas interfaces son en realidad los puntos de entrada (*head_*) a los vínculos (*link*) por los cuales deberá enviarse los paquetes multicast; en este caso se supone que son objetos *networkinterface*. Los otros objetos a los cuales debe enviarse la información en el nodo, son los agentes mencionados, ag1 y ag2, objetos derivados en este caso de la clase *Agent*.

A continuación se muestran los objetos *replicators* (y sus conjuntos de objetos destino) que deberá administrar el objeto *multiclassifier* del nodo para una organización basada en el paradigma source based trees (el provisto por Ns) y cómo podría modificarse si se tratara de un modelo orientado al paradigma shared trees.

El conjunto de replicators necesarios para el paradigma source based trees es el siguiente:

[41] Puede considerarse, de acuerdo a la implementación Ns, que cada *replicator* es asociado por el objeto *multiclassifier* con una "entrada" <S, G, interfaz de arribo>.

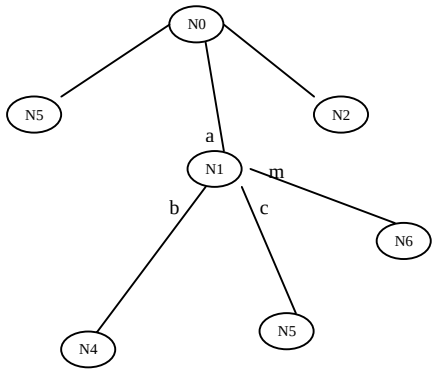

Fig. 6.1. Topología integrando parte de uno o varios árboles de distribución.

<5, G, a> => (b, c, ag1, ag2)
<2, G, a> => (b, c, ag1, ag2)
<0, G, a> => (b, c, ag1, ag2)
<4, G, b> => (a, c, ag1, ag2)
<3, G, c> => (a, b, ag1, ag2)

En este caso el objeto *multiclassifier* tendrá cinco *replicators* de acuerdo al nodo de origen y al grupo; el tercer elemento agregado a este par es la interfaz de arribo esperada de los paquetes. Si bien no constituye un índice para acceder al replicator cuando el *multiclassifier* recibe un paquete, esta interfaz de arribo es comparada con la de llegada del paquete, y en caso de no coincidir se lo indica al protocolo multicast a través del procedimiento *handle-wrong-iif*; de no indicarlo, se podría producir un ciclo en el correspondiente árbol de distribución ya que una de las interfaces de salida podría coincidir con la de entrada.

Cada objeto *replicator* tiene asociado un conjunto de destinos a los cuales debe enviar el paquete recibido; en este caso los objetos *networkinterface*, que representan la distribución de paquetes en el árbol, y los objetos ag1 y ag2, que representan a las aplicaciones locales.

Para un paradigma shared trees, se tendría un conjunto más reducido de *replicators*, debido a que existe un único arbol de distribución para el grupo, no importando el origen de los paquetes.

<*, G, b> => (a, c, ag1, ag2)
<*, G, c> => (a, b, ag1, ag2)
<*, G, a> => (b, c, ag1, ag2)

En esta implementación, se genera el conjunto de *replicators* mostrado en primer término. Para generar los tres *replicators* asociados a la interfaz a, por ejemplo, deben haberse producido, luego del armado de las tablas de reenvío, tres invocaciones a *handle-cache-miss*, una para cada origen (5, 2 y 0).

En el caso de producirse la eliminación de las interfaces CBT asociadas a la interfaz a, involucrando los orígenes 5, 2 y 0, se producirán los siguientes cambios: eliminación de *replicators* <5, G, a>, <2, G, a> y <0, G, a> y eliminación de la interfaz de salida a en los *replicators* <4, G, b> y <3, G, c>. Como fue mencionado, esta última acción es necesaria debido a la bidireccionalidad del árbol de distribución CBT.

En el caso de producirse el agregado de una nueva entrada child (a nivel CBT), por ejemplo vía interfaz m y proveniente del nodo N6, se agregará un nuevo objeto *replicator* al producirse la invocación a *handle-cache-miss*:

<6, G, m> => (a, b, c, ag1, ag2)

y los *replicators* existentes hasta el momento serán modificados al agregarse la entrada (antes de que se produzca el cache-miss) como se muestra a continuación.

<5, G, a> => (b, c, ag1, ag2, m)
<2, G, a> => (b, c, ag1, ag2, m)
<0, G, a> => (b, c, ag1, ag2, m)
<4, G, b> => (a, c, ag1, ag2, m)
<3, G, c> => (a, b, ag1, ag2, m)

6.1.1.7.6 Alternativas para los destinos de los objetos *replicator*

A efectos de explicar el funcionamiento del reenvío, se ha supuesto que los objetos que almacena un *replicator* a efectos de entregar un paquete a múltiples destinos son instancias de la clase n*etworkinterface*.

Hay dos casos en que estos objetos serán instancias de otras clases: cuando se utiliza la capacidad de Ns de crear vínculos (*links*) dinámicos y cuando se utilizan vínculos unicast o túneles.

- **Uso de vínculos dinámicos:** Es una capacidad provista por Ns que permite indicar en el script de simulación ciertos comportamientos de los vínculos que unen los nodos. De esta manera, se podrá especificar que en determinados momentos un vínculo deje de estar operativo, con lo cual se producirá la correspondiente pérdida de paquetes. Para las diferentes alternativas de especificación de comportamiento de vínculos (manual, exponencial, etc, refrirse a la documentación Ns [13]).
  Cuando se define un vínculo como dinámico, Ns agrega como cabecera (*head_*) del vínculo un objeto de clase *DynamicLink*, cuya función es almacenar el estado del vínculo, y en base a él descartar un paquete (vínculo no operativo) o entregarlo al siguiente objeto (*target_*), en este caso el objeto *networkinterface*. Debido a esta característica, cuando se crea un objeto *replicator* como consecuencia de una invocación al método *handle_cache_miss* del protocolo, es necesario determinar, para todos los vínculos involucrados (conocidos a través de sus rótulos de interfaz), si son o no dinámicos; en caso de serlo, el objeto a agregar en el replicator será el de clase *DynamicLink* en lugar del objeto *Networkinterface*.

- **Uso de vínculos unicast o túneles:** Tanto en los vínculos que no ofrecen capacidad multicast como en los túneles, es necesario realizar una encapsulación del paquete multicast en el nodo que lo envía (unicast) y la respectiva desencapsulación en el nodo que lo recibe. Estas funciones están a cargo de agentes de encapsulación (clase *encap*) y agentes de desencapsulación (clase *decap*) respectivamente.
  En el caso de utilizar un agente de encapsulación, dicho agente se convertirá en el objeto de destino a agregar al *replicator*.

6.1.2 Clase CBTInterface

Pueden existir varios objetos de esta clase en cada nodo. Cada instancia está asociada con una interfaz en el nodo, la cual se corresponde con una interfaz física descripta en la especificación. Su objeto es independizar, en cierta medida, al código CBT de las características de las interfaces tal como son definidas en Ns. De esta manera, se podrá determinar, por

ejemplo, el tipo de un vínculo (multiacceso, multicast, etc) sin necesidad de conocer detalles internos de Ns.

Cada objeto CBTInterface almacena valores referidos a las características físicas de la interfaz (punto a punto, etc) y valores propios de CBT, especialmente en lo que se refiere a mecanismos para la elección del Designated router en vínculos multiacceso (protocolo Hello).

Los métodos implementados permiten la consulta y asignación de valores a algunas de las variables correspondientes.

A continuación se describe brevemente la función de cada variable definida; algunas de ellas corresponden a la MIB CBT[8] y MIB IP Multicast[17] y otras son internas de la implementación.

- **Label**: es el rótulo (label) de la interfaz física a través del cual se identifica, en el medioambiente Ns, la interfaz dentro del nodo.
- **Linktype**: indica el tipo de vínculo asociado a la interfaz, tal como se describe en la especificación.
- **Pref_value**: Valor de preferencia para protocolo Hello, no implementado.
- **Des_router**: Indica si el nodo es o no Designated router sobre el vínculo (multiacceso). Este valor se configura de manera estática en el script de cada simulación.
- **Hello_int**: intervalo para protocolo Hello, no implementado.
- **Address**: En la MIB CBT, este valor identifica la dirección IP asociada a la placa. En esta implementación se utiliza el valor de dirección (id) que Ns asocia al nodo. Debe tenerse en cuenta que en este caso, una interfaz se identificará globalmente a través del valor address y del valor label.

6.1.3 Clase MFC_Table

Existe un objeto de esta clase por instancia del protocolo CBT y representa su tabla de reenvío (MFC). Esta tabla, a la cual se accede por grupo multicast debido a que CBT está orientado al paradigma shared trees, está compuesta por entradas (objetos *MFC_Entry*), una por grupo para el cual el estado de la entidad CBT es on-tree. Es en base al contenido de esta tabla que el protocolo crea o elimina los objetos necesarios para el reenvío de paquetes multicast.

Los métodos de esta clase permiten la obtención de información respectiva al conjunto de entradas (por ejemplo obtener la totalidad de los grupos pertenecientes a la tabla) y la obtención y manipulación de información específica de un grupo (por ejemplo obtener el parent del grupo, agregar una entrada child, etc). En este último caso, se invoca a los objetos *MFC_Entry* respectivos.

6.1.4 Clase MFC_Entry

Existe un objeto de este tipo por entrada (grupo multicast) en el objeto *MFC_Table*. Se proveen métodos para agregar, eliminar consultar o modificar entradas. Estos métodos no son accesibles directamente, sino que deben ser invocados a través de *MFC_Table*. Cada objeto contiene información acerca del nodo parent, nodos child e interfaces correspondientes para la entrada de la tabla. Debe considerarse que desde el punto de vista de la construcción del árbol de distribución es necesario el concepto de parent y child, esta distinción no se utiliza en la fase de reenvío de los paquetes.

Existe además un objeto timer asociado a la entrada, para eliminarla después de un cierto tiempo de no detectar actividad por parte del parent (ausencia de ECHO_REPLY).

6.1.5 Clase Transient_Table

Existe un objeto de esta clase por entidad CBT. Es una tabla organizada de manera similar a *MFC_Table*, pero que representa entradas transitorias (el nodo no forma parte aún del árbol de distribución), que podrán pasar a ser permanentes e integrar la *MFC_Table* al recibir el JOIN_ACK correspondiente o serán eliminadas luego de un cierto tiempo. Los métodos provistos tienen la misma funcionalidad que para *MFC_Table*.

6.1.6 Clase Transient_Entry

Los objetos de esta clase representan entradas en el objeto *Transient_Table*. Se corresponden en contenido y funcionalidad con los objetos *MFC_Entry*, con la excepción que cada entrada contiene un campo adicional que indica su condición o no de leaf. Esta condición indicará a la entidad CBT si debe realizar reintentos de conexión al árbol de distribución para el grupo en caso de no obtener respuesta.

6.1.7 Clase CBTTimer

Es la clase base para los diferentes timers que tienen funciones específicas y son administrados por la entidad CBT.

Con el objeto de simplificar el código CBT, se permite activar un timer inicializando su tiempo de expiración (timeout) y la cantidad de veces que ésta se producirá antes de ser cancelado. Esta característica es útil en casos tales como el reenvío del QUIT_NOTIFICATION, que debe realizarse un cierto número de veces. Como caso particular se puede asignar un valor uno, lo que producirá una única expiración. Cada vez que se produzca la expiración, será ejecutado un procedimiento (*tout*) que debe ser provisto por la clase derivada.

Al finalizar el número especificado de reintentos, se produce la cancelación del timer, que consiste en la ejecución opcional de un procedimiento de cancelación provisto por la clase derivada (*tfin*) y la eliminación del objeto *CBTTimer* para producir la liberación de los recursos asignados. Esta característica es importante debido a que se crean diferentes timers según los grupos para los cuales se integre el árbol de distribución; en simulaciones con una cantidad de nodos y número de grupos considerable, el mantenimiento de timers no utilizados podría dar lugar al agotamiento de la memoria.

Si bien Ns provee timers implementados en C++, se decidió implementar una nueva clase, totalmente en Otcl, para conservar la portabilidad[42]. Como consecuencia, los objetos *CBTTimer* envían eventos al scheduler utilizando la capacidad "*at-events*" provista por Ns.

El método *sched* permite la activación de un objeto *CBTTimer*, indicando el intervalo de tiempo luego del cual se producirá el timeout y la cantidad de veces que se permite la producción de un evento por vencimiento del intervalo.

El método *resched* permite la anulación del evento pendiente como consecuencia de una invocación previa a *sched*. Es posible indicar un nuevo valor para el intervalo y para la cantidad de veces a producir el timeout.

El método *cancel* produce la cancelación anormal (no por terminación de la cantidad de expiraciones especificada) del timer, la ejecución opcional del procedimiento *tfin* ya mencionado, y la destrucción del objeto *CBTTimer*. Se provee un parámetro de uso opcional para el procedimiento *tfin*, que indica si la cancelación se produjo normalmente o explícitamente a través de la invocación del método cancel.

[42] La clase provista por Ns no permite especificar reintentos.

Los objetos derivados de la clase *CBTTimer* corresponden a los timers descriptos en el capítulo de especificación del protocolo.

6.1.8 Clase Echorq_adm

Como fue mencionado, a nivel de la especificación por razones de simplicidad, se consideró un timer para envío de ECHO_REQUEST para cada uno de los grupos multicast presentes en el nodo. Sin embargo, considerando que la PDU no hace referencia a grupos, es posible utilizar un único timer por interfaz, lo que significa un menor consumo de recursos en el nodo en el caso en que otro nodo actúe como parent para más de un grupo. Por esta razón se crea un único timer para envío de ECHO_REQUEST por interfaz CBT, debiendo administrarse su uso entre los diferentes grupos involucrados.

El objetivo de la clase *Ehorq_adm*, es administrar la creación y eliminación de timers cuando sea necesario, y el agregado y eliminación de grupos a cada uno de los timers. Los métodos provistos permiten determinar si existe un timer para una determinada interfaz, eliminar un timer (en caso en que no existan grupos que lo utilicen), y agregar y eliminar grupos a un timer.

6.1.9 Clase Agent/Message/CBT (Agent/Message)

Existe un objeto de esta clase asociado a cada entidad CBT. Su función es el envío y recepción de mensajes a y desde otras entidades CBT remotas, como fue descripto en las secciones 6.1.1.5 y 6.1.1.6. Este tipo de agentes es capaz de recibir PDUs a través de vínculos multicast o unicast. Se utiliza la funcionalidad provista por la clase *Agent/Message*, consistente en la capacidad de envío de mensajes. La funcionalidad agregada consiste, en la recepción, en la decodificación parcial de la PDU recibida a efectos de determinar su tipo y en base a él invocar al procedimiento correspondiente de la entidad CBT.

6.1.10 Clases encap y decap (Agent/Message)

Tanto en los vínculos que no ofrecen capacidad multicast como en los túneles y en los casos de envío a un grupo (a su core[43]) por parte de un nodo no miembro del grupo, es necesario realizar una encapsulación del paquete multicast en el nodo que lo envía (transmisión unicast) y la respectiva desencapsulación en el nodo que lo recibe para entregarlo a los destinatarios (aplicaciones locales o elementos de reenvío). Estas funciones están a cargo de agentes de encapsulación y desencapsulación respectivamente.

Existe un agente de desencapsulación por nodo (clase *decap*), creado al iniciar CBT. Su dirección unicast es accedida por agentes de encapsulación remotos a través de mecanismos provistos por Ns. Cuando el agente recibe un paquete (unicast), desencapsula el paquete multicast contenido en él y lo inyecta en el nodo en su forma original (multicast), con sus direcciones de origen (unicast) y de destino (multicast) correspondientes[44]. De esta manera, los componentes del nodo perciben la llegada de un paquete multicast y actúan de la manera habitual.

[43] La especificación de CBT versión 2 determina que en caso de que un router que no sea miembro de un grupo desee enviar información a dicho grupo, debe encapsular el datagram multicat en uno unicast dirigido al core del grupo.

[44] Además de las direcciones de origen y destino, debe conservarse la interfaz de arribo del paquete, ya que los elementos de reenvío del nodo se basan también en ella (grupo, origen, interfaz de arribo).

Los agentes de encapsulación (clase *encap*) pueden residir en un nodo sin capacidad multicast (por ejemplo, en el caso de envío a un grupo por parte de un nodo no miembro) o formar parte de los elementos CBT (en los casos de routers con capacidad CBT e interfaces sin capacidad multicast o túneles).

En el primer caso, dichos agentes podrían ser creados al inicializar la aplicación correspondiente o en el momento en que sea necesario el envío multicast.

En el segundo caso, los agentes serán creados dinámicamente por los elementos de reenvío del nodo al recibir un paquete multicast para una interfaz unicast. En ese momento, los agentes serán asociados a las interfaces correspondientes de acuerdo a las entradas existentes en la tabla de reenvío multicast.

Respecto de la dirección de origen de un paquete multicast encapsulado, debe mencionarse que, al implementarse los objetos de encapsulación y desencapsulación totalmente en OTcl, se pierde la información respecto al port (del nodo origen) asociado al objeto que originó el paquete. Esta característica, que no afecta a la construcción y mantenimiento de los árboles de distribución, se debe a que el método *new-group* del nodo invocado por la función de reenvío Ns, entrega sólo la dirección (id) del nodo origen.

6.2 Soporte para comprobación de la actividad del protocolo

Tanto en la etapa de prueba del código implementado como cuando se desea monitorear la actividad del protocolo en los diferentes nodos, es importante contar con soporte para registrar las acciones de las entidades que conforman el protocolo.

En la presente implementación se proveen facilidades que permiten producir mensajes por la salida standard cuando ocurren ciertos eventos o mostrar determinada información cuando se lo solicita explícitamente. La emisión de estos mensajes puede controlarse desde el script de la simulación o en forma dinámica desde las aplicaciones (agentes) o desde el protocolo y debe discriminarse por cada objeto CBT (por ejemplo, podría activarse determinadas características sólo en los cores de ciertos grupos o en uno o varios nodos en particular).

Los mensajes producidos al ocurrir ciertos eventos son los siguientes:

- Emisión de mensajes por recepción y/o envío de determinadas PDUs: Es posible configurar a una entidad CBT de un determinado nodo para que emita mensajes relativos a la recepción y envío de las PDUs definidas por el protocolo (método *enable_pdus*), a excepción de HELLO (no implementada). Una vez habilitada, la entidad permanece en este estado hasta que es reconfigurada (método *disable_pdus*). La invocación a los métodos mencionados permite la especificación de una o la totalidad (ALL) de los tipos de PDUs.
- Emisión de mensajes por realización de operaciones sobre las tablas de reenvío (MFC_Table, Transient_Table): Los métodos *enable_tables* y *disable_tables* permiten configurar a la entidad CBT para la emisión de mensajes cuando se crea o elimina una entrada (child o parent) en la tabla permanente o transitoria. Mientras la entidad esté habilitada, producirá este tipo de mensajes.
- Emisión de mensajes por operaciones y eventos relacionados con los timers: Los métodos *enable_timers* y *disable_timers* permiten configurar a la entidad CBT de manera que emita mensajes al producirse operaciones relacionadas con los timers: creación, activación, desactivación y eliminación. Como en los casos anteriores, los mensajes serán producidos mientras la entidad permanezca habilitada para ello.
- Emisión de mensajes por PDUs recibidas y descartadas: Los métodos *enable_discard* y *disable_discard* permiten configurar a la entidad CBT para que emita mensajes cada vez que descarta una PDU. Estos casos pueden producirse como consecuencia de acciones normales del protocolo (por ejemplo el envío de tres QUIT-NOTIFICATION al parent o

recepción de un JOIN-ACK referido a un grupo para el cual no existe entrada transitoria -en un vínculo de acceso múltiple-). Como en los casos anteriores, los mensajes serán producidos mientras la entidad CBT esté habilitada para ello.

Otro grupo de facilidades consiste en el listado de características de determinados objetos. Estas deben solicitarse de manera explícita en el script de simulación o en las aplicaciones, siendo las que se enumeran a continuación:

- Listado de contenido de tablas de reenvío: Consiste en el listado del contenido de las tablas permanente y transitoria, con su grupo, su interfaz parent y el conjunto de interfaces child. Debe solicitarse explícitamente a la entidad CBT utilizando los métodos *list_MFCTable* y *list_TransientTable* respectivamente.
- Listado de características de las interfaces: Consiste en el listado de las características de

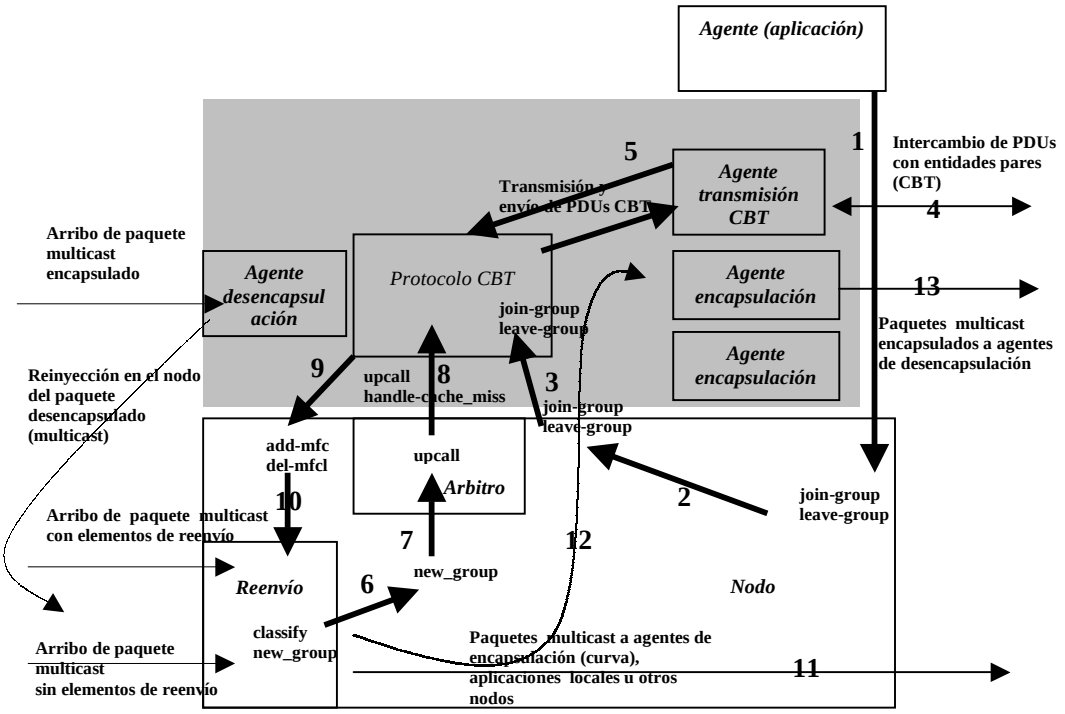

Fig. 6.2. Esquema simplificado de interacciones CBT-Nodo*. Los objetos en la parte sombreada corresponden a la implementación. Las líneas finas representan pasaje de datos (paquetes multicast).*

cada una de las interfaces en el nodo. Se solicita invocando el método *list_interfaces.*

- Listado de los nodos componentes del árbol de distribución para un determinado grupo: Consiste en el listado de los nodos que integran el árbol de distribución de un determinado grupo. Dicho listado especifica en primer lugar la raíz del árbol (el nodo core) con su lista de nodos child y luego repite el listado para cada child hasta llegar a las hojas del árbol. Esta facilidad se implementó agregando el método *map-tree* a la clase *Simulator* provista por Ns.

6.3 Interacción entre los componentes

En la figura 6.2 se muestran algunas de las componentes implementadas (parte sombreada) y la interacción entre ellas y con los demás elementos del nodo. Los intercambios de datos se muestran con línea fina y las demás interacciones con línea gruesa.

Se distingue por un lado las acciones que producen la actualización de las tablas de reenvío (Multicast Forwarding Cache) que maneja el nodo. Estas acciones pueden originarse en requerimientos de aplicaciones locales (o IGMP) que realizan un join-group o leave-group (1, 2, 3) o de PDUs recibidas (a través del agente de transmisión asociado) – JOIN-REQUEST, JOIN-ACK – (4, 5).

En base al contenido de la MFC, y disparado por la llegada de datos, se produce (cuando es necesario) la creación de los elementos de reenvío que posibilitarán que el paquete multicast sea entregado a las aplicaciones locales, a los vínculos hacia otros nodos con capacidad multicast, o luego de la creación del agente de encapsulación correspondiente, a vínculos a nodos sin capacidad multicast o a túneles.

El arribo de un paquete multicast para el cual ya fueron creados los elementos de reenvío, es entregado por el nodo a su(s) correspondiente(s) destinos(s) (11, 12, 13): vínculos a otros nodos, agentes de encapsulación y aplicaciones multicast locales (no mostrado en la figura).

El arribo de un paquete multicast sin elementos de reenvío creados y con entrada en la tabla MFC desencadena las interacciones que se muestran en el gráfico (6, 7, 8, 9 10), las que

```
set ns [new Simulator]
Simulator set EnableMcast_ 1
Simulator set NumberInterfaces_ 1
source cbtmain.tcl
#Instanciacion de nodos
for {set i 0} {i <=11} {incr i 1} {set n($i) [$ns node]}
#topologia
$ns duplex-link $n(0) $n(1) 1.5Mb 10ms DropTail
$ns duplex-link $n(4) $n(5) 1.5Mb 10ms DropTail
$ns duplex-link $n(7) $n(8) 1.5Mb 10ms DropTail
$ns multi-link-of-interfaces $n(1) $n(2) $n(3) $n(4)
10Mb 1ms
$ns multi-link-of-interfaces $n(4) $n(6) $n(7) $n(9)
1.5Mb 10ms
$ns multi-link-of-interfaces $n(9) $n(10) $n(11) 10Mb
1ms
#Instanciacion de agentes CBT en los nodos
for {set i 0} {i <=11} {incr i 1} {set cbt($i) [new CBT
$ns $n($i)]}
#Definicion de core
$cbt(0) set-core 65490
#Nodos 2, 7 y 9 Drs
$cbt(2) set_des_router 1 2
$cbt(1) set_des_router 2 2
$cbt(3) set_des_router 2 2
$cbt(4) set_des_router 2 2
$cbt(7) set_des_router 6 7
$cbt(6) set_des_router 7 7
$cbt(9) set_des_router 7 7
$cbt(4) set_des_router 7 7
$cbt(9) set_des_router 10 9
$cbt(10) set_des_router 9 9
$cbt(11) set_des_router 9 9
#Configuracion de vinculos unicast
$cbt(4) set_unicast 6
$cbt(6) set_unicast 4
$cbt(7) set_unicast 4
$cbt(9) set_unicast 4
#Arranque de los agentes CBT en tiempo 1.0
for {set i 0} {i <=11} {incr i 1} {$ns at 1.0 “cbt($i)
start”}
#Debugging
for {set i 0} {i<=11} {incr i 1} {
    $cbt($i) enable_pdus JOIN_REQ
    $cbt($i) enable_pdus JOIN_ACK
    $cbt($i) enable_tables
}
#Definicion de agentes multicast
set e(7) [new Agent/Message/Emisor 0.01]
$e(7) set dst_ 65490
$ns attach-agent $n(7) $e(7)
set r(9) [new Agent/Message/Receptor]
$ns attach-agent $n(7) $r(7)
set r(5) [new Agent/Message/Receptor]
$ns attach-agent $n(5) $r(5)
#Join de grupos, arranque y parada de emisor
$ns at 2.5 “$r(9) join-group 65490”
$ns at 2.8 “$r(5) join-group 65490”
$ns at 4.0 “$e(7) start”
$ns at 8.0 “$e(7) stop”
#Visualizacion del arbol
$ns at 8.0 “$ns map-tree”
$ns run
```

Fig. 6.3. Script de simulación: *definiendo y utilizando la topología de la figura 6.4.*

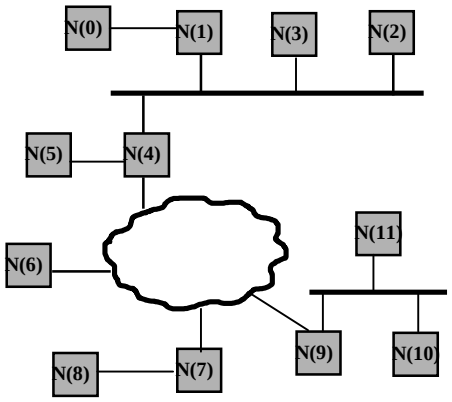

Fig 6.4. Topología ejemplo*: compuesta de dos vínculos multiacceso multicast, uno multiacceso no multicast y tres punto a punto multicast.*

provocan la creación de los elementos de reenvío y, en los casos en que el próximo destino del paquete sea accedido através de un vínculo no multicast o un túnel, la creación del agente de encapsulación correspondiente. Inmediatamente después, se provocan las acciones descriptas en el párrafo anterior para efectuar el envío del paquete.

El arribo de un paquete encapsulado (unicast) origina que el agente de desencapsulación lo desencapsule y lo reinyecte en el nodo, como si fuera un paquete multicast. Luego se tomarán las acciones que correspondan según lo descripto arriba.

6.4 Ejemplo

En la figura 6.4 se muestra un ejemplo de una topología simple y en la figura 6.3 el correspondiente script de simulación. Se definen doce nodos (desde el punto de vista de Ns todos tienen la funcionalidad de routers), conectados por una topología consistente en tres vínculos punto a punto y tres vínculos de acceso múltiple.

Las características propias de la implementación son la instanciación de las entidades CBT, la configuración de cores (en este caso nodo N(0) es core para el grupo 65490) y la definición de los nodos que actúan como DR en cada vínculo de acceso múltiple (los parámetros consisten de la identificación de un nodo adyacente que determina el vínculo multiacceso y la identificación del nodo DR).

El mecanismo de configuración de cores tiene alcance global (para todos los nodos de la simulación) mientras que el DR debe especificarse para cada nodo involucrado. Esta decisión se basa en el alcance más reducido en este último caso: sólo los nodos conectados al vínculo de acceso múltiple.

Se muestra en este caso la habilitación en todos los nodos de la emisión de mensajes por recepción y emisión de JOIN-REQUESTs y JOIN-ACKs y por modificación de tablas, transitorias y permanentes.

Se definen luego agentes multicast de emisión y recepción en diferentes nodos, y se solicita, en el tiempo de simulación 5.0, un mapa del árbol de distribución producido (*map-tree*).

En el Apéndice B se muestran ejemplos de topologías y operación del protocolo en vínculos punto a punto y multiacceso y las salidas correspondientes a cada ejecución.

6.5 Resumen

La implementación realizada abarcó la operación del protocolo con excepción de túneles (envío desde un nodo no miembro del grupo) y del protocolo Hello para vínculos multiacceso por razones de extensión.

Se mantuvo la premisa de lograr portabilidad implementando la totalidad del código en Otcl, lo que trajo como consecuencia la necesidad de adaptar el código implementado respecto de la interacción con el soporte de reenvío provisto por Ns, orientado al paradigma source based trees.

Se logró aislar claramente la operación del protocolo tal como fue descripta en el capítulo 5, de las características del soporte, a través de las clases que representan las interfaces tales como las percibe el protocolo, y las que representan las tablas de reenvío CBT.

Las mayores dificultades se presentaron en los aspectos relativos a la interacción con los elementos de reenvío Ns, y en los relacionados con la posibilidad de permitir la operación del protocolo sobre vínculos de acceso múltiple, utilizando el soporte Ns.

Las facilidades de debugging desarrolladas resultaron de gran utilidad tanto en la etapa de depuración del código como para el posterior análisis de la operación del protocolo.

Capítulo 7

Análisis

El análisis realizado tuvo por objeto comprobar el funcionamiento de la implementación y su performance en el medioambiente de simulación para evaluar su uso en análisis más detallados y comparaciones con otros protocolos. Abarcó un aspecto simple del protocolo, referido a su escalabilidad respecto de recuperación ante fallas en los vínculos que componen el árbol de distribución.

7.1 Motivación y objetivo del análisis

CBT tiene la característica de designar un nodo como core de un grupo[45]. El correspondiente árbol de distribución se construye, a medida que los nodos interesados en integrar el grupo lo demandan, utilizando información provista por el ruteo unicast. Dicho árbol es mantenido operativo a través de la actualización periódica de la información concerniente a las interfaces involucradas en los nodos que lo componen.

Periódicamente un nodo child debe verificar el correcto funcionamiento de su parent, a través del intercambio de PDUs ECHO-REQUEST y ECHO-REPLY. En caso de detectar ausencia de esta última, el nodo child inicia la reconstrucción del árbol.

Una característica importante a tener en cuenta es que un nodo CBT, a efectos de evitar posibles ciclos en el árbol de distribución, no intenta reconectarse al mismo manteniendo conectado el subárbol del cual es raíz, sino que inicia un proceso recursivo en dirección hacia los nodos hoja[46], a través del cual todos ellos se desconectan del árbol.

Esta característica que apunta a mantener la simplicidad del protocolo, produce una mayor carga en la red y posiblemente un tiempo de reconexión mayor que si se mantuviera conectado el subárbol. Los factores que influyen en los costos estarán dados por la conectividad de la red y por la distribución en la misma de los nodos hoja (leaf) integrantes del subárbol desconectado.

7.2 Experiencias realizadas

Las experiencias realizadas se basaron en una red con una topología similar a Arpanet (figura 7.1), consistente de 47 nodos conectados por 69 vínculos. Cada vínculo se supuso de 1.5 Mbps de capacidad con una demora de propagación de 1 ms.

Para realizar las pruebas se seleccionó un conjunto de diez nodos pertenecientes a la red de manera que los grupos multicast constituidos al considerar a cada uno de los nodos de la red como core resultasen, en promedio, con un grado medio de densidad[47] .

[45] Un nodo puede actuar como core para más de un grupo.
[46] Este proceso consiste en el envío de FLUSH-TREE a los nodos hijos.
[47] Se considera el grado de densidad de un grupo como el cociente entre la totalidad de los vínculos de la red y la cantidad de vínculos que conforman el árbol de distribución. Esta métrica hace referencia a la densidad o dispersidad de los grupos multicast.

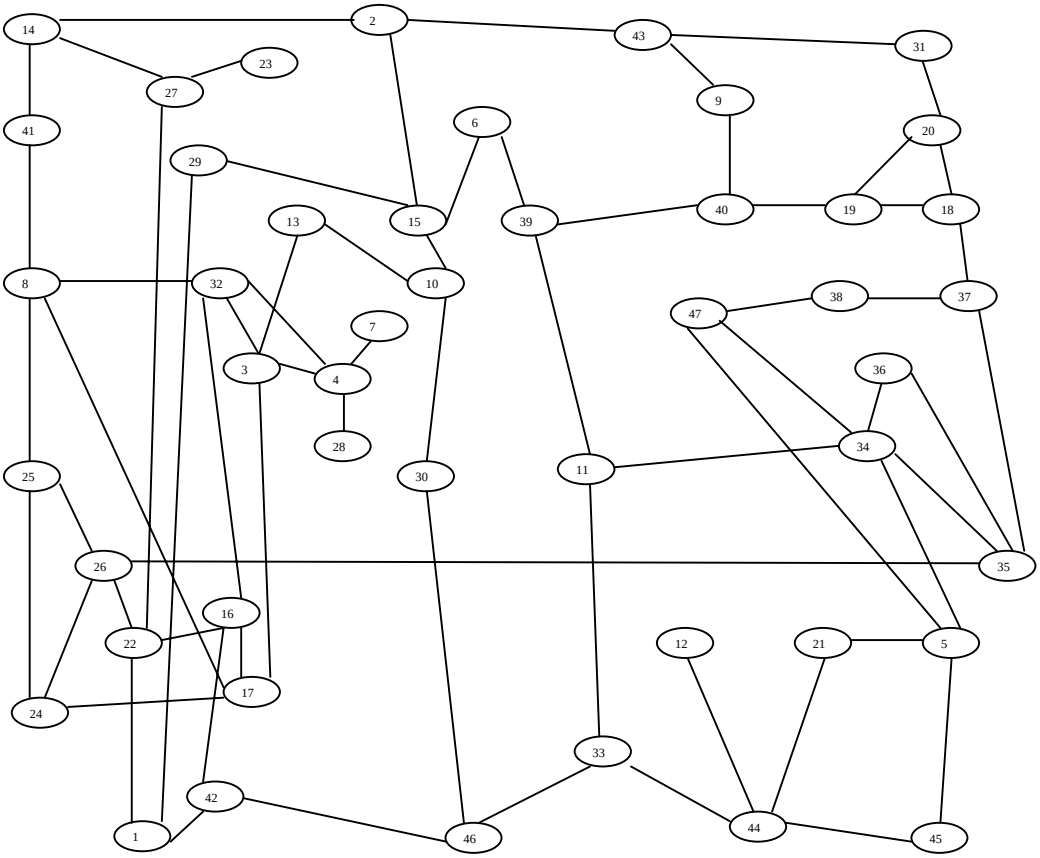

Fig 7.1. Topología utilizada para el análisis de costos y demoras

Se supuso que la totalidad de los routers tiene capacidad CBT y los vínculos son punto a punto con capacidad multicast.

Para simular la caída de los vínculos se utilizó la capacidad de caída y recuperación dinámica de vínculos provista por Ns, en este caso especificada manualmente desde el script de simulación a tiempos prefijados.

Para la recuperación de los datos necesarios, se utilizó la capacidad de debugging implementada en la clase CBT.

Las pruebas consistieron en seleccionar un nodo de la red como core para el grupo, luego provocar la caída de cada uno de los vínculos que integran el árbol de distribución, y medir el tiempo de recuperación y el costo de armado del nuevo subárbol resultante.

Al comienzo de la simulación, se activaron los agentes CBT en cada nodo de la red, y luego los procesos multicast especificos (agentes emisores y receptores desarrollados) en cada uno de los diez nodos elegidos, lo que provocó el agregado (join) de cada uno de ellos al grupo.

Estas acciones determinaron la construcción del árbol de distribución correspondiente basado en las tablas de ruteo unicast. Los nodos y vínculos integrantes del árbol, fueron determinados a través de las facilidades de debugging implementadas (método *map-tree* agregado a la clase *Simulator*).

Luego de un determinado tiempo, se produjo la falla en cada uno de los vínculos (uno en cada corrida de la simulación) componentes del árbol de distribución y se analizó el costo demandado y la demora producida para la construcción de un nuevo subárbol que lo reemplazara.

Los resultados obtenidos (demoras en la construcción de un nuevo subárbol de distribución y costos de transmisión), se registraron por cada vínculo en falla, y luego se los agrupó por nodo core y de acuerdo al diámetro del árbol de distribución que quedó aislado en cada caso.

Con respecto a la reconstrucción del subárbol de distribución, se supuso que los nodos, cuando comienzan las acciones de recuperación, disponen de información actualizada de ruteo unicast y que éste converge, lo cual impide la producción de ciclos transitorios durante el armado del árbol.

Esto implica que a las demoras obtenidas debería agregársele un tiempo mínimo fijo, consecuencia del tiempo de convergencia del algoritmo de ruteo unicast, tiempo a partir del cual deberían comenzar las acciones para la recuperación del subárbol de distribución.

7.3 Análisis de resultados

Con la información obtenida, se analizaron dos aspectos referidos a la recuperación del protocolo ante fallas en los vínculos, el primero referido a la escalabilidad de CBT en cuanto a la distancia máxima entre los nodos que conforman el árbol de distribución[48], y el segundo referido a la comparación de cada uno de los posibles nodos a ser seleccionados como core.

El primer aspecto consistió en agrupar los resultados obtenidos por longitud (en vínculos) del subárbol que quedó aislado por la falla. Este aspecto permitió determinar la escalabilidad del protocolo respecto al diámetro de la red.

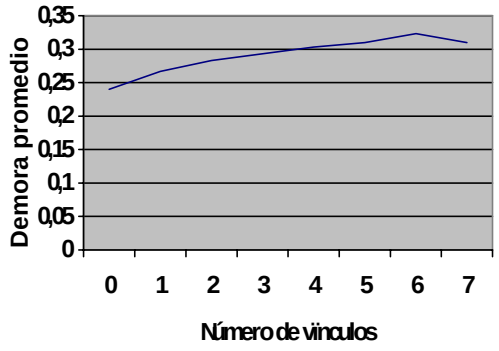

***Fig. 7.2. Demora promedio para la reconstrucción del árbol de distribución**: en función del número de vínculos del subárbol* aislado

Los resultados indican que las demoras en la reconstrucción del árbol aumentan lentamente (figura 7.2), lo cual indica una buena capacidad de escalabilidad del protocolo al aumentar el diámetro de la red.

[48] Esta distancia estará directamente relacionada con el diámetro de la red sobre la cual opera el protocolo en el caso de grupos dispersos.

Los costos medidos en intercambio de PDUs (figura 7.3), en cambio, aumentan con mayor rapidez, pero debe tenerse en cuenta que éste escala con un orden O(|G|), y que además se ha considerado el costo total, que deberá distribuirse entre los vínculos afectados.

En el aspecto referido a la comparación de cada posible nodo actuando como core, de acuerdo con los resultados obtenidos, existe una alta correlación entre los costos y las demoras producidos (figura 7.4), es decir, aquellos cores que producen costos más altos producen a su vez demoras mayores. Por otra parte, la elección del core demostró tener mayor peso respecto de los costos producidos, ya que la desviación standard de los costos resultó ser un 10,87% de su promedio, mientras que la de las demoras fue de un 6,70%.

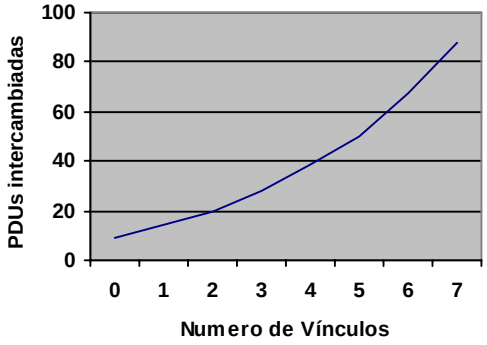

***Fig. 7.3. Cantidad de PDUs intercambiadas para la reconstrucción del árbol de distribución**: en función del número de vínculos del subárbol* aislado

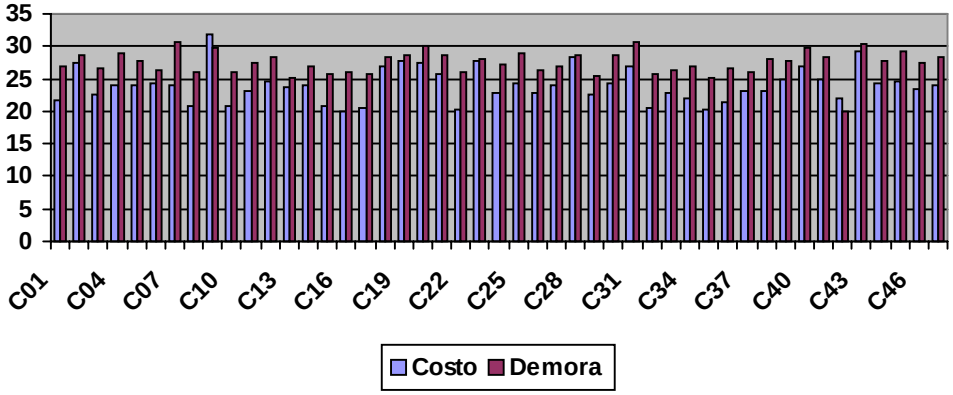

Fig 7.4. Relación entre costos y demoras promedio para cada nodo actuando como core

La performance de la implementación, puede considerarse aceptable, demandando en un Pentium 100MHz con sistema operativo Linux, aproximadamente 0,66 seg por cada segundo simulado con tasas bajas de envío por parte de los emisores[49], incluyendo tiempos de

[49] Debe notarse que para los objetivos establecidos no fue necesario un envío masivo de datos ni provocar carga en la red a través de los agentes provistos por Ns.

inicialización de nodos, vínculos y agentes CBT y produciendo mensajes de debugging por salida standard.

Debe tenerse en cuenta al considerar estos valores, que se trata de una implementación detallada del protocolo. Los tiempos de simulación crecen a medida que se agregan agentes multicast o se incrementa la tasa de paquetes enviados, resultando viables para los valores utilizados (10 paquetes/segundo por emisor, 10 emisores) en la topología descripta.

Los altos tiempos de simulación obtenidos utilizando el algoritmo de ruteo Distance Vector no centralizado provisto por Ns, no hicieron posible en la topología descripta el análisis de la interacción entre ruteo unicast y multicast.

El script de simulación utilizado para el análisis descripto en este capítulo, un ejemplo de la salida producida y las tablas con los valores obtenidos en las diferentes corridas se presentan en el Apéndice C.

7.4 Resumen

En este capítulo se presentó un caso simple de utilización de la implementación CBT desarrollada, que permitió determinar la factibilidad de su uso para analizar aspectos del protocolo debido a su performance aceptable. Fue de suma importancia para facilitar la recopilación de resultados la salida producida por las facilidades de debugging implememtadas, procesadas posteriormente por awk.

Bibliografía

[1] S. Deering, "Host Extensions for IP Multicasting", RFC 1112, August 1989. http://www.isi.edu.
[2] W. Fenner, “Internet Group Management Protocol, Version 2”, RFC 2236, November 1997, http://www.isi.edu.
[3] S. Deering, A.Thyagarajan, “Internet Group Management Protocol, Version 3”, Internet Draft, November 1997, http://www.isi.edu.(draft-ietf-idmr-igmp-v3-00.txt).
[4] S. Deering, C. Partridge, D. Waitzman, “Distance Vector Multicast Routing Protocol”, RFC 1075, November 1988. http://www.isi.edu.
[5] S. Deering, D. Estrin, D. Farinacci, Van Jacobson, A. Helmy, L. Wei, “Protocol Independent Multicast Version 2, Dense Mode Specification”, Internet Draft, May 1997, http://www.isi.edu.(draft-ietf-idmr-pim-dm-05.txt).
[6] A. Ballardie, "Core based Trees (CBT) Multicast Routing Architecture", RFC 2201, September 1997. http://www.isi.edu.
[7] A. Ballardie, "Core Based Trees (CBT version 2) Multicast Routing. Protocol Specification", RFC 2189, September 1997. http://www.isi.edu.
[8]A. Ballardie, "Core Based Trees (CBT) Multicast Routing MIB, Internet Draft, July 1997. http://www.isi.edu. (draft-ietf-idmr-cbt-mib-00.txt).
[9] Estrin, D. et. al., "Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification", RFC 2117, June 1997. http://www.isi.edu.
[10] T. Pusateri, “Distance Vector Multicast Routing Protocol”, Internet Draft, March 1998, http://www.isi.edu.(draft-ietf-idmr-dvmrp-v3-06.txt).
[11] J. Moy, "Multicast Extensions to OSPF", RFC 1584, March 1994. http://www.isi.edu
[12] A. Ballardie, B. Cain, Z. Zhang, “Core Based Trees (CBT version 3) Multicast Routing. Protocol Specification”, Internet draft, March 1998. http://www.isi.edu.(draft-ietf-idmr-cbt-spec-v3-00.txt).
[13] K. Varadhan, K. Fall, "Ns Notes and Documentation", (VINT Project), UC Berkeley, May 1998. http://www.isi.edu/~kannan/nsDoc.ps.gz
[14] D. Wetherall, C. Lindblad, “Extending Tcl for Dynamic Object-Oriented Programming”, Proceedings of the Tcl/Tk Workshop 95, Toronto, Ontario, July 1995.
[15] D. Wetherall, “Otcl Tutorial, version 0.96”, September 1995, distribuido con versión Ns 2.0 (otcl 1.0b4).
[16] J. Osterhoust, “An Introduction to Tcl and Tk”, Addison Wesley, 1994.
[17] K. Mc Cloghrie, D. Farinacci, D. Thaler, “IP Multicast Routing MIB”, Internet Draft, November 1997, http://www.isi.edu, (draft-ietf-idmr-multicast-routmib-06.txt).

Bibliografía Comentada

[1] S. Deering, "Host Extensions for IP Multicasting", RFC 1112, August 1989. http://www.isi.edu.
Describe el protocolo IGMP y las interacciones que a través de él tienen lugar entre un router y los hosts locales a efectos de que el primero obtenga información acerca de los grupos multicast a los cuales debe responder. Es un documento básico en multicasting en la Internet. Fue utilizado para definir la ubicación de las entidades IGMP en los routers y su comunicación con el agente CBT. Dichas entidades (no implementadas) son vistas por la entidad CBT como si generaran requerimientos de integración o abandono de un grupo (join y leave) de la misma manera en que lo hacen las aplicaciones locales. Se utilizó como lectura introductoria.

[2] W. Fenner, “Internet Group Management Protocol, Version 2”, RFC 2236, November 1997, http://www.isi.edu.
Constituye una nueva versión de [1], conteniendo modificaciones entre las cuales se destaca la menor latencia presentada por el protocolo respecto del abandono de un grupo. Se utilizó como lectura de contexto.

[3] S. Deering, A.Thyagarajan, “Internet Group Management Protocol, Version 3”, Internet Draft, November 1997, http://www.isi.edu.(draft-ietf-idmr-igmp-v3-00.txt).
Es la tercera versión de IGMP, en estado borrador, que incorpora la capacidad que permite que un sistema solicite recibir o filtrar información de determinados emisores. Se utilizó como lectura de contexto.

[4] S. Deering, C. Partridge, D. Waitzman, “Distance Vector Multicast Routing Protocol”, RFC 1075, November 1988. http://www.isi.edu.
Constituye la descripción original de DVMRP. Desde dicha especificación, se han ido produciendo cambios menores, actualmente reflejados en documentos drafts. Se utilizó como lectura introductoria.

[5] S. Deering, D. Estrin, D. Farinacci, Van Jacobson, A. Helmy, L. Wei, “Protocol Independent Multicast Version 2, Dense Mode Specification”, Internet Draft, May 1997, http://www.isi.edu.(draft-ietf-idmr-pim-dm-05.txt).
Contiene una descripción general y una especificación de la operación del protocolo, haciendo referencia a la especificación de PIM-SM [9] para el formato de los mensajes del protocolo. Fue utilizado como lectura introductoria.

[6] A. Ballardie, "Core based Trees (CBT) Multicast Routing Architecture", RFC 2201, September 1997. http://www.isi.edu.
Presenta una visión general del protocolo CBT, justificando su desarrollo a través de argumentos que tienen en cuenta su simplicidad y escalabilidad. Presenta en forma sintética los elementos del protocolo, que son tratados en mayor detalle en el documento de especificación. Fue fundamental para definir la propuesta de trabajo.

[7] A. Ballardie, "Core Based Trees (CBT version 2) Multicast Routing. Protocol Specification", RFC 2189, September 1997. http://www.isi.edu.
Define en forma detallada los elementos y operación del protocolo CBT versión 2. Se definen los timers utilizados por el protocolo, las PDUs intercambiadas y la operación detallada en lenguaje natural. Constituyó un elemento importante para definir la propuesta de trabajo y fue utilizado posteriormente para definir en forma más precisa la operación del protocolo, definición en la cual se basó la implementación.

[8]A. Ballardie, "Core Based Trees (CBT) Multicast Routing MIB, Internet Draft, July 1997. http://www.isi.edu. (draft-ietf-idmr-cbt-mib-00.txt).
En este documento se define (hasta el estado de draft mencionado arriba) parcialmente los objetos de la MIB CBTv2. Fue utilizado fundamentalmente el grupo cbtInterfaceGroup para definir la estructura de datos a ser mantenida por la clase CBTInterface. No se utilizaron los grupos cbtBootstrapGroup ni cbtBorderGroup debido a los alcances del trabajo.

[9] Estrin, D. et. al., "Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification", RFC 2117, June 1997. http://www.isi.edu.
Define el protocolo PIM-SM, detallando su manera de operar, el cambio de árboles de distribución compartidos a árboles con raíz en el emisor, timers, etc. Fue utilizado como lectura introductoria y luego para comprender las diferencias entre este protocolo y CBT (por ejemplo la direccionalidad de los árboles de distribución).

[10] T. Pusateri, “Distance Vector Multicast Routing Protocol”, Internet Draft, March 1998, http://www.isi.edu.(draft-ietf-idmr-dvmrp-v3-06.txt).
Es una actualización a la definición de DVMRP [4], que incluye modificaciones en el aspecto túneles y en cuanto a la transmisión de mensajes de poda del árbol en forma confiable. Se utilizó como lectura de contexto, complementando a [4].

[11] J. Moy, "Multicast Extensions to OSPF", RFC 1584, March 1994. http://www.isi.edu.
Define mecanismos adicionales a ser incorporados a OSPF para soportar la diseminación de información respecto a grupos multicast a efectos de que los routers puedan intercambiar la información multicast que reciben y en base a ella y a la aportada por OSPF construir los árboles de distribución. Se utilizó como lectura introductoria.

[12] A. Ballardie, B. Cain, Z. Zhang, “Core Based Trees (CBT version 3) Multicast Routing. Protocol Specification”, Internet draft, March 1998. http://www.isi.edu.(draft-ietf-idmr-cbt-spec-v3-00.txt).
En este documento se define una nueva versión de CBT. En él se especifican, además de algunas modificaciones menores (mejora del protocolo Hello), el agregado de funcionalidad (tipos de entradas soportados y prefijos de ruteo) para un mejor comportamiento del protocolo en dominios de tránsito. Se contempla continuar el trabajo en el aspecto implementación, para soportar estas características.

[13] K. Varadhan, K. Fall, "Ns Notes and Documentation", (VINT Project), UC Berkeley, May 1998. http://www.isi.edu/~kannan/nsDoc.ps.gz
Describe las facilidades provistas por el simulador utilizado y la jerarquía de clases que lo conforma. Fue utilizado como referencia durante el desarrollo de la implementación.

[14] D. Wetherall, C. Lindblad, “Extending Tcl for Dynamic Object-Oriented Programming”, Proceedings of the Tcl/Tk Workshop 95, Toronto, Ontario, July 1995.
En este artículo se describen las mejoras introducidas a Tcl para soportar programación dinámica orientada a objetos y algunos cambios menores en la sintaxis (respecto a Tcl). Se utilizó como referencia durante el desarrollo de la implementación.

[15] D. Wetherall, “Otcl Tutorial, version 0.96”, September 1995, distribuido con versión Ns 2.0 (otcl 1.0b4).
Contiene detalles de implementación de Otcl no cubiertos en [14], tales como autoloading de clases y el API C provisto para manipular objetos y clases Otcl. Fue utilizado como lectura complementaria a [14].

[16] J. Osterhoust, “An Introduction to Tcl and Tk”, Addison Wesley, 1994.
En este libro se presenta el lenguaje Tcl y su soporte gráfico Tk. Los capítulos que describen Tcl fueron utilizados como referencia al lenguaje durante el desarrollo de la implementación.

[17] K. Mc Cloghrie, D. Farinacci, D. Thaler, “IP Multicast Routing MIB”, Internet Draft, November 1997, http://www.isi.edu, (draft-ietf-idmr-multicast-routmib-06.txt).
En este documento se define una MIB general para los protocolos de ruteo multicast, que es completada para cada protocolo por su correspondiente MIB. Se lo utilizó en forma conjunta con la definición de la MIB para CBTv2, siendo el aspecto utilizado la definición de la conformación de las tablas de ruteo multicast.

Apéndice A: Código

Se muestra a continuación el código implementado, que fue organizado de la siguiente manera en archivos tcl:

- **cbtmain.tcl**: Contiene referencias a los demás archivos para facilitar la carga del código con una única sentencia: "source cbtmain.tcl".
- **cbt.tcl**: Contiene procedimientos de la clase CBT para interacción con las demás clases implementadas y el soporte provisto por el simulador.
- **timer.tcl**: Contiene la clase base *CBTTimer* y las derivadas que implementan los timers específicos.
- **agents.tcl**: Contiene las clases que definen los distintos agentes (derivados de *Agent/Message*): agente de emisión/recepción asociado a CBT, agentes de encapsulación y desencapsulación y algunos agentes para prueba del protocolo (*Prueba*, *Emisor_delay*, *Receptor_delay*).
- **simul.tcl**: Contiene métodos agregados a la clase *Simulator*, *map-tree*, *set-core*, etc.
- **debug.tcl**: Contiene métodos de varias de las clases relativos a las funciones de debugging implementadas.
- **recvr.tcl**: Contiene métodos CBT destinados a procesar las PDUs recibidas y al envío de PDUs a otros nodos.
- **interface.tcl**: Código asociado a las interfaces.
- **mfc.tcl**: Código correspondiente a las tablas transitoria y permanente mantenidas en cada nodo.

A.1 cbtmain.tcl

```
source cbt.tcl
source agents.tcl
source timer.tcl
source simul.tcl
source mfc.tcl
source recvr.tcl
source interface.tcl
source debug.tcl
```

A.2 cbt.tcl

```
Class CBT -superclass McastProtocol

# Definicion de constantes

# Tipos de PDU
CBT set HELLO        0
CBT set JOIN_REQUEST    1
CBT set JOIN_ACK     2
CBT set QUIT_NOTIFICATION   3
CBT set ECHO_REQUEST    4
CBT set ECHO_REPLY      5
CBT set FLUSH_TREE      6
```

```
# Timers and contadores (si son redefinidos tener en cuenta que
algunos
# estan expresados en funcion del valor de otros). Los tiempos en
segundos
CBT set HOLDTIME     3
CBT set RTX_INTERVAL     5
CBT set ECHO_INTERVAL    60
CBT set EXPECTED_REPLY_TIME 70
CBT set HELLO_INTERVAL      60
CBT set JOIN_TIMEOUT     [expr [CBT set RTX_INTERVAL] * 3.5]
CBT set TRANSIENT_TIMEOUT   [expr [CBT set RTX_INTERVAL] * 1.5]
CBT set CACHE_DEL_TIMER     [expr [CBT set HOLDTIME] * 1.5]
CBT set GROUP_EXPIRE_TIME   [expr [CBT set ECHO_INTERVAL] * 1.5]
CBT set HELLO_PREFERENCE 255
CBT set MAX_RTX          3


# Direccion multicast routers CBT
CBT set ALL_CBT_ROUTERS 0xFFCF


# Instanciacion de la entidad CBT en el nodo
CBT instproc init { sim node } {
   $self next
   $self instvar ns Node type enbl
   set ns $sim
   set Node $node
   set type "CBT"
   #Inicializa elementos que usara el agente (interfaces, tablas, etc)
   $self initialize
}


# Inicializacion de grupos en el nodo, agente de desencapsulacion,
# interfaces, tablas transitoria y permanente, agente de transmision y
# modulo para debugging
CBT instproc initialize { } {
   $self instvar ns Node messager  decapag joined_groups MFCTable
   $self instvar echorq TransientTable interfaces
   set joined_groups ""
   [$Node getArbiter] addproto $self
   set decapag [new decap]
   $Node attach $decapag
   #Inicializa Tabla Multicast Forwarding Cache
   set MFCTable [new MFC_Table $self]
   #Inicializa el objeto administrador de echo requests
   set echorq [new Echorq_adm $ns $Node $self]
   #Inicializa Tabla Multicast Transient Forwarding Cache
   set TransientTable [new Transient_Table $self]
   #Inicializa interfaces
   $self init_interfaces
   #Agentes messager(intercambio de PDUs) y decapag (decapsulacion)
   set messager [new Agent/Message/CBT]
   $Node attach $messager
   $messager set proto $self
   $messager set dst_ [CBT set ALL_CBT_ROUTERS]
   $Node join-group $messager [CBT set ALL_CBT_ROUTERS]
   #Inicializa modulo para debugging
   $self init_debug
}
```

```
# Arranque del protocolo, variable status en up (McastProtocol).
CBT instproc start {} {
   $self next
}


CBT instproc stop {} {
        $self instvar MRTArray groupArray cacheByGroup sourceArray
        if [info exists MRTArray] {
                unset MRTArray
        }
        if [info exists groupArray] {
                unset groupArray
        }
        if [info exists cacheByGroup] {
                unset cacheByGroup
        }
        if [info exists sourceArray] {
                unset sourceArray
        }
}



##########################################################################
###
# Metodos invocados por la parte de reenvio (upcall)

# Upcall invocado con code, srcID, group, interfaz de entrada
# Es invocado por cache_miss (no existe replicator para el par source
group o
# por wrong_iif (existe el par pero la interfaz de arribo es diferente
a la
# asociada con el replicator.
CBT instproc upcall { argslist } {
        set code [lindex $argslist 0]
        set argslist [lreplace $argslist 0 0]
        switch $code {
                "CACHE_MISS" { $self handle-cache-miss $argslist }
                "WRONG_IIF" { $self handle-wrong-iif $argslist }
                default { puts "codigo de upcall desconocido, $code" }
        }
}


# Para compatibilidad con las funciones del nodo (add-mfc, y del-mfc,
sobre
# todo). Se guarda aqui una lista propia de src, grp, iif para cada
entrada
# que se crea
CBT instproc handle-cache-miss { argslist } {
#  $self instvar sources groups iifs
   $self instvar ns interfaces
   $self instvar rep rep_obj rep_int
        set srcID [lindex $argslist 0]
        set group [lindex $argslist 1]
        set iface [lindex $argslist 2]
        $self instvar Node MFCTable
   #Para grupo CBT manda siempre
   if { $group == [CBT set ALL_CBT_ROUTERS] } {
      #Dada la interfaz de arribo, determina la totalidad de las
```

```
      #interfaces restantes (labels) en el nodo, (outifaces) para
crear los
      #elementos de forwarding para ellas.
      set outifaces ""
      foreach interf [$Node set ifaces_] {
              set la [$interf set id]
              if { $la != $iface } {
                 set tt [$interf set ifaceout]
                 lappend outifaces $tt
              }
           }
      if {![info exists sources]} {
         set sources ""
         set groups ""
         set iifs ""
      }
      #si alguna interfaz es un dynamic link, en lugar de objeto
      #networkinterface debe ponerse el objeto DynaLink
      set cnt 0
      foreach x $outifaces {
         set y [$Node set neighbor_]
         foreach yy $y {
            set xx [$ns set link_([$Node id]:[$yy id])]
            set in [$xx set ifacein_]
            if {$in == $x} {
               set he [$xx set head_]
                    set cl [$he info class]
                    if {$cl == "DynamicLink" } {
                       set outifaces [lreplace $outifaces $cnt $cnt
$he]
               }
            }
         }
         incr cnt 1
      }
      #Agregado del elemento de forwarding
      $Node add-mfc $srcID $group $iface $outifaces
      return
   }

   #Grupo diferente a CBT routers. En este caso, en funcion de la MFC
   #determina las interfaces para el grupo. Luego crea el replicators
   #para S,G y le asocia la interfaz de llegada del paquete; por
ultimo
   #inserta las demas ifaces del grupo como targets del replicator (lo
   #hace utilizando add-mfc).
   #Debe ademas insertarse el nuevo elemento (iface de arribo) en
   #TODOS los replicators para el grupo, para que se le envie info
   #Esto no se puede hacer data driven, entonces lo hace al agregar
   #la entrada en la tabla a traves del proc CBT actualizar-
replicators
   #Si la interfaz de arribo del paquete no corresponde a alguna de
las
   #del grupo, ignora y sale sin actualizar elementos de reenvio.
   set ent [$MFCTable get_entry $group]
   #No existe grupo en tabla, ignora
   if {$ent == -1 } {return}
   #Obtiene interfaces (iflist) y nexthops (iflist1) para el grupo
   set cc [$self get_ifaces $group]
        set iflist [lindex $cc 0]
   set iflist1 [lindex $cc 1]
```

```
   #Existe la iif de entrada para el grupo?
   set ex [lsearch -exact $iflist $iface]
   if {$ex == -1} {return}
   set iflist [lreplace $iflist $ex $ex]
   set iflist1 [lreplace $iflist1 $ex $ex]
   #Debe eliminarse todas las interfaces iguales, porque en un link
   #multiacceso se puede tener mas de una (en el caso de un  DR). Si
se
   #deja mas de una, el DR generara una copia adicional del paquete
   set lx [lsearch -exact $iflist $iface]
   while {$lx != -1} {
      set iflist [lreplace $iflist $lx $lx]
      set iflist1 [lreplace $iflist1 $ex $ex]
      set lx [lsearch -exact $iflist $iface]
   }
   #En olist, se colocan los objetos interface (el head_ del link)
   set olist ""
   foreach xx $iflist {
      set ifo [$self lbl2if $xx]
      lappend olist $ifo
   }
   #Para el caso de links unicast, debe reemplazarse el objeto
   #networkinterface por un agente encap. Para acceder a la info
   #respecto del tipo de link, se debe recurrir a la informacion
   #almacenada en cada objeto interfaz (CBT)
   #(Tener en cuenta que en en iflist estan los labels)
   set cnt 0
   foreach inf $iflist {
      #Pregunta por label -2, interfaz local
      if {$inf != -2} {
             set inf1 $interfaces($inf)
         set pp [$inf1 get_linktype]
         if {$pp == 3 || $pp == 4} {
            set nxhop [lindex $iflist1 $cnt]
            set kk [lindex $olist $cnt]
            #Si no existe el agente encap para S,G,iif,oif,nhop, se
crea
            if ![info exists ag($srcID:$group:$iface:$inf:$nxhop)] {
                   set ag($srcID:$group:$iface:$inf:$nxhop) [new
encap $srcID $group $iface $inf $nxhop $Node]
               #puts "Nodo [$Node id] crea
ag($srcID:$group:$iface:$inf:$nxhop)"
               $ns attach-agent $Node
$ag($srcID:$group:$iface:$inf:$nxhop)
               #Reemplaza en olist el objeto por el agente.
               set olist [lreplace $olist $cnt $cnt
$ag($srcID:$group:$iface:$inf:$nxhop)]
                }
         }
      }
      incr cnt 1
   }
   #Si alguna interfaz es un dynamic link, debe reemplazarse
   #el objeto networkinterface por el dynamiclink.
   set cnt 0
   foreach x $olist {
      set c ""
      if {$x != -1} {
        set c [$x info class]
      }
```

```
      #Si es un encap, la salida va unicast, no es necesario testear
el
      #target del replicator
      if {$c != "encap" } {
         set y [$Node set neighbor_]
         foreach yy $y {
            set xx [$ns set link_([$Node id]:[$yy id])]
            set in [$xx set ifacein_]
            if {$in == $x} {
               set he [$xx set head_]
               set cl [$he info class]
               if {$cl == "DynamicLink" } {
                  set olist [lreplace $olist $cnt $cnt $he]
               }
            }
         }
      }
      incr cnt 1
   }
   #Actualizacion de los elementos de reenvio, crea nuevo replicator
   #con los targets
   #Aca es necesario conocer el objeto replicator agregado. Como
   #add-mfc no lo devuelve, se compara una lista previa con la actual
   set previa [$Node array names replicator_]
   $Node add-mfc $srcID $group $iface $olist
   set actual [$Node array names replicator_]
   #Se supone que se agrega un replicator
   set ind [llength previa]
   foreach a $actual {
      set esta 0
      foreach p $previa {
         if {$a == $p} {set esta 1}
      }
      if {$esta == 0} break
   }
   set nrep [$Node set replicator_($a)]
   #Actualizacion de los elementos usados por CBT
   if {![info exists rep($srcID:$group:$iface)]} {
      set rep($srcID:$group:$iface) $nrep
      set rep_obj($srcID:$group:$iface) $olist
      set rep_int($srcID:$group:$iface) $iflist
   }
   #No else, si entro por miss no habia replicator
}


# En este caso se recibe un paquete por s,g por una interfaz diferente
a la
# asociada al replicator. Si bien en CBT no se tiene informacion de
source
# en las tablas, el replicator no puede procesar (reenviar) el paquete
ya
# que se produciria un loop, porque entre las interfaces de salida
podria
# estar la de arribo. Se decide descartar el paquete.
# Normalmente esta situacion no se podria producir en CBT.
CBT instproc handle-wrong-iif {argslist} {
        $self instvar Node MFCTable
   $self instvar sources groups iifs ns interfaces
   $self instvar rep rep_obj rep_int
        set srcID [lindex $argslist 0]
```

```
        set group [lindex $argslist 1]
        set iface [lindex $argslist 2]
   puts "Node [$Node id]  handle-wrong-iif. src:$srcID  gr: $group
iface: $iface"
   #Se debe cambiar la interfaz de arribo. Para ello se busca s,g y se
   #obtiene la interfaz anterior, luego se invoca al cambio con la
   #nueva (la que se informa al handle), si es que esta pertenece a la
   #entrada para el grupo
   #Si nueva interfaz (iface) no esta en el grupo, retorna
   #Obtiene interfaces (iflist) y nexthops (iflist1) para el grupo
   set cc [$self get_ifaces $group]
        set iflist [lindex $cc 0]
   set iflist1 [lindex $cc 1]
   set ex [lsearch -exact $iflist $iface]
   if {$ex == -1} {return}
   #Si nueva interfaz esta en el grupo
   #Busca el par src,grp y obtiene interfaz
   set repli array names rep
   foreach e $repli {
      set xx [split e :]
      set ss [lindex $xx 0]
      set gg [lindex $xx 1]
      set ii [lindex $xx 2]
      if { $ss == $srcID && $gg == $group } break
   }
   set oldif $ii
   set mc [$Node set multiclassifier_]
   $mc change-iface $srcID $group $oldif $iface
   #Cambia la info CBT
   set rep($srcID:$group:$iface) $rep($srcID:$group:$oldif)
   unset rep($srcID:$group:$oldif)
   set rep_obj($srcID:$group:$iface) $rep_obj($srcID:$group:$oldif)
   unset rep_obj($srcID:$group:$oldif)
   set rep_int($srcID:$group:$iface) $rep_int($srcID:$group:$oldif)
   unset rep_int($srcID:$group:$oldif)
   #Si la nueva interfaz de arribo estaba como salida en el rep, debe
   #anularse, de lo contrario ocurririan loops
   set cnt 0
   foreach oif $rep_int($srcID:$group:$iface) {
      if {$oif == $iface} {
      #Si iface == -2, reemplazo?
         set xoif [lindex $rep_obj($srcID:$group:$iface) $cnt]
         set rep_int($srcID:$group:$iface) [lreplace
$rep_int($srcID:$group:$iface) $cnt $cnt]        set rep_obj
         set rep_obj($srcID:$group:$iface) [lreplace
$rep_obj($srcID:$group:$iface) $cnt $cnt]
         $Node del-mfc $srcID $group $xoif
         break
      }
      incr cnt 1
   }
}



#Procedimiento invocado por mfc al agregar una entrada child para un
grupo.
#El nuevo replicator (asociado a la nueva interfaz) sera creado cuando
se
#produzca un cache-miss. Pero hay que actualizar todos los demas
replicators
```

```
#para el grupo, ya que el agregado de un nuevo target en el replicator
no
#es disparado por llegada de datos
CBT instproc actualizar-replicators {group ch_interface} {
        $self instvar Node MFCTable
    $self instvar sources groups iifs ns interfaces
    $self instvar rep rep_obj rep_int
    set oif [$self lbl2if $ch_interface]
    set todos [$self array names rep]
        set y [$Node set neighbor_]
    foreach yy $y {
       set xx [$ns set link_([$Node id]:[$yy id])]
       set in [$xx set ifacein_]
       if {$in == $oif} {
          set he [$xx set head_]
          set cl [$he info class]
          if {$cl == "DynamicLink" } {
             set oif  $he
          }
       }
         }
    if {[llength $todos] >= 1} {
       foreach elem $todos {
              set pp [split $elem :]
          set ss [lindex $pp 0]
          set gg [lindex $pp 1]
          set ii [lindex $pp 2]
               if {$gg == $group} {
             set rrp [$Node set replicator_($ss:$gg)]
                  if {$oif != -1} {
                $rrp insert $oif
                #actualiza CBT
                set ax rep_obj($elem)
                set ax [lappend $ax $oif]
                set rep_obj($elem) $ax
                set ax rep_int($elem)
                set ax [lappend $ax $ch_interface]
                set rep_int($elem) $ax
             }
             #Que pasa si provenia de un join-group, aca queda una
             #unica entrada pero el replicator puede tener mas de
             #una ya que es el join del nodo el que inserta el nuevo
             #target. Tampoco guardo aqui info del agente.Quiza no se
         #use
               if {$ch_interface == -2} {
            set ax rep_obj($elem)
                set ax [lappend $ax $oif]
                set rep_obj($elem) $ax
                set ax rep_int($elem)
                set ax [lappend $ax $ch_interface]
                set rep_int($elem) $ax
             }
               }
          }
    }
}



CBT instproc drop {rep src dst} {
    $self instvar Node
```

```
   puts "Node [$Node id]  drop.  replicator: $rep source: $src dest:
$dst"
}


#Dado un grupo, devuelve una lista con las interfaces involucradas
(sus
#labels) y otra con los nexthops. Como es a efectos de distribuir la
#informacion, no tiene en cuenta parent ni child.
CBT instproc get_ifaces { group } {
        $self instvar Node MFCTable
   #determina interfaz (cld) y nexthop (cld1) parent.
   set cld ""
   set pnt [$MFCTable get_parent $group]
   lappend cld [lindex $pnt 1]
   set cld1 ""
   lappend cld1 [lindex $pnt 2]
   #Si el nodo es core (interfaz -1) , no hay interfaz parent
   if {[lindex $cld 0] == -1 } {
           set cld ""
      set cld1 ""
       }
   #Determina la(s) interfaces y nexthop(s) child
   set iflist [$MFCTable get_child_if $group]
   set iflist1 [$MFCTable get_child_nh $group]
   if { [lindex $iflist 0] != -1} {
      if {[llength $cld] > 0}  {
         lappend iflist $cld
         lappend iflist1 $cld1
      }
   }
   set nlist [list $iflist $iflist1]
   return $nlist
}


##########################################################################
###
#Procedimientos para join y leave

#Este procedimiento es invocado cuando un agente realiza join a un
grupo.
#Es invocado luego del join del nodo. Si el grupo es CBT,no hace el
join ya
#no es necesario arbol de distribucion
#En el caso de IGMP, se debera hacer a traves de un agente local
CBT instproc join-group {group} {
        $self instvar joined_groups Node
   if { $group == [CBT set ALL_CBT_ROUTERS] } { return 0}
        if { [lsearch -exact $joined_groups [expr $group]] == -1 } {
      #Agregado de la entrada leaf
      set r [$self add-leaf $group -2]
      if {$r == 0} {
         lappend joined_groups [expr $group]
              puts "Grupo $group joined."
      }
   }
   puts "Grupo $group, ya joined."

}
```

```
CBT instproc add-leaf {group interface} {
   $self instvar ns interfaces MFCTable TransientTable qntimer
   set gr [expr $group]
   #Nodo es on-tree para el grupo, agrega child
   if { [$MFCTable exists_entry $gr] != -1 } {
      #Ver si hay que validar una relacion parent/leaf-child
      $MFCTable add_child $gr $interface -1
      return 0
   }
   #Nodo no es on-tree para el grupo
   #Existe entrada transitoria para el grupo
   if { [$TransientTable exists_entry $gr] != -1 } {
      #Ver si hay que validar relacion parent/leaf-child
      $TransientTable add_child $gr $interface -1
      return 0
   } else {
   #No existe entrada transitoria
      #Obtiene core para el grupo, por ahora del simulador, esto
      #dependera de si se implementa seteo estatico de los cores o
      #de si hay un procedimiento de  bootstrapping
      #Obtiene nexthop para el core (next_router_to_core)
      set core [$ns get-core $gr]
      if {$core == -1 } {
         puts "No es posible encontrar core para grupo $gr"
         return -1
      }
      set lst [$self next_router $core]
      #No hay ruta al core
      if { [lindex $lst 0] == 0 } {
         return -1
      }
      set iface [lindex $lst 2]
      set nxthop [lindex $lst 1]
      #Si link no mcast, el nexthop es el id del nodo.
      #Si link mcast y DR, el nexhop es el id del nodo
      #Si link mcast y no DR, nexthop es *
      set tpl [$interfaces($iface) get_linktype]
      set dr [$interfaces($iface) is_dr]
      if { $tpl <= 2 &&  $dr != 1} {
            set nxthop  *
      }
      #Ver si hay que validar la relacion parent/child
      #Nodo es leaf, crea asi la entrada transitoria
      set leaf 1
      #Si hay Quit_Notification_Timer para la entrada, lo elimina
      if [info exists qntimer($gr:$iface:$nxthop)] {
         $qntimer($gr:$iface:$nxthop) cancel
      }
      #Crea entrada transitoria
      $TransientTable add_entry $gr $core $iface $nxthop $leaf
      #Agrega child
      $TransientTable add_child $gr $interface -1
      return 0
   }
}


# Invocado por mcastproto cuando un agente deja un grupo. Si no quedan
mas
# agentes locales para el grupo, borra child.
CBT instproc leave-group {group} {
```

```
   $self instvar Node joined_groups
   set ag [$Node get_Agents $group]
   if { [llength $ag] == 0 } {
      set idx [lsearch -exact $joined_groups [expr $group]]
      if {$idx > -1} {
         set joined_groups [lreplace $joined_groups $idx $idx]
         $self delete-leaf $group -2
         puts "Leaving group $group"
      }
   }
}

CBT instproc delete-leaf {group interface} {
   $self instvar MFCTable
   set gr [expr $group]
   if { [$MFCTable exists_entry $gr] != -1 } {
      $MFCTable delete_child $gr $interface -1
      return
   }
}


##############################################################################
####
# Interaccion con el agente para envio de PDUs
# Resuelve la menera de invocar al agente de mensajes CBT, en base al
tipo
# de envio.
CBT instproc send_msg { msg lbloif type dest} {
   $self instvar messager ns Node
   #En primer termino determina el target del messager
(DynamicLink,etc)
        set oif [$self lbl2if $lbloif]
   set x $oif
   set y [$Node set neighbor_]
   foreach yy $y {
      set xx [$ns set link_([$Node id]:[$yy id])]
      set in [$xx set ifacein_]
      if {$in == $x} {
         set he [$xx set head_]
              set cl [$he info class]
              if {$cl == "DynamicLink" } {
                 #set olist [lreplace $olist $cnt $cnt $he]
            set oif $he
              }
          }
   }
   $messager target $oif
   #En segundo termino, determina el destino del messager y su ttl
   if {$type == "M"} {
          $messager set ttl_ 2
      $messager set dst_ [CBT set ALL_CBT_ROUTERS]
   } else {
          $messager set ttl_ 32
          set n [$ns set Node_($dest)]
          set ar [$n set mcastproto_]
          set cb [$ar getType CBT]
          set agen [$cb set messager]
          $ns connect $messager $agen
   }
   #Por ultimo envia PDU
```

```
    $messager transmit $msg
}



##########################################################################
####
#Metodos para configuracion

#Configuracion de Designated Router en vinculos multiacceso
#Se pasa, un nodo de la red, para individualizar la interfaz, y el
#designated router. Esto debe hacerse p/c nodo en un maccess,
manualmente
#Lo deberia hacer a traves de Hello

#Configura el DR en la interfaz del nodo hacia dstnode
CBT instproc set_des_router { dstnode des_router } {
    $self instvar interfaces
    set lab [$self label_to_neighbor $dstnode]
    if {$lab == -1} { return }
    $interfaces($lab) set_des_router $des_router
}

#Devuelve el DR de una interfaz identificada por su label
CBT instproc get_des_router {lbl} {
    $self instvar interfaces
    if { ![info existe interfaces($lbl)] } {
        return -1
    }
    return [$interfaces($lbl) get_des_router]
}

#Configuracion de vinculos solo con capacidad unicast
#Configura el vinculo desde el nodo hacia dstnode como unicast (debe
#configurarse independientemente el otro nodo -dstnode-)
CBT instproc set_unicast { dstnode }  {
    $self instvar interfaces
    set lbl [$self label_to_neighbor $dstnode]
    set lkt [$interfaces($lbl) get_linktype]
    if {$lkt == 1 || $lkt == 2} {
        incr lkt 2
        $interfaces($lbl) set_linktype $lkt
    }
}


#Configuracion de cores
#Convierte al CBT del nodo en core para el grupo dado.
CBT instproc set-core { group } {
    $self instvar  Node ns MFCTable
    #Lista global de cores
    $ns set-core $group [$Node id]
    set gr [expr $group]
    $MFCTable add_core_entry $gr [$Node id]
}



#Protocolo Hello no implementado
CBT instproc recv-hello { msg } {
```

```
   puts "PROTOCOLO CBT $self : RECIBE  HELLO, no implementado"

}



#########################################################################
######
# Otros metodos



CBT instproc get_simulator {} {
   $self instvar ns
   return $ns
}

#Devuelve la multicast forwarding cache del agente CBT
CBT instproc get_MFCTable {} {
   $self instvar MFCTable
   return $MFCTable
}


#Devuelve la multicast transient forwarding cache del agente CBT
CBT instproc get_TMFCTable {} {
   $self instvar TMFCTable
   return $TMFCTable
}

#Devuelve el objeto que administra los timers para echo request
CBT instproc get_Echorq {} {
   $self instvar echorq
   return $echorq
}


#Devuelve el id del nodo
CBT instproc get_nodeid {} {
   $self instvar Node
   return [$Node id]
}


#Dado un rotulo de interfaz, retorna el objeto networkinterface de
salida
CBT instproc lbl2if {lbl} {
   $self instvar Node
   foreach ifx [$Node set ifaces_] {
     set ifc [$ifx set iface]
     set lb [$ifc set intf_label_]
     if {$lb == $lbl} { return [$ifx set ifaceout]}
   }
        return -1
}


#Dado un neighbor, devuelve el label de la interfaz que va hacia el
CBT instproc label_to_neighbor {neighbor} {
   $self instvar Node ns
   set ne [$Node set neighbor_]
```

```
   set cnt [llength $ne]
   if {$cnt > 0} {
      set nids ""
      set idx 0
      while {$idx < $cnt} {
         set nd [lindex $ne $idx]
         lappend nids [$nd id]
         incr idx 1
      }
   }
   set x [lsearch -exact $nids $neighbor ]
   if {$x < 0} {
      puts "Node $neighbor no es neighbor"
      return -1
   }
   set lnk [$ns set link_([$Node id]:$neighbor)]
   set ifn [$lnk set ifacein_]
   set lbl [$ifn set intf_label_]
   return $lbl
}


#Inicializa las interfaces del nodo
CBT instproc init_interfaces {}  {
   $self instvar Node interfaces ns
   #En primer termino determina las interfaces (labels) y por c/u
   #crea un objeto CBTinterface que almacena en interfaces($lbl)
   #por ahora para interfaz local, -2 creada a mano
   set interfaces(-2) [new CBTinterface $self -2]
   $interfaces(-2) set_linktype 1
   set li [$Node set ifaces_]
   foreach l $li {
      set ou [$l set ifaceout]
      set lbl [$ou set intf_label_]
      set interfaces($lbl) [new CBTinterface $self $lbl]
   }
   #Determina para cada iff, el tipo de link 1:pap-mcast;
   #2:macc-mcast; 3:pap-nmcast; 4:macc-nmcast.
   #Hasta ahora, ns solo permite multicast (todo es multicast), si
   #luego se quiere poner alguna como unicast, debe hacerselo
   #explicitamente
   #Existe una manera mejor de hacerlo?
   set ne [$Node set neighbor_]
   foreach nn $ne {
      set lnk [$ns set link_([$Node id]:[$nn id])]
      set ifn [$lnk set ifacein_]
      set lbl [$ifn set intf_label_]
      #Aca podria seleccionarse no hacer dos veces el poner el tipo
      #en las ifaces maccess, a mejorar posteriormente
      set typ 1
      if {[$lnk info class] == "DummyLink"} {
         set typ 2
      }
      $interfaces($lbl) set_linktype $typ
   }
}



#Se borra un child para el grupo. Pasos:
```

```
#1-Buscar todos los replicators <cualquier src,grupo>
#    si el iif del replicator es la if child, inhabilitar todos los
oifs
#    si el iif del replicator NO es la if child, inhabilitar el oif =
child
#Para tener en cuenta: al handle cache miss entra una sola vez, cuando
crea
#el replicator. Por eso, al inhabilitar replicators, no se debe
sacarlos de
#las listas <sources, ....).
CBT instproc borrar_child {group child} {
   $self instvar sources iifs groups MFCTable Node
   $self instvar rep rep_int rep_obj
   set ent [$MFCTable get_entry $group]
   #No group
   if {$ent == -1 } {return}
   #lista de interfaces para el grupo, parent y child
   set cld ""
   set pnt [$MFCTable get_parent $group]
   lappend cld [lindex $pnt 1]
   if {[lindex $cld 0] == -1 } { set cld ""}
   set iflist [$MFCTable get_child_if $group]
   if { [lindex $iflist 0] != -1} {
      if {[llength $cld] > 0}  {
         lappend iflist $cld
      }
   }
   #Primero elimina el replicator asociado al grupo y al rotulo de
interfaz
   #(puede ser uno o mas, en el caso de dos sources enviando por la
misma iif)
   #Si detecta igual grupo y alguna iface de salida igual a la
recibida,
   #elimina el target
   set todos [$self array names rep]
   if {[llength $todos] >= 1} {
      foreach elem $todos {
              set pp [split $elem :]
         set ss [lindex $pp 0]
         set gg [lindex $pp 1]
         set ii [lindex $pp 2]
         #Elimina todos los targets
              if {$gg == $group && $ii == $child} {
            #Actualiza replicators
            set rrp [$Node set replicator_($ss:$gg)]
            set targets [$rrp array names elements_]
            $Node del-mfc $ss $gg $targets
            #Actualiza CBT
            unset rep($ss:$gg:$ii)
            unset rep_obj($ss:$gg:$ii)
            unset rep_int($ss:$gg:$ii)
         } else {
            #Para todo el grupo (la entrada a eliminar) ve si la iif
es
            #de salida
            if {$gg == $group} {
               set e [lsearch -exact $rep_int($ss:$gg:$ii) $child]
               if { $e != -1 } {
                  #Elimina la interfaz de salida del replicator
                  set rp $rep($ss:$gg:$ii)
                  set obj [lindex $rep_obj($ss:$gg:$ii) $e]
```

```
                  if { $obj != -1 } {
                     $Node del-mfc $ss $gg $obj
                  }
                  #Actualiza CBT
                  set $rep_int($ss:$gg:$ii) [lreplace
$rep_int($ss:$gg:$ii) $e $e]
                  set $rep_obj($ss:$gg:$ii) [lreplace
$rep_obj($ss:$gg:$ii) $e $e]
                  if { [llength $rep_int($ss:$gg:$ii)] == 0} {
                     unset $rep($ss:$gg:$ii)
                     unset $rep_int($ss:$gg:$ii)
                     unset $rep_obj($ss:$gg:$ii)
                  }
               }
            }
         }
      }
   }

   #Recorre la lista propia que guarda CBT con S,G,iif
   #Esto no es necesario ya que con el esquema de replicators de Ns
   #no es posible un par S,G asociado a dos interfaces de entrada, es
   #decir, bastara con encontrar el replicator_(S,G)
   #Primer paso: para el replicator de entrada inhabilitar todos los
   #targets
}

#Determina proximo router camino al core y la interfaz asociada
#target: id del nodo core
#Devuelve una lista con los elementos:
#code: codigo de retorno (0 OK, 1 error)
#nexthop: id del nodo next-hop camino al core
#if_lab: label de la interfaz
#if_obj: objeto networkinterface
CBT instproc next_router {target} {
        $self instvar ns Node
        set lr ""
        set code 0
   #Determinar proximo router camino al core
   set nexthop [ [ $ns set routingTable_] lookup [$Node set id_]
$target ]
        #No hay ruta al core del grupo
   if {$nexthop == -1} {
      puts "Nodo [$Node id]: JR .No ruta al core $target"
           lappend lr $code
      return $lr
   }
        #Es este nodo el next hop al core
   if {[$Node id] == $nexthop} {
      puts "Nodo $nexthop: JR para CORE $target "
      puts "Descartado porque no hay ruta al core"
           lappend lr $code
      return $lr
   }
   #En base al next hop para el core, obtiene la interfaz
   set ifaz [$Node get-oif [$ns set link_([$Node id]:$nexthop)]]
        set if_lab [lindex $ifaz 0]
        set if_obj [lindex $ifaz 1]
        set code 1
        lappend lr $code
        lappend lr $nexthop
```

```
        lappend lr $if_lab
        lappend lr $if_obj
        return $lr
}

#Dada una interfaz de arribo y un origen o tipo de envio devuelve una
lista
#con los grupos que son parents para el arribo.
#Los pares pueden ser (iif, nxthop) si el envio fue unicast
#o (iif, *) si el envio fue multicast
CBT instproc get_parents {ifce nhop} {
   $self instvar MFCTable
   set lst ""
   set gr [$MFCTable get_groups]
   foreach g $gr {
      set ex [$MFCTable lookup_child $g $ifce $nhop]
      if {$ex != -1} {lappend lst $g}
   }
   return $lst
}


#Dada una interfaz y una lista de grupos (-1 significa todos) devuelve
#los grupos de la lista para los cuales la interfaz es parent
CBT instproc if_parent {interface listgroups} {
   $self instvar MFCTable
   if { [lindex $listgroups 0] == -1 } {
      set listgroups [$MFCTable get_groups]
   }
   set xlist ""
   foreach g $listgroups {
      set x [$MFCTable get_parent $g]
      if {[lindex $x 1] == $interface} {
         lappend xlist $g
      }
   }
   return $xlist
}



#Dado un conjunto de grupos, y una interfaz, devuelve la interseccion
#entre el conjunto de grupos parametro y el conjunto de grupos child
de la
#interfaz.
#Si conjunto vacio devuelve -2
#Devuelve  -1 conjunto grupos parametro igual a conjunto child
#Devuelve ademas los nexthops para cada interfaz
CBT instproc childs {grps} {
   $self instvar MFCTable ifs nhp
   set gr [$MFCTable get_groups]
   foreach g $gr {
      set lch [$MFCTable get_child_if $g]
      set lnh [$MFCTable get_child_nh $g]
      set cnt -1
      foreach ii $lch {
      incr cnt 1
         if {![info exists ifs($ii)]} {
            set l ""
            set ifs($ii) $l
            set nhp($ii) [lindex $lnh $cnt]
```

```
         }
         set l $ifs($ii)
         lappend l $g
         set ifs($ii) $l
      }
   }
   set aa [array names ifs]
   foreach a $aa {
      set l1 $ifs($a)
      set l2 ""
      foreach g $grps {
         set s [lsearch -exact $l1 $g]
         if {$s > -1} {
            lappend l2 $g
         }
      }
      if { [llength $l1] == [llength $l2] } {
         set l2 ""
         lappend l2 -1
         set ifs($a) $l2
      } else {
         if { [llength $l2] == 0} {
            lappend l2 -2
            set ifs($a) $l2
         } else {
            set ifs($a) $l2
         }
      }
   }
}


#Determina si la relacion parent child es correcta. Devuelve 0 si lo
es, 1
#si no
# Link PaP con o sin  mcast:if child no puede coincidir con if parent
# Link maccess con mcast: si nodo NO DR, if child no puede coincidir
con parent
# Link maccess sin mcast: si nodo es DR, pares (if,nxthop) no pueden
coincidir
# Link maccess sin mcast: si nodo NO DR, if child no puede coincidir
con parent
CBT instproc child_is_valid { int1 nhop1 int2 nhop2 } {
   $self instvar interfaces
   if {$int1 == $int2 && $nhop1 == $nhop2} {return 1}
   if {$int1 == $int2} {
      #set itype [$self ifce_type $int1]
      set itype [$interfaces($int1) get_linktype]
      set dr [$interfaces($int1) is_dr]
      switch itype {
        case 1 {return 1}
        case 3 {return 1}
        case 2 {if { $dr != 1} {return 1}}
        case 4 {if { $dr != 1} {return 1}}
        }
      }
   return 0
}
```

A.3 timer.tcl

```
#########################################################################
######
#Timer handlers

#Los objetos timer, de clases propias CBT, se ejecutan un cierto
numero de
#veces (puede ser una) ejecutando una determinada accion (por ejemplo,
#enviando una PDU). Cuando el timer finaliza (luego de una o mas
ejecuciones
#del proc timeout definido para la nueva clase), la clase CBTtimer se
#encarga de hacer un delete del objeto timer y llamar a una funcion
del
#objeto cuya clase se deriva de timer. Los metodos a crear son tout y
tfin



Class CBTtimer

CBTtimer instproc init {tid simulator cbt tpars fpars} {
   $self instvar id ns fprocc object fparams tprocc tparams cbtagent
   set id $tid
   set fparams $fpars
   set tparams $tpars
   set ns $simulator
   set cbtagent $cbt
   $cbtagent tmr_puts "Node $id (Creando en tiempo: [$ns now])"
   return $self
}

#El sched contiene el tiempo del timer y se agrega las veces q se
ejecutara
CBTtimer instproc sched { delay times} {
   $self instvar cnt del ns id_
   if [info exists id_] {
      puts "Timer ya activo"
      return
   }
   set cnt $times
   set del $delay
   set id_ [$ns at [expr [$ns now] + $del] "$self timeout"]
}


#El resched larga de nuevo un timer ya existente
CBTtimer instproc resched { delay times} {
   $self instvar cnt del id ns id_ cbtagent
   $cbtagent tmr_puts "Node $id (Rescheduling en tiempo: [$ns now])"
   if [info exists id_] {
           $ns cancel $id_
           unset id_
       }
   set cnt $times
   set del $delay
   set id_ [$ns at [expr [$ns now] + $del] "$self timeout"]
}


CBTtimer instproc sched_ { delay times} {
   $self instvar cnt del ns id_
```

```
   set id_ [$ns at [expr [$ns now] + $del] "$self timeout"]
}


CBTtimer instproc cancel {} {
   $self instvar ns id_ object fprocc fparams tproc tparams id
cbtagent
   if [info exists id_] {
      $ns cancel $id_
      unset id_
   }
   $cbtagent tmr_puts "Node $id (Cancelando en tiempo: [$ns now])"
   #delete $self, despues del return !!! (si no se cuelga)
   $ns at [$ns now] "delete $self"
       $self tfin CANCEL $fparams
}


CBTtimer instproc timeout {} {
   $self instvar id cnt del ns object fprocc fparams tprocc tparams
   $self instvar cbtagent
       if {$cnt == "*"} {
   $cbtagent tmr_puts "Node $id (Timeout en tiempo: [$ns now])"
          $self sched_ $del 0
      $self tout $tparams
          return
       }
   incr cnt -1
   if {$cnt <= 0} {
      #delete $self, despues del return !!!
      $ns at [$ns now] "delete $self"
      #Aca hay que invocar al proceso timeout que qquiero que se
ejecute c/vez q
      #vence el timer.
          $self tout $tparams
      #Aca invoca al proc final
      $self tfin NORMAL $fparams
      return
   }
   if {$cnt != 0} {
      $self sched_ $del 0
      $cbtagent tmr_puts "Node $id (Timeout en tiempo: [$ns now])"
      #Aca hay que invocar al proceso timeout que qquiero que se
ejecute c/vez q
      #vence el timer.
      $self tout $tparams
      return
   }
}



#############################################################################
#######

#Clase Delete_Entry_Timer: representa objetos timer que invocan al
delete
#entry al finalizar.

Class Delete_Entry_Timer  -superclass CBTtimer
```

```
#Timeout no hace nada (se invoca con 0 reintentos), solo tiene efecto
el cancel
Delete_Entry_Timer instproc tout {tparams} {
}

#Procedimiento que se ejecuta al vencer el tiempo
#Invoca al borrado de la entrada correspondiente
#Recibe como parametros la tabla mfc y el grupo a borrar
Delete_Entry_Timer instproc tfin {code fparams} {
   $self instvar ns
   #puts "Tiempo: [$ns now]: Eliminacion de entrada on tree, grupo
[lindex $fparams 1]"
   [lindex $fparams 0] delete_entry [lindex $fparams 1]
   #Debe reengancharse los miembros locales del grupo
   $self instvar cbtagent
   $cbtagent reeng [lindex $fparams 1]
}


##########################################################################
#######

#Clase Delete_T_Entry_Timer: representa objetos timer que invocan al
delete
#entry al finalizar.Pero al transient, que requiere code de
terminacion

Class Delete_T_Entry_Timer  -superclass CBTtimer

#Timeout no hace nada (se invoca con 0 reintentos), solo tiene efecto
el cancel
Delete_T_Entry_Timer instproc tout {tparams} {
}

#Procedimiento que se ejecuta al vencer el tiempo
#Invoca al borrado de la entrada correspondiente
#Recibe como parametros la tabla mfc y el grupo a borrar
Delete_T_Entry_Timer instproc tfin {code fparams} {
   $self instvar ns
   #puts "Tiempo: [$ns now]: Eliminacion de entrada transitoria, grupo
[lindex $fparams 1]"
   [lindex $fparams 0] delete_entry [lindex $fparams 1]
}


##########################################################################
########
#Clase Echo_Request_Timer: Timer encargado de enviar echo request

Class Echo_Request_Timer -superclass CBTtimer

#Cada vez que se cumple un timeout, se envia un echo request
Echo_Request_Timer instproc tout {params} {
   set msg   [lindex $params 0]
   set iflbl [lindex $params 1]
        set envio [lindex $params 2]
        set dest  [lindex $params 3]
   set cbt   [lindex $params 4]
   #puts " $self Enviando echo request ...($msg) a Node $dest"
   $cbt send_msg $msg $iflbl $envio $dest
}
```

```
Echo_Request_Timer instproc tfin {code params} {
   $self instvar echorq_timer
   set interface [lindex $params 0]
        set nexthop [lindex $params 1]
   set adm_echorq [lindex $params 2]
        set exists [$adm_echorq exists $interface $nexthop]
   if {$exists == 1} {
            $adm_echorq elim $interface $nexthop
      #puts " $self unsetting echo_request($interface:$nexthop)"
   }
}

#########################################################################
######
#Clase Quit_Not_Timer: cada vez que se cumple timeout, envia qn

Class Quit_Not_Timer -superclass CBTtimer

#tfin no hace nada
Quit_Not_Timer instproc tfin {code params} {
   set gr [lindex $params 0]
   set pif [lindex $params 1]
   set dest [lindex $params 2]
   set cbt [lindex $params 3]
   $cbt unset qntimer($gr:$pif:$dest)
}


Quit_Not_Timer instproc tout {params} {
   set msg   [lindex $params 0]
   set iflbl [lindex $params 1]
        set envio [lindex $params 2]
        set dest  [lindex $params 3]
   set cbt   [lindex $params 4]
   $cbt send_msg $msg $iflbl $envio $dest
}

#########################################################################
######
#Clase Resend_Join_Timer: Se utiliza para eliminar una entrada
temporaria
#por timeout no hace nada, por final invoca al delete entry

Class Resend_Join_Timer -superclass CBTtimer

#tfin no hace nada
Resend_Join_Timer instproc tfin {code params} {
}

Resend_Join_Timer instproc tout {params} {
   set msg   [lindex $params 0]
   set iflbl [lindex $params 1]
        set envio [lindex $params 2]
        set dest  [lindex $params 3]
   set cbt   [lindex $params 4]
   $cbt send_msg $msg $iflbl $envio $dest
}
```

```
#########################################################################
######
#Clase Send_Join_Timer: cada vez que se cumple timeout, envia join
request

Class Send_Join_Timer -superclass CBTtimer

#tfin no hace nada
Send_Join_Timer instproc tfin {code params} {
}


Send_Join_Timer instproc tout {params} {
   set msg   [lindex $params 0]
   set iflbl [lindex $params 1]
        set envio [lindex $params 2]
        set dest  [lindex $params 3]
   set cbt   [lindex $params 4]
   $cbt send_msg $msg $iflbl $envio $dest
}



#########################################################################
######
#Clase Cache_Del_Timer: cada vez que se cumple timeout, elimina la
entrada
#child para el grupo

Class Cache_Del_Timer -superclass CBTtimer

#tfin unsetting de cache_del_timer de cbt
Cache_Del_Timer instproc tfin {code params} {
   set gr [lindex $params 0]
   set iif [lindex $params 1]
   set cbt [lindex $params 2]
   $cbt unset cache_del_timer($gr:$iif)
}


Cache_Del_Timer instproc tout {tparams} {
   [lindex $tparams 0] delete_child [lindex $tparams 1] [lindex
$tparams 2] [lindex $tparams 3]
}
```

A.4 agents.tcl

```
#########################################################################
######
# El agente de transmision y recepcion es el encargado de enviar y
recibir
# las PDUs CBT invocando en este ultimo caso a la funcion
correspondiente del
# protocolo

Class Agent/Message/CBT -superclass Agent/Message


# transmit tiene en cuenta directivas de debugging y luego invoca al
metodo
```

```
# send definido en la clase Agent/Message
Agent/Message/CBT instproc transmit msg {
   $self instvar proto node_
   $proto debug_send  $msg
   $self send $msg
}


# handle es invocado cada vez que llega un mensaje direccionado
(unicast o
# multicast al agente. Realiza una decodificacion parcial que le
permite
# determinar el rotulo de la interfaz local de arribo y el tipo de
PDU. En
# base a el invoca al metodo CBT correspondiente
Agent/Message/CBT instproc handle msg  {
   $self instvar proto  node_ iif_
   #Obtencion de la interfaz de arribo. En base al ultimo de los
campos
   #de la PDU, router origen, se obtiene el nodo adyacente (se supone
   #que no hay politicas de ruteo implementadas, ya que Ns no lo
soporta)
   set simul [$proto set ns]
   set rr [$simul set routingTable_]
   set list [split $msg /]
   set ll [llength $list]
   incr ll -1
   set nnod [lindex $list $ll]
   set origen [$rr lookup [$node_ id] $nnod]
   set oi [$node_ get-oif [$simul set link_([$node_ id]:$origen)]]
   set oi1 [lindex $oi 1]
   set iif_ [$oi1 set intf_label_]
   set pdu_type [lindex $list 0]
   set list [lreplace $list 0 0]
   $proto debug_recv $msg
   switch $pdu_type {
      0       { $proto recv-hello $list }
      1       { $proto recv-join_request $list }
      2       { $proto recv-join_ack $list }
      3       { $proto recv-quit_notification $list }
      4       { $proto recv-echo_request $list }
      5       { $proto recv-echo_reply $list }
      6       { $proto recv-flush_tree $list }
      default { puts "$self Tipo de CBT recibida desconocido:
$pdu_type" }
   }
}



#########################################################################
######
#Agentes para prueba del protocolo

Class Agent/Message/Prueba -superclass Agent/Message

Agent/Message/Prueba instproc handle { msg } {
   $self instvar node_
   set ns [$node_ set ns_]
   puts "Node:[$node_ id] Agent/Message/Prueba recibe: $msg at [$ns
now]"
```

```
}


Agent/Message/Prueba instproc transmit {msg} {
   $self instvar ttl_ node_
   set ns [$node_ set ns_]
#       puts "Node:[$node_ id] Agent(self: $self  ttl: $ttl_) send:
$msg at [$ns now]"
   #OJO con ttl, cuando era 2 propagaba JR a otros agentes cbt, ahora
   #,1, solo lo envia al local
   #$self set ttl_ 2
   $self send $msg
}



# Los agentes Receptor_delay y Emisor_delay se utilizan para medicion
de
# demoras punta a punta, desde una fuente a un destino en particular.
#
# El agente de recepcion recibe un descriptor de archivo donde generar
# informacion correspondiente a cada paquete recibido. Esta
informacion incluye
# numero de secuencia del paquete, tiempo de envio y tiempo de demora.
#
# El agente de emision recibe un intervalo de tiempo para enviar sus
paquetes
# regularmente al grupo multicast, desde que es invocado su arranque
hasta
# su terminacion. Esta caracteristica puede cambiarse para que genere
trafico
# de acuerdo a laguna distribucion.
#
# Ambos agentes deben configurarse con la direccion adecuada

Class Agent/Message/Receptor_delay -superclass Agent/Message

Agent/Message/Receptor_delay instproc init {file} {
   $self instvar file_
   set file_ $file
   $self next
}

Agent/Message/Receptor_delay instproc handle msg {
   $self instvar node_ file_
   set ns [$node_ set ns_]
   set list [split $msg /]
   set nod [lindex $list 0]
   set time [lindex $list 1]
   set seq [lindex $list 2]
   set n [$ns now]
   set dif [expr $n - $time]
# set dif 5
#       puts "Node:[$node_ id] Agente recibe: $msg at [$ns now]
(delay: $dif)"
       puts $file_ "Node:[$node_ id] Recibe:$msg at [$ns now] Delay:
$dif"
#  puts $file_ "$nod [$node_ id] $seq $dif"
}
```

```
Class Agent/Message/Emisor_delay -superclass Agent/Message

Agent/Message/Emisor_delay instproc init {interval file} {
   $self instvar interval_ fid_ stop file_
   set interval_ $interval
   set file_ $file
   set fid_ 1
   set stop 1
   $self next
}

Agent/Message/Emisor_delay instproc start {} {
   $self instvar seq stop
   set seq 1
   $self transmit
}

Agent/Message/Emisor_delay instproc transmit {} {
   $self instvar ttl_ node_ interval_ fid_ file_ seq stop
   set ns [$node_ set ns_]
   set msg "[$node_ id]/[$ns now]/$seq"
   incr seq 1
        puts $file_ "Node:[$node_ id] Envia: $msg at [$ns now] "
#       puts "Node:[$node_ id] Agente(self: $self  ttl: $ttl_) send:
$msg at [$ns now]"
   if {$stop == 1} {
      $ns at [expr [$ns now] + $interval_] "$self transmit"
   }
   $self send $msg
}


Agent/Message/Emisor_delay instproc stop {} {
   $self instvar stop
   set stop 0
}




# Un objeto encap es el encargado de tomar paquetes multicast que son
para un
# destino unicast o para un tunel, encapsularlos y mandarlos (unicast)
a
# destino.
# Los paquetes llegan a el porque siempre que el proc cache_miss
detecta que
# hay un paquete multicast para una interfaz unicast, en lugar de
poner en
# el replicator un objeto networkinterface o un dynalink, pone un
agente
# encap creado especialmente. Esto quiere decir que para cada
src,dst,iif se
# podran crear varios agentes encap, uno por destino (igual que si
fueran
# objetos networkinterface o dynamiclink).
# La encapsulacion consistira en poner como dato el grupo de destino y
el
# origen del paquete multicast
```

```
# El destino (el decap correspondiente) lo averiguara via ruteo o por
# configuracion manual en caso de tunneling

Class encap -superclass Agent/Message

encap instproc init { src grp iif oif nxhop nod} {
   $self instvar source group iintf ointf nexthop node
   set source $src
   set group $grp
   set iintf $iif
   set ointf $oif
   set nexthop $nxhop
   set node $nod
   #puts "Encap creado, src:$source grp:$group iif:$iintf oif:$ointf"
   $self next
}

encap instproc transmit {msg} {
   $self instvar dst_ source group iintf ointf nexthop node
   #puts "ENCAP: $self (nodo [$node id]: transmite"
   #Determina la direccion del agente decap del nexthop (MFCTable)
   set sim [$node set ns_]
   set nn [$sim set Node_($nexthop)]
   #puts "ES [$nn id]"
   set mc [$nn set mcastproto_]
   set lp [$mc set protocols]
   #Se asume que existe CBT
   set prot -1
   foreach pr $lp {
      set t [$pr set type]
      if { $t == "CBT" } {
         set prot $pr
      }
   }
   set cc [$prot set decapag]
   set dst_ [$cc set addr_]
   #Encapsulacion: source y group se conocen por construccion del
objeto
   set msg "$source/$group/$msg"
   $self send $msg
}

# Handle recibe desde el propio nodo (multiclassifier y replicators
...). Lo
# que hace es ubicar su decap, encapsular y enviar
encap instproc handle {msg} {
   $self transmit $msg
}


# Existe un agente decap por nodo.
# Su funcion es recibir paquetes multicast que han sido encapsulados,
debido
# a que el vinculo de arribo es unicast o a que se ha configurado
# (manualmente y en forma estatica -no admite dynamics-) un tunel. (La
# diferencia entre el tunel y el vinculo unicast es que en este ultimo
caso
# el next hop se obtiene interactuando con la funcion de ruteo (en
este caso
# el ruteo unicast provisto por ns) y en el primero debe ser
configurado
```

```
# manualmente.
# La manera de operar del agente es la siguiente: al recibir un
paquete
# unicast direccionado a el (proveniente de un agente encap remoto)
por una
# interfaz (ns) determinada, toma el origen del paquete (src) y el
grupo de
# destino (group) y genera un paquete multicast con estas
caracteristicas.
# El paquete multicast se inyecta en el nodo, de manera tal que a
efectos
# del manejo multicast (forwarding en base a replicators existentes o
# creacion de nuevos elementos de forwarding en base a las tables de
ruteo
# multicast (MFCTable), se comporte como un paquete normal. Para ello
hay
# que tener en cuenta la interfaz de llegada e inyectarlo (target) en
el
# objeto networkinterface (iface) correspondiente (recordar que el
# forwarding se realiza teniendo en cuenta src, dest e iif)

Class decap -superclass Agent/Message

decap instproc handle {msg} {
   $self instvar node_ dst_ addr_ iif_
#      puts "Decap de node [$node_ id] recibe mensaje: $msg"
   set list [split $msg /]
   set source [lindex $list 0]
   set group [lindex $list 1]
   set msg [lindex $list 2]
   set int -1
   #Se asume que la iif existe
   set xx [$node_ set ifaces_]
   foreach x $xx {
      set l [$x set id]
      if {$l == $iif_} {
         set int $x
      }
   }
   set ifac [$x set iface]
   $self target $ifac
   #addr_ se cambia temporariamente para el send, luego debe
   #restaurarse para que el agente pueda ser encontrado y recibir
   set tmp $addr_
   ####set dst_ 65490
   set dst_ $group
   ####set addr_ 2818
   set addr_ [expr $source << 8]
   $self send $msg
   set addr_ $tmp
}
```

A.5 simul.tcl

```
# Metodos agregados a clases Ns


# Devuelve los agentes locales joined a un grupo
Node instproc get_Agents {group} {
   $self instvar Agents_
```

```
   if { ![info exists Agents_($group)] } {
      return -1
   } else {
      return $Agents_($group)
   }
}


#Proc agregado al simulator para setear cores. Esto se utiliza solo si
los
#CBTs son estaticos. Consiste en tener una lista en el simulador con
#los grupos (grplist) y otra con los nodos core asociados (corelist)
#Aca solo se setean las listas; cada agente cbt es responsable se
colocar su
#nodo (id) y su grupo
Simulator instproc set-core { group node } {
   $self instvar grplist corelist
   if {![info exists grplist]} { set grplist ""}
        if {![info exists corelist]} {set corelist ""}
   lappend grplist $group
   lappend corelist $node
}

# Para obtener el core de unn grupo
Simulator instproc get-core {group} {
   $self instvar grplist corelist
   set idx [lsearch -exact $grplist $group]
   if {$idx < 0 } { return -1 }
   set core [lindex $corelist $idx]
   return $core
}

# Para obtener lista de cores y grupos
Simulator instproc get-cores {} {
   $self instvar grplist corelist
   puts "Cores: $corelist"
   puts "Groups: $grplist"
}


# Muestra por salida standard el arbol de distribucion para  un grupo
Simulator instproc map-tree {group} {
   set lista ""
   set co [$self get-core $group]
   if {$co == -1} {
      puts "No core para grupo $group"
      return
   }
   lappend lista $co
   set longi [llength $lista]
   while {$longi > 0} {
      set co [lindex $lista 0]
      set nodo [$self set Node_($co)]
      set prot [$nodo set mcastproto_]
           set cb [$prot getType CBT]
      set tab [$cb get_MFCTable]
      set lis [$tab get_child_if $group]
      set lis1 ""
      set cnt 0
      while { [llength $lis] > $cnt} {
         set ll [lindex $lis $cnt]
```

```
         if {$ll > -2} {
            set nei [$nodo set neighbor_]
            foreach x $nei {
             set lnk [$self set link_([$nodo id]:[$x id])]
                 set ifn [$lnk set ifacein_]
                 set lbl [$ifn set intf_label_]
                 if {$lbl == $ll} {
                    lappend lis1 [$x id]
                    lappend lista [$x id]
                 }
              }

         }
         incr cnt 1
      }


#SACAR REPETIDOS de lis1 (para vinculos multiacceso)
      set list2 ""
           foreach l1 $lis1 {
         set esta 0
         foreach l2 $list2 {
            if { $l1 == $l2 } {set esta 1}
              }
             if { $esta == 0 } { lappend list2 $l1 }
           }
      set lis1 $list2

# y de lista
      set list2 ""
           foreach l1 $lista {
         set esta 0
         foreach l2 $list2 {
            if { $l1 == $l2 } {set esta 1}
              }
              if { $esta == 0 } { lappend list2 $l1 }
           }
      set lista $list2


      puts "Nodo: $co, child: $lis1"
      set lista [lreplace $lista 0 0]
      set longi [llength $lista]
   }
}
```

A.6 debug.tcl

```
#Procedimientos para debugging
#Son metodos de varias clases agrupados en este archivo

#Lista interfaces
CBT instproc list_interfaces {} {
   $self instvar interfaces Node ns
   puts "Agente CBT en nodo [$Node id].Tiempo: [$ns now]. Interfaces:"
   set inf [array names interfaces]
   if { [llength $inf] == 0 } {
      puts "No existen interfaces"
      return
   } else {
```

```
      foreach g $inf {
         $interfaces($g) list
      }
   }
}


#Lista MFCTable
CBT instproc list_MFCTable {} {
   $self instvar MFCTable Node ns
   puts "Agente CBT en nodo [$Node id]. Tiempo: [$ns now]. MFCTable:"
   $MFCTable list
}



#Lista TransientTable
CBT instproc list_TransientTable {} {
   $self instvar TransientTable Node ns
   puts "Agente CBT en nodo [$Node id]. Tiempo: [$ns now].
Transient_Table:"
   $TransientTable list
}


#Lista la interfaz
CBTinterface instproc list {} {
   $self instvar CBTagent label linktype pref_value des_router address
   $self instvar hello_int
   set lt UNKNOWN
   switch $linktype {
     1 {set lt "PaP/M"}
     2 {set lt "Mac/M"}
     3 {set lt "PaP/U"}
     4 {set lt "Mac/U"}
   }
   puts "IF($self): lbl($label) -link($lt) -DR($des_router) -
Pr.val($pref_value) -Hello($hello_int) -Addr($address)"
}

#Lista la MFCTable
MFC_Table instproc list {} {
   $self instvar entry
   set x [array names entry]
   if { [llength $x] == 0 } {
      puts "No existen entradas en MFCTable"
   } else {
      foreach g $x {
         $entry($g) list
      }
   }
}



#Lista la TransientTable
Transient_Table instproc list {} {
   $self instvar entry
   set x [array names entry]
   if { [llength $x] == 0 } {
      puts "No existen entradas en Transient_Table"
```

```
   } else {
      foreach g $x {
         $entry($g) list
      }
   }
}



MFC_entry instproc list {} {
   $self instvar group core_id parent_label parent_nexthop del_entry
   $self instvar Mfctable CBTagent child_if child_nh
   if {[info exists del_entry]} {
      puts "Group:$group - Core:$core_id - if to parent:$parent_label
- nexthop: $parent_nexthop - delete timer: $del_entry"
   } else {
      puts "Group:$group - Core:$core_id - if to parent:$parent_label
- nexthop: $parent_nexthop - delete timer: ---"
   }
   puts "   Child iifs    : $child_if"
   puts "   Child nexthops: $child_nh"
}




Transient_entry instproc list {} {
   $self instvar group core_id parent_label parent_nexthop del_entry
   $self instvar TMfctable CBTagent child_if child_nh leaf retransmit
   set l N
   set ret N
   if {$leaf == 1} {
      set l Y
      set ret $retransmit
   }
   puts "Group:$group - Core:$core_id - if to parent:$parent_label -
nexthop: $parent_nexthop - leaf: $l"
   puts "delete timer: $del_entry - retransmit timer: $ret"
   puts "   Child iifs    : $child_if"
   puts "   Child nexthops: $child_nh"
}


CBT instproc debug_send {msg} {
   $self instvar enbl Node ns
   if {![info exists enbl]} {return}
   set list [split $msg /]
   set pdu_type [lindex $list 0]
   set list [lreplace $list 0 0]
   set p "@"
   switch $pdu_type {
      0  { if {$enbl(0) == 1} { set p "HELLO" } }
      1  { if {$enbl(1) == 1} { set p "JOIN_REQ" } }
      2  { if {$enbl(2) == 1} { set p "JOIN_ACK" } }
      3  { if {$enbl(3) == 1} { set p "QUIT_NOT" } }
      4  { if {$enbl(4) == 1} { set p "ECHO_REQ"} }
      5  { if {$enbl(5) == 1} { set p "ECHO_RPL" } }
      6  { if {$enbl(6) == 1} { set p "FLSH_TRE" } }
      default { set p "UNKNOUN PDU" }
```

```
   }
   if {$p == "@"} {return}
        puts "Node:[$Node set id_] ==>$p: $list  Tiempo:[$ns now]"
}


CBT instproc debug_recv {msg} {
   $self instvar Node ns enbl
   if {![info exists enbl]} {return}
   set list [split $msg /]
   set pdu_type [lindex $list 0]
   set list [lreplace $list 0 0]
   set p @
   switch $pdu_type {
      0  { if {$enbl(0) == 1} { set p "HELLO" } }
      1  { if {$enbl(1) == 1} { set p "JOIN_REQ" } }
      2  { if {$enbl(2) == 1} { set p "JOIN_ACK" } }
      3  { if {$enbl(3) == 1} { set p "QUIT_NOT" } }
      4  { if {$enbl(4) == 1} { set p "ECHO_REQ"} }
      5  { if {$enbl(5) == 1} { set p "ECHO_RPL" } }
      6  { if {$enbl(6) == 1} { set p "FLSH_TRE" } }
      default { set p "UNKNOUN PDU" }
   }
   if {$p == "@"} {return}
        puts "Node:[$Node set id_] <== $p: $list  Tiempo:[$ns now]"
}



#Inicializa las variables de debugging. Debe ser corrido en el init
del
#agente CBT. Pone todas las variables en cero
CBT instproc init_debug { } {
   $self instvar enbl tmr dtables disc
   #enbl: determinan si se producen mensajes por PDUs recibidas y
enviadas
   set enbl(0) 0
   set enbl(1) 0
   set enbl(2) 0
   set enbl(3) 0
   set enbl(4) 0
   set enbl(5) 0
   set enbl(6) 0
   #tmr: Determina si se produce mensaje por acciones sobre los timers
   set tmr 0
   #dtables: Determina si se producen mensajes por modificaciones en
   #         las tablas (agregado/borrado child o agregado/borrado
entry)
   set dtables 0
   #disc: Determina si se producen mensajes por PDU descartadas (por
   #      ejemplo un ack para el que no hay gupo)
   set disc 0
}

CBT instproc enable_timers {} {
   $self instvar tmr
   set tmr 1
}

CBT instproc disable_timers {} {
   $self instvar tmr
```

```
   set tmr 0
}

CBT instproc tmr_puts {msg} {
   $self instvar tmr
   if {$tmr == 1} {puts "$msg"}
}

CBT instproc enable_tables {} {
   $self instvar dtables
   set dtables 1
}

CBT instproc disable_tables {} {
   $self instvar dtables
   set dtables 0
}

CBT instproc tab_puts {msg} {
   $self instvar dtables
   if {$dtables == 1} {puts "$msg"}
}


CBT instproc enable_discard {} {
   $self instvar disc
   set disc 1
}

CBT instproc disable_discard {} {
   $self instvar disc
   set disc 0
}

CBT instproc disc_puts {msg} {
   $self instvar disc
   if {$disc == 1} {puts "$msg"}
}


CBT instproc enable_ALL {} {
   $self enable_pdus ALL
   $self enable_timers
   $self enable_discard
   $self enable_tables
}


CBT instproc disable_ALL {} {
   $self disable_pdus ALL
   $self disable_timers
   $self disable_discard
   $self disable_tables
}



CBT instproc enable_pdus {pdu} {
   $self instvar enbl
   switch $pdu {
     "HELLO" {set enbl(0) 1}
```

```
     "JOIN_REQ" {set enbl(1) 1}
     "JOIN_ACK" {set enbl(2) 1}
     "QUIT_NOT" {set enbl(3) 1}
     "ECHO_REQ" {set enbl(4) 1}
     "ECHO_RPL" {set enbl(5) 1}
     "FLSH_TRE" {set enbl(6) 1}
     "ALL"      {set cnt 0
                    while {$cnt <= 6} {
                       set enbl($cnt) 1
             incr cnt 1
                    }
              }
     default    {puts "CBT:enable UNKNOUN PDU, optios are:"
           puts " HELLO JOIN_REQ JOIN_ACK QUIT_NOT ECHO_REQ ECHO_RPL
FLSH_TRE ALL"
          }
   }
}


CBT instproc disable_pdus {pdu} {
   $self instvar enbl
   switch $pdu {
     "HELLO" {set enbl(0) 0}
     "JOIN_REQ" {set enbl(1) 0}
     "JOIN_ACK" {set enbl(2) 0}
     "QUIT_NOT" {set enbl(3) 0}
     "ECHO_REQ" {set enbl(4) 0}
     "ECHO_RPL" {set enbl(5) 0}
     "FLSH_TRE" {set enbl(6) 0}
     "ALL"      {set cnt 0
                    while {$cnt < 6} {
                       set enbl($cnt) 0
             incr cnt 1
                    }
              }
     default    { puts "CBT:disable UNKNOUN PDU, opciones:"
           puts "HELLO JOIN_REQ JOIN_ACK QUIT_NOT ECHO_REQ ECHO_RPL
FLSH_TRE ALL"
          }
   }
}
```

A.7 recvr.tcl

```
####################JOIN ACK

######### SEND
#Este proceso se encarga de recibir requerimientos para envio de join
acks
CBT instproc send_join_ack {group interface nexthop} {
   $self instvar MFCTable ns Node node_
   set orig_router [$Node id]
   set type [CBT set JOIN_ACK]
   set ls1 [$MFCTable get_parent $group]
   set target [lindex $ls1 0]
   set dest $nexthop
   set ifce $interface
   if {$dest == "*"} {
      set envio M
```

```
   } else {
      set envio U
   }
   set msg "$type/$group/$target/$envio/$orig_router"
   $self send_msg $msg $ifce $envio $dest
}


############## RECEIVE
#Recibe join_ack, reenvia y actualiza tablas
CBT instproc recv-join_ack { ja_pdu } {
   $self instvar ns Node node_ TransientTable qntimer
   #Separacion de las componentes del join ack
   set group [lindex $ja_pdu 0]
   set target [lindex $ja_pdu 1]
   set type [lindex $ja_pdu 2]
   set nexthop [lindex $ja_pdu 3]
   set interface [[$self set messager] set iif_]
   #Si el ack no coincide con una entrada transitoria, lo descarta
   set ent [$TransientTable get_parent $group]
   set int [lindex $ent 1]
   set nhp [lindex $ent 2]
   #Acepta J-ack si interfaz coincide, y si nexthop coincide
   set acc [$self jac_accept $group $interface $type $nexthop]
   if { $acc == -1} {
     set m "Nodo:[$Node id] descarta join_ack (no grupo) tiempo [$ns
now]"
     $self disc_puts $m
     return
   }
   #Convierte entrada transitoria a entrada on-tree (elimina entrada
   #transitoria)
   $self transient2ontree $group
}


#Convierte una entrada transitoria en entrada on-tree. Es invocado al
#recibir un join-ack.
#   - 1 Crea una entrada (group) en tabla on-tree, si ya existe,
aborta
#       el proceso
#   - 2 Copia las child transient en la nueva entrada on-tree (add
entry,
#       que implica el envio de join ack)
#   - 3 Elimina las entradas transient
CBT instproc transient2ontree {group} {
   $self instvar TransientTable MFCTable
   #Si existe la entrada, retorna
   set ex [$MFCTable get_entry $group]
   if {$ex != -1 } {return}
   set l1 [$TransientTable get_parent $group]
   $MFCTable add_entry $group [lindex $l1 0] [lindex $l1 1] [lindex
$l1 2]
   set ifaces [$TransientTable get_child_if $group]
   set nhops  [$TransientTable get_child_nh $group]
   set cnt 0
   set lng [llength $ifaces]
   while {$cnt < $lng} {
      $MFCTable add_child $group [lindex $ifaces $cnt] [lindex $nhops
$cnt]
```

```
      $TransientTable delete_child $group [lindex $ifaces $cnt]
[lindex $nhops $cnt]
      incr cnt 1
   }
}


#Dado un grupo para el que llega un join ack, y la interface, tipo de
envio y
#nexthop por las que arribo, determina si el agente lo acepta o no.
#Si las interfaces no coinciden, no se acepta
#Si el nodo no es DR, se testea que coincida solo la iif si envio M
#Si el nodo no es DR, se testea que coincida iif y nexthop si envio U
#Si el nodo es DR, solo acepto envios U, y debe coincidir iif nexthop
CBT instproc jac_accept { group interface envio nexthop} {
   $self instvar TransientTable interfaces
   set ent [$TransientTable get_parent $group]
   set int [lindex $ent 1]
   set nhp [lindex $ent 2]
   #Coinciden interfaces
   if { $int != $interface } { return -1}
   #Node es DR
   if { [$interfaces($int) is_dr] == 1} {
      #Nodo DR, envio no U
      if { $envio != "U" } {
         return -1
      } else {
      #Nodo DR, envio U
         #iif y nexthop no coinciden
         if { $int != $interface || $nexthop != $nhp } {
       return -1
         } else {
       return 0
         }
      }
   } else {
   #Nodo no es DR
      if {$envio == "U" } {
         if {$int == $interface && $nexthop == $nhp } {
       return 0
         } else {
       return -1
         }
      } else {
      #Envio M
         return 0
      }
   }
}



############### JOIN REQUEST

######## SEND
#Envia join request; si la entrada del grupo es leaf, setea el timer
para
#reintentos. en este caso devuelve el timer creado
CBT instproc send_jr { group } {
   $self instvar TransientTable ns Node node_ interfaces
   set orig_router [$Node id]
```

```
   #Obtiene core, interface, nexthop para el grupo
   set list1 [$TransientTable get_parent $group]
   set target [lindex $list1 0]
   set pif  [lindex $list1 1]
   set dest [lindex $list1 2]
   set envio [$interfaces($pif) sendmode]
   if {$envio == "M"} {set dest *}
   #Nodo es DR en parent, JR se envia unicast
   #Envio del primer join request
   set type [CBT set JOIN_REQUEST]
   set msg "$type/$group/$target/$orig_router/$envio/$orig_router"
   $self send_msg $msg $pif $envio $dest
   set leaf [$TransientTable get_leaf $group]
   #Si nodo es leaf, setea timer para reenvio
   if {$leaf == 1} {
      set tout ""
      lappend tout $msg
      lappend tout $pif
      lappend tout $envio
      lappend tout $dest
      lappend tout $self
      set ptimeout $tout
      set fin ""
      set pfin $fin
      set id "[$Node id] Resend_Join_Timer($pif,$dest)"
      set timer [new Resend_Join_Timer $id $ns $self $ptimeout $pfin]
      $timer sched [CBT set RTX_INTERVAL] *
      $TransientTable set_retransmit_timer $group $timer
   }
}


########## RECEIVE

CBT instproc recv-join_request { jr_pdu } {
   $self instvar  ns Node MFCTable
   $self instvar TransientTable jr_pending interfaces
   $self instvar qntimer
   #Separacion de las componentes del join request (target=id del
core)
   set group [lindex $jr_pdu 0]
   set target [lindex $jr_pdu 1]
   set orig_router [lindex $jr_pdu 2]
   set type [lindex $jr_pdu 3]
   set sender [lindex $jr_pdu 4]
   #sender es la node id del nodo que envia (o reenvia) la PDU -no el
origen
   set gr [expr $group]
   #Determinar interfaz de arribo, si es no valida descartar
   #Casos especiales de iif:
   # -1 es interfaz no valida
   #La iif -2, indica que recibo el request a traves de un agente
local
   #(puede ser un agente IGMP o un agente para introducir la pdu desde
   #el simulador)
   set iif [[$self set messager] set iif_]
   if {$iif == -1} {
      set m "Nodo:[$Node id] Descarta JREQ (recibido por interfaz
incorrecta, $iif) tiempo [$ns now]"
      $self disc_puts $m
      return
```

```
    }
    #Determinar si se acepta o no el JR. Un JR es rechazado si:
    #vinculo multiacceso, join multicast, nodo NO es DR
    #vinculo multiacceso, join unicast, nodo DR (==> Error, dos DRs?)
    set linktype [$interfaces($iif) get_linktype]
    set dr [$interfaces($iif) is_dr ]
    if { $linktype == 2 && $type == "M" && $dr != 1 } {
       return
    }
    if { $linktype == 2 && $type == "U" && $dr == 1 } {
       puts "Error dos DRs?"
       return
    }
    #Determina nexthop en base al tipo de join ??
    if {$type == "U"} {
       set nhop $sender
    } else {
       set nhop *
    }
    #Nodo es on-tree para el grupo.
    if { [$MFCTable exists_entry $gr] != -1 } {
       set l1 [$MFCTable get_entry $gr]
       #PDU recibido por parent
       set ip [lindex $l1 1]
       set tp [lindex $l1 2]
       #Si recibe por parent, ignora excepto que sea link mcast maccess
       #el join recibido Mcast y el nodo NO DR
       if { $ip == $iif && $tp == $nhop} {
          if { $type == "M" && $linktype == 2 } {
           #si hay jreq scheduled, lo elimina
        if [info exists jr_pending($gr)] {
            $jr_pending($gr) cancel
         }
          } else {
         set m "Nodo:[$Node id] Descarta JREQ (recibido por interfaz
parent, $iif) at [$ns now]"
         $self disc_puts $m
          }
          return
       } else {
       #Si if no parent, se valida la relacion parent/child, si OK,
       #se hace directamente un add_child, que agrega
       #la entrada si no esta y envia (siempre) un JACK
          if {[$self child_is_valid $iif $nhop $ip $tp] == 1} {return}
          $MFCTable add_child $gr $iif $nhop
          return
       }
    }
    #Nodo NO es on tree para el grupo
    #Existe entrada transitoria para el grupo: Debe agregarse al nodo
    #como child. Se testea que no sea invalida la relacion parent-child
    if { [$TransientTable exists_entry $gr] != -1 } {
       set l1 [$TransientTable get_parent $gr]
       if {[$self child_is_valid $iif $nhop [lindex $l1 1] [lindex $l1
2]] == 1} {
          return
       }
       $TransientTable add_child $gr $iif $nhop
    } else {
    #No existe entrada transitoria
       #Obtiene core para el grupo, es target que vino en PDU
```

```
      #Obtiene nexthop para el core (next_router_to_core)
      set lst [$self next_router $target]
      #No hay ruta al core
      if { [lindex $lst 0] == 0 } { return }
      set iface [lindex $lst 2]
      set nxthop [lindex $lst 1]
      #Si link no mcast, el nexthop es el id del nodo.
      #Si link mcast y DR, el nexhop es el id del nodo
      #Si link mcast y no DR, nexthop es *
      set tpl [$interfaces($iface) get_linktype]
      set dr [$interfaces($iface) is_dr]
      if { $tpl <= 2 &&  $dr != 1} {
       set nxthop  *
      }
      #Si la relacion parent/child es no valida, retorna
      if { [$self child_is_valid $iif $nhop $iface $nxthop] == 1 }
{return}
      #Si nodo es leaf, (iif = -2, por ahora) crea asi la entrada
      set leaf 0
      if { $iif == -2 } { set leaf 1}
      #Elimina Quit_Notificatio_Timer, si existe
      if [info exists qntimer($gr:$iface:$nxthop)] {
         $qntimer($gr:$iface:$nxthop) cancel
      }
      #Crea entrada transitoria
      $TransientTable add_entry $gr $target $iface $nxthop $leaf
      #Agrega child
      $TransientTable add_child $gr $iif $nhop
   }
}


######### ECHO REQUEST


######## SEND ADM


##########################################################################
########

#Clase Echorq_adm
#Existe un objeto de este tipo por cada agente CBT
#Debido a que los echo requests son manejados por timers y que estos
pueden
#involucrar a varios grupos, aca se administra esto. Cuando una
entrada en
#MFC pide agregarse a echo request, se crea un timer si no existe, o
se
#agrega el grupo al ya existente. Cuando una entrada se elimina, se
testea
#si este es el ultimo usuario del timer, y si lo es, se borra
Class Echorq_adm

Echorq_adm instproc init {simul node cbtagent} {
   $self instvar ns Node CBTagent
   set CBTagent $cbtagent
   set Node $node
   set ns $simul
}
```

```
Echorq_adm instproc exists {interface nexthop}  {
   $self instvar echorq_timer
   if [info exists echorq_timer($interface:$nexthop)] {
      return 1
   } else {
      return 0
   }
}

Echorq_adm instproc elim {interface  nexthop} {
   $self instvar ns Node echorq_timer
   unset echorq_timer($interface:$nexthop)
}

Echorq_adm instproc add_group {group interface nexthop} {
   $self instvar ns Node echorq_timer echorq_list CBTagent
   if [info exists echorq_timer($interface:$nexthop)] {
      set lis $echorq_list($interface:$nexthop)
      set idx [lsearch -exact $lis $group]
      if {$idx < 0} {
         lappend lis $group
         set echorq_list($interface:$nexthop) $lis
      }
   } else {
      set lis ""
      lappend lis $group
      set echorq_list($interface:$nexthop) $lis
      #Instancia el objeto timer
      set type [CBT set ECHO_REQUEST]
      set origen [$CBTagent get_nodeid]
      if {$nexthop == "*"} {
         set envio M
      } else {
         set envio U
      }
      set msg "$type/$origen/$envio/$origen"
      #Parametros para el proc timeout
      set tout ""
      lappend tout $msg
      lappend tout $interface
      lappend tout $envio
      lappend tout $nexthop
      lappend tout $CBTagent
      set ptimeout $tout
      #Parametros para el proc final
      set fin ""
      lappend fin $interface
      lappend fin $nexthop
      lappend fin $self
      set pfin $fin
      set simul [$CBTagent get_simulator]
      set id "[$Node id] Echo_Request_Timer($interface,$nexthop)"
      set echorq_timer($interface:$nexthop) [new Echo_Request_Timer
$id $simul $CBTagent $ptimeout $pfin]
      $echorq_timer($interface:$nexthop) sched [CBT set ECHO_INTERVAL]
*
   }
}

#La busqueda se hace por grupo, ya que esto identifica univocamente al
#timer involucrado
```

```
Echorq_adm instproc delete_group {group} {
   $self instvar ns Node echorq_timer echorq_list CBTagent
   set nam [array names echorq_list]
   foreach timer $nam {
      set lis $echorq_list($timer)
      set idx [lsearch -exact $lis $group]
      if {$idx >= 0} {
         set lis [lreplace $lis $idx $idx]
         if {[llength $lis] == 0} {
          set table [$CBTagent get_MFCTable]
          set li [$table get_parent $group]
          set interface [lindex $li 1]
          set nexthop [lindex $li 2]
          $echorq_timer($interface:$nexthop) cancel
          unset echorq_list($timer)
         } else {
       set echorq_list($timer) $lis
         }
      }
   }
}


########### RECEIVE
CBT instproc recv-echo_request { echorq_pdu } {
   $self instvar ns Node node_
   $self instvar cache_del_timer join_request_pending
   set iif [[$self set messager] set iif_]
   #Separacion de las componentes del echo request
   set orig_router [lindex $echorq_pdu 0]
   set type_join [lindex $echorq_pdu 1]
   set dest_join [lindex $echorq_pdu 2]
   #Obtiene los grupos que son parents para (iif, hop) en pgrps
   if {$type_join == "M" } {set nh *}
   if {$type_join == "U" } {set nh $dest_join}
   set pgrps [$self get_parents $iif $nh]
   #Envio de echo reply para los grupos
   #hay grupos parents en la interfaz
   if {[llength $pgrps] > 0} {
      set type [CBT set ECHO_REPLY]
      set org [$Node id]
      set dd 0
      if {$type_join == "U"} {set dd $nh}
      #Hay que ver si mando todos los grupos o uno solo en el reply???
      set msg "$type/$org/$pgrps/$type_join/$org"
      $self send_msg $msg $iif $type_join $dd
   }
   #Interfaz de arribo es parent para algun grupo y echo_rq multicast
   #Es posible preguntar directamente por los timers para echo request
   set nam [array names echorq_timers]
   set idx [lsearch -exact $nam $iif:*]
   if {$idx >= 0 && $type_join == "M" } {
      #Rescheduling del timer para envio del echo request
      $send_echorq($iif:*) resched [CBT set ECHO_INTERVAL] *
   }
}


############### ECHO REPLY

##########  SEND
```

```
########### RECEIVE

CBT instproc recv-echo_reply { echoreply_pdu } {
   $self instvar ns Node MFCTable
   $self instvar cache_del_timer join_request_pending delete_entry
   set iif [[$self set messager] set iif_]
   #Separacion de las componentes del echo reply
   #Puede venir una unica direccion o varias
   set long [llength $echoreply_pdu]
   set orig_router [lindex $echoreply_pdu 0]
   set type [lindex $echoreply_pdu [expr $long -2]]
   set sender [lindex $echoreply_pdu [expr $long -1]]
   set listgroups [lreplace $echoreply_pdu [expr $long -2] [expr $long
-1] ]
   set listgroups [lreplace $listgroups 0 0]
   foreach grp $listgroups {
      set x [$MFCTable get_parent $grp]
      if {[lindex $x 0] > -1} {
         set l1 [lindex $x 1]
         set l2 [lindex $x 2]
         #si la interfaz, nexthop es parent para el grupo, refresh
         if {$l1 == $iif && ($l2 == $sender || $l2 == "*") } {
       $MFCTable resched_delete $grp
         }
      }
   }

}



##################### FLUSH-TREE

############## SEND
#Este proceso se encarga de recibir requerimientos para envio de flush
#tree. Por ahora solo envia el flush tree para cada requerimiento por
#separado. En el futuro podra estar controlado por un timer para
acumular
#posibles flush-tree. Esto en realidad arma el flush tree. Habra q
modif
CBT instproc send_flush_tree {group} {
   $self instvar listg MFCTable ns Node node_
   set gr [$MFCTable get_groups]
   foreach g $gr {
      set lch [$MFCTable get_child_if $g]
      set lnh [$MFCTable get_child_nh $g]
      set cnt -1
      foreach ii $lch {
      incr cnt 1
         if {![info exists ifs($ii)]} {
       set l ""
       set ifs($ii) $l
       set nhp($ii) [lindex $lnh $cnt]
         }
         set l $ifs($ii)
         lappend l $g
         set ifs($ii) $l
      }
```

```
   }
   set aa [array names ifs]
   foreach a $aa {
      set l1 $ifs($a)
      set l2 ""
      set grps ""
      lappend grps $group
      foreach g $grps {
         set s [lsearch -exact $l1 $g]
         if {$s > -1} {
       lappend l2 $g
         }
      }
      if { [llength $l1] == [llength $l2] } {
         set l2 ""
         lappend l2 -1
         set ifs($a) $l2
      } else {
         if { [llength $l2] == 0} {
       lappend l2 -2
       set ifs($a) $l2
         } else {
       set ifs($a) $l2
         }
      }
   }
   #Genera y envia los flush tree para cada interfaz (interfaz,
nexthop)
   set type [CBT set FLUSH_TREE]
   set sender [$Node id]
   foreach yy [array names ifs] {
      set lis $ifs($yy)
      set nhop $nhp($yy)
      set tm U
      if {$nhop == "*"} {
         set tm M
      }
      if {[lindex $lis 0] != -2} {
         set msg "$type/$lis/$tm/$sender"
         #Como se pudo generar la interfaz local, no le envia el flush
tree
         if {$yy != -2 } {
        $self send_msg $msg $yy $tm $nhop
         }
      }
   }
}


########### RECEIVE

#El flush tree se envia U o M segun
#la capacidad del link, y todo el que lo recibe por una interfaz
parent lo
#toma como valido
#Esto debe desencadenar el reenganche ..............

CBT instproc recv-flush_tree {ft_pdu} {
   $self instvar ns Node node_ TransientTable MFCTable
   $self instvar cache_del_timer join_request_pending delete_entry
   set iif [[$self set messager] set iif_]
```

```
   #Separacion de las componentes del flush tree
   #Se adopta que viene una unica  direccion, individual o todos los
grp
   #Todos los grupos, se asume grp = -1
   set long [llength $ft_pdu]
   set type [lindex $ft_pdu [expr $long -2]]
   set sender [lindex $ft_pdu [expr $long -1]]
   set listgroups [lreplace $ft_pdu [expr $long -2] [expr $long -1] ]
   #Se determina la totalidad de grupos para los cuales la int. es
parent
   set flushlist [$self if_parent $iif $listgroups]
   #Eliminacion de la informacion de los grupos involucrados
   foreach g $flushlist {
     #Aca hay que borrar tambien el timer (del_entry o invocarlo)
     #Si se invoca el proc de borrado de la entrada directamente,no se
     #cancelara el timer de borrado asociado. Esto puede ocasionar
     #problemas cuando se crea una nueva entrada.
     #En su lugar, se invoca el timer y se lo cancela.
     ###$MFCTable delete_entry $g
     set ent [$MFCTable get_entry $g]
     set tim [$ent set del_entry]
     $tim cancel
   }
   #Reenganche de los grupos locales
   $self instvar joined_groups
   foreach g $flushlist {
      set lx [lsearch -exact  $joined_groups $g]
      if { $lx != -1 } {
         $self add-leaf $g -2
      }
   }
}


CBT instproc reeng {g} {
   $self instvar joined_groups
   set lx [lsearch -exact  $joined_groups $g]
   if { $lx != -1 } {
      $self add-leaf $g -2
   }
}


###################QUIT NOTIFICATION

####### SEND
CBT instproc send_qn { group } {
   $self instvar MFCTable ns Node node_ interfaces
   $self instvar qntimer
   set orig_router [$Node id]
   #Obtiene core, interface, nexthop para el grupo
   set list1 [$MFCTable get_parent $group]
   set pif  [lindex $list1 1]
   set dest [lindex $list1 2]
   set envio [$interfaces($pif) sendmode]
   if {$envio == "M"} { set dest * }
   #Envio del primer quit notification
   set type [CBT set QUIT_NOTIFICATION]
   set msg "$type/$group/$orig_router/$envio/$orig_router"
   $self send_msg $msg $pif $envio $dest
   #setea timer y cuenta de reenvio
```

```
   set tout ""
   lappend tout $msg
   lappend tout $pif
   lappend tout $envio
   lappend tout $dest
   lappend tout $self
   set ptimeout $tout
   set fin ""
   lappend fin $group
   lappend fin $pif
   lappend fin $dest
   lappend fin $self
   set pfin $fin
   set pfin $fin
   set id "[$Node id] Quit_Notification_Timer($group:$pif,$dest)"
   set qntimer($group:$pif:$dest) [new Quit_Not_Timer $id $ns $self
$ptimeout $pfin]
   $qntimer($group:$pif:$dest) sched [CBT set HOLDTIME] [CBT set
MAX_RTX]
}


### RECEIVE

CBT instproc recv-quit_notification { qn_pdu } {
   $self instvar  ns Node MFCTable TransientTable cache_del_timer
   $self instvar jr_pending interfaces
   #Separacion de las componentes del quit notification
   set group [lindex $qn_pdu 0]
   set orig_router [lindex $qn_pdu 1]
   set type [lindex $qn_pdu 2]
   set sender [lindex $qn_pdu 3]
   set gr [expr $group]
   #Determinar interfaz de arribo, si es no valida descartar
   #Casos especiales de iif:
   # -1 es interfaz no valida
   #La iif -2, indica que recibo el request a traves de un agente
local
   #(puede ser un agente IGMP o un agente para introducir la pdu desde
   #el simulador)
   set iif [[$self set messager] set iif_]
   if {$iif == -1} {
      set m "Nodo:[$Node id]  Descarta QUIT-NOT (recibido por interfaz
incorrecta, $iif) tiempo [$ns now]"
      $self disc_puts $m
      return
   }
   if {$type == "U"} {
      set nhop $sender
   } else {
      set nhop *
   }
   #Nodo no es on-tree para el grupo.
   if { [$MFCTable exists_entry $gr] == -1 } {
      set m "Nodo:[$Node id]  Descarta QUIT-NOT (grupo no on tree)
tiempo [$ns now]"
      $self disc_puts $m
      return
   }
   #Nodo es on tree para el grupo
   if { [$MFCTable exists_entry $gr] != -1 } {
```

```
      set l1 [$MFCTable get_parent $gr]
      set ip [lindex $l1 1]
      set tp [lindex $l1 2]
      #Si soy el core del grupo lo ignoro
      if {$ip == -1 && $tp == -1} {
         set m "Nodo:[$Node id]  Descarta QUIT-NOT (core para grupo)
tiempo [$ns now]"
         $self disc_puts $m
         return
      }
      #QN recibido por parent
      if { $ip == $iif && $tp == $nhop} {
         #QN enviado unicast, es descartado
         if { $type != "M" } {
        set m "Nodo:[$Node id]  Descarta QUIT-NOT (unicast recibido por
interfaz parent) tiempo [$ns now]"
        $self disc_puts $m
        return
         } else {
         #QN enviado multicast, otro nodo desea prune, debe enviarse
JR
        #Pone un timer al azar entre 0 y Holdtime para enviar join
request
        #JR sera enviado multicast
        #Puede haber un problema, los nodos mcast cambiaron de rol
        #debido al rearmado del arbol, ahora lo recibo por parent
        #mcast pero no es maccess; debo ignorarlo en este ultimo
        #caso. Es decir, si link es pap mcast (1) descarto
        set lt [$interfaces($iif) get_linktype]
        if {$lt == 1} { return}
        set fin ""
        set pfin $fin
        #Crea params para el proc timeout
        set type [CBT set JOIN_REQUEST]
        #set core [lindex $parent 0]
        set core [lindex $l1 0]
        set orig [$Node id]
        set msg "$type/$group/$core/$orig/M/$orig"
        set tout ""
        lappend tout $msg
        lappend tout $iif
        lappend tout M
        lappend tout $nhop
        lappend tout $self
        set ptimeout $tout
        set id "[$Node id] Send_Join_Timer($iif, $nhop)"
        set jr_pending($gr) [new Send_Join_Timer $id $ns $self
$ptimeout $pfin]
        #Genera numero al azar entre 0 y HOLDTIME
        set rand [expr [ns-random]/double(0x7fffffff)]
        set rand [expr $rand * [CBT set HOLDTIME]]
        $jr_pending($gr) sched $rand 0
        return
         }
      } else {
      #QN recibido por una interfaz no parent
         set ischild [$MFCTable lookup_child $gr $iif $nhop]
         #Interfaz es child
         if {$ischild >= 0} {
        #QN multicast, timer para eliminar entrada child
        if {$type == "M"} {
```

```
           #Si aun no hay timer
           if {![info exists cache_del_timer($gr:$iif)]} {
              set fin ""
              lappend fin $gr
              lappend fin $iif
              lappend fin $self
              set pfin $fin
              #Crea params para el proc timeout
              set tout ""
              lappend tout $MFCTable
              lappend tout $gr
              lappend tout $iif
              lappend tout $nhop
              set ptimeout $tout
              set id "[$Node id] Cache_Del_Timer($gr,$iif,$nhop)"
              set cache_del_timer($gr:$iif) [new Cache_Del_Timer $id
$ns $self $ptimeout $pfin]
              $cache_del_timer($gr:$iif) sched [CBT set
CACHE_DEL_TIMER]  0
              return
           }
        } else {
        #QN no es multicast, se elimina la interfaz
           $MFCTable delete_child $gr $iif $nhop
        }
          }
       }
   }
}
```

A.8 interface.tcl

```
#Clase CBTinterface
#Cada objeto interface representa una interfaz del nodo

Class CBTinterface

#Inicializa las variables, pone valores por defecto para los
configurables
#Los valores para cada interfaz, son tomados de CBTv2 MIB
CBTinterface instproc init {agent lbl} {
   $self instvar CBTagent label linktype pref_value des_router address
   $self instvar hello_int
   set CBTagent $agent
   #Label: cbtInterfaceIfIndex
   set label $lbl
   #Address: cbtInterfaceAddress
   set address [[$CBTagent set Node] id]
   #Designated router
   set des_router -1
   #Preference value for hello protocol: cbtInterfaceHelloPreference
   set pref_value [CBT set HELLO_PREFERENCE]
   #Intervalo para el hello
   set hello_int [CBT set HELLO_INTERVAL]
   #Status: si esta habilitado o no el CBT en esta interface. A
agregar
   #Tipo de link, al inicializar, -1
   set linktype -1
}

#Setea tipo de link
```

```
CBTinterface instproc set_linktype {type} {
   $self instvar linktype
   set linktype $type
}


#Devuelve tipo de link
CBTinterface instproc get_linktype {} {
   $self instvar linktype
   return $linktype
}


#Setea preference value
CBTinterface instproc set_preference_value {value} {
   $self instvar pref_value
   set pref_value $value
}


#Devuelve preference value
CBTinterface instproc get_preference_value {} {
   $self instvar pref_value
   return $pref_value
}


#Setea valor de designated router en la interfaz
CBTinterface instproc set_des_router {value} {
   $self instvar des_router linktype
   if { ($linktype == 1 || $linktype == 3) && $value != -1 } {
      puts "No es posible setear designated router en link no maccess"
      return
   }
   set des_router $value
}


#Devuelve valor de designated router en la interfaz
CBTinterface instproc get_des_router {} {
   $self instvar des_router
   return $des_router
}

#Devuelve 1 si el nodo es el dr en la interfaz
CBTinterface instproc is_dr { } {
   $self instvar des_router CBTagent
   set n [[$CBTagent set Node] id ]
   if {$n == $des_router} {
      return 1
   } else {
      return 0
   }
}


#Devuelve el modo en que se debe enviar una PDU: U si vinculo no
soporta
#mcast o si vinculo es macces y es DR
CBTinterface instproc sendmode { } {
   set dr [$self is_dr]
```

```
   set lt [$self get_linktype]
   set mode U
   if {$lt == 1 } {set mode M}
   if {$dr == 0 && $lt ==2 } {set mode M}
   return $mode
}
```

A.9 mfc.tcl

```
#########################################################################
#######
#Clase MFC_entry : Entradas en la multicast forwarding cache
#Estos procedimientos contienen acciones especificas del protocolo

Class MFC_entry

# 1 Crea una nueva entrada en la tabla. Recibe los siguientes
parametros:
#   - Table: la tabla a la cual pertenece la entrada
#   - CBTag: el agente CBT correspondiente
#   - core: id del nodo core para el grupo. Si es core este nodo el id
es el de el
#   - pnt_interface: Label de la interfaz local por donde se accede al
core
#   - pnt_nexthop: Proximo nodo camino al core (* si es multicast)
# 2 - Si no existe un timer para generar ECHO_REQUEST para el par
interfaz,
#     nexthop, lo crea.
# 3 Crea e inicializa un timer para el borrado de la entrada en caso
de no
#   recibirse en echo reply en el tiempo EXPIRE_GROUP_TIME
MFC_entry instproc init {Table CBTag grp core pnt_interface
pnt_nexthop} {
   $self instvar group core_id parent_label parent_nexthop del_entry
   $self instvar Mfctable CBTagent child_if child_nh
   set group $grp
   set child_if ""
   set child_nh ""
   set CBTagent $CBTag
   set Mfctable $Table
   set core_id $core
        set parent_label $pnt_interface
        set parent_nexthop $pnt_nexthop
   set n [$CBTagent set Node]
   set s [$CBTagent set ns]
   set m "Node:[$n id] Agregando entrada MFC, group:$group core:$core
if:$parent_label nxp:$parent_nexthop at [$s now]"
   $CBTagent tab_puts $m
   #Si la entrada no es core . . .
   if {$parent_nexthop != -1 && $parent_label != -1} {
      #Creacion e inicializacion del timer para borrado de la entrada
      #Parametros para el proc timeout
      set tout ""
      lappend tout $Mfctable
      lappend tout $group
      set ptimeout $tout
      #Parametros para el proc final
           set fin ""
      lappend fin $Mfctable
      lappend fin $group
```

```
      set pfin $fin
      set simul [$CBTagent get_simulator]
      set Node [$CBTagent set Node]
      set id "[$Node id] Delete_Entry_Timer($group)"
      set del_entry [new Delete_Entry_Timer $id $simul $CBTagent
$ptimeout $pfin]
      $del_entry sched [CBT set GROUP_EXPIRE_TIME] 0
      #Crea o se agrega al envio de echo requests por la interfaz
parent
      set adm [$CBTagent get_Echorq]
      $adm add_group $group $parent_label $parent_nexthop
   }
        return $self
}

# Rescheduling del timer asociaado a la entrada, ejecutado al recibir
# ECHO_REPLY
MFC_entry instproc resched_delete {} {
   $self instvar del_entry
   $del_entry resched [CBT set GROUP_EXPIRE_TIME] 0
}


#Procedimientos a llevar a cabo cuando se elimina una entrada en la
tabla
# 1 Enviar quit notfication al parent
# 2 Enviar flush tree a cada child
# 3 Eliminarse de la lista de grupos involucrados en el echo request
MFC_entry instproc delete  {} {
   $self instvar group core_id parent_label parent_nexthop
delete_entry
   $self instvar Mfctable CBTagent del_entry
   #Envio de flush tree a cada child. Esto debe ser un notificar
pedido
   #de envio
   set n [$CBTagent set Node]
   set s [$CBTagent set ns]
   set m "Node:[$n id] Borrando entrada MFC, group:$group
core:$core_id if:$parent_label nxp:$parent_nexthop at [$s now]"
   $CBTagent tab_puts $m
   $CBTagent send_flush_tree $group
   #Envio de quit notification al parent
   $CBTagent send_qn $group
   #Eliminacion del timer para echorequest
   set adm [$CBTagent get_Echorq]
   $adm delete_group $group
   #Cada child, se elimina usando delete_child
   set ifz [$self get_child_if]
   set nhp [$self get_child_nh]
   set cnt 0
   foreach iz $ifz {
      incr cnt 1
      set nh [lindex $nhp $cnt]
      $self delete_child $iz $nh
   }
   return 0
}


#Devuelve una lista con info del parent: core, interface, nexthop
MFC_entry instproc get_parent {} {
```

```
   $self instvar core_id parent_label parent_nexthop
   set list1 ""
   lappend list1 $core_id
   lappend list1 $parent_label
   lappend list1 $parent_nexthop
   return $list1
}


#Devuelve una lista con info de child: interface
MFC_entry instproc get_child_if {} {
   $self instvar child_if
   return $child_if
}


#Devuelve una lista con info de child: nexthop
MFC_entry instproc get_child_nh {} {
   $self instvar child_nh
   return $child_nh
}


#Agrega un child en la entrada. Devuelve el indice de la entrada
agregada,
#si ya estaba, devuelve el indice sin agregarla
#La agrege o no, debe generar un join ack.
#La generacion del ack depende de si la interfaz de origen del JR es
local
#(-2) o es causada por un agente CBT.
#En ambos casos, el nexthop es -1, y no se genera un join ack
MFC_entry instproc add_child {ch_interface ch_nexthop} {
   $self instvar child_if child_nh CBTagent group core_id
   set resul [$self lookup_child $ch_interface $ch_nexthop]
   #Encontro la entrada, envia ack y devuelve indice
   if { $resul != -1 } {
      if {$ch_nexthop != -1 && $ch_nexthop != -2 } {
         $CBTagent send_join_ack $group $ch_interface $ch_nexthop
      }
      return $resul
   } else {
   #No encontro la entrada, la agrega, envia ack y devuelve indice
      set n [$CBTagent set Node]
      set s [$CBTagent set ns]
      set m "Node:[$n id] MFC:Agregando child, group:$group
core:$core_id if_ch:$ch_interface nxp_ch:$ch_nexthop at [$s now]"
      $CBTagent tab_puts $m
      lappend child_if $ch_interface
      lappend child_nh $ch_nexthop
      if {$ch_nexthop != -1  && $ch_interface != -2 } {
         $CBTagent send_join_ack $group $ch_interface $ch_nexthop
      }
      set resul [$self lookup_child $ch_interface $ch_nexthop]
      #Al agregar una entrada se debe hacer algo para que la proxima
vez
      #que se reciba un paquete src, gr, iif, se produzca un cache
miss
      #y esto provoque el agregado del replicator correspondiente
      #esto se produce automaticamente ya que el nuevo S lo producira
      #si manda algo; Lo que debe hacerse aparte es agregar la nueva
      #interfaz en los replicators ya existentes
```

```
      #Para todos los replicators del group, se hace el insert oif
      #Para esto se invoca a actualizar-replicators de CBT
      $CBTagent actualizar-replicators $group $ch_interface

   return $resul
   }
}


#Elimina un child en la entrada. Devuelve -1 si no existe
MFC_entry instproc delete_child {ch_interface ch_nexthop} {
   $self instvar child_if child_nh group CBTagent core_id
   set resul [$self lookup_child $ch_interface $ch_nexthop]
   if {$resul == -1} {
      return $resul
   } else {
      set n [$CBTagent set Node]
      set s [$CBTagent set ns]
      set m "Node:[$n id] MFC:Eliminando child, group:$group
core:$core_id if_ch:$ch_interface nxp_ch:$ch_nexthop at [$s now]"
      $CBTagent tab_puts $m
      #Eliminacion de los elementos de reenvio (inhabilita targets en
      #los replicators)
      $CBTagent borrar_child $group $ch_interface
      set child_if [lreplace $child_if $resul $resul]
      set child_nh [lreplace $child_nh $resul $resul]
      return 0
   }
}


#Dada una interfaz y un nexthop, busca la entrada child
correspondiente
#Devuelve el indice si lo encuentra y si no -1
MFC_entry instproc lookup_child {ch_interface ch_nexthop} {
   $self instvar child_if child_nh
   set cnt [llength $child_if]
   if {$cnt == 0} {
      return -1
   } else {
      set idx 0
      while {$idx < $cnt} {
         set inf [lindex $child_if $idx]
         set nhp [lindex $child_nh $idx]
         if {$inf == $ch_interface && $nhp == $ch_nexthop} {
            return $idx
         }
         incr idx 1
      }
      return -1
   }
}

#Devuelve la cantidad de entradas child
MFC_entry instproc num_child {} {
   $self instvar child_if
   return [llength $child_if]
}
```

```
##########################################################################
########
#Clase Transient_entry : Entradas en la transient table
#Estos procedimientos contienen acciones especificas del protocolo

Class Transient_entry

# 1 Crea una nueva entrada en la tabla. Recibe los siguientes
parametros:
#   -Table: objeto Table al que pertenece
#   -CBTag: Objeto agente CBT
#   -core: id del nodo core para el grupo. Si es core este nodo el id
es el de el
#   -pnt_interface: Label de la interfaz local por donde se accede al
core
#   -pnt_nexthop: Proximo nodo camino al core (* si es multicast)
#   -leaf: Indicacion si el nodo es leaf para la entrada (1) o no (0)
# 2 - Inicializa los datos child (vacio)
# 3 - Crea un timer para eliminar la entrada si no se recibe ack
Transient_entry instproc init {Table CBTag grp core pnt_interface
pnt_nexthop lf} {
   $self instvar group core_id parent_label parent_nexthop del_entry
   $self instvar TMfctable CBTagent child_if child_nh leaf retransmit
   set leaf $lf
   set retransmit 0
   set group $grp
   set child_if ""
   set child_nh ""
   set CBTagent $CBTag
   set TMfctable $Table
   set core_id $core
        set parent_label $pnt_interface
        set parent_nexthop $pnt_nexthop
   set n [$CBTagent set Node]
   set s [$CBTagent set ns]
   set y Y
   if {$leaf == 0} {set y N}
   set m "Node:[$n id] Agregando entrada Transitoria, group:$group
core:$core leaf:$y if:$parent_label nxp:$parent_nexthop at [$s now]"
   $CBTagent tab_puts $m
   #Creacion e inicializacion del timer para borrado de la entrada
   #Parametros para el proc timeout
   set tout ""
   set ptimeout $tout
   #Parametros para el proc final
        set fin ""
   lappend fin $TMfctable
   lappend fin $group
   #agrega code =1 , terminacion anormal
   lappend fin 1
   set pfin $fin
   set simul [$CBTagent get_simulator]
   set Node [$CBTagent set Node]
   set id "[$Node id] Delete_T_Entry_Timer($group)"
   set del_entry [new Delete_T_Entry_Timer $id $simul $CBTagent
$ptimeout $pfin]
   if {$leaf == 0} {
      $del_entry sched [CBT set TRANSIENT_TIMEOUT] 0
   } else {
      $del_entry sched [CBT set JOIN_TIMEOUT] 0
   }
```

```
   #Envia primer join request al parent (si leaf seteara timer y
   #devuelve objeto timer)
   #Esto lo debe hacer la tabla, ya q el jreq pide a la tabla el entry
   #y hasta q este proc no retorne, no existe
   ##$CBTagent send_jr $group
        return $self
}


#Procedimientos a llevar a cabo cuando se elimina una entrada en la
tabla
#La eliminacion puede ocurrir en forma normal (se recibio un ack) o en
#forma anormal (expiro el tiempo). Se recibe esta indicacion en code
(0 si
#terminacion normal, 1 si anormal)
#Terminacion anormal:
#   1-Cancelar el timer de reenvio si leaf
#Terminacion normal
#   1-Cancelar timer de reenvio si leaf
#   * El timer de expiracion no se puede cancelar, porque invoca a
este
#     procedimiento. Lo que ocurrira es que cuando cancele por tiempo
#     no tendra entrada para eliminar
Transient_entry instproc delete {} {
   $self instvar group core_id parent_label parent_nexthop
delete_entry
   $self instvar TMfctable CBTagent retransmit leaf
   set n [$CBTagent set Node]
   set s [$CBTagent set ns]
   set m "Node:[$n id] Eliminando entrada Transitoria, group:$group
core:$core_id if:$parent_label nxp:$parent_nexthop at [$s now]"
   $CBTagent tab_puts $m
   #Eliminacion del timer de reenvio
   if {$leaf == 1} {
      $retransmit cancel
   }
   return 0
}

#Devuelve informacion acerca de si el nodo es leaf en la entrada
transitoria
Transient_entry instproc get_leaf { } {
   $self instvar leaf
   return $leaf
}

Transient_entry instproc set_retransmit_timer {timer} {
   $self instvar retransmit
   set retransmit $timer
}

#Devuelve una lista con info del parent: core, interface, nexthop
Transient_entry instproc get_parent {} {
   $self instvar core_id parent_label parent_nexthop
   set lis ""
   lappend lis $core_id
   lappend lis $parent_label
   lappend lis $parent_nexthop
   return $lis
}
```

```
#Devuelve una lista con info de child: interface
Transient_entry instproc get_child_if {} {
   $self instvar child_if
   return $child_if
}


#Devuelve una lista con info de child: nexthop
Transient_entry instproc get_child_nh {} {
   $self instvar child_nh
   return $child_nh
}


#Agrega un child en la entrada. Devuelve el indice de la entrada
agregada,
#si ya estaba, devuelve el indice sin agregarla
Transient_entry instproc add_child {ch_interface ch_nexthop} {
   $self instvar child_if child_nh CBTagent group core_id
   set resul [$self lookup_child $ch_interface $ch_nexthop]
   if {$resul != -1} {
      return $resul
   } else {
      set n [$CBTagent set Node]
      set s [$CBTagent set ns]
      set m "Node:[$n id] Agregando child (transitorio), group:$group
core:$core_id if_ch:$ch_interface nxp_ch:$ch_nexthop at [$s now]"
      $CBTagent tab_puts $m
      lappend child_if $ch_interface
      lappend child_nh $ch_nexthop
      return 0
   }
}


#Elimina un child en la entrada. Devuelve -1 si no existe
Transient_entry instproc delete_child {ch_interface ch_nexthop} {
   $self instvar child_if child_nh core_id group CBTagent
   set resul [$self lookup_child $ch_interface $ch_nexthop]
   if {$resul == -1} {
      return $resul
   } else {
      set n [$CBTagent set Node]
      set s [$CBTagent set ns]
      set m "Node:[$n id] Eliminando child (transitorio), group:$group
core:$core_id if_ch:$ch_interface nxp_ch:$ch_nexthop at [$s now]"
      $CBTagent tab_puts $m
      set child_if [lreplace $child_if $resul $resul]
      set child_nh [lreplace $child_nh $resul $resul]
      return 0
   }
}


#Dada una interfaz y un nexthop, busca la entrada child
correspondiente
#Devuelve el indice si lo encuentra y si no -1
Transient_entry instproc lookup_child {ch_interface ch_nexthop} {
   $self instvar child_if child_nh
   set cnt [llength $child_if]
```

```
   if {$cnt == 0} {
      return -1
   } else {
      set idx 0
      while {$idx < $cnt} {
         set inf [lindex $child_if $idx]
         set nhp [lindex $child_nh $idx]
         if {$inf == $ch_interface && $nhp == $ch_nexthop} {
            return $idx
         }
         incr idx 1
      }
      return -1
   }
}

#Devuelve la cantidad de entradas child
Transient_entry instproc num_child {} {
   $self instvar child_if
   return [llength $child_if]
}




##########################################################################
######

#Clase MFC_Table: multicast forwarding cache, contiene MFC-entries

Class MFC_Table

#Inicializa un objeto MFC_Table: pone su tabla en 0
MFC_Table instproc init {cbt_ag} {
   $self instvar entry cbt_agent
        set cbt_agent $cbt_ag
   return $self
}

#Devuelve el objeto Entry asociado al grupo
#-1 si no existe entrada para el grupo
MFC_Table instproc get_entry {group} {
   $self instvar entry
   if [info exists entry($group)] {
           return $entry($group)
        } else {
           return -1
        }
}


#Borra la  entrada asociada al grupo. Invoca al procedimiento delete
de la
#entrada
#Si el grupo no existe o si la entrada da error, devuelve -1
MFC_Table instproc delete_entry {group} {
   $self instvar entry
   if [info exists entry($group)] {
           set ent $entry($group)
           set result [$ent delete]
           if {$result != -1} {unset entry($group)}
```

```
            return $result
   } else {
            return -1
        }
}


#Crea una nueva entrada en la tabla. Devuelve -1 si la entrada ya
existe o si
#el procedimiento de creacion del objeto entry da error
MFC_Table instproc add_entry {group core interface nexthop} {
   $self instvar entry cbt_agent
   if [info exists entry($group)] {
            return -1
        } else {
      set ent [new MFC_entry $self $cbt_agent $group $core $interface
$nexthop]
            if {$ent == -1} {
               return -1
            } else {
               set entry($group) $ent
               return 0
            }
   }
}



#Crea una entrada core en la tabla. Falta: Devuelve -1 si la entrada
ya existe
# o si ya hay otro core, etc ....
MFC_Table instproc add_core_entry {group core} {
   $self instvar entry cbt_agent
   if [info exists entry($group)] {
            return -1
        } else {
      set ent [new MFC_entry $self $cbt_agent $group $core -1 -1]
            if {$ent == -1} {
               return -1
            } else {
               set entry($group) $ent
               return 0
            }
   }
}




#Dado un grupo, devuelve informacion del parent: core, interface,
nexthop
MFC_Table instproc get_parent {group} {
   $self instvar entry
   if {![info exists entry($group)]} {
            return -1
        } else {
      return [$entry($group) get_parent]
   }
}
```

```
MFC_Table instproc resched_delete {group} {
   $self instvar entry
   if {![info exists entry($group)]} {
           return -1
        } else {
      return [$entry($group) resched_delete]
   }
}



#Dado un grupo, una interface y un nexthop, devuelve -1 si no existe
child
MFC_Table instproc lookup_child {group interface nexthop} {
   $self instvar entry
   if {![info exists entry($group)]} {
           return -1
        } else {
      return [$entry($group) lookup_child $interface $nexthop]
   }
}



#Dado un grupo, devuelve informacion de child: interface
MFC_Table instproc get_child_if {group} {
   $self instvar entry
   if {![info exists entry($group)]} {
           return -1
        } else {
      return [$entry($group) get_child_if]
   }
}


#Dado un grupo, devuelve informacion de child: nexthop
MFC_Table instproc get_child_nh {group} {
   $self instvar entry
   if {![info exists entry($group)]} {
           return -1
        } else {
      return [$entry($group) get_child_nh]
   }
}


#Dado un grupo, crea una entrada child en el entry correspondiente
#Esto implica el envio del join-ack por la child interface (a cargo
del add
#que realiza el entry)
MFC_Table instproc add_child {group interface nexthop} {
   $self instvar entry cbt_agent
   if {![info exists entry($group)]} {
           return -1
        } else {
           set resul [$entry($group) add_child $interface $nexthop]
      return $resul
   }
}
```

```
#Dado un grupo, elimina una entrada child en el entry correspondiente
#Si es la ultima child para el grupo debe eliminar la entrada
MFC_Table instproc delete_child {group interface nexthop} {
   $self instvar entry cbt_agent
   if {![info exists entry($group)]} {
           return -1
        } else {
           set resul [$entry($group) delete_child $interface $nexthop]
      if { [$entry($group) num_child] == 0 } {
         $self delete_entry $group
      }
      return $resul
   }
}


#Dado un grupo una interface de arribo y un nexthop (previous)
devuelve
#-1 si la entrada no existe, 0 si existe
MFC_Table instproc exists_entry {group } {
#interface nexthop
   $self instvar entry cbt_agent
   if {![info exists entry($group)]} {
           return -1
   }
   return 0
}


#Devuelve los grupos on tree en el nodo
MFC_Table instproc get_groups {} {
   $self instvar entry
   set xl [array names entry]
   return $xl
}


##############################################################################
#######

#Clase Transient_Table: transient table, contiene Transient_entries

Class Transient_Table

#Inicializa un objeto Transient_Table: pone su tabla en 0
Transient_Table instproc init {cbt_ag} {
   $self instvar entry cbt_agent
        set cbt_agent $cbt_ag
   return $self
}

#Devuelve el objeto Entry asociado al grupo
#-1 si no existe entrada para el grupo
Transient_Table instproc get_entry {group} {
   $self instvar entry
   if [info exists entry($group)] {
           return $entry($group)
        } else {
           return -1
        }
}
```

```
#Borra la  entrada asociada al grupo. Invoca al procedimiento delete
de la
#entrada
#Recibe un codigo de terminacion normal(0) o anormal (1)
#Si el grupo no existe o si la entrada da error, devuelve -1
#Este caso puede ocurrir,  ya que el timer que cancela la entrada
(timeout)
#al no haber recibido un ack, puede querer borrar la entrada que ya ha
#sido borrada normalmente
Transient_Table instproc delete_entry {group} {
   $self instvar entry
   if [info exists entry($group)] {
           set ent $entry($group)
           set result [$ent delete ]
           if {$result != -1} {unset entry($group)}
           return $result
   } else {
           return -1
        }
}


#Crea una nueva entrada en la tabla. Devuelve -1 si la entrada ya
existe o si
#el procedimiento de creacion del objeto entry da error
Transient_Table instproc add_entry {group core interface nexthop leaf}
{
   $self instvar entry cbt_agent
   if [info exists entry($group)] {
           return -1
       } else {
      set ent [new Transient_entry $self $cbt_agent $group $core
$interface $nexthop $leaf]
           if {$ent == -1} {
              return -1
           } else {
              set entry($group) $ent
         $cbt_agent send_jr $group
              return 0
           }
   }
}

#Dado un grupo, devuelve informacion acerca de si la entrada es leaf o
no
Transient_Table instproc get_leaf {group} {
   $self instvar entry
   if {![info exists entry($group)]} {
           return -1
        } else {
      return [$entry($group) get_leaf]
   }
}


#Dado un grupo, devuelve informacion del parent: core, interface,
nexthop
Transient_Table instproc get_parent {group} {
   $self instvar entry
```

```
   if {![info exists entry($group)]} {
           return -1
        } else {
      return [$entry($group) get_parent]
   }
}


#Dado un grupo, devuelve informacion de child: interface
Transient_Table instproc get_child_if {group} {
   $self instvar entry
   if {![info exists entry($group)]} {
           return -1
        } else {
      return [$entry($group) get_child_if]
   }
}


#Dado un grupo, devuelve informacion de child: nexthop
Transient_Table instproc get_child_nh {group} {
   $self instvar entry
   if {![info exists entry($group)]} {
           return -1
        } else {
      return [$entry($group) get_child_nh]
   }
}

#Dado un grupo una interface de arribo y un nexthop (previous)
devuelve
#-1 si la entrada no existe, 0 si existe
Transient_Table instproc exists_entry {group} {
   $self instvar entry cbt_agent
   if {![info exists entry($group)]} {
           return -1
   }
   return 0
}


#Dado un grupo, crea una entrada child en el entry correspondiente
Transient_Table instproc add_child {group interface nexthop} {
   $self instvar entry cbt_agent
   if {![info exists entry($group)]} {
           return -1
        } else {
           set resul [$entry($group) add_child $interface $nexthop]
      return $resul
   }
}


#Dado un grupo, elimina una entrada child en el entry correspondiente
#Si es la ultima child para el grupo debe eliminar la entrada
#Hay que ver si es necesario enviar code de terminacion normal o
anormal
Transient_Table instproc delete_child {group interface nexthop} {
   $self instvar entry cbt_agent
   if {![info exists entry($group)]} {
           return -1
```

```
        } else {
           set resul [$entry($group) delete_child $interface $nexthop]
       if { [$entry($group) num_child] == 0 } {
          $self delete_entry $group
       }
       return $resul
   }
}

Transient_Table instproc set_retransmit_timer {group timer} {
   $self instvar entry
   if {![info exists entry($group)]} {
           return -1
        } else {
           $entry($group) set_retransmit_timer $timer
   }
}
```

Apéndice B: Ejemplos

Se presenta a continuación ejemplos de simulaciones que utilizan las facilidades implementadas. Los ejemplos reflejan la operación del protocolo CBT y las facilidades de configuración y debugging implementadas.

Para cada ejemplo se muestra el código correspondiente al script de simulación y las salidas producidas.

B.1 Ejemplo de las facilidades de debugging activadas por eventos

El objetivo de este ejemplo es mostrar la configuración y la salida producida por las facilidades de debugging consistentes en la producción de mensajes cuando ocurren ciertos eventos. Se muestra a continuación el script de simulación y luego las salidas generadas por el tracing Ns y el debugging.

B.1.1 Código

```
# Ejemplo de una topologia compuesta por 8 nodos. Los vinculos son
punto a
# punto.
# Por salida standard se muestra el intercambio de PDUs entre los
agentes
# CBT para la secuencia dada, utilizando la capacidad de debugging
implementada.
# En el archivo mensaje se muestra el intercambio de mensajes entre
los
# agentes emisores y receptores implementados para prueba
# En el archivo out.tr se muestra el tracing completo provisto por Ns

set ns [new Simulator]
set dels [open mensajes w]
Simulator set EnableMcast_ 1
Simulator set NumberInterfaces_ 1

# Habilitacion de las facilidades de tracing Ns
set f [open out.tr w]
$ns trace-all $f

# Carga del codigo implementado
source cbtmain.tcl

#Generacion de nodos
puts "Generando nodos . . ."
for {set i 0} {$i <= 7} {incr i 1} {
   set n($i) [$ns node]
}

#Topologia
puts "Generando links .  . . "
$ns duplex-link $n(0) $n(1) 1.5Mb 10ms DropTail
$ns duplex-link $n(1) $n(2) 1.5Mb 10ms DropTail
$ns duplex-link $n(1) $n(5) 1.5Mb 10ms DropTail
$ns duplex-link $n(1) $n(7) 1.5Mb 10ms DropTail
```

```
$ns duplex-link $n(2) $n(3) 1.5Mb 10ms DropTail
$ns duplex-link $n(2) $n(5) 1.5Mb 10ms DropTail
$ns duplex-link $n(3) $n(4) 1.5Mb 10ms DropTail
$ns duplex-link $n(4) $n(5) 1.5Mb 10ms DropTail
$ns duplex-link $n(5) $n(6) 1.5Mb 10ms DropTail

#Agentes CBT
puts "Generando agentes CBT . . . "
for {set i 0} {$i <= 7} {incr i 1} {
   set cbt($i) [new CBT $ns $n($i)]
   $cbt($i) enable_ALL
}

#Arranque de los agentes CBT
for {set i 0} {$i <= 7} {incr i 1} {
   $cbt($i) start
}


#Configuracion de nodo n(3) como core para el grupo 65490
$cbt(3) set-core 65490

#Creacion de los agentes multicast de prueba
#Emisores al grupo, en nodos n(0) y n(4), emiten 10 paquetes/segundo
#de 200 y 300 bytes c/u respectivamente
#Los agentes emisores deben realizar el join group debido a que no
esta
#implementado el envio desde un nodo no miembro del grupo
   set  e(0) [new Agent/Message/Emisor_delay 0.1 $dels]
   $ns attach-agent $n(0) $e(0)
   $ns at 0.0 "$e(0) join-group 65490"
   $e(0) set dst_ 65490
   $e(0) set packetSize_ 300
   set  e(4) [new Agent/Message/Emisor_delay 0.1 $dels]
   $ns attach-agent $n(4) $e(4)
   $ns at 0.0 "$e(4) join-group 65490"
   $e(4) set dst_ 65490
   $e(4) set packetSize_ 200

#Agentes receptores en nodos 6, 2 y 1. Registran los mensajes en el
archivo
#mensajes
   set re(6) [new Agent/Message/Receptor_delay $dels]
   $ns attach-agent $n(6) $re(6)
   set re(2) [new Agent/Message/Receptor_delay $dels]
   $ns attach-agent $n(2) $re(2)
   set re(1) [new Agent/Message/Receptor_delay $dels]
   $ns attach-agent $n(1) $re(1)

#Arranques de los agentes emisores
   $ns at 2.0 "$e(0) start"
   $ns at 10.0 "$e(4) start"
   $ns at 32.0 "$e(0) stop"
   $ns at 32.0 "$e(4) stop"

#Join y leaves de los receptores
   $ns at 2.0 "$re(6) join-group 65490"
   $ns at 2.0 "$re(1) join-group 65490"
   $ns at 5.0 "$re(2) join-group 65490"
   $ns at 12.0 "$re(6) leave-group 65490"
   $ns at 140.0 "final"
```

```
proc final {} {
   global ns f dels
   $ns flush-trace
   close $f
   close $dels
   exit 0
}

$ns run
```

B.1.2 Archivo de tracing generado por Ns (Se muestra sólo parcialmente por razones de extensión)

```
+ 0 0 1 message 180 ------ 0 0.1 255.207 -1 0
- 0 0 1 message 180 ------ 0 0.1 255.207 -1 0
+ 0 4 3 message 180 ------ 0 4.1 255.207 -1 1
- 0 4 3 message 180 ------ 0 4.1 255.207 -1 1
+ 0.01096 3 2 message 180 ------ 0 4.1 255.207 -1 1
- 0.01096 3 2 message 180 ------ 0 4.1 255.207 -1 1
+ 0.01096 3 4 message 180 ------ 0 3.1 255.207 -1 2
- 0.01096 3 4 message 180 ------ 0 3.1 255.207 -1 2
+ 0.01096 1 2 message 180 ------ 0 0.1 255.207 -1 0
- 0.01096 1 2 message 180 ------ 0 0.1 255.207 -1 0
+ 0.01096 1 5 message 180 ------ 0 0.1 255.207 -1 0
- 0.01096 1 5 message 180 ------ 0 0.1 255.207 -1 0
+ 0.01096 1 7 message 180 ------ 0 0.1 255.207 -1 0
- 0.01096 1 7 message 180 ------ 0 0.1 255.207 -1 0
+ 0.01096 1 2 message 180 ------ 0 1.1 255.207 -1 3
- 0.01192 1 2 message 180 ------ 0 1.1 255.207 -1 3
d 0.02192 3 2 message 180 ------ 0 4.1 255.207 -1 1
+ 0.02192 4 5 message 180 ------ 0 3.1 255.207 -1 2
- 0.02192 4 5 message 180 ------ 0 3.1 255.207 -1 2
d 0.02192 1 2 message 180 ------ 0 0.1 255.207 -1 0
d 0.02192 1 5 message 180 ------ 0 0.1 255.207 -1 0
d 0.02192 1 7 message 180 ------ 0 0.1 255.207 -1 0
+ 0.02288 2 3 message 180 ------ 0 1.1 255.207 -1 3
- 0.02288 2 3 message 180 ------ 0 1.1 255.207 -1 3
+ 0.02288 2 5 message 180 ------ 0 1.1 255.207 -1 3
- 0.02288 2 5 message 180 ------ 0 1.1 255.207 -1 3
+ 0.02288 2 3 message 180 ------ 0 2.1 255.207 -1 4
- 0.02384 2 3 message 180 ------ 0 2.1 255.207 -1 4
d 0.03288 4 5 message 180 ------ 0 3.1 255.207 -1 2
d 0.03384 2 3 message 180 ------ 0 1.1 255.207 -1 3
d 0.03384 2 5 message 180 ------ 0 1.1 255.207 -1 3
+ 0.0348 3 4 message 180 ------ 0 2.1 255.207 -1 4
- 0.0348 3 4 message 180 ------ 0 2.1 255.207 -1 4
+ 0.0348 3 2 message 180 ------ 0 3.1 255.207 -1 5
- 0.0348 3 2 message 180 ------ 0 3.1 255.207 -1 5
d 0.04576 3 4 message 180 ------ 0 2.1 255.207 -1 4
+ 0.04576 2 1 message 180 ------ 0 3.1 255.207 -1 5
- 0.04576 2 1 message 180 ------ 0 3.1 255.207 -1 5
+ 0.04576 2 5 message 180 ------ 0 3.1 255.207 -1 5
- 0.04576 2 5 message 180 ------ 0 3.1 255.207 -1 5
+ 0.04576 2 1 message 180 ------ 0 2.1 255.207 -1 6
- 0.04672 2 1 message 180 ------ 0 2.1 255.207 -1 6
d 0.05672 2 1 message 180 ------ 0 3.1 255.207 -1 5
d 0.05672 2 5 message 180 ------ 0 3.1 255.207 -1 5
+ 0.05768 1 0 message 180 ------ 0 2.1 255.207 -1 6
```

```
- 0.05768 1 0 message 180 ------ 0 2.1 255.207 -1 6
+ 0.05768 1 5 message 180 ------ 0 2.1 255.207 -1 6
- 0.05768 1 5 message 180 ------ 0 2.1 255.207 -1 6
+ 0.05768 1 7 message 180 ------ 0 2.1 255.207 -1 6
- 0.05768 1 7 message 180 ------ 0 2.1 255.207 -1 6
+ 0.05768 1 0 message 180 ------ 0 1.1 255.207 -1 7
- 0.05864 1 0 message 180 ------ 0 1.1 255.207 -1 7
d 0.06864 1 0 message 180 ------ 0 2.1 255.207 -1 6
d 0.06864 1 5 message 180 ------ 0 2.1 255.207 -1 6
d 0.06864 1 7 message 180 ------ 0 2.1 255.207 -1 6
+ 2 0 1 message 300 ------ 0 0.2 255.210 -1 8
- 2 0 1 message 300 ------ 0 0.2 255.210 -1 8
+ 2 6 5 message 180 ------ 0 6.1 255.207 -1 9
- 2 6 5 message 180 ------ 0 6.1 255.207 -1 9
+ 2.01096 5 1 message 180 ------ 0 6.1 255.207 -1 9
- 2.01096 5 1 message 180 ------ 0 6.1 255.207 -1 9
+ 2.01096 5 2 message 180 ------ 0 6.1 255.207 -1 9
- 2.01096 5 2 message 180 ------ 0 6.1 255.207 -1 9
+ 2.01096 5 4 message 180 ------ 0 6.1 255.207 -1 9
- 2.01096 5 4 message 180 ------ 0 6.1 255.207 -1 9
+ 2.01096 5 2 message 180 ------ 0 5.1 255.207 -1 10
+ 2.0116 1 2 message 300 ------ 0 0.2 255.210 -1 8
- 2.0116 1 2 message 300 ------ 0 0.2 255.210 -1 8
- 2.01192 5 2 message 180 ------ 0 5.1 255.207 -1 10
d 2.02192 5 1 message 180 ------ 0 6.1 255.207 -1 9
d 2.02192 5 2 message 180 ------ 0 6.1 255.207 -1 9
d 2.02192 5 4 message 180 ------ 0 6.1 255.207 -1 9
+ 2.02288 2 1 message 180 ------ 0 5.1 255.207 -1 10
- 2.02288 2 1 message 180 ------ 0 5.1 255.207 -1 10
+ 2.02288 2 3 message 180 ------ 0 5.1 255.207 -1 10
- 2.02288 2 3 message 180 ------ 0 5.1 255.207 -1 10
+ 2.02288 2 5 message 180 ------ 0 2.1 255.207 -1 11
- 2.02288 2 5 message 180 ------ 0 2.1 255.207 -1 11
+ 2.0232 2 5 message 300 ------ 0 0.2 255.210 -1 8
+ 2.0232 2 3 message 300 ------ 0 0.2 255.210 -1 8
- 2.02384 2 3 message 300 ------ 0 0.2 255.210 -1 8
- 2.02384 2 5 message 300 ------ 0 0.2 255.210 -1 8
d 2.03384 2 1 message 180 ------ 0 5.1 255.207 -1 10
d 2.03384 2 3 message 180 ------ 0 5.1 255.207 -1 10
+ 2.03384 5 1 message 180 ------ 0 2.1 255.207 -1 11
- 2.03384 5 1 message 180 ------ 0 2.1 255.207 -1 11
+ 2.03384 5 4 message 180 ------ 0 2.1 255.207 -1 11
- 2.03384 5 4 message 180 ------ 0 2.1 255.207 -1 11
+ 2.03384 5 6 message 180 ------ 0 2.1 255.207 -1 11
- 2.03384 5 6 message 180 ------ 0 2.1 255.207 -1 11
+ 2.03384 5 6 message 180 ------ 0 5.1 255.207 -1 12
- 2.0348 5 6 message 180 ------ 0 5.1 255.207 -1 12
+ 2.03544 3 4 message 300 ------ 0 0.2 255.210 -1 8
- 2.03544 3 4 message 300 ------ 0 0.2 255.210 -1 8
+ 2.03544 5 6 message 300 ------ 0 0.2 255.210 -1 8
- 2.03576 5 6 message 300 ------ 0 0.2 255.210 -1 8
d 2.0448 5 1 message 180 ------ 0 2.1 255.207 -1 11
d 2.0448 5 4 message 180 ------ 0 2.1 255.207 -1 11
d 2.0448 5 6 message 180 ------ 0 2.1 255.207 -1 11
+ 2.1 0 1 message 300 ------ 0 0.2 255.210 -1 13
- 2.1 0 1 message 300 ------ 0 0.2 255.210 -1 13
+ 2.1116 1 2 message 300 ------ 0 0.2 255.210 -1 13
- 2.1116 1 2 message 300 ------ 0 0.2 255.210 -1 13
+ 2.1232 2 5 message 300 ------ 0 0.2 255.210 -1 13
- 2.1232 2 5 message 300 ------ 0 0.2 255.210 -1 13
+ 2.1232 2 3 message 300 ------ 0 0.2 255.210 -1 13
```

```
- 2.1232 2 3 message 300 ------ 0 0.2 255.210 -1 13
+ 2.1348 5 6 message 300 ------ 0 0.2 255.210 -1 13
- 2.1348 5 6 message 300 ------ 0 0.2 255.210 -1 13
+ 2.1348 3 4 message 300 ------ 0 0.2 255.210 -1 13
- 2.1348 3 4 message 300 ------ 0 0.2 255.210 -1 13
+ 2.2 0 1 message 300 ------ 0 0.2 255.210 -1 14
- 2.2 0 1 message 300 ------ 0 0.2 255.210 -1 14
+ 2.2116 1 2 message 300 ------ 0 0.2 255.210 -1 14
- 2.2116 1 2 message 300 ------ 0 0.2 255.210 -1 14
+ 2.2232 2 5 message 300 ------ 0 0.2 255.210 -1 14
- 2.2232 2 5 message 300 ------ 0 0.2 255.210 -1 14
+ 2.2232 2 3 message 300 ------ 0 0.2 255.210 -1 14
- 2.2232 2 3 message 300 ------ 0 0.2 255.210 -1 14
+ 2.2348 5 6 message 300 ------ 0 0.2 255.210 -1 14
- 2.2348 5 6 message 300 ------ 0 0.2 255.210 -1 14
+ 2.2348 3 4 message 300 ------ 0 0.2 255.210 -1 14
- 2.2348 3 4 message 300 ------ 0 0.2 255.210 -1 14
+ 2.3 0 1 message 300 ------ 0 0.2 255.210 -1 15
- 2.3 0 1 message 300 ------ 0 0.2 255.210 -1 15
+ 2.3116 1 2 message 300 ------ 0 0.2 255.210 -1 15
- 2.3116 1 2 message 300 ------ 0 0.2 255.210 -1 15
+ 2.3232 2 5 message 300 ------ 0 0.2 255.210 -1 15
- 2.3232 2 5 message 300 ------ 0 0.2 255.210 -1 15
+ 2.3232 2 3 message 300 ------ 0 0.2 255.210 -1 15
- 2.3232 2 3 message 300 ------ 0 0.2 255.210 -1 15
+ 2.3348 5 6 message 300 ------ 0 0.2 255.210 -1 15
- 2.3348 5 6 message 300 ------ 0 0.2 255.210 -1 15
+ 2.3348 3 4 message 300 ------ 0 0.2 255.210 -1 15
- 2.3348 3 4 message 300 ------ 0 0.2 255.210 -1 15
+ 2.4 0 1 message 300 ------ 0 0.2 255.210 -1 16
- 2.4 0 1 message 300 ------ 0 0.2 255.210 -1 16
+ 2.4116 1 2 message 300 ------ 0 0.2 255.210 -1 16
- 2.4116 1 2 message 300 ------ 0 0.2 255.210 -1 16
+ 2.4232 2 5 message 300 ------ 0 0.2 255.210 -1 16
- 2.4232 2 5 message 300 ------ 0 0.2 255.210 -1 16
+ 2.4232 2 3 message 300 ------ 0 0.2 255.210 -1 16
- 2.4232 2 3 message 300 ------ 0 0.2 255.210 -1 16
+ 2.4348 5 6 message 300 ------ 0 0.2 255.210 -1 16
- 2.4348 5 6 message 300 ------ 0 0.2 255.210 -1 16
+ 2.4348 3 4 message 300 ------ 0 0.2 255.210 -1 16
- 2.4348 3 4 message 300 ------ 0 0.2 255.210 -1 16
+ 2.5 0 1 message 300 ------ 0 0.2 255.210 -1 17
- 2.5 0 1 message 300 ------ 0 0.2 255.210 -1 17
+ 2.5116 1 2 message 300 ------ 0 0.2 255.210 -1 17
- 2.5116 1 2 message 300 ------ 0 0.2 255.210 -1 17
+ 2.5232 2 5 message 300 ------ 0 0.2 255.210 -1 17
- 2.5232 2 5 message 300 ------ 0 0.2 255.210 -1 17
+ 2.5232 2 3 message 300 ------ 0 0.2 255.210 -1 17
- 2.5232 2 3 message 300 ------ 0 0.2 255.210 -1 17
+ 2.5348 5 6 message 300 ------ 0 0.2 255.210 -1 17
- 2.5348 5 6 message 300 ------ 0 0.2 255.210 -1 17
+ 2.5348 3 4 message 300 ------ 0 0.2 255.210 -1 17
- 2.5348 3 4 message 300 ------ 0 0.2 255.210 -1 17
+ 2.6 0 1 message 300 ------ 0 0.2 255.210 -1 18
- 2.6 0 1 message 300 ------ 0 0.2 255.210 -1 18
+ 2.6116 1 2 message 300 ------ 0 0.2 255.210 -1 18
- 2.6116 1 2 message 300 ------ 0 0.2 255.210 -1 18
+ 2.6232 2 5 message 300 ------ 0 0.2 255.210 -1 18
- 2.6232 2 5 message 300 ------ 0 0.2 255.210 -1 18
+ 2.6232 2 3 message 300 ------ 0 0.2 255.210 -1 18
- 2.6232 2 3 message 300 ------ 0 0.2 255.210 -1 18
```

B.1.3 Salida generada por las facilidades de debugging

```
Generando nodos . . .
Generando links .  . .
Generando agentes CBT . . .
Node:3 Agregando entrada MFC, group:65490 core:3 if:-1 nxp:-1 at 0
Node:0 Agregando entrada Transitoria, group:65490 core:3 leaf:Y if:0
nxp:* at 0
Node 0 Delete_T_Entry_Timer(65490) (Creando en tiempo: 0)
Node:0 ==>JOIN_REQ: 65490 3 0 M 0  Tiempo:0
Node 0 Resend_Join_Timer(0,*) (Creando en tiempo: 0)
Node:0 Agregando child (transitorio), group:65490 core:3 if_ch:-2
nxp_ch:-1 at 0
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:4 Agregando entrada Transitoria, group:65490 core:3 leaf:Y if:13
nxp:* at 0
Node 4 Delete_T_Entry_Timer(65490) (Creando en tiempo: 0)
Node:4 ==>JOIN_REQ: 65490 3 4 M 4  Tiempo:0
Node 4 Resend_Join_Timer(13,*) (Creando en tiempo: 0)
Node:4 Agregando child (transitorio), group:65490 core:3 if_ch:-2
nxp_ch:-1 at 0
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:3 <== JOIN_REQ: 65490 3 4 M 4  Tiempo:0.010960000000000001
Node:3 MFC:Agregando child, group:65490 core:3 if_ch:12 nxp_ch:* at
0.010960000000000001
Node:3 ==>JOIN_ACK: 65490 3 M 3  Tiempo:0.010960000000000001
Node:1 <== JOIN_REQ: 65490 3 0 M 0  Tiempo:0.010960000000000001
Node:1 Agregando entrada Transitoria, group:65490 core:3 leaf:N if:2
nxp:* at 0.010960000000000001
Node 1 Delete_T_Entry_Timer(65490) (Creando en tiempo:
0.010960000000000001)
Node:1 ==>JOIN_REQ: 65490 3 1 M 1  Tiempo:0.010960000000000001
Node:1 Agregando child (transitorio), group:65490 core:3 if_ch:1
nxp_ch:* at 0.010960000000000001
Node:4 <== JOIN_ACK: 65490 3 M 3  Tiempo:0.021920000000000002
Node:4 Agregando entrada MFC, group:65490 core:3 if:13 nxp:* at
0.021920000000000002
Node 4 Delete_Entry_Timer(65490) (Creando en tiempo:
0.021920000000000002)
Node 4 Echo_Request_Timer(13,*) (Creando en tiempo:
0.021920000000000002)
Node:4 MFC:Agregando child, group:65490 core:3 if_ch:-2 nxp_ch:-1 at
0.021920000000000002
Node:4 Eliminando child (transitorio), group:65490 core:3 if_ch:-2
nxp_ch:-1 at 0.021920000000000002
Node:4 Eliminando entrada Transitoria, group:65490 core:3 if:13 nxp:*
at 0.021920000000000002
Node 4 Resend_Join_Timer(13,*) (Cancelando en tiempo:
0.021920000000000002)
Node:2 <== JOIN_REQ: 65490 3 1 M 1  Tiempo:0.022880000000000005
Node:2 Agregando entrada Transitoria, group:65490 core:3 leaf:N if:8
nxp:* at 0.022880000000000005
Node 2 Delete_T_Entry_Timer(65490) (Creando en tiempo:
0.022880000000000005)
Node:2 ==>JOIN_REQ: 65490 3 2 M 2  Tiempo:0.022880000000000005
```

```
Node:2 Agregando child (transitorio), group:65490 core:3 if_ch:3
nxp_ch:* at 0.022880000000000005
Node:3 <== JOIN_REQ: 65490 3 2 M 2  Tiempo:0.034800000000000005
Node:3 MFC:Agregando child, group:65490 core:3 if_ch:9 nxp_ch:* at
0.034800000000000005
Node:3 ==>JOIN_ACK: 65490 3 M 3  Tiempo:0.034800000000000005
Node:2 <== JOIN_ACK: 65490 3 M 3  Tiempo:0.045760000000000009
Node:2 Agregando entrada MFC, group:65490 core:3 if:8 nxp:* at
0.045760000000000009
Node 2 Delete_Entry_Timer(65490) (Creando en tiempo:
0.045760000000000009)
Node 2 Echo_Request_Timer(8,*) (Creando en tiempo:
0.045760000000000009)
Node:2 MFC:Agregando child, group:65490 core:3 if_ch:3 nxp_ch:* at
0.045760000000000009
Node:2 ==>JOIN_ACK: 65490 3 M 2  Tiempo:0.045760000000000009
Node:2 Eliminando child (transitorio), group:65490 core:3 if_ch:3
nxp_ch:* at 0.045760000000000009
Node:2 Eliminando entrada Transitoria, group:65490 core:3 if:8 nxp:*
at 0.045760000000000009
Node:1 <== JOIN_ACK: 65490 3 M 2  Tiempo:0.057680000000000009
Node:1 Agregando entrada MFC, group:65490 core:3 if:2 nxp:* at
0.057680000000000009
Node 1 Delete_Entry_Timer(65490) (Creando en tiempo:
0.057680000000000009)
Node 1 Echo_Request_Timer(2,*) (Creando en tiempo:
0.057680000000000009)
Node:1 MFC:Agregando child, group:65490 core:3 if_ch:1 nxp_ch:* at
0.057680000000000009
Node:1 ==>JOIN_ACK: 65490 3 M 1  Tiempo:0.057680000000000009
Node:1 Eliminando child (transitorio), group:65490 core:3 if_ch:1
nxp_ch:* at 0.057680000000000009
Node:1 Eliminando entrada Transitoria, group:65490 core:3 if:2 nxp:*
at 0.057680000000000009
Node:0 <== JOIN_ACK: 65490 3 M 1  Tiempo:0.069600000000000009
Node:0 Agregando entrada MFC, group:65490 core:3 if:0 nxp:* at
0.069600000000000009
Node 0 Delete_Entry_Timer(65490) (Creando en tiempo:
0.069600000000000009)
Node 0 Echo_Request_Timer(0,*) (Creando en tiempo:
0.069600000000000009)
Node:0 MFC:Agregando child, group:65490 core:3 if_ch:-2 nxp_ch:-1 at
0.069600000000000009
Node:0 Eliminando child (transitorio), group:65490 core:3 if_ch:-2
nxp_ch:-1 at 0.069600000000000009
Node:0 Eliminando entrada Transitoria, group:65490 core:3 if:0 nxp:*
at 0.069600000000000009
Node 0 Resend_Join_Timer(0,*) (Cancelando en tiempo:
0.069600000000000009)
Node:6 Agregando entrada Transitoria, group:65490 core:3 leaf:Y if:17
nxp:* at 2
Node 6 Delete_T_Entry_Timer(65490) (Creando en tiempo: 2)
Node:6 ==>JOIN_REQ: 65490 3 6 M 6  Tiempo:2
Node 6 Resend_Join_Timer(17,*) (Creando en tiempo: 2)
Node:6 Agregando child (transitorio), group:65490 core:3 if_ch:-2
nxp_ch:-1 at 2
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:1 MFC:Agregando child, group:65490 core:3 if_ch:-2 nxp_ch:-1 at 2
Grupo 65490 joined.
Grupo 65490, ya joined.
```

```
Node:5 <== JOIN_REQ: 65490 3 6 M 6  Tiempo:2.0109599999999999
Node:5 Agregando entrada Transitoria, group:65490 core:3 leaf:N if:11
nxp:* at 2.0109599999999999
Node 5 Delete_T_Entry_Timer(65490) (Creando en tiempo:
2.0109599999999999)
Node:5 ==>JOIN_REQ: 65490 3 5 M 5  Tiempo:2.0109599999999999
Node:5 Agregando child (transitorio), group:65490 core:3 if_ch:16
nxp_ch:* at 2.0109599999999999
Node:2 <== JOIN_REQ: 65490 3 5 M 5  Tiempo:2.0228799999999998
Node:2 MFC:Agregando child, group:65490 core:3 if_ch:10 nxp_ch:* at
2.0228799999999998
Node:2 ==>JOIN_ACK: 65490 3 M 2  Tiempo:2.0228799999999998
Node:5 <== JOIN_ACK: 65490 3 M 2  Tiempo:2.0338399999999996
Node:5 Agregando entrada MFC, group:65490 core:3 if:11 nxp:* at
2.0338399999999996
Node 5 Delete_Entry_Timer(65490) (Creando en tiempo:
2.0338399999999996)
Node 5 Echo_Request_Timer(11,*) (Creando en tiempo:
2.0338399999999996)
Node:5 MFC:Agregando child, group:65490 core:3 if_ch:16 nxp_ch:* at
2.0338399999999996
Node:5 ==>JOIN_ACK: 65490 3 M 5  Tiempo:2.0338399999999996
Node:5 Eliminando child (transitorio), group:65490 core:3 if_ch:16
nxp_ch:* at 2.0338399999999996
Node:5 Eliminando entrada Transitoria, group:65490 core:3 if:11 nxp:*
at 2.0338399999999996
Node:6 <== JOIN_ACK: 65490 3 M 5  Tiempo:2.0457599999999996
Node:6 Agregando entrada MFC, group:65490 core:3 if:17 nxp:* at
2.0457599999999996
Node 6 Delete_Entry_Timer(65490) (Creando en tiempo:
2.0457599999999996)
Node 6 Echo_Request_Timer(17,*) (Creando en tiempo:
2.0457599999999996)
Node:6 MFC:Agregando child, group:65490 core:3 if_ch:-2 nxp_ch:-1 at
2.0457599999999996
Node:6 Eliminando child (transitorio), group:65490 core:3 if_ch:-2
nxp_ch:-1 at 2.0457599999999996
Node:6 Eliminando entrada Transitoria, group:65490 core:3 if:17 nxp:*
at 2.0457599999999996
Node 6 Resend_Join_Timer(17,*) (Cancelando en tiempo:
2.0457599999999996)
Node:2 MFC:Agregando child, group:65490 core:3 if_ch:-2 nxp_ch:-1 at 5
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:6 MFC:Eliminando child, group:65490 core:3 if_ch:-2 nxp_ch:-1 at
12
Node:6 Borrando entrada MFC, group:65490 core:3 if:17 nxp:* at 12
Node:6 ==>QUIT_NOT: 65490 6 M 6  Tiempo:12
Node 6 Quit_Notification_Timer(65490:17,*) (Creando en tiempo: 12)
Node 6 Echo_Request_Timer(17,*) (Cancelando en tiempo: 12)
Leaving group 65490
Node:5 <== QUIT_NOT: 65490 6 M 6  Tiempo:12.010960000000001
Node 5 Cache_Del_Timer(65490,16,*) (Creando en tiempo:
12.010960000000001)
Node 6  drop.  replicator: _o391 source: 4 dest: 65490
Node 6  drop.  replicator: _o384 source: 0 dest: 65490
Node 6 Quit_Notification_Timer(65490:17,*) (Timeout en tiempo: 15)
Node:6 ==>QUIT_NOT: 65490 6 M 6  Tiempo:15
Node:5 <== QUIT_NOT: 65490 6 M 6  Tiempo:15.010960000000001
Node:5 MFC:Eliminando child, group:65490 core:3 if_ch:16 nxp_ch:* at
16.510999999999999
```

```
Node:5 Borrando entrada MFC, group:65490 core:3 if:11 nxp:* at
16.510999999999999
Node:5 ==>QUIT_NOT: 65490 5 M 5  Tiempo:16.510999999999999
Node 5 Quit_Notification_Timer(65490:11,*) (Creando en tiempo:
16.510999999999999)
Node 5 Echo_Request_Timer(11,*) (Cancelando en tiempo:
16.510999999999999)
Node:2 <== QUIT_NOT: 65490 5 M 5  Tiempo:16.52196
Node 2 Cache_Del_Timer(65490,10,*) (Creando en tiempo: 16.52196)
Node 5  drop.  replicator: _o389 source: 4 dest: 65490
Node 5  drop.  replicator: _o378 source: 0 dest: 65490
Node 6 Quit_Notification_Timer(65490:17,*) (Timeout en tiempo: 18)
Node:6 ==>QUIT_NOT: 65490 6 M 6  Tiempo:18
Node:5 <== QUIT_NOT: 65490 6 M 6  Tiempo:18.010960000000001
Nodo:5  Descarta QUIT-NOT (grupo no on tree) tiempo 18.010960000000001
Node 5 Quit_Notification_Timer(65490:11,*) (Timeout en tiempo:
19.510999999999999)
Node:5 ==>QUIT_NOT: 65490 5 M 5  Tiempo:19.510999999999999
Node:2 <== QUIT_NOT: 65490 5 M 5  Tiempo:19.52196
Node:6 ==>QUIT_NOT: 65490 6 M 6  Tiempo:21
Node:5 <== QUIT_NOT: 65490 6 M 6  Tiempo:21.010960000000001
Nodo:5  Descarta QUIT-NOT (grupo no on tree) tiempo 21.010960000000001
Node:2 MFC:Eliminando child, group:65490 core:3 if_ch:10 nxp_ch:* at
21.021999999999998
Node 5 Quit_Notification_Timer(65490:11,*) (Timeout en tiempo:
22.510999999999999)
Node:5 ==>QUIT_NOT: 65490 5 M 5  Tiempo:22.510999999999999
Node:2 <== QUIT_NOT: 65490 5 M 5  Tiempo:22.52196
Node:5 ==>QUIT_NOT: 65490 5 M 5  Tiempo:25.510999999999999
Node:2 <== QUIT_NOT: 65490 5 M 5  Tiempo:25.52196
Node 4 Echo_Request_Timer(13,*) (Timeout en tiempo:
60.021900000000002)
Node:4 ==>ECHO_REQ: 4 M 4  Tiempo:60.021900000000002
Node:3 <== ECHO_REQ: 4 M 4  Tiempo:60.032859999999999
Node:3 ==>ECHO_RPL: 3 65490 M 3  Tiempo:60.032859999999999
Node:4 <== ECHO_RPL: 3 65490 M 3  Tiempo:60.043819999999997
Node 4 Delete_Entry_Timer(65490) (Rescheduling en tiempo:
60.043819999999997)
Node 2 Echo_Request_Timer(8,*) (Timeout en tiempo: 60.0458)
Node:2 ==>ECHO_REQ: 2 M 2  Tiempo:60.0458
Node:3 <== ECHO_REQ: 2 M 2  Tiempo:60.056759999999997
Node:3 ==>ECHO_RPL: 3 65490 M 3  Tiempo:60.056759999999997
Node 1 Echo_Request_Timer(2,*) (Timeout en tiempo: 60.057699999999997)
Node:1 ==>ECHO_REQ: 1 M 1  Tiempo:60.057699999999997
Node:2 <== ECHO_RPL: 3 65490 M 3  Tiempo:60.067719999999994
Node 2 Delete_Entry_Timer(65490) (Rescheduling en tiempo:
60.067719999999994)
Node:2 <== ECHO_REQ: 1 M 1  Tiempo:60.068659999999994
Node:2 ==>ECHO_RPL: 2 65490 M 2  Tiempo:60.068659999999994
Node 0 Echo_Request_Timer(0,*) (Timeout en tiempo: 60.069600000000001)
Node:0 ==>ECHO_REQ: 0 M 0  Tiempo:60.069600000000001
Node:1 <== ECHO_RPL: 2 65490 M 2  Tiempo:60.079639999999991
Node 1 Delete_Entry_Timer(65490) (Rescheduling en tiempo:
60.079639999999991)
Node:1 <== ECHO_REQ: 0 M 0  Tiempo:60.080559999999998
Node:1 ==>ECHO_RPL: 1 65490 M 1  Tiempo:60.080559999999998
Node:0 <== ECHO_RPL: 1 65490 M 1  Tiempo:60.091559999999987
Node 0 Delete_Entry_Timer(65490) (Rescheduling en tiempo:
60.091559999999987)
Node 4 Echo_Request_Timer(13,*) (Timeout en tiempo:
120.02200000000001)
```

```
Node:4 ==>ECHO_REQ: 4 M 4  Tiempo:120.02200000000001
Node:3 <== ECHO_REQ: 4 M 4  Tiempo:120.03296
Node:3 ==>ECHO_RPL: 3 65490 M 3  Tiempo:120.03296
Node:4 <== ECHO_RPL: 3 65490 M 3  Tiempo:120.04392
Node 4 Delete_Entry_Timer(65490) (Rescheduling en tiempo: 120.04392)
Node 2 Echo_Request_Timer(8,*) (Timeout en tiempo: 120.04600000000001)
Node:2 ==>ECHO_REQ: 2 M 2  Tiempo:120.04600000000001
Node:3 <== ECHO_REQ: 2 M 2  Tiempo:120.05696
Node:3 ==>ECHO_RPL: 3 65490 M 3  Tiempo:120.05696
Node 1 Echo_Request_Timer(2,*) (Timeout en tiempo: 120.05800000000001)
Node:1 ==>ECHO_REQ: 1 M 1  Tiempo:120.05800000000001
Node:2 <== ECHO_RPL: 3 65490 M 3  Tiempo:120.06792
Node 2 Delete_Entry_Timer(65490) (Rescheduling en tiempo: 120.06792)
Node:2 <== ECHO_REQ: 1 M 1  Tiempo:120.06896
Node:2 ==>ECHO_RPL: 2 65490 M 2  Tiempo:120.06896
Node 0 Echo_Request_Timer(0,*) (Timeout en tiempo: 120.06999999999999)
Node:0 ==>ECHO_REQ: 0 M 0  Tiempo:120.06999999999999
Node:1 <== ECHO_RPL: 2 65490 M 2  Tiempo:120.07992
Node 1 Delete_Entry_Timer(65490) (Rescheduling en tiempo: 120.07992)
Node:1 <== ECHO_REQ: 0 M 0  Tiempo:120.08095999999999
Node:1 ==>ECHO_RPL: 1 65490 M 1  Tiempo:120.08095999999999
Node:0 <== ECHO_RPL: 1 65490 M 1  Tiempo:120.09191999999999
Node 0 Delete_Entry_Timer(65490) (Rescheduling en tiempo:
120.09191999999999)
```

B.1.4 Salida generada por los agentes para registro de demoras (Se muestra sólo parcialmente por razones de extensión)

```
Node:0 Envia: 0/2/1 at 2
Node:1 Recibe:0/2/1 at 2.0116000000000001 Delay: 0.0116
Node:6 Recibe:0/2/1 at 2.0473599999999998 Delay: 0.04736
Node:0 Envia: 0/2.1000000000000001/2 at 2.1000000000000001
Node:1 Recibe:0/2.1000000000000001/2 at 2.1116000000000001 Delay:
0.0116
Node:6 Recibe:0/2.1000000000000001/2 at 2.1464000000000003 Delay:
0.0464
Node:0 Envia: 0/2.2000000000000002/3 at 2.2000000000000002
Node:1 Recibe:0/2.2000000000000002/3 at 2.2116000000000002 Delay:
0.0116
Node:6 Recibe:0/2.2000000000000002/3 at 2.2464000000000004 Delay:
0.0464
Node:0 Envia: 0/2.2999999999999998/4 at 2.2999999999999998
Node:1 Recibe:0/2.2999999999999998/4 at 2.3115999999999999 Delay:
0.0116
Node:6 Recibe:0/2.2999999999999998/4 at 2.3464 Delay: 0.0464
Node:0 Envia: 0/2.3999999999999999/5 at 2.3999999999999999
Node:1 Recibe:0/2.3999999999999999/5 at 2.4116 Delay: 0.0116
Node:6 Recibe:0/2.3999999999999999/5 at 2.4464000000000001 Delay:
0.0464
Node:0 Envia: 0/2.5/6 at 2.5
Node:1 Recibe:0/2.5/6 at 2.5116000000000001 Delay: 0.0116
Node:6 Recibe:0/2.5/6 at 2.5464000000000002 Delay: 0.0464
Node:0 Envia: 0/2.6000000000000001/7 at 2.6000000000000001
Node:1 Recibe:0/2.6000000000000001/7 at 2.6116000000000001 Delay:
0.0116
Node:6 Recibe:0/2.6000000000000001/7 at 2.6464000000000003 Delay:
0.0464
Node:0 Envia: 0/2.7000000000000002/8 at 2.7000000000000002
```

```
Node:1 Recibe:0/2.7000000000000002/8 at 2.7116000000000002 Delay:
0.0116
Node:6 Recibe:0/2.7000000000000002/8 at 2.7464000000000004 Delay:
0.0464
Node:0 Envia: 0/2.7999999999999998/9 at 2.7999999999999998
Node:1 Recibe:0/2.7999999999999998/9 at 2.8115999999999999 Delay:
0.0116
Node:6 Recibe:0/2.7999999999999998/9 at 2.8464 Delay: 0.0464
Node:0 Envia: 0/2.8999999999999999/10 at 2.8999999999999999
Node:1 Recibe:0/2.8999999999999999/10 at 2.9116 Delay: 0.0116
Node:6 Recibe:0/2.8999999999999999/10 at 2.9464000000000001 Delay:
0.0464
Node:0 Envia: 0/3/11 at 3
Node:1 Recibe:0/3/11 at 3.0116000000000001 Delay: 0.0116
Node:6 Recibe:0/3/11 at 3.0464000000000002 Delay: 0.0464
Node:0 Envia: 0/3.1000000000000001/12 at 3.1000000000000001
Node:1 Recibe:0/3.1000000000000001/12 at 3.1116000000000001 Delay:
0.0116
Node:6 Recibe:0/3.1000000000000001/12 at 3.1464000000000003 Delay:
0.0464
Node:0 Envia: 0/3.2000000000000002/13 at 3.2000000000000002
Node:1 Recibe:0/3.2000000000000002/13 at 3.2116000000000002 Delay:
0.0116
Node:6 Recibe:0/3.2000000000000002/13 at 3.2464000000000004 Delay:
0.0464
Node:0 Envia: 0/3.2999999999999998/14 at 3.2999999999999998
Node:1 Recibe:0/3.2999999999999998/14 at 3.3115999999999999 Delay:
0.0116
Node:6 Recibe:0/3.2999999999999998/14 at 3.3464 Delay: 0.0464
Node:0 Envia: 0/3.3999999999999999/15 at 3.3999999999999999
Node:1 Recibe:0/3.3999999999999999/15 at 3.4116 Delay: 0.0116
Node:6 Recibe:0/3.3999999999999999/15 at 3.4464000000000001 Delay:
0.0464
Node:0 Envia: 0/3.5/16 at 3.5
Node:1 Recibe:0/3.5/16 at 3.5116000000000001 Delay: 0.0116
Node:6 Recibe:0/3.5/16 at 3.5464000000000002 Delay: 0.0464
Node:0 Envia: 0/3.6000000000000001/17 at 3.6000000000000001
Node:1 Recibe:0/3.6000000000000001/17 at 3.6116000000000001 Delay:
0.0116
Node:6 Recibe:0/3.6000000000000001/17 at 3.6464000000000003 Delay:
0.0464
Node:0 Envia: 0/3.7000000000000002/18 at 3.7000000000000002
Node:1 Recibe:0/3.7000000000000002/18 at 3.7116000000000002 Delay:
0.0116
Node:6 Recibe:0/3.7000000000000002/18 at 3.7464000000000004 Delay:
0.0464
Node:0 Envia: 0/3.7999999999999998/19 at 3.7999999999999998
Node:1 Recibe:0/3.7999999999999998/19 at 3.8115999999999999 Delay:
0.0116
Node:6 Recibe:0/3.7999999999999998/19 at 3.8464 Delay: 0.0464
Node:0 Envia: 0/3.8999999999999999/20 at 3.8999999999999999
Node:1 Recibe:0/3.8999999999999999/20 at 3.9116 Delay: 0.0116
Node:6 Recibe:0/3.8999999999999999/20 at 3.9464000000000001 Delay:
0.0464
Node:0 Envia: 0/4/21 at 4
Node:1 Recibe:0/4/21 at 4.0115999999999996 Delay: 0.0116
Node:6 Recibe:0/4/21 at 4.0463999999999984 Delay: 0.0464
Node:0 Envia: 0/4.0999999999999996/22 at 4.0999999999999996
Node:1 Recibe:0/4.0999999999999996/22 at 4.1115999999999993 Delay:
0.0116
```

```
Node:6 Recibe:0/4.0999999999999996/22 at 4.1463999999999981 Delay:
0.0464
Node:0 Envia: 0/4.2000000000000002/23 at 4.2000000000000002
Node:1 Recibe:0/4.2000000000000002/23 at 4.2115999999999998 Delay:
0.0116
Node:6 Recibe:0/4.2000000000000002/23 at 4.2463999999999986 Delay:
0.0464
Node:0 Envia: 0/4.2999999999999998/24 at 4.2999999999999998
Node:1 Recibe:0/4.2999999999999998/24 at 4.3115999999999994 Delay:
0.0116
Node:6 Recibe:0/4.2999999999999998/24 at 4.3463999999999983 Delay:
0.0464
Node:0 Envia: 0/4.4000000000000004/25 at 4.4000000000000004
Node:1 Recibe:0/4.4000000000000004/25 at 4.4116 Delay: 0.0116
Node:6 Recibe:0/4.4000000000000004/25 at 4.4463999999999988 Delay:
0.0464
Node:0 Envia: 0/4.5/26 at 4.5
Node:1 Recibe:0/4.5/26 at 4.5115999999999996 Delay: 0.0116
Node:6 Recibe:0/4.5/26 at 4.5463999999999984 Delay: 0.0464
Node:0 Envia: 0/4.5999999999999996/27 at 4.5999999999999996
Node:1 Recibe:0/4.5999999999999996/27 at 4.6115999999999993 Delay:
0.0116
Node:6 Recibe:0/4.5999999999999996/27 at 4.6463999999999981 Delay:
0.0464
Node:0 Envia: 0/4.7000000000000002/28 at 4.7000000000000002
Node:1 Recibe:0/4.7000000000000002/28 at 4.7115999999999998 Delay:
0.0116
Node:6 Recibe:0/4.7000000000000002/28 at 4.7463999999999986 Delay:
0.0464
Node:0 Envia: 0/4.7999999999999998/29 at 4.7999999999999998
Node:1 Recibe:0/4.7999999999999998/29 at 4.8115999999999994 Delay:
0.0116
Node:6 Recibe:0/4.7999999999999998/29 at 4.8463999999999983 Delay:
0.0464
Node:0 Envia: 0/4.9000000000000004/30 at 4.9000000000000004
Node:1 Recibe:0/4.9000000000000004/30 at 4.9116 Delay: 0.0116
Node:6 Recibe:0/4.9000000000000004/30 at 4.9463999999999988 Delay:
0.0464
Node:0 Envia: 0/5/31 at 5
Node:1 Recibe:0/5/31 at 5.0115999999999996 Delay: 0.0116
Node:2 Recibe:0/5/31 at 5.0231999999999992 Delay: 0.0232
Node:6 Recibe:0/5/31 at 5.0463999999999984 Delay: 0.0464
Node:0 Envia: 0/5.0999999999999996/32 at 5.0999999999999996
Node:1 Recibe:0/5.0999999999999996/32 at 5.1115999999999993 Delay:
0.0116
Node:2 Recibe:0/5.0999999999999996/32 at 5.1231999999999989 Delay:
0.0232
Node:6 Recibe:0/5.0999999999999996/32 at 5.1463999999999981 Delay:
0.0464
Node:0 Envia: 0/5.2000000000000002/33 at 5.2000000000000002
Node:1 Recibe:0/5.2000000000000002/33 at 5.2115999999999998 Delay:
0.0116
Node:2 Recibe:0/5.2000000000000002/33 at 5.2231999999999994 Delay:
0.0232
Node:6 Recibe:0/5.2000000000000002/33 at 5.2463999999999986 Delay:
0.0464
Node:0 Envia: 0/5.2999999999999998/34 at 5.2999999999999998
Node:1 Recibe:0/5.2999999999999998/34 at 5.3115999999999994 Delay:
0.0116
Node:2 Recibe:0/5.2999999999999998/34 at 5.323199999999999 Delay:
0.0232
```

B.2 Ejemplo de las facilidades de debugging invocadas explícitamente

El objetivo de este ejemplo es mostrar la salida producida por las facilidades de debugging consistentes en la producción de mensajes relativos al contenido de las tablas en los nodos, las características de las interfaces y el mapa del árbol de distribución para un grupo.

B.2.1 Código

```
# Ejemplo de una topologia compuesta por 8 nodos. Los vinculos son
punto a
# punto.
# Por salida standard se muestra el estado de las tablas en los nodos
y el
# arbol de distribucion del grupo luego de los cambios producidos por
los
# aplicaciones multicast en los nodos
# No se generan archivos con intercambio de PDUs ni de mensajes entre
las
# aplicaciones, ni el tracing provisto por Ns

set ns [new Simulator]
set dels [open mensajes w]
Simulator set EnableMcast_ 1
Simulator set NumberInterfaces_ 1

# Carga del codigo implementado
source cbtmain.tcl

#Generacion de nodos
puts "Generando nodos . . ."
for {set i 0} {$i <= 7} {incr i 1} {
   set n($i) [$ns node]
}

#Topologia
puts "Generando links .  . . "
$ns duplex-link $n(0) $n(1) 1.5Mb 10ms DropTail
$ns duplex-link $n(1) $n(2) 1.5Mb 10ms DropTail
$ns duplex-link $n(1) $n(5) 1.5Mb 10ms DropTail
$ns duplex-link $n(1) $n(7) 1.5Mb 10ms DropTail
$ns duplex-link $n(2) $n(3) 1.5Mb 10ms DropTail
$ns duplex-link $n(2) $n(5) 1.5Mb 10ms DropTail
$ns duplex-link $n(3) $n(4) 1.5Mb 10ms DropTail
$ns duplex-link $n(4) $n(5) 1.5Mb 10ms DropTail
$ns duplex-link $n(5) $n(6) 1.5Mb 10ms DropTail

#Agentes CBT
puts "Generando agentes CBT . . . "
for {set i 0} {$i <= 7} {incr i 1} {
   set cbt($i) [new CBT $ns $n($i)]
}

#Arranque de los agentes CBT
for {set i 0} {$i <= 7} {incr i 1} {
   $cbt($i) start
}


#Configuracion de nodo n(3) como core para el grupo 65490
```

```
$cbt(3) set-core 65490

#Creacion de los agentes multicast de prueba
#Emisores al grupo, en nodos n(0) y n(4), emiten 10 paquetes/segundo
#de 200 y 300 bytes c/u respectivamente
#Los agentes emisores deben realizar el join group debido a que no
esta
#implementado el envio desde un nodo no miembro del grupo
   set  e(0) [new Agent/Message/Emisor_delay 0.1 $dels]
   $ns attach-agent $n(0) $e(0)
   $ns at 0.0 "$e(0) join-group 65490"
   $e(0) set dst_ 65490
   $e(0) set packetSize_ 300
   set  e(4) [new Agent/Message/Emisor_delay 0.1 $dels]
   $ns attach-agent $n(4) $e(4)
   $ns at 0.0 "$e(4) join-group 65490"
   $e(4) set dst_ 65490
   $e(4) set packetSize_ 200

#Agentes receptores en nodos 6, 2 y 1. Registran los mensajes en el
archivo
#mensajes
   set re(6) [new Agent/Message/Receptor_delay $dels]
   $ns attach-agent $n(6) $re(6)
   set re(2) [new Agent/Message/Receptor_delay $dels]
   $ns attach-agent $n(2) $re(2)
   set re(1) [new Agent/Message/Receptor_delay $dels]
   $ns attach-agent $n(1) $re(1)

#Arranques de los agentes emisores
   $ns at 2.0 "$e(0) start"
   $ns at 10.0 "$e(4) start"
   $ns at 32.0 "$e(0) stop"
   $ns at 32.0 "$e(4) stop"

#Join y leaves de los receptores
   $ns at 2.0 "$re(6) join-group 65490"
   $ns at 2.0 "$re(1) join-group 65490"
   $ns at 5.0 "$re(2) join-group 65490"
   $ns at 12.0 "$re(6) leave-group 65490"
   $ns at 140.0 "final"

#Listado de interfaces en los nodos
   $ns at 1.01 "$cbt(0) list_interfaces"
   $ns at 1.02 "$cbt(1) list_interfaces"
   $ns at 1.03 "$cbt(2) list_interfaces"
   $ns at 1.04 "$cbt(3) list_interfaces"
   $ns at 1.05 "$cbt(4) list_interfaces"
   $ns at 1.06 "$cbt(5) list_interfaces"
   $ns at 1.07 "$cbt(6) list_interfaces"
   $ns at 1.08 "$cbt(7) list_interfaces"

# Sentencias de listado cada 1 segundo, desde 1 hasta 13
   for {set i 2} {$i < 14} {incr i 1} {
          for {set k 0 } {$k < 7} {incr k 1} {
         $ns at $i.$k "$cbt($k) list_MFCTable"
         $ns at $i.$k "$cbt($k) list_TransientTable"
      }
      $ns at $i.8 "$ns map-tree 65490"
   }
```

```
proc final {} {
   global ns f dels
   $ns flush-trace
   close $dels
   exit 0
}

$ns run
```

B.2.2 Salida generada (Se muestra parcialmente por razones de extensión)

```
Generando nodos . . .
Generando links .  . .
Generando agentes CBT . . .
Grupo 65490 joined.
Grupo 65490, ya joined.
Grupo 65490 joined.
Grupo 65490, ya joined.
Agente CBT en nodo 0.Tiempo: 1.01. Interfaces:
IF(_o197): lbl(0) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(0)
IF(_o196): lbl(-2) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(0)
Agente CBT en nodo 1.Tiempo: 1.02. Interfaces:
IF(_o208): lbl(4) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(1)
IF(_o206): lbl(1) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(1)
IF(_o209): lbl(6) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(1)
IF(_o207): lbl(2) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(1)
IF(_o205): lbl(-2) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(1)
Agente CBT en nodo 2.Tiempo: 1.03. Interfaces:
IF(_o219): lbl(8) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(2)
IF(_o220): lbl(10) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(2)
IF(_o218): lbl(3) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(2)
IF(_o217): lbl(-2) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(2)
Agente CBT en nodo 3.Tiempo: 1.04. Interfaces:
IF(_o229): lbl(9) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(3)
IF(_o230): lbl(12) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(3)
IF(_o228): lbl(-2) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(3)
Agente CBT en nodo 4.Tiempo: 1.05. Interfaces:
IF(_o239): lbl(13) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(4)
IF(_o240): lbl(14) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(4)
IF(_o238): lbl(-2) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(4)
Agente CBT en nodo 5.Tiempo: 1.0600000000000001. Interfaces:
```

```
IF(_o249): lbl(5) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(5)
IF(_o251): lbl(15) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(5)
IF(_o250): lbl(11) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(5)
IF(_o252): lbl(16) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(5)
IF(_o248): lbl(-2) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(5)
Agente CBT en nodo 6.Tiempo: 1.0700000000000001. Interfaces:
IF(_o261): lbl(17) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(6)
IF(_o260): lbl(-2) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(6)
Agente CBT en nodo 7.Tiempo: 1.0800000000000001. Interfaces:
IF(_o270): lbl(7) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(7)
IF(_o269): lbl(-2) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(7)
Grupo 65490 joined.
Grupo 65490, ya joined.
Agente CBT en nodo 0. Tiempo: 2. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 0. Tiempo: 2. MFCTable:
Group:65490 - Core:3 - if to parent:0 - nexthop: * - delete timer:
_o307
   Child iifs    : -2
   Child nexthops: -1
Grupo 65490 joined.
Grupo 65490, ya joined.
Agente CBT en nodo 1. Tiempo: 2.1000000000000001. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 1. Tiempo: 2.1000000000000001. MFCTable:
Group:65490 - Core:3 - if to parent:2 - nexthop: * - delete timer:
_o303
   Child iifs    : 1 -2
   Child nexthops: * -1
Agente CBT en nodo 2. Tiempo: 2.2000000000000002. MFCTable:
Group:65490 - Core:3 - if to parent:8 - nexthop: * - delete timer:
_o299
   Child iifs    : 3 10
   Child nexthops: * *
Agente CBT en nodo 2. Tiempo: 2.2000000000000002. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 3. Tiempo: 2.2999999999999998. MFCTable:
Group:65490 - Core:3 - if to parent:-1 - nexthop: -1 - delete timer: -
--
   Child iifs    : 12 9
   Child nexthops: * *
Agente CBT en nodo 3. Tiempo: 2.2999999999999998. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 4. Tiempo: 2.3999999999999999. MFCTable:
Group:65490 - Core:3 - if to parent:13 - nexthop: * - delete timer:
_o291
   Child iifs    : -2
   Child nexthops: -1
Agente CBT en nodo 4. Tiempo: 2.3999999999999999. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 5. Tiempo: 2.5. MFCTable:
```

```
Group:65490 - Core:3 - if to parent:11 - nexthop: * - delete timer:
_o321
   Child iifs    : 16
   Child nexthops: *
Agente CBT en nodo 5. Tiempo: 2.5. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 6. Tiempo: 2.6000000000000001. MFCTable:
Group:65490 - Core:3 - if to parent:17 - nexthop: * - delete timer:
_o327
   Child iifs    : -2
   Child nexthops: -1
Agente CBT en nodo 6. Tiempo: 2.6000000000000001. Transient_Table:
No existen entradas en Transient_Table
Nodo: 3, child: 4 2
Nodo: 4, child:
Nodo: 2, child: 1 5
Nodo: 1, child: 0
Nodo: 5, child: 6
Nodo: 0, child:
Nodo: 6, child:
Agente CBT en nodo 0. Tiempo: 3. MFCTable:
Group:65490 - Core:3 - if to parent:0 - nexthop: * - delete timer:
_o307
   Child iifs    : -2
   Child nexthops: -1
Agente CBT en nodo 0. Tiempo: 3. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 1. Tiempo: 3.1000000000000001. MFCTable:
Group:65490 - Core:3 - if to parent:2 - nexthop: * - delete timer:
_o303
   Child iifs    : 1 -2
   Child nexthops: * -1
Agente CBT en nodo 1. Tiempo: 3.1000000000000001. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 2. Tiempo: 3.2000000000000002. MFCTable:
Group:65490 - Core:3 - if to parent:8 - nexthop: * - delete timer:
_o299
   Child iifs    : 3 10
   Child nexthops: * *
Agente CBT en nodo 2. Tiempo: 3.2000000000000002. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 3. Tiempo: 3.2999999999999998. MFCTable:
Group:65490 - Core:3 - if to parent:-1 - nexthop: -1 - delete timer: -
--
   Child iifs    : 12 9
   Child nexthops: * *
Agente CBT en nodo 3. Tiempo: 3.2999999999999998. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 4. Tiempo: 3.3999999999999999. MFCTable:
Group:65490 - Core:3 - if to parent:13 - nexthop: * - delete timer:
_o291
   Child iifs    : -2
   Child nexthops: -1
Agente CBT en nodo 4. Tiempo: 3.3999999999999999. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 5. Tiempo: 3.5. MFCTable:
Group:65490 - Core:3 - if to parent:11 - nexthop: * - delete timer:
_o321
   Child iifs    : 16
   Child nexthops: *
Agente CBT en nodo 5. Tiempo: 3.5. Transient_Table:
```

```
No existen entradas en Transient_Table
Agente CBT en nodo 6. Tiempo: 3.6000000000000001. MFCTable:
Group:65490 - Core:3 - if to parent:17 - nexthop: * - delete timer:
_o327
   Child iifs    : -2
   Child nexthops: -1
Agente CBT en nodo 6. Tiempo: 3.6000000000000001. Transient_Table:
No existen entradas en Transient_Table
Nodo: 3, child: 4 2
Nodo: 4, child:
Nodo: 2, child: 1 5
Nodo: 1, child: 0
Nodo: 5, child: 6
Nodo: 0, child:
Nodo: 6, child:
Agente CBT en nodo 0. Tiempo: 4. MFCTable:
Group:65490 - Core:3 - if to parent:0 - nexthop: * - delete timer:
_o307
   Child iifs    : -2
   Child nexthops: -1
Agente CBT en nodo 0. Tiempo: 4. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 1. Tiempo: 4.0999999999999996. MFCTable:
Group:65490 - Core:3 - if to parent:2 - nexthop: * - delete timer:
_o303
   Child iifs    : 1 -2
   Child nexthops: * -1
Agente CBT en nodo 1. Tiempo: 4.0999999999999996. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 2. Tiempo: 4.2000000000000002. MFCTable:
Group:65490 - Core:3 - if to parent:8 - nexthop: * - delete timer:
_o299
   Child iifs    : 3 10
   Child nexthops: * *
Agente CBT en nodo 2. Tiempo: 4.2000000000000002. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 3. Tiempo: 4.2999999999999998. MFCTable:
Group:65490 - Core:3 - if to parent:-1 - nexthop: -1 - delete timer: -
--
   Child iifs    : 12 9
   Child nexthops: * *
Agente CBT en nodo 3. Tiempo: 4.2999999999999998. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 4. Tiempo: 4.4000000000000004. MFCTable:
Group:65490 - Core:3 - if to parent:13 - nexthop: * - delete timer:
_o291
   Child iifs    : -2
   Child nexthops: -1
Agente CBT en nodo 4. Tiempo: 4.4000000000000004. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 5. Tiempo: 4.5. MFCTable:
Group:65490 - Core:3 - if to parent:11 - nexthop: * - delete timer:
_o321
   Child iifs    : 16
   Child nexthops: *
Agente CBT en nodo 5. Tiempo: 4.5. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 6. Tiempo: 4.5999999999999996. MFCTable:
Group:65490 - Core:3 - if to parent:17 - nexthop: * - delete timer:
_o327
   Child iifs    : -2
```

B.3 Ejemplo de las facilidades de configuración: intercambio de PDUs

El objetivo de este ejemplo es mostrar el uso de las facilidades de configuración implementadas: designated routers y configuración de cores. Se muestra en este caso las PDUs intercambiadas por los nodos.

B.3.1 Código

```
# Topologia ejemplo de redes de acceso multiple/ vinculos sin
capacidad
# multicast
# Una de las redes es multiacceso multicast
# Otra red es multiacceso no multicast
# Un vinculo es punto a punto no multicast
# Por salida standard se muestra el intercambio de PDUs entre los
agentes
# CBT para la secuencia dada, utilizando la capacidad de debugging
implementada.
# En el archivo mensaje se muestra el intercambio de mensajes entre
los
# agentes emisores y receptores implementados para prueba

set ns [new Simulator]
set dels [open mensajes w]
Simulator set EnableMcast_ 1
Simulator set NumberInterfaces_ 1

# Carga del codigo implementado
source cbtmain.tcl

#Generacion de nodos
puts "Generando nodos . . ."
for {set i 0} {$i <= 7} {incr i 1} {
   set n($i) [$ns node]
}

#Topologia
puts "Generando links .  . . "
$ns duplex-link $n(0) $n(1) 1.5Mb 10ms DropTail
$ns duplex-link $n(1) $n(2) 1.5Mb 10ms DropTail
$ns multi-link-of-interfaces "$n(1) $n(3) $n(4) $n(5)" 1.5Mb 10ms
DropTail
$ns multi-link-of-interfaces "$n(5) $n(6) $n(7)" 1.5Mb 10ms DropTail

#Agentes CBT
puts "Generando agentes CBT . . . "
for {set i 0} {$i <= 7} {incr i 1} {
   set cbt($i) [new CBT $ns $n($i)]
   $cbt($i) enable_ALL
}

#Configuracion de vinculos no multicast
$cbt(0) set_unicast 1
$cbt(1) set_unicast 0

$cbt(1) set_unicast 3
$cbt(5) set_unicast 3
$cbt(4) set_unicast 3
$cbt(3) set_unicast 4
```

```
#Configuracion de DRs
#Nodo 3 en red no multiacceso no multicast
$cbt(3) set_des_router 4 3
$cbt(4) set_des_router 3 3
$cbt(1) set_des_router 3 3
$cbt(5) set_des_router 3 3

#Nodo 5 en red multiacceso multicast
$cbt(5) set_des_router 6 5
$cbt(6) set_des_router 5 5
$cbt(7) set_des_router 5 5


#Arranque de los agentes CBT
for {set i 0} {$i <= 7} {incr i 1} {
   $cbt($i) start
}



#Configuracion de nodo n(0) como core para el grupo 65490
$cbt(0) set-core 65490

#Creacion de los agentes multicast de prueba
#Emisores al grupo, en nodos n(7) y n(4), emiten 10 paquetes/segundo
#de 200 y 300 bytes c/u respectivamente
#Los agentes emisores deben realizar el join group debido a que no
esta
#implementado el envio desde un nodo no miembro del grupo
   set  e(7) [new Agent/Message/Emisor_delay 0.1 $dels]
   $ns attach-agent $n(7) $e(7)
   $ns at 0.0 "$e(7) join-group 65490"
   $e(7) set dst_ 65490
   $e(7) set packetSize_ 300
   set  e(4) [new Agent/Message/Emisor_delay 0.1 $dels]
   $ns attach-agent $n(4) $e(4)
   $ns at 0.0 "$e(4) join-group 65490"
   $e(4) set dst_ 65490
   $e(4) set packetSize_ 200

#Agentes receptores en nodos 6, 2 y 1. Registran los mensajes en el
archivo
#mensajes
   set re(6) [new Agent/Message/Receptor_delay $dels]
   $ns attach-agent $n(6) $re(6)
   set re(2) [new Agent/Message/Receptor_delay $dels]
   $ns attach-agent $n(2) $re(2)
   set re(1) [new Agent/Message/Receptor_delay $dels]
   $ns attach-agent $n(1) $re(1)

#Arranques de los agentes emisores
   $ns at 2.0 "$e(7) start"
   $ns at 10.0 "$e(4) start"
   $ns at 32.0 "$e(7) stop"
   $ns at 32.0 "$e(4) stop"

#Join y leaves de los receptores
   $ns at 2.0 "$re(6) join-group 65490"
   $ns at 2.0 "$re(1) join-group 65490"
```

```
   $ns at 5.0 "$re(2) join-group 65490"
   $ns at 12.0 "$re(6) leave-group 65490"
   $ns at 140.0 "final"


proc final {} {
   global ns dels
   close $dels
   exit 0
}

$ns run
```

B.3.2 Salida generada (Se muestra parcialmente por razones de extensión)

```
Generando nodos . . .
Generando links .  . .
Generando agentes CBT . . .
Node:0 Agregando entrada MFC, group:65490 core:0 if:-1 nxp:-1 at 0
Node:7 Agregando entrada Transitoria, group:65490 core:0 leaf:Y if:10
nxp:* at 0
Node 7 Delete_T_Entry_Timer(65490) (Creando en tiempo: 0)
Node:7 ==>JOIN_REQ: 65490 0 7 M 7  Tiempo:0
Node 7 Resend_Join_Timer(10,*) (Creando en tiempo: 0)
Node:7 Agregando child (transitorio), group:65490 core:0 if_ch:-2
nxp_ch:-1 at 0
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:4 Agregando entrada Transitoria, group:65490 core:0 leaf:Y if:6
nxp:1 at 0
Node 4 Delete_T_Entry_Timer(65490) (Creando en tiempo: 0)
Node:4 ==>JOIN_REQ: 65490 0 4 U 4  Tiempo:0
Node 4 Resend_Join_Timer(6,1) (Creando en tiempo: 0)
Node:4 Agregando child (transitorio), group:65490 core:0 if_ch:-2
nxp_ch:-1 at 0
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:1 <== JOIN_REQ: 65490 0 4 U 4  Tiempo:0.010960000000000001
Node:1 Agregando entrada Transitoria, group:65490 core:0 leaf:N if:1
nxp:0 at 0.010960000000000001
Node 1 Delete_T_Entry_Timer(65490) (Creando en tiempo:
0.010960000000000001)
Node:1 ==>JOIN_REQ: 65490 0 1 U 1  Tiempo:0.010960000000000001
Node:1 Agregando child (transitorio), group:65490 core:0 if_ch:4
nxp_ch:4 at 0.010960000000000001
Node:6 <== JOIN_REQ: 65490 0 7 M 7  Tiempo:0.010960000000000001
Node:5 <== JOIN_REQ: 65490 0 7 M 7  Tiempo:0.010960000000000001
Node:5 Agregando entrada Transitoria, group:65490 core:0 leaf:N if:7
nxp:1 at 0.010960000000000001
Node 5 Delete_T_Entry_Timer(65490) (Creando en tiempo:
0.010960000000000001)
Node:5 ==>JOIN_REQ: 65490 0 5 U 5  Tiempo:0.010960000000000001
Node:5 Agregando child (transitorio), group:65490 core:0 if_ch:8
nxp_ch:* at 0.010960000000000001
Node:0 <== JOIN_REQ: 65490 0 1 U 1  Tiempo:0.021920000000000002
Node:0 MFC:Agregando child, group:65490 core:0 if_ch:0 nxp_ch:1 at
0.021920000000000002
Node:0 ==>JOIN_ACK: 65490 0 U 0  Tiempo:0.021920000000000002
Node:1 <== JOIN_REQ: 65490 0 5 U 5  Tiempo:0.022880000000000005
```

```
Node:1 Agregando child (transitorio), group:65490 core:0 if_ch:4
nxp_ch:5 at 0.022880000000000005
Node:1 <== JOIN_REQ: 65490 0 4 U 4  Tiempo:0.023840000000000004
Node:1 <== JOIN_REQ: 65490 0 4 U 4  Tiempo:0.024800000000000003
Node:1 <== JOIN_ACK: 65490 0 U 0  Tiempo:0.032880000000000006
Node:1 Agregando entrada MFC, group:65490 core:0 if:1 nxp:0 at
0.032880000000000006
Node 1 Delete_Entry_Timer(65490) (Creando en tiempo:
0.032880000000000006)
Node 1 Echo_Request_Timer(1,0) (Creando en tiempo:
0.032880000000000006)
Node:1 MFC:Agregando child, group:65490 core:0 if_ch:4 nxp_ch:4 at
0.032880000000000006
Node:1 ==>JOIN_ACK: 65490 0 U 1  Tiempo:0.032880000000000006
Node:1 Eliminando child (transitorio), group:65490 core:0 if_ch:4
nxp_ch:4 at 0.032880000000000006
Node:1 MFC:Agregando child, group:65490 core:0 if_ch:4 nxp_ch:5 at
0.032880000000000006
Node:1 ==>JOIN_ACK: 65490 0 U 1  Tiempo:0.032880000000000006
Node:1 Eliminando child (transitorio), group:65490 core:0 if_ch:4
nxp_ch:5 at 0.032880000000000006
Node:1 Eliminando entrada Transitoria, group:65490 core:0 if:1 nxp:0
at 0.032880000000000006
Node:1 <== JOIN_REQ: 65490 0 5 U 5  Tiempo:0.033840000000000009
Node:1 ==>JOIN_ACK: 65490 0 U 1  Tiempo:0.033840000000000009
Node:1 <== JOIN_REQ: 65490 0 5 U 5  Tiempo:0.034800000000000005
Node:1 ==>JOIN_ACK: 65490 0 U 1  Tiempo:0.034800000000000005
Node:4 <== JOIN_ACK: 65490 0 U 1  Tiempo:0.043840000000000004
Node:4 Agregando entrada MFC, group:65490 core:0 if:6 nxp:1 at
0.043840000000000004
Node 4 Delete_Entry_Timer(65490) (Creando en tiempo:
0.043840000000000004)
Node 4 Echo_Request_Timer(6,1) (Creando en tiempo:
0.043840000000000004)
Node:4 MFC:Agregando child, group:65490 core:0 if_ch:-2 nxp_ch:-1 at
0.043840000000000004
Node:4 Eliminando child (transitorio), group:65490 core:0 if_ch:-2
nxp_ch:-1 at 0.043840000000000004
Node:4 Eliminando entrada Transitoria, group:65490 core:0 if:6 nxp:1
at 0.043840000000000004
Node 4 Resend_Join_Timer(6,1) (Cancelando en tiempo:
0.043840000000000004)
Node:5 <== JOIN_ACK: 65490 0 U 1  Tiempo:0.044800000000000006
Node:5 Agregando entrada MFC, group:65490 core:0 if:7 nxp:1 at
0.044800000000000006
Node 5 Delete_Entry_Timer(65490) (Creando en tiempo:
0.044800000000000006)
Node 5 Echo_Request_Timer(7,1) (Creando en tiempo:
0.044800000000000006)
Node:5 MFC:Agregando child, group:65490 core:0 if_ch:8 nxp_ch:* at
0.044800000000000006
Node:5 ==>JOIN_ACK: 65490 0 M 5  Tiempo:0.044800000000000006
Node:5 Eliminando child (transitorio), group:65490 core:0 if_ch:8
nxp_ch:* at 0.044800000000000006
Node:5 Eliminando entrada Transitoria, group:65490 core:0 if:7 nxp:1
at 0.044800000000000006
Node:5 <== JOIN_ACK: 65490 0 U 1  Tiempo:0.045760000000000009
Nodo:5 descarta join_ack (no grupo) tiempo 0.045760000000000009
Node:5 <== JOIN_ACK: 65490 0 U 1  Tiempo:0.046720000000000012
Nodo:5 descarta join_ack (no grupo) tiempo 0.046720000000000012
Node:4 <== JOIN_ACK: 65490 0 U 1  Tiempo:0.054800000000000001
```

```
Nodo:4 descarta join_ack (no grupo) tiempo 0.054800000000000001
Node:5 <== JOIN_ACK: 65490 0 U 1  Tiempo:0.055760000000000004
Nodo:5 descarta join_ack (no grupo) tiempo 0.055760000000000004
Node:6 <== JOIN_ACK: 65490 0 M 5  Tiempo:0.055760000000000004
Nodo:6 descarta join_ack (no grupo) tiempo 0.055760000000000004
Node:7 <== JOIN_ACK: 65490 0 M 5  Tiempo:0.055760000000000004
Node:7 Agregando entrada MFC, group:65490 core:0 if:10 nxp:* at
0.055760000000000004
Node 7 Delete_Entry_Timer(65490) (Creando en tiempo:
0.055760000000000004)
Node 7 Echo_Request_Timer(10,*) (Creando en tiempo:
0.055760000000000004)
Node:7 MFC:Agregando child, group:65490 core:0 if_ch:-2 nxp_ch:-1 at
0.055760000000000004
Node:7 Eliminando child (transitorio), group:65490 core:0 if_ch:-2
nxp_ch:-1 at 0.055760000000000004
Node:7 Eliminando entrada Transitoria, group:65490 core:0 if:10 nxp:*
at 0.055760000000000004
Node 7 Resend_Join_Timer(10,*) (Cancelando en tiempo:
0.055760000000000004)
Node:4 <== JOIN_ACK: 65490 0 U 1  Tiempo:0.055760000000000004
Nodo:4 descarta join_ack (no grupo) tiempo 0.055760000000000004
Node:5 <== JOIN_ACK: 65490 0 U 1  Tiempo:0.056720000000000007
Nodo:5 descarta join_ack (no grupo) tiempo 0.056720000000000007
Node:5 <== JOIN_ACK: 65490 0 U 1  Tiempo:0.057680000000000009
Nodo:5 descarta join_ack (no grupo) tiempo 0.057680000000000009
Node:5 <== JOIN_ACK: 65490 0 U 1  Tiempo:0.058640000000000012
Nodo:5 descarta join_ack (no grupo) tiempo 0.058640000000000012
Node:5 <== JOIN_ACK: 65490 0 U 1  Tiempo:0.059600000000000014
Nodo:5 descarta join_ack (no grupo) tiempo 0.059600000000000014
Node:5 <== JOIN_ACK: 65490 0 U 1  Tiempo:0.060560000000000017
Nodo:5 descarta join_ack (no grupo) tiempo 0.060560000000000017
Node:6 Agregando entrada Transitoria, group:65490 core:0 leaf:Y if:9
nxp:* at 2
Node 6 Delete_T_Entry_Timer(65490) (Creando en tiempo: 2)
Node:6 ==>JOIN_REQ: 65490 0 6 M 6  Tiempo:2
Node 6 Resend_Join_Timer(9,*) (Creando en tiempo: 2)
Node:6 Agregando child (transitorio), group:65490 core:0 if_ch:-2
nxp_ch:-1 at 2
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:1 MFC:Agregando child, group:65490 core:0 if_ch:-2 nxp_ch:-1 at 2
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:5 <== JOIN_REQ: 65490 0 6 M 6  Tiempo:2.0109599999999999
Node:5 ==>JOIN_ACK: 65490 0 M 5  Tiempo:2.0109599999999999
Node:7 <== JOIN_REQ: 65490 0 6 M 6  Tiempo:2.0109599999999999
Node:6 <== JOIN_ACK: 65490 0 M 5  Tiempo:2.0219199999999997
Node:6 Agregando entrada MFC, group:65490 core:0 if:9 nxp:* at
2.0219199999999997
Node 6 Delete_Entry_Timer(65490) (Creando en tiempo:
2.0219199999999997)
Node 6 Echo_Request_Timer(9,*) (Creando en tiempo: 2.0219199999999997)
Node:6 MFC:Agregando child, group:65490 core:0 if_ch:-2 nxp_ch:-1 at
2.0219199999999997
Node:6 Eliminando child (transitorio), group:65490 core:0 if_ch:-2
nxp_ch:-1 at 2.0219199999999997
Node:6 Eliminando entrada Transitoria, group:65490 core:0 if:9 nxp:*
at 2.0219199999999997
Node 6 Resend_Join_Timer(9,*) (Cancelando en tiempo:
2.0219199999999997)
```

```
Node:7 <== JOIN_ACK: 65490 0 M 5  Tiempo:2.0219199999999997
Nodo:7 descarta join_ack (no grupo) tiempo 2.0219199999999997
Node 0  drop.  replicator: _o268 source: 7 dest: 65490
Node:2 Agregando entrada Transitoria, group:65490 core:0 leaf:Y if:3
nxp:* at 5
Node 2 Delete_T_Entry_Timer(65490) (Creando en tiempo: 5)
Node:2 ==>JOIN_REQ: 65490 0 2 M 2  Tiempo:5
Node 2 Resend_Join_Timer(3,*) (Creando en tiempo: 5)
Node:2 Agregando child (transitorio), group:65490 core:0 if_ch:-2
nxp_ch:-1 at 5
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:1 <== JOIN_REQ: 65490 0 2 M 2  Tiempo:5.0109599999999999
Node:1 MFC:Agregando child, group:65490 core:0 if_ch:2 nxp_ch:* at
5.0109599999999999
Node:1 ==>JOIN_ACK: 65490 0 M 1  Tiempo:5.0109599999999999
Node:2 <== JOIN_ACK: 65490 0 M 1  Tiempo:5.0219199999999997
Node:2 Agregando entrada MFC, group:65490 core:0 if:3 nxp:* at
5.0219199999999997
Node 2 Delete_Entry_Timer(65490) (Creando en tiempo:
5.0219199999999997)
Node 2 Echo_Request_Timer(3,*) (Creando en tiempo: 5.0219199999999997)
Node:2 MFC:Agregando child, group:65490 core:0 if_ch:-2 nxp_ch:-1 at
5.0219199999999997
Node:2 Eliminando child (transitorio), group:65490 core:0 if_ch:-2
nxp_ch:-1 at 5.0219199999999997
Node:2 Eliminando entrada Transitoria, group:65490 core:0 if:3 nxp:*
at 5.0219199999999997
Node 2 Resend_Join_Timer(3,*) (Cancelando en tiempo:
5.0219199999999997)
Node 0  drop.  replicator: _o284 source: 4 dest: 65490
Node:6 MFC:Eliminando child, group:65490 core:0 if_ch:-2 nxp_ch:-1 at
12
Node:6 Borrando entrada MFC, group:65490 core:0 if:9 nxp:* at 12
Node:6 ==>QUIT_NOT: 65490 6 M 6  Tiempo:12
Node 6 Quit_Notification_Timer(65490:9,*) (Creando en tiempo: 12)
Node 6 Echo_Request_Timer(9,*) (Cancelando en tiempo: 12)
Leaving group 65490
Node:5 <== QUIT_NOT: 65490 6 M 6  Tiempo:12.010960000000001
Node 5 Cache_Del_Timer(65490,8,*) (Creando en tiempo:
12.010960000000001)
Node:7 <== QUIT_NOT: 65490 6 M 6  Tiempo:12.010960000000001
Node 7 Send_Join_Timer(10, *) (Creando en tiempo: 12.010960000000001)
Node:7 ==>JOIN_REQ: 65490 0 7 M 7  Tiempo:12.010999999999999
Node 6  drop.  replicator: _o269 source: 7 dest: 65490
Node:5 <== JOIN_REQ: 65490 0 7 M 7  Tiempo:12.02196
Node:5 ==>JOIN_ACK: 65490 0 M 5  Tiempo:12.02196
Node:6 <== JOIN_REQ: 65490 0 7 M 7  Tiempo:12.02196
Node:6 <== JOIN_ACK: 65490 0 M 5  Tiempo:12.032920000000001
Nodo:6 descarta join_ack (no grupo) tiempo 12.032920000000001
Node:7 <== JOIN_ACK: 65490 0 M 5  Tiempo:12.032920000000001
Nodo:7 descarta join_ack (no grupo) tiempo 12.032920000000001
Node 6 Quit_Notification_Timer(65490:9,*) (Timeout en tiempo: 15)
Node:6 ==>QUIT_NOT: 65490 6 M 6  Tiempo:15
Node:5 <== QUIT_NOT: 65490 6 M 6  Tiempo:15.010960000000001
Node:7 <== QUIT_NOT: 65490 6 M 6  Tiempo:15.010960000000001
Node 7 Send_Join_Timer(10, *) (Creando en tiempo: 15.010960000000001)
Node:7 ==>JOIN_REQ: 65490 0 7 M 7  Tiempo:15.4056
Node:5 <== JOIN_REQ: 65490 0 7 M 7  Tiempo:15.41656
Node:5 ==>JOIN_ACK: 65490 0 M 5  Tiempo:15.41656
Node:6 <== JOIN_REQ: 65490 0 7 M 7  Tiempo:15.41656
```

```
Node:6 <== JOIN_ACK: 65490 0 M 5  Tiempo:15.427520000000001
Nodo:6 descarta join_ack (no grupo) tiempo 15.427520000000001
Node:7 <== JOIN_ACK: 65490 0 M 5  Tiempo:15.427520000000001
Nodo:7 descarta join_ack (no grupo) tiempo 15.427520000000001
Node:5 MFC:Eliminando child, group:65490 core:0 if_ch:8 nxp_ch:* at
16.510999999999999
Node:5 Borrando entrada MFC, group:65490 core:0 if:7 nxp:1 at
16.510999999999999
Node:5 ==>QUIT_NOT: 65490 5 U 5  Tiempo:16.510999999999999
Node 5 Quit_Notification_Timer(65490:7,1) (Creando en tiempo:
16.510999999999999)
Node 5 Echo_Request_Timer(7,1) (Cancelando en tiempo:
16.510999999999999)
Node 5  drop.  replicator: _o262 source: 7 dest: 65490
Node:1 <== QUIT_NOT: 65490 5 U 5  Tiempo:16.52196
Node:1 MFC:Eliminando child, group:65490 core:0 if_ch:4 nxp_ch:5 at
16.52196
Node:1 <== QUIT_NOT: 65490 5 U 5  Tiempo:16.532920000000001
Node:1 <== QUIT_NOT: 65490 5 U 5  Tiempo:16.53388
Node 1  drop.  replicator: _o282 source: 4 dest: 65490
Node 6 Quit_Notification_Timer(65490:9,*) (Timeout en tiempo: 18)
Node:6 ==>QUIT_NOT: 65490 6 M 6  Tiempo:18
Node:5 <== QUIT_NOT: 65490 6 M 6  Tiempo:18.010960000000001
Nodo:5  Descarta QUIT-NOT (grupo no on tree) tiempo 18.010960000000001
Node:7 <== QUIT_NOT: 65490 6 M 6  Tiempo:18.010960000000001
Node 7 Send_Join_Timer(10, *) (Creando en tiempo: 18.010960000000001)
Node 5 Quit_Notification_Timer(65490:7,1) (Timeout en tiempo:
19.510999999999999)
Node:5 ==>QUIT_NOT: 65490 5 U 5  Tiempo:19.510999999999999
Node:1 <== QUIT_NOT: 65490 5 U 5  Tiempo:19.52196
Node:1 <== QUIT_NOT: 65490 5 U 5  Tiempo:19.532920000000001
Node:1 <== QUIT_NOT: 65490 5 U 5  Tiempo:19.53388
Node:7 ==>JOIN_REQ: 65490 0 7 M 7  Tiempo:20.277799999999999
Node:5 <== JOIN_REQ: 65490 0 7 M 7  Tiempo:20.28876
Node 5 Quit_Notification_Timer(65490:7,1) (Cancelando en tiempo:
20.28876)
Node:5 Agregando entrada Transitoria, group:65490 core:0 leaf:N if:7
nxp:1 at 20.28876
Node 5 Delete_T_Entry_Timer(65490) (Creando en tiempo: 20.28876)
Node:5 ==>JOIN_REQ: 65490 0 5 U 5  Tiempo:20.28876
Node:5 Agregando child (transitorio), group:65490 core:0 if_ch:8
nxp_ch:* at 20.28876
Node:6 <== JOIN_REQ: 65490 0 7 M 7  Tiempo:20.28876
Node:1 <== JOIN_REQ: 65490 0 5 U 5  Tiempo:20.30068
Node:1 MFC:Agregando child, group:65490 core:0 if_ch:4 nxp_ch:5 at
20.30068
Node:1 ==>JOIN_ACK: 65490 0 U 1  Tiempo:20.30068
Node:5 <== JOIN_ACK: 65490 0 U 1  Tiempo:20.311640000000001
Node:5 Agregando entrada MFC, group:65490 core:0 if:7 nxp:1 at
20.311640000000001
Node 5 Delete_Entry_Timer(65490) (Creando en tiempo:
20.311640000000001)
Node 5 Echo_Request_Timer(7,1) (Creando en tiempo: 20.311640000000001)
Node:5 MFC:Agregando child, group:65490 core:0 if_ch:8 nxp_ch:* at
20.311640000000001
Node:5 ==>JOIN_ACK: 65490 0 M 5  Tiempo:20.311640000000001
Node:5 Eliminando child (transitorio), group:65490 core:0 if_ch:8
nxp_ch:* at 20.311640000000001
Node:5 Eliminando entrada Transitoria, group:65490 core:0 if:7 nxp:1
at 20.311640000000001
Node:1 <== JOIN_REQ: 65490 0 5 U 5  Tiempo:20.311920000000001
```

B.4 Ejemplo de las facilidades de configuración: estado de los nodos

El objetivo de este ejemplo es mostrar el uso de las facilidades de configuración implementadas: designated routers y configuración de cores. Se muestra en este caso el estado de tablas e interfaces en los nodos.

B.4.1 Código

```
# Topologia ejemplo de redes de acceso multiple/ vinculos sin
capacidad
# multicast
# Una de las redes es multiacceso multicast
# Otra red es multiacceso no multicast
# Un vinculo es punto a punto no multicast
# Por salida standard se muestra el estado de las tablas en los nodos
y el
# arbol de distribucion del grupo luego de los cambios producidos por
los
# aplicaciones multicast en los nodos
# No se generan archivos con intercambio de PDUs ni de mensajes entre
las
# aplicaciones, ni el tracing provisto por Ns

set ns [new Simulator]
Simulator set EnableMcast_ 1
Simulator set NumberInterfaces_ 1

set dels [open mensajes w]

# Carga del codigo implementado
source cbtmain.tcl

#Generacion de nodos
puts "Generando nodos . . ."
for {set i 0} {$i <= 7} {incr i 1} {
   set n($i) [$ns node]
}

#Topologia
puts "Generando links .  . . "
$ns duplex-link $n(0) $n(1) 1.5Mb 10ms DropTail
$ns duplex-link $n(1) $n(2) 1.5Mb 10ms DropTail
$ns multi-link-of-interfaces "$n(1) $n(3) $n(4) $n(5)" 1.5Mb 10ms
DropTail
$ns multi-link-of-interfaces "$n(5) $n(6) $n(7)" 1.5Mb 10ms DropTail

#Agentes CBT
puts "Generando agentes CBT . . . "
for {set i 0} {$i <= 7} {incr i 1} {
   set cbt($i) [new CBT $ns $n($i)]
}

#Configuracion de vinculos no multicast
$cbt(0) set_unicast 1
$cbt(1) set_unicast 0

$cbt(1) set_unicast 3
$cbt(5) set_unicast 3
$cbt(4) set_unicast 3
```

```
$cbt(3) set_unicast 4


#Configuracion de DRs
#Nodo 3 en red no multiacceso no multicast
$cbt(3) set_des_router 4 3
$cbt(4) set_des_router 3 3
$cbt(1) set_des_router 3 3
$cbt(5) set_des_router 3 3

#Nodo 5 en red multiacceso multicast
$cbt(5) set_des_router 6 5
$cbt(6) set_des_router 5 5
$cbt(7) set_des_router 5 5


#Arranque de los agentes CBT
for {set i 0} {$i <= 7} {incr i 1} {
   $cbt($i) start
}



#Configuracion de nodo n(0) como core para el grupo 65490
$cbt(0) set-core 65490

#Creacion de los agentes multicast de prueba
#Emisores al grupo, en nodos n(7) y n(4), emiten 10 paquetes/segundo
#de 200 y 300 bytes c/u respectivamente
#Los agentes emisores deben realizar el join group debido a que no
esta
#implementado el envio desde un nodo no miembro del grupo
   set  e(7) [new Agent/Message/Emisor_delay 0.1 $dels]
   $ns attach-agent $n(7) $e(7)
   $ns at 0.0 "$e(7) join-group 65490"
   $e(7) set dst_ 65490
   $e(7) set packetSize_ 300
   set  e(4) [new Agent/Message/Emisor_delay 0.1 $dels]
   $ns attach-agent $n(4) $e(4)
   $ns at 0.0 "$e(4) join-group 65490"
   $e(4) set dst_ 65490
   $e(4) set packetSize_ 200

#Agentes receptores en nodos 6, 2 y 1. Registran los mensajes en el
archivo
#mensajes
   set re(6) [new Agent/Message/Receptor_delay $dels]
   $ns attach-agent $n(6) $re(6)
   set re(2) [new Agent/Message/Receptor_delay $dels]
   $ns attach-agent $n(2) $re(2)
   set re(1) [new Agent/Message/Receptor_delay $dels]
   $ns attach-agent $n(1) $re(1)

#Arranques de los agentes emisores
   $ns at 2.0 "$e(7) start"
   $ns at 10.0 "$e(4) start"
   $ns at 32.0 "$e(7) stop"
   $ns at 32.0 "$e(4) stop"

#Join y leaves de los receptores
   $ns at 2.0 "$re(6) join-group 65490"
```

```
   $ns at 2.0 "$re(1) join-group 65490"
   $ns at 5.0 "$re(2) join-group 65490"
   $ns at 12.0 "$re(6) leave-group 65490"
   $ns at 140.0 "final"

#Listado de interfaces en los nodos
   $ns at 1.01 "$cbt(0) list_interfaces"
   $ns at 1.02 "$cbt(1) list_interfaces"
   $ns at 1.03 "$cbt(2) list_interfaces"
   $ns at 1.04 "$cbt(3) list_interfaces"
   $ns at 1.05 "$cbt(4) list_interfaces"
   $ns at 1.06 "$cbt(5) list_interfaces"
   $ns at 1.07 "$cbt(6) list_interfaces"
   $ns at 1.08 "$cbt(7) list_interfaces"

# Sentencias de listado cada 1 segundo, desde 1 hasta 13
   for {set i 2} {$i < 14} {incr i 1} {
          for {set k 0 } {$k < 7} {incr k 1} {
         $ns at $i.$k "$cbt($k) list_MFCTable"
         $ns at $i.$k "$cbt($k) list_TransientTable"
      }
      $ns at $i.8 "$ns map-tree 65490"
   }



proc final {} {
   global ns dels
   close $dels
   exit 0
}

$ns run
```

B.4.2 Salida generada (Se muestra parcialmente por razones de extensión)

```
Generando nodos . . .
Generando links .  . .
Generando agentes CBT . . .
Grupo 65490 joined.
Grupo 65490, ya joined.
Grupo 65490 joined.
Grupo 65490, ya joined.
Agente CBT en nodo 0.Tiempo: 1.01. Interfaces:
IF(_o154): lbl(0) -link(PaP/U) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(0)
IF(_o153): lbl(-2) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(0)
Agente CBT en nodo 1.Tiempo: 1.02. Interfaces:
IF(_o165): lbl(4) -link(Mac/U) -DR(3) -Pr.val(255) -Hello(60) -Addr(1)
IF(_o163): lbl(1) -link(PaP/U) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(1)
IF(_o164): lbl(2) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(1)
IF(_o162): lbl(-2) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(1)
Agente CBT en nodo 2.Tiempo: 1.03. Interfaces:
IF(_o174): lbl(3) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(2)
```

```
IF(_o173): lbl(-2) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(2)
Agente CBT en nodo 3.Tiempo: 1.04. Interfaces:
IF(_o183): lbl(5) -link(Mac/U) -DR(3) -Pr.val(255) -Hello(60) -Addr(3)
IF(_o182): lbl(-2) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(3)
Agente CBT en nodo 4.Tiempo: 1.05. Interfaces:
IF(_o192): lbl(6) -link(Mac/U) -DR(3) -Pr.val(255) -Hello(60) -Addr(4)
IF(_o191): lbl(-2) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(4)
Agente CBT en nodo 5.Tiempo: 1.0600000000000001. Interfaces:
IF(_o202): lbl(8) -link(Mac/M) -DR(5) -Pr.val(255) -Hello(60) -Addr(5)
IF(_o201): lbl(7) -link(Mac/U) -DR(3) -Pr.val(255) -Hello(60) -Addr(5)
IF(_o200): lbl(-2) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(5)
Agente CBT en nodo 6.Tiempo: 1.0700000000000001. Interfaces:
IF(_o211): lbl(9) -link(Mac/M) -DR(5) -Pr.val(255) -Hello(60) -Addr(6)
IF(_o210): lbl(-2) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(6)
Agente CBT en nodo 7.Tiempo: 1.0800000000000001. Interfaces:
IF(_o220): lbl(10) -link(Mac/M) -DR(5) -Pr.val(255) -Hello(60) -
Addr(7)
IF(_o219): lbl(-2) -link(PaP/M) -DR(-1) -Pr.val(255) -Hello(60) -
Addr(7)
Grupo 65490 joined.
Grupo 65490, ya joined.
Agente CBT en nodo 0. Tiempo: 2. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 0. Tiempo: 2. MFCTable:
Group:65490 - Core:0 - if to parent:-1 - nexthop: -1 - delete timer: -
--
   Child iifs    : 0
   Child nexthops: 1
Grupo 65490 joined.
Grupo 65490, ya joined.
Node 0  drop.  replicator: _o268 source: 7 dest: 65490
Agente CBT en nodo 1. Tiempo: 2.1000000000000001. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 1. Tiempo: 2.1000000000000001. MFCTable:
Group:65490 - Core:0 - if to parent:1 - nexthop: 0 - delete timer:
_o242
   Child iifs    : 4 4 -2
   Child nexthops: 4 5 -1
Agente CBT en nodo 2. Tiempo: 2.2000000000000002. MFCTable:
No existen entradas en MFCTable
Agente CBT en nodo 2. Tiempo: 2.2000000000000002. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 3. Tiempo: 2.2999999999999998. MFCTable:
No existen entradas en MFCTable
Agente CBT en nodo 3. Tiempo: 2.2999999999999998. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 4. Tiempo: 2.3999999999999999. MFCTable:
Group:65490 - Core:0 - if to parent:6 - nexthop: 1 - delete timer:
_o245
   Child iifs    : -2
   Child nexthops: -1
Agente CBT en nodo 4. Tiempo: 2.3999999999999999. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 5. Tiempo: 2.5. MFCTable:
Group:65490 - Core:0 - if to parent:7 - nexthop: 1 - delete timer:
_o248
```

```
   Child iifs    : 8
   Child nexthops: *
Agente CBT en nodo 5. Tiempo: 2.5. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 6. Tiempo: 2.6000000000000001. MFCTable:
Group:65490 - Core:0 - if to parent:9 - nexthop: * - delete timer:
_o266
   Child iifs    : -2
   Child nexthops: -1
Agente CBT en nodo 6. Tiempo: 2.6000000000000001. Transient_Table:
No existen entradas en Transient_Table
Nodo: 0, child: 1
Nodo: 1, child: 3 4 5
Nodo: 3, child:
Nodo: 4, child:
Nodo: 5, child: 6 7
Nodo: 6, child:
Nodo: 7, child:
Agente CBT en nodo 0. Tiempo: 3. MFCTable:
Group:65490 - Core:0 - if to parent:-1 - nexthop: -1 - delete timer: -
--
   Child iifs    : 0
   Child nexthops: 1
Agente CBT en nodo 0. Tiempo: 3. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 1. Tiempo: 3.1000000000000001. MFCTable:
Group:65490 - Core:0 - if to parent:1 - nexthop: 0 - delete timer:
_o242
   Child iifs    : 4 4 -2
   Child nexthops: 4 5 -1
Agente CBT en nodo 1. Tiempo: 3.1000000000000001. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 2. Tiempo: 3.2000000000000002. MFCTable:
No existen entradas en MFCTable
Agente CBT en nodo 2. Tiempo: 3.2000000000000002. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 3. Tiempo: 3.2999999999999998. MFCTable:
No existen entradas en MFCTable
Agente CBT en nodo 3. Tiempo: 3.2999999999999998. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 4. Tiempo: 3.3999999999999999. MFCTable:
Group:65490 - Core:0 - if to parent:6 - nexthop: 1 - delete timer:
_o245
   Child iifs    : -2
   Child nexthops: -1
Agente CBT en nodo 4. Tiempo: 3.3999999999999999. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 5. Tiempo: 3.5. MFCTable:
Group:65490 - Core:0 - if to parent:7 - nexthop: 1 - delete timer:
_o248
   Child iifs    : 8
   Child nexthops: *
Agente CBT en nodo 5. Tiempo: 3.5. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 6. Tiempo: 3.6000000000000001. MFCTable:
Group:65490 - Core:0 - if to parent:9 - nexthop: * - delete timer:
_o266
   Child iifs    : -2
   Child nexthops: -1
Agente CBT en nodo 6. Tiempo: 3.6000000000000001. Transient_Table:
No existen entradas en Transient_Table
```

```
Nodo: 0, child: 1
Nodo: 1, child: 3 4 5
Nodo: 3, child:
Nodo: 4, child:
Nodo: 5, child: 6 7
Nodo: 6, child:
Nodo: 7, child:
Agente CBT en nodo 0. Tiempo: 4. MFCTable:
Group:65490 - Core:0 - if to parent:-1 - nexthop: -1 - delete timer: -
--
   Child iifs    : 0
   Child nexthops: 1
Agente CBT en nodo 0. Tiempo: 4. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 1. Tiempo: 4.0999999999999996. MFCTable:
Group:65490 - Core:0 - if to parent:1 - nexthop: 0 - delete timer:
_o242
   Child iifs    : 4 4 -2
   Child nexthops: 4 5 -1
Agente CBT en nodo 1. Tiempo: 4.0999999999999996. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 2. Tiempo: 4.2000000000000002. MFCTable:
No existen entradas en MFCTable
Agente CBT en nodo 2. Tiempo: 4.2000000000000002. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 3. Tiempo: 4.2999999999999998. MFCTable:
No existen entradas en MFCTable
Agente CBT en nodo 3. Tiempo: 4.2999999999999998. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 4. Tiempo: 4.4000000000000004. MFCTable:
Group:65490 - Core:0 - if to parent:6 - nexthop: 1 - delete timer:
_o245
   Child iifs    : -2
   Child nexthops: -1
Agente CBT en nodo 4. Tiempo: 4.4000000000000004. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 5. Tiempo: 4.5. MFCTable:
Group:65490 - Core:0 - if to parent:7 - nexthop: 1 - delete timer:
_o248
   Child iifs    : 8
   Child nexthops: *
Agente CBT en nodo 5. Tiempo: 4.5. Transient_Table:
No existen entradas en Transient_Table
Agente CBT en nodo 6. Tiempo: 4.5999999999999996. MFCTable:
Group:65490 - Core:0 - if to parent:9 - nexthop: * - delete timer:
_o266
   Child iifs    : -2
   Child nexthops: -1
Agente CBT en nodo 6. Tiempo: 4.5999999999999996. Transient_Table:
No existen entradas en Transient_Table
Nodo: 0, child: 1
Nodo: 1, child: 3 4 5
Nodo: 3, child:
Nodo: 4, child:
Nodo: 5, child: 6 7
Nodo: 6, child:
Nodo: 7, child:
Grupo 65490 joined.
Grupo 65490, ya joined.
Agente CBT en nodo 0. Tiempo: 5. MFCTable:
```

Apéndice C: Código y resultados de la simulación

En las experiencias realizadas se seleccionó un grupo de cores fijo, y luego se tomó para este grupo cada nodo integrante de la red como posible core.

A continuación se muestra el conjunto de corridas correspondientes al nodo 28 actuando como core.

En primer lugar se realiza una corrida cuyo objeto es la formación del árbol de distribución para el grupo. Una vez determinados los vínculos involucrados, se realiza una corrida por cada uno de ellos, provocando su falla y midiendo la demora máxima en reconstruir un nuevo árbol (en el caso en que ninguno de los nodos hoja quede aislado debido a la topología de la red) y la cantidad de PDUs intercambiadas entre los nodos.

C.1 Código para determinar el árbol de distribución

```
   #Prueba para determinar el costo de la caida de un vinculo
# - medido en tiempo necesario para restaurar el arbol
# - medido en costo de transmision (cantidad de mensajes)

#Caso particular de la corrida: Core: Nodo 28
#No se produce falla, solo se realiza la corrida para determinar el
'arbol
#de distribucion generado

#Se prueba con N=10, estando estos nodos distribuidos en la red, con
una
#densidad media
#Nodos: 1 - 7 - 11 - 12 - 25 - 30 - 31 - 34 - 45 - 47

set ns [new Simulator]

#Archivo de salida para las aplicaciones que reciben paquetes
multicast
set dels [open delaySG w]

Simulator set EnableMcast_ 1
Simulator set NumberInterfaces_ 1
source cbtmain.tcl

#Ruteo dinamico en la totalidad de los nodos
$ns rtproto Session

#Se genera nodo n(0) que no se utiliza, para que los id de los nodos
#coincidan con su subindice
set n(0) [$ns node]

#NODOS
puts "Generando nodos . . ."
for {set i 1} {$i <= 47} {incr i 1} {
   set n($i) [$ns node]
}

#TOPOLOGIA
puts "Generando links .  . . "
$ns duplex-link $n(1) $n(22) 1.5Mb 10ms DropTail
```

```
$ns duplex-link $n(1) $n(29) 1.5Mb 10ms DropTail
$ns duplex-link $n(1) $n(42) 1.5Mb 10ms DropTail
$ns duplex-link $n(2) $n(43) 1.5Mb 10ms DropTail
$ns duplex-link $n(2) $n(14) 1.5Mb 10ms DropTail
$ns duplex-link $n(2) $n(15) 1.5Mb 10ms DropTail
$ns duplex-link $n(3) $n(17) 1.5Mb 10ms DropTail
$ns duplex-link $n(3) $n(4) 1.5Mb 10ms DropTail
$ns duplex-link $n(3) $n(13) 1.5Mb 10ms DropTail
$ns duplex-link $n(3) $n(32) 1.5Mb 10ms DropTail
$ns duplex-link $n(4) $n(32) 1.5Mb 10ms DropTail
$ns duplex-link $n(4) $n(7) 1.5Mb 10ms DropTail
$ns duplex-link $n(4) $n(28) 1.5Mb 10ms DropTail
$ns duplex-link $n(5) $n(21) 1.5Mb 10ms DropTail
$ns duplex-link $n(5) $n(47) 1.5Mb 10ms DropTail
$ns duplex-link $n(5) $n(34) 1.5Mb 10ms DropTail
$ns duplex-link $n(5) $n(45) 1.5Mb 10ms DropTail
$ns duplex-link $n(6) $n(15) 1.5Mb 10ms DropTail
$ns duplex-link $n(6) $n(39) 1.5Mb 10ms DropTail
$ns duplex-link $n(8) $n(17) 1.5Mb 10ms DropTail
$ns duplex-link $n(8) $n(25) 1.5Mb 10ms DropTail
$ns duplex-link $n(8) $n(41) 1.5Mb 10ms DropTail
$ns duplex-link $n(8) $n(32) 1.5Mb 10ms DropTail
$ns duplex-link $n(9) $n(40) 1.5Mb 10ms DropTail
$ns duplex-link $n(9) $n(43) 1.5Mb 10ms DropTail
$ns duplex-link $n(10) $n(13) 1.5Mb 10ms DropTail
$ns duplex-link $n(10) $n(15) 1.5Mb 10ms DropTail
$ns duplex-link $n(10) $n(30) 1.5Mb 10ms DropTail
$ns duplex-link $n(11) $n(39) 1.5Mb 10ms DropTail
$ns duplex-link $n(11) $n(34) 1.5Mb 10ms DropTail
$ns duplex-link $n(11) $n(33) 1.5Mb 10ms DropTail
$ns duplex-link $n(12) $n(44) 1.5Mb 10ms DropTail
$ns duplex-link $n(14) $n(41) 1.5Mb 10ms DropTail
$ns duplex-link $n(14) $n(27) 1.5Mb 10ms DropTail
$ns duplex-link $n(15) $n(29) 1.5Mb 10ms DropTail
$ns duplex-link $n(16) $n(22) 1.5Mb 10ms DropTail
$ns duplex-link $n(16) $n(32) 1.5Mb 10ms DropTail
$ns duplex-link $n(16) $n(42) 1.5Mb 10ms DropTail
$ns duplex-link $n(16) $n(17) 1.5Mb 10ms DropTail
$ns duplex-link $n(17) $n(24) 1.5Mb 10ms DropTail
$ns duplex-link $n(18) $n(19) 1.5Mb 10ms DropTail
$ns duplex-link $n(18) $n(20) 1.5Mb 10ms DropTail
$ns duplex-link $n(18) $n(37) 1.5Mb 10ms DropTail
$ns duplex-link $n(19) $n(20) 1.5Mb 10ms DropTail
$ns duplex-link $n(19) $n(40) 1.5Mb 10ms DropTail
$ns duplex-link $n(20) $n(31) 1.5Mb 10ms DropTail
$ns duplex-link $n(21) $n(44) 1.5Mb 10ms DropTail
$ns duplex-link $n(22) $n(26) 1.5Mb 10ms DropTail
$ns duplex-link $n(22) $n(27) 1.5Mb 10ms DropTail
$ns duplex-link $n(23) $n(27) 1.5Mb 10ms DropTail
$ns duplex-link $n(24) $n(25) 1.5Mb 10ms DropTail
$ns duplex-link $n(24) $n(26) 1.5Mb 10ms DropTail
$ns duplex-link $n(25) $n(26) 1.5Mb 10ms DropTail
$ns duplex-link $n(26) $n(35) 1.5Mb 10ms DropTail
$ns duplex-link $n(30) $n(46) 1.5Mb 10ms DropTail
$ns duplex-link $n(31) $n(43) 1.5Mb 10ms DropTail
$ns duplex-link $n(33) $n(44) 1.5Mb 10ms DropTail
$ns duplex-link $n(33) $n(46) 1.5Mb 10ms DropTail
$ns duplex-link $n(34) $n(35) 1.5Mb 10ms DropTail
$ns duplex-link $n(34) $n(36) 1.5Mb 10ms DropTail
$ns duplex-link $n(34) $n(47) 1.5Mb 10ms DropTail
$ns duplex-link $n(35) $n(36) 1.5Mb 10ms DropTail
```

```
$ns duplex-link $n(35) $n(37) 1.5Mb 10ms DropTail
$ns duplex-link $n(37) $n(38) 1.5Mb 10ms DropTail
$ns duplex-link $n(38) $n(47) 1.5Mb 10ms DropTail
$ns duplex-link $n(39) $n(40) 1.5Mb 10ms DropTail
$ns duplex-link $n(42) $n(46) 1.5Mb 10ms DropTail
$ns duplex-link $n(44) $n(45) 1.5Mb 10ms DropTail

#Listado del arbol de distribucion en 3.0, con el arbol original
$ns at 3.0 "$ns map-tree 65490"

#AGENTES CBT
puts "Generando agentes CBT . . . "
for {set i 1} {$i <= 47} {incr i 1} {
   set cbt($i) [new CBT $ns $n($i)]
   #Habilitacion de  mensajes por ciertas PDUs para controlar
   #datos de interes
   $cbt($i) enable_pdus JOIN_REQ
   $cbt($i) enable_pdus JOIN_ACK
   $cbt($i) enable_pdus QUIT_NOT
   $cbt($i) enable_pdus FLSH_TRE
}


#ARRANQUE DE AGENTES CBT
for {set i 1} {$i <= 47} {incr i 1} {
   $cbt($i) start
}

#Lista con los nodos del grupo
set lista {1 7 11 12 25 30 31 34 45 47 }
#Lista con la tasa de envio de cada nodo; en este caso es baja porque
no
#es de interes evaluar aspectos referidos al trafico multicast
set acti {1 1 1 1 1 1 1 1 1 1}

#CONFIGURACION DE CORE(S)

#UN UNICO CORE, 28
$cbt(28) set-core 65490

#Creacion de los agentes multicast de prueba, cada  emisor reside en
#uno de los nodos del grupo. Todos ellos envian al grupo
#Un receptor por nodo
set cnt 0
foreach i $lista {
   #emisor
   set  e($i) [new Agent/Message/Emisor_delay [lindex $acti $cnt]
$dels]
   $ns attach-agent $n($i) $e($i)
   $e($i) set dst_ 65490
   $e($i) set packetSize_ 200
   $ns at 118.0 "$e($i) start"
   $ns at 125.0 "$e($i) stop"
   #receptor
   set re($i) [new Agent/Message/Receptor_delay $dels]
   $ns attach-agent $n($i) $re($i)
        $ns at 0.1 "$re($i) join-group 65490"
   incr cnt 1
}

#Evento de fin
```

```
$ns at 130.0 "final"

proc final {} {
   global dels
   close $dels
   exit 0
}

#Inicio de la corrida
$ns run
```

C.2 Salida obtenida

```
Generando nodos . . .
Generando links .  . .
Generando agentes CBT . . .
Node:25 ==>JOIN_REQ: 65490 28 25 M 25  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:47 ==>JOIN_REQ: 65490 28 47 M 47  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:45 ==>JOIN_REQ: 65490 28 45 M 45  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:34 ==>JOIN_REQ: 65490 28 34 M 34  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:31 ==>JOIN_REQ: 65490 28 31 M 31  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:30 ==>JOIN_REQ: 65490 28 30 M 30  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:7 ==>JOIN_REQ: 65490 28 7 M 7  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:1 ==>JOIN_REQ: 65490 28 1 M 1  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:11 ==>JOIN_REQ: 65490 28 11 M 11  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:12 ==>JOIN_REQ: 65490 28 12 M 12  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:8 <== JOIN_REQ: 65490 28 25 M 25  Tiempo:0.11096
Node:8 ==>JOIN_REQ: 65490 28 8 M 8  Tiempo:0.11096
Node:34 <== JOIN_REQ: 65490 28 47 M 47  Tiempo:0.11096
Node:44 <== JOIN_REQ: 65490 28 12 M 12  Tiempo:0.11096
Node:44 ==>JOIN_REQ: 65490 28 44 M 44  Tiempo:0.11096
Node:33 <== JOIN_REQ: 65490 28 11 M 11  Tiempo:0.11096
Node:33 ==>JOIN_REQ: 65490 28 33 M 33  Tiempo:0.11096
Node:22 <== JOIN_REQ: 65490 28 1 M 1  Tiempo:0.11096
Node:22 ==>JOIN_REQ: 65490 28 22 M 22  Tiempo:0.11096
Node:4 <== JOIN_REQ: 65490 28 7 M 7  Tiempo:0.11096
Node:4 ==>JOIN_REQ: 65490 28 4 M 4  Tiempo:0.11096
Node:10 <== JOIN_REQ: 65490 28 30 M 30  Tiempo:0.11096
Node:10 ==>JOIN_REQ: 65490 28 10 M 10  Tiempo:0.11096
Node:43 <== JOIN_REQ: 65490 28 31 M 31  Tiempo:0.11096
```

```
Node:43 ==>JOIN_REQ: 65490 28 43 M 43  Tiempo:0.11096
Node:35 <== JOIN_REQ: 65490 28 34 M 34  Tiempo:0.11096
Node:35 ==>JOIN_REQ: 65490 28 35 M 35  Tiempo:0.11096
Node:44 <== JOIN_REQ: 65490 28 45 M 45  Tiempo:0.11096
Node:32 <== JOIN_REQ: 65490 28 8 M 8  Tiempo:0.12288
Node:32 ==>JOIN_REQ: 65490 28 32 M 32  Tiempo:0.12288
Node:33 <== JOIN_REQ: 65490 28 44 M 44  Tiempo:0.12288
Node:46 <== JOIN_REQ: 65490 28 33 M 33  Tiempo:0.12288
Node:46 ==>JOIN_REQ: 65490 28 46 M 46  Tiempo:0.12288
Node:16 <== JOIN_REQ: 65490 28 22 M 22  Tiempo:0.12288
Node:16 ==>JOIN_REQ: 65490 28 16 M 16  Tiempo:0.12288
Node:28 <== JOIN_REQ: 65490 28 4 M 4  Tiempo:0.12288
Node:28 ==>JOIN_ACK: 65490 28 M 28  Tiempo:0.12288
Node:13 <== JOIN_REQ: 65490 28 10 M 10  Tiempo:0.12288
Node:13 ==>JOIN_REQ: 65490 28 13 M 13  Tiempo:0.12288
Node:2 <== JOIN_REQ: 65490 28 43 M 43  Tiempo:0.12288
Node:2 ==>JOIN_REQ: 65490 28 2 M 2  Tiempo:0.12288
Node:26 <== JOIN_REQ: 65490 28 35 M 35  Tiempo:0.12288
Node:26 ==>JOIN_REQ: 65490 28 26 M 26  Tiempo:0.12288
Node:4 <== JOIN_ACK: 65490 28 M 28  Tiempo:0.13384000000000001
Node:4 ==>JOIN_ACK: 65490 28 M 4  Tiempo:0.13384000000000001
Node:4 <== JOIN_REQ: 65490 28 32 M 32  Tiempo:0.1348
Node:4 ==>JOIN_ACK: 65490 28 M 4  Tiempo:0.1348
Node:42 <== JOIN_REQ: 65490 28 46 M 46  Tiempo:0.1348
Node:42 ==>JOIN_REQ: 65490 28 42 M 42  Tiempo:0.1348
Node:32 <== JOIN_REQ: 65490 28 16 M 16  Tiempo:0.1348
Node:3 <== JOIN_REQ: 65490 28 13 M 13  Tiempo:0.1348
Node:3 ==>JOIN_REQ: 65490 28 3 M 3  Tiempo:0.1348
Node:15 <== JOIN_REQ: 65490 28 2 M 2  Tiempo:0.1348
Node:15 ==>JOIN_REQ: 65490 28 15 M 15  Tiempo:0.1348
Node:24 <== JOIN_REQ: 65490 28 26 M 26  Tiempo:0.1348
Node:24 ==>JOIN_REQ: 65490 28 24 M 24  Tiempo:0.1348
Node:32 <== JOIN_ACK: 65490 28 M 4  Tiempo:0.14576
Node:32 ==>JOIN_ACK: 65490 28 M 32  Tiempo:0.14576
Node:32 ==>JOIN_ACK: 65490 28 M 32  Tiempo:0.14576
Node:7 <== JOIN_ACK: 65490 28 M 4  Tiempo:0.14576
Node:16 <== JOIN_REQ: 65490 28 42 M 42  Tiempo:0.14671999999999999
Node:4 <== JOIN_REQ: 65490 28 3 M 3  Tiempo:0.14671999999999999
Node:4 ==>JOIN_ACK: 65490 28 M 4  Tiempo:0.14671999999999999
Node:10 <== JOIN_REQ: 65490 28 15 M 15  Tiempo:0.14671999999999999
Node:17 <== JOIN_REQ: 65490 28 24 M 24  Tiempo:0.14671999999999999
Node:17 ==>JOIN_REQ: 65490 28 17 M 17  Tiempo:0.14671999999999999
Node:3 <== JOIN_ACK: 65490 28 M 4  Tiempo:0.15767999999999999
Node:3 ==>JOIN_ACK: 65490 28 M 3  Tiempo:0.15767999999999999
Node:8 <== JOIN_ACK: 65490 28 M 32  Tiempo:0.15767999999999999
Node:8 ==>JOIN_ACK: 65490 28 M 8  Tiempo:0.15767999999999999
Node:16 <== JOIN_ACK: 65490 28 M 32  Tiempo:0.15767999999999999
Node:16 ==>JOIN_ACK: 65490 28 M 16  Tiempo:0.15767999999999999
Node:16 ==>JOIN_ACK: 65490 28 M 16  Tiempo:0.15767999999999999
Node:3 <== JOIN_REQ: 65490 28 17 M 17  Tiempo:0.15863999999999998
Node:3 ==>JOIN_ACK: 65490 28 M 3  Tiempo:0.15863999999999998
Node:17 <== JOIN_ACK: 65490 28 M 3  Tiempo:0.16959999999999997
Node:17 ==>JOIN_ACK: 65490 28 M 17  Tiempo:0.16959999999999997
Node:13 <== JOIN_ACK: 65490 28 M 3  Tiempo:0.16959999999999997
Node:13 ==>JOIN_ACK: 65490 28 M 13  Tiempo:0.16959999999999997
Node:25 <== JOIN_ACK: 65490 28 M 8  Tiempo:0.16959999999999997
Node:22 <== JOIN_ACK: 65490 28 M 16  Tiempo:0.16959999999999997
Node:22 ==>JOIN_ACK: 65490 28 M 22  Tiempo:0.16959999999999997
Node:42 <== JOIN_ACK: 65490 28 M 16  Tiempo:0.16959999999999997
Node:42 ==>JOIN_ACK: 65490 28 M 42  Tiempo:0.16959999999999997
Node:24 <== JOIN_ACK: 65490 28 M 17  Tiempo:0.18151999999999996
```

```
Node:24 ==>JOIN_ACK: 65490 28 M 24  Tiempo:0.18151999999999996
Node:10 <== JOIN_ACK: 65490 28 M 13  Tiempo:0.18151999999999996
Node:10 ==>JOIN_ACK: 65490 28 M 10  Tiempo:0.18151999999999996
Node:10 ==>JOIN_ACK: 65490 28 M 10  Tiempo:0.18151999999999996
Node:1 <== JOIN_ACK: 65490 28 M 22  Tiempo:0.18151999999999996
Node:46 <== JOIN_ACK: 65490 28 M 42  Tiempo:0.18151999999999996
Node:46 ==>JOIN_ACK: 65490 28 M 46  Tiempo:0.18151999999999996
Node:26 <== JOIN_ACK: 65490 28 M 24  Tiempo:0.19343999999999995
Node:26 ==>JOIN_ACK: 65490 28 M 26  Tiempo:0.19343999999999995
Node:15 <== JOIN_ACK: 65490 28 M 10  Tiempo:0.19343999999999995
Node:15 ==>JOIN_ACK: 65490 28 M 15  Tiempo:0.19343999999999995
Node:30 <== JOIN_ACK: 65490 28 M 10  Tiempo:0.19343999999999995
Node:33 <== JOIN_ACK: 65490 28 M 46  Tiempo:0.19343999999999995
Node:33 ==>JOIN_ACK: 65490 28 M 33  Tiempo:0.19343999999999995
Node:33 ==>JOIN_ACK: 65490 28 M 33  Tiempo:0.19343999999999995
Node:35 <== JOIN_ACK: 65490 28 M 26  Tiempo:0.20535999999999993
Node:35 ==>JOIN_ACK: 65490 28 M 35  Tiempo:0.20535999999999993
Node:2 <== JOIN_ACK: 65490 28 M 15  Tiempo:0.20535999999999993
Node:2 ==>JOIN_ACK: 65490 28 M 2  Tiempo:0.20535999999999993
Node:11 <== JOIN_ACK: 65490 28 M 33  Tiempo:0.20535999999999993
Node:44 <== JOIN_ACK: 65490 28 M 33  Tiempo:0.20535999999999993
Node:44 ==>JOIN_ACK: 65490 28 M 44  Tiempo:0.20535999999999993
Node:44 ==>JOIN_ACK: 65490 28 M 44  Tiempo:0.20535999999999993
Node:34 <== JOIN_ACK: 65490 28 M 35  Tiempo:0.21727999999999992
Node:34 ==>JOIN_ACK: 65490 28 M 34  Tiempo:0.21727999999999992
Node:43 <== JOIN_ACK: 65490 28 M 2  Tiempo:0.21727999999999992
Node:43 ==>JOIN_ACK: 65490 28 M 43  Tiempo:0.21727999999999992
Node:12 <== JOIN_ACK: 65490 28 M 44  Tiempo:0.21727999999999992
Node:45 <== JOIN_ACK: 65490 28 M 44  Tiempo:0.21727999999999992
Node:47 <== JOIN_ACK: 65490 28 M 34  Tiempo:0.2291999999999999
Node:31 <== JOIN_ACK: 65490 28 M 43  Tiempo:0.2291999999999999
Nodo: 28, child: 4
Nodo: 4, child: 7 32 3
Nodo: 7, child:
Nodo: 32, child: 8 16
Nodo: 3, child: 13 17
Nodo: 8, child: 25
Nodo: 16, child: 22 42
Nodo: 13, child: 10
Nodo: 17, child: 24
Nodo: 25, child:
Nodo: 22, child: 1
Nodo: 42, child: 46
Nodo: 10, child: 30 15
Nodo: 24, child: 26
Nodo: 1, child:
Nodo: 46, child: 33
Nodo: 30, child:
Nodo: 15, child: 2
Nodo: 26, child: 35
Nodo: 33, child: 11 44
Nodo: 2, child: 43
Nodo: 35, child: 34
Nodo: 11, child:
Nodo: 44, child: 12 45
Nodo: 43, child: 31
Nodo: 34, child: 47
Nodo: 12, child:
Nodo: 45, child:
Nodo: 31, child:
Nodo: 47, child:
```

C.3 Corrida provocando falla de un vínculo (entre los nodos 3 y 4)

```
#Prueba para determinar el costo de la caida de un vinculo
# - medido en tiempo necesario para restaurar el arbol
# - medido en costo de transmision (cantidad de mensajes)

#Caso particular de la corrida: Core: Nodo 28
#                               Vinculo en falla: vinculo entre nodos
3-4

#Se prueba con N=10, estando estos nodos distribuidos en la red, con
una
#densidad media
#Nodos: 1 - 7 - 11 - 12 - 25 - 30 - 31 - 34 - 45 - 47

set ns [new Simulator]

#Archivo de salida para las aplicaciones que reciben paquetes
multicast
set dels [open delaySG w]

Simulator set EnableMcast_ 1
Simulator set NumberInterfaces_ 1
source cbtmain.tcl

#Ruteo dinamico en la totalidad de los nodos
$ns rtproto Session

#Se genera nodo n(0) que no se utiliza, para que los id de los nodos
#coincidan con su subindice
set n(0) [$ns node]

#NODOS
puts "Generando nodos . . ."
for {set i 1} {$i <= 47} {incr i 1} {
   set n($i) [$ns node]
}

#TOPOLOGIA
puts "Generando links .  . . "
$ns duplex-link $n(1) $n(22) 1.5Mb 10ms DropTail
$ns duplex-link $n(1) $n(29) 1.5Mb 10ms DropTail
$ns duplex-link $n(1) $n(42) 1.5Mb 10ms DropTail
$ns duplex-link $n(2) $n(43) 1.5Mb 10ms DropTail
$ns duplex-link $n(2) $n(14) 1.5Mb 10ms DropTail
$ns duplex-link $n(2) $n(15) 1.5Mb 10ms DropTail
$ns duplex-link $n(3) $n(17) 1.5Mb 10ms DropTail
$ns duplex-link $n(3) $n(4) 1.5Mb 10ms DropTail
$ns duplex-link $n(3) $n(13) 1.5Mb 10ms DropTail
$ns duplex-link $n(3) $n(32) 1.5Mb 10ms DropTail
$ns duplex-link $n(4) $n(32) 1.5Mb 10ms DropTail
$ns duplex-link $n(4) $n(7) 1.5Mb 10ms DropTail
$ns duplex-link $n(4) $n(28) 1.5Mb 10ms DropTail
$ns duplex-link $n(5) $n(21) 1.5Mb 10ms DropTail
$ns duplex-link $n(5) $n(47) 1.5Mb 10ms DropTail
$ns duplex-link $n(5) $n(34) 1.5Mb 10ms DropTail
$ns duplex-link $n(5) $n(45) 1.5Mb 10ms DropTail
$ns duplex-link $n(6) $n(15) 1.5Mb 10ms DropTail
```

```
$ns duplex-link $n(6) $n(39) 1.5Mb 10ms DropTail
$ns duplex-link $n(8) $n(17) 1.5Mb 10ms DropTail
$ns duplex-link $n(8) $n(25) 1.5Mb 10ms DropTail
$ns duplex-link $n(8) $n(41) 1.5Mb 10ms DropTail
$ns duplex-link $n(8) $n(32) 1.5Mb 10ms DropTail
$ns duplex-link $n(9) $n(40) 1.5Mb 10ms DropTail
$ns duplex-link $n(9) $n(43) 1.5Mb 10ms DropTail
$ns duplex-link $n(10) $n(13) 1.5Mb 10ms DropTail
$ns duplex-link $n(10) $n(15) 1.5Mb 10ms DropTail
$ns duplex-link $n(10) $n(30) 1.5Mb 10ms DropTail
$ns duplex-link $n(11) $n(39) 1.5Mb 10ms DropTail
$ns duplex-link $n(11) $n(34) 1.5Mb 10ms DropTail
$ns duplex-link $n(11) $n(33) 1.5Mb 10ms DropTail
$ns duplex-link $n(12) $n(44) 1.5Mb 10ms DropTail
$ns duplex-link $n(14) $n(41) 1.5Mb 10ms DropTail
$ns duplex-link $n(14) $n(27) 1.5Mb 10ms DropTail
$ns duplex-link $n(15) $n(29) 1.5Mb 10ms DropTail
$ns duplex-link $n(16) $n(22) 1.5Mb 10ms DropTail
$ns duplex-link $n(16) $n(32) 1.5Mb 10ms DropTail
$ns duplex-link $n(16) $n(42) 1.5Mb 10ms DropTail
$ns duplex-link $n(16) $n(17) 1.5Mb 10ms DropTail
$ns duplex-link $n(17) $n(24) 1.5Mb 10ms DropTail
$ns duplex-link $n(18) $n(19) 1.5Mb 10ms DropTail
$ns duplex-link $n(18) $n(20) 1.5Mb 10ms DropTail
$ns duplex-link $n(18) $n(37) 1.5Mb 10ms DropTail
$ns duplex-link $n(19) $n(20) 1.5Mb 10ms DropTail
$ns duplex-link $n(19) $n(40) 1.5Mb 10ms DropTail
$ns duplex-link $n(20) $n(31) 1.5Mb 10ms DropTail
$ns duplex-link $n(21) $n(44) 1.5Mb 10ms DropTail
$ns duplex-link $n(22) $n(26) 1.5Mb 10ms DropTail
$ns duplex-link $n(22) $n(27) 1.5Mb 10ms DropTail
$ns duplex-link $n(23) $n(27) 1.5Mb 10ms DropTail
$ns duplex-link $n(24) $n(25) 1.5Mb 10ms DropTail
$ns duplex-link $n(24) $n(26) 1.5Mb 10ms DropTail
$ns duplex-link $n(25) $n(26) 1.5Mb 10ms DropTail
$ns duplex-link $n(26) $n(35) 1.5Mb 10ms DropTail
$ns duplex-link $n(30) $n(46) 1.5Mb 10ms DropTail
$ns duplex-link $n(31) $n(43) 1.5Mb 10ms DropTail
$ns duplex-link $n(33) $n(44) 1.5Mb 10ms DropTail
$ns duplex-link $n(33) $n(46) 1.5Mb 10ms DropTail
$ns duplex-link $n(34) $n(35) 1.5Mb 10ms DropTail
$ns duplex-link $n(34) $n(36) 1.5Mb 10ms DropTail
$ns duplex-link $n(34) $n(47) 1.5Mb 10ms DropTail
$ns duplex-link $n(35) $n(36) 1.5Mb 10ms DropTail
$ns duplex-link $n(35) $n(37) 1.5Mb 10ms DropTail
$ns duplex-link $n(37) $n(38) 1.5Mb 10ms DropTail
$ns duplex-link $n(38) $n(47) 1.5Mb 10ms DropTail
$ns duplex-link $n(39) $n(40) 1.5Mb 10ms DropTail
$ns duplex-link $n(42) $n(46) 1.5Mb 10ms DropTail
$ns duplex-link $n(44) $n(45) 1.5Mb 10ms DropTail



#Network Dynamics, evento de caida de link 3-4 configurado manualmente
$ns rtmodel-at 5.0 down  $n(3) $n(4)

#Trace solo para control
[$ns link $n(3) $n(4)] trace-dynamics $ns stdout

#Listado del arbol de distribucion en 3.0, con el arbol original y en
120.0
```

```
#con el nuevo arbol construido luego de la falla
$ns at 3.0 "$ns map-tree 65490"
$ns at 120.0 "$ns map-tree 65490"

#AGENTES CBT
puts "Generando agentes CBT . . . "
for {set i 1} {$i <= 47} {incr i 1} {
   set cbt($i) [new CBT $ns $n($i)]
   #Habilitacion de  mensajes por ciertas PDUs para controlar
   #datos de interes
   $cbt($i) enable_pdus JOIN_REQ
   $cbt($i) enable_pdus JOIN_ACK
   $cbt($i) enable_pdus QUIT_NOT
   $cbt($i) enable_pdus FLSH_TRE
}


#ARRANQUE DE AGENTES CBT
for {set i 1} {$i <= 47} {incr i 1} {
   $cbt($i) start
}

#Lista con los nodos del grupo
set lista {1 7 11 12 25 30 31 34 45 47 }
#Lista con la tasa de envio de cada nodo; en este caso es baja porque
no
#es de interes evaluar aspectos referidos al trafico multicast
set acti {1 1 1 1 1 1 1 1 1 1}

#CONFIGURACION DE CORE(S)

#UN UNICO CORE, 28
$cbt(28) set-core 65490

#Creacion de los agentes multicast de prueba, cada  emisor reside en
#uno de los nodos del grupo. Todos ellos envian al grupo
#Un receptor por nodo
set cnt 0
foreach i $lista {
   #emisor
   set  e($i) [new Agent/Message/Emisor_delay [lindex $acti $cnt]
$dels]
   $ns attach-agent $n($i) $e($i)
   $e($i) set dst_ 65490
   $e($i) set packetSize_ 200
   $ns at 118.0 "$e($i) start"
   $ns at 155.0 "$e($i) stop"
   #receptor
   set re($i) [new Agent/Message/Receptor_delay $dels]
   $ns attach-agent $n($i) $re($i)
        $ns at 0.1 "$re($i) join-group 65490"
   incr cnt 1
}

#Evento de fin
$ns at 130.0 "final"

proc final {} {
   global dels
   close $dels
   exit 0
```

```
}

#Inicio de la corrida
$ns run
```

C.4 Salida para determinación de costos y demoras

Los costos y las demoras fueron determinados a través del proceso de los archivos de salida con scripts awk. En este caso particular, se obtuvo un costo de 71 PDUs intercambiadas y una demora máxima de reconstrucción del nuevo subárbol de 0,3465 seg.

```
Generando nodos . . .
Generando links .  . .
Generando agentes CBT . . .
Node:45 ==>JOIN_REQ: 65490 28 45 M 45  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:47 ==>JOIN_REQ: 65490 28 47 M 47  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:34 ==>JOIN_REQ: 65490 28 34 M 34  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:31 ==>JOIN_REQ: 65490 28 31 M 31  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:30 ==>JOIN_REQ: 65490 28 30 M 30  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:7 ==>JOIN_REQ: 65490 28 7 M 7  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:1 ==>JOIN_REQ: 65490 28 1 M 1  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:11 ==>JOIN_REQ: 65490 28 11 M 11  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:12 ==>JOIN_REQ: 65490 28 12 M 12  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:25 ==>JOIN_REQ: 65490 28 25 M 25  Tiempo:0.10000000000000001
Grupo 65490 joined.
Grupo 65490, ya joined.
Node:44 <== JOIN_REQ: 65490 28 45 M 45  Tiempo:0.11096
Node:44 ==>JOIN_REQ: 65490 28 44 M 44  Tiempo:0.11096
Node:34 <== JOIN_REQ: 65490 28 47 M 47  Tiempo:0.11096
Node:8 <== JOIN_REQ: 65490 28 25 M 25  Tiempo:0.11096
Node:8 ==>JOIN_REQ: 65490 28 8 M 8  Tiempo:0.11096
Node:44 <== JOIN_REQ: 65490 28 12 M 12  Tiempo:0.11096
Node:33 <== JOIN_REQ: 65490 28 11 M 11  Tiempo:0.11096
Node:33 ==>JOIN_REQ: 65490 28 33 M 33  Tiempo:0.11096
Node:22 <== JOIN_REQ: 65490 28 1 M 1  Tiempo:0.11096
Node:22 ==>JOIN_REQ: 65490 28 22 M 22  Tiempo:0.11096
Node:4 <== JOIN_REQ: 65490 28 7 M 7  Tiempo:0.11096
Node:4 ==>JOIN_REQ: 65490 28 4 M 4  Tiempo:0.11096
Node:10 <== JOIN_REQ: 65490 28 30 M 30  Tiempo:0.11096
Node:10 ==>JOIN_REQ: 65490 28 10 M 10  Tiempo:0.11096
```

```
Node:43 <== JOIN_REQ: 65490 28 31 M 31  Tiempo:0.11096
Node:43 ==>JOIN_REQ: 65490 28 43 M 43  Tiempo:0.11096
Node:35 <== JOIN_REQ: 65490 28 34 M 34  Tiempo:0.11096
Node:35 ==>JOIN_REQ: 65490 28 35 M 35  Tiempo:0.11096
Node:33 <== JOIN_REQ: 65490 28 44 M 44  Tiempo:0.12288
Node:32 <== JOIN_REQ: 65490 28 8 M 8  Tiempo:0.12288
Node:32 ==>JOIN_REQ: 65490 28 32 M 32  Tiempo:0.12288
Node:46 <== JOIN_REQ: 65490 28 33 M 33  Tiempo:0.12288
Node:46 ==>JOIN_REQ: 65490 28 46 M 46  Tiempo:0.12288
Node:16 <== JOIN_REQ: 65490 28 22 M 22  Tiempo:0.12288
Node:16 ==>JOIN_REQ: 65490 28 16 M 16  Tiempo:0.12288
Node:28 <== JOIN_REQ: 65490 28 4 M 4  Tiempo:0.12288
Node:28 ==>JOIN_ACK: 65490 28 M 28  Tiempo:0.12288
Node:13 <== JOIN_REQ: 65490 28 10 M 10  Tiempo:0.12288
Node:13 ==>JOIN_REQ: 65490 28 13 M 13  Tiempo:0.12288
Node:2 <== JOIN_REQ: 65490 28 43 M 43  Tiempo:0.12288
Node:2 ==>JOIN_REQ: 65490 28 2 M 2  Tiempo:0.12288
Node:26 <== JOIN_REQ: 65490 28 35 M 35  Tiempo:0.12288
Node:26 ==>JOIN_REQ: 65490 28 26 M 26  Tiempo:0.12288
Node:4 <== JOIN_ACK: 65490 28 M 28  Tiempo:0.13384000000000001
Node:4 ==>JOIN_ACK: 65490 28 M 4  Tiempo:0.13384000000000001
Node:4 <== JOIN_REQ: 65490 28 32 M 32  Tiempo:0.1348
Node:4 ==>JOIN_ACK: 65490 28 M 4  Tiempo:0.1348
Node:42 <== JOIN_REQ: 65490 28 46 M 46  Tiempo:0.1348
Node:42 ==>JOIN_REQ: 65490 28 42 M 42  Tiempo:0.1348
Node:32 <== JOIN_REQ: 65490 28 16 M 16  Tiempo:0.1348
Node:3 <== JOIN_REQ: 65490 28 13 M 13  Tiempo:0.1348
Node:3 ==>JOIN_REQ: 65490 28 3 M 3  Tiempo:0.1348
Node:15 <== JOIN_REQ: 65490 28 2 M 2  Tiempo:0.1348
Node:15 ==>JOIN_REQ: 65490 28 15 M 15  Tiempo:0.1348
Node:24 <== JOIN_REQ: 65490 28 26 M 26  Tiempo:0.1348
Node:24 ==>JOIN_REQ: 65490 28 24 M 24  Tiempo:0.1348
Node:32 <== JOIN_ACK: 65490 28 M 4  Tiempo:0.14576
Node:32 ==>JOIN_ACK: 65490 28 M 32  Tiempo:0.14576
Node:32 ==>JOIN_ACK: 65490 28 M 32  Tiempo:0.14576
Node:7 <== JOIN_ACK: 65490 28 M 4  Tiempo:0.14576
Node:16 <== JOIN_REQ: 65490 28 42 M 42  Tiempo:0.14671999999999999
Node:10 <== JOIN_REQ: 65490 28 15 M 15  Tiempo:0.14671999999999999
Node:17 <== JOIN_REQ: 65490 28 24 M 24  Tiempo:0.14671999999999999
Node:17 ==>JOIN_REQ: 65490 28 17 M 17  Tiempo:0.14671999999999999
Node:4 <== JOIN_REQ: 65490 28 3 M 3  Tiempo:0.14671999999999999
Node:4 ==>JOIN_ACK: 65490 28 M 4  Tiempo:0.14671999999999999
Node:3 <== JOIN_ACK: 65490 28 M 4  Tiempo:0.15767999999999999
Node:3 ==>JOIN_ACK: 65490 28 M 3  Tiempo:0.15767999999999999
Node:8 <== JOIN_ACK: 65490 28 M 32  Tiempo:0.15767999999999999
Node:8 ==>JOIN_ACK: 65490 28 M 8  Tiempo:0.15767999999999999
Node:16 <== JOIN_ACK: 65490 28 M 32  Tiempo:0.15767999999999999
Node:16 ==>JOIN_ACK: 65490 28 M 16  Tiempo:0.15767999999999999
Node:16 ==>JOIN_ACK: 65490 28 M 16  Tiempo:0.15767999999999999
Node:3 <== JOIN_REQ: 65490 28 17 M 17  Tiempo:0.15863999999999998
Node:3 ==>JOIN_ACK: 65490 28 M 3  Tiempo:0.15863999999999998
Node:17 <== JOIN_ACK: 65490 28 M 3  Tiempo:0.16959999999999997
Node:17 ==>JOIN_ACK: 65490 28 M 17  Tiempo:0.16959999999999997
Node:13 <== JOIN_ACK: 65490 28 M 3  Tiempo:0.16959999999999997
Node:13 ==>JOIN_ACK: 65490 28 M 13  Tiempo:0.16959999999999997
Node:25 <== JOIN_ACK: 65490 28 M 8  Tiempo:0.16959999999999997
Node:22 <== JOIN_ACK: 65490 28 M 16  Tiempo:0.16959999999999997
Node:22 ==>JOIN_ACK: 65490 28 M 22  Tiempo:0.16959999999999997
Node:42 <== JOIN_ACK: 65490 28 M 16  Tiempo:0.16959999999999997
Node:42 ==>JOIN_ACK: 65490 28 M 42  Tiempo:0.16959999999999997
Node:24 <== JOIN_ACK: 65490 28 M 17  Tiempo:0.18151999999999996
```

```
Node:24 ==>JOIN_ACK: 65490 28 M 24  Tiempo:0.18151999999999996
Node:10 <== JOIN_ACK: 65490 28 M 13  Tiempo:0.18151999999999996
Node:10 ==>JOIN_ACK: 65490 28 M 10  Tiempo:0.18151999999999996
Node:10 ==>JOIN_ACK: 65490 28 M 10  Tiempo:0.18151999999999996
Node:1 <== JOIN_ACK: 65490 28 M 22  Tiempo:0.18151999999999996
Node:46 <== JOIN_ACK: 65490 28 M 42  Tiempo:0.18151999999999996
Node:46 ==>JOIN_ACK: 65490 28 M 46  Tiempo:0.18151999999999996
Node:26 <== JOIN_ACK: 65490 28 M 24  Tiempo:0.19343999999999995
Node:26 ==>JOIN_ACK: 65490 28 M 26  Tiempo:0.19343999999999995
Node:15 <== JOIN_ACK: 65490 28 M 10  Tiempo:0.19343999999999995
Node:15 ==>JOIN_ACK: 65490 28 M 15  Tiempo:0.19343999999999995
Node:30 <== JOIN_ACK: 65490 28 M 10  Tiempo:0.19343999999999995
Node:33 <== JOIN_ACK: 65490 28 M 46  Tiempo:0.19343999999999995
Node:33 ==>JOIN_ACK: 65490 28 M 33  Tiempo:0.19343999999999995
Node:33 ==>JOIN_ACK: 65490 28 M 33  Tiempo:0.19343999999999995
Node:35 <== JOIN_ACK: 65490 28 M 26  Tiempo:0.20535999999999993
Node:35 ==>JOIN_ACK: 65490 28 M 35  Tiempo:0.20535999999999993
Node:2 <== JOIN_ACK: 65490 28 M 15  Tiempo:0.20535999999999993
Node:2 ==>JOIN_ACK: 65490 28 M 2  Tiempo:0.20535999999999993
Node:11 <== JOIN_ACK: 65490 28 M 33  Tiempo:0.20535999999999993
Node:44 <== JOIN_ACK: 65490 28 M 33  Tiempo:0.20535999999999993
Node:44 ==>JOIN_ACK: 65490 28 M 44  Tiempo:0.20535999999999993
Node:44 ==>JOIN_ACK: 65490 28 M 44  Tiempo:0.20535999999999993
Node:34 <== JOIN_ACK: 65490 28 M 35  Tiempo:0.21727999999999992
Node:34 ==>JOIN_ACK: 65490 28 M 34  Tiempo:0.21727999999999992
Node:43 <== JOIN_ACK: 65490 28 M 2  Tiempo:0.21727999999999992
Node:43 ==>JOIN_ACK: 65490 28 M 43  Tiempo:0.21727999999999992
Node:12 <== JOIN_ACK: 65490 28 M 44  Tiempo:0.21727999999999992
Node:45 <== JOIN_ACK: 65490 28 M 44  Tiempo:0.21727999999999992
Node:47 <== JOIN_ACK: 65490 28 M 34  Tiempo:0.2291999999999999
Node:31 <== JOIN_ACK: 65490 28 M 43  Tiempo:0.2291999999999999
Nodo: 28, child: 4
Nodo: 4, child: 7 32 3
Nodo: 7, child:
Nodo: 32, child: 8 16
Nodo: 3, child: 13 17
Nodo: 8, child: 25
Nodo: 16, child: 22 42
Nodo: 13, child: 10
Nodo: 17, child: 24
Nodo: 25, child:
Nodo: 22, child: 1
Nodo: 42, child: 46
Nodo: 10, child: 30 15
Nodo: 24, child: 26
Nodo: 1, child:
Nodo: 46, child: 33
Nodo: 30, child:
Nodo: 15, child: 2
Nodo: 26, child: 35
Nodo: 33, child: 11 44
Nodo: 2, child: 43
Nodo: 35, child: 34
Nodo: 11, child:
Nodo: 44, child: 45 12
Nodo: 43, child: 31
Nodo: 34, child: 47
Nodo: 45, child:
Nodo: 12, child:
Nodo: 31, child:
Nodo: 47, child:
```

```
v 5 link-down 3 4
Node:3 ==>FLSH_TRE: -1 M 3  Tiempo:90.157700000000006
Node:3 ==>FLSH_TRE: -1 M 3  Tiempo:90.157700000000006
Node:3 ==>QUIT_NOT: 65490 3 M 3  Tiempo:90.157700000000006
Node:17 <== FLSH_TRE: -1 M 3  Tiempo:90.168660000000003
Node:17 ==>FLSH_TRE: -1 M 17  Tiempo:90.168660000000003
Node:17 ==>QUIT_NOT: 65490 17 M 17  Tiempo:90.168660000000003
Node:13 <== FLSH_TRE: -1 M 3  Tiempo:90.168660000000003
Node:13 ==>FLSH_TRE: -1 M 13  Tiempo:90.168660000000003
Node:13 ==>QUIT_NOT: 65490 13 M 13  Tiempo:90.168660000000003
Node:3 <== QUIT_NOT: 65490 17 M 17  Tiempo:90.17962
Node:3 <== QUIT_NOT: 65490 13 M 13  Tiempo:90.17962
Node:24 <== FLSH_TRE: -1 M 17  Tiempo:90.180580000000006
Node:24 ==>FLSH_TRE: -1 M 24  Tiempo:90.180580000000006
Node:24 ==>QUIT_NOT: 65490 24 M 24  Tiempo:90.180580000000006
Node:10 <== FLSH_TRE: -1 M 13  Tiempo:90.180580000000006
Node:10 ==>FLSH_TRE: -1 M 10  Tiempo:90.180580000000006
Node:10 ==>FLSH_TRE: -1 M 10  Tiempo:90.180580000000006
Node:10 ==>QUIT_NOT: 65490 10 M 10  Tiempo:90.180580000000006
Node:17 <== QUIT_NOT: 65490 24 M 24  Tiempo:90.191540000000003
Node:13 <== QUIT_NOT: 65490 10 M 10  Tiempo:90.191540000000003
Node:26 <== FLSH_TRE: -1 M 24  Tiempo:90.19250000000001
Node:26 ==>FLSH_TRE: -1 M 26  Tiempo:90.19250000000001
Node:26 ==>QUIT_NOT: 65490 26 M 26  Tiempo:90.19250000000001
Node:15 <== FLSH_TRE: -1 M 10  Tiempo:90.19250000000001
Node:15 ==>FLSH_TRE: -1 M 15  Tiempo:90.19250000000001
Node:15 ==>QUIT_NOT: 65490 15 M 15  Tiempo:90.19250000000001
Node:30 <== FLSH_TRE: -1 M 10  Tiempo:90.19250000000001
Node:30 ==>QUIT_NOT: 65490 30 M 30  Tiempo:90.19250000000001
Node:30 ==>JOIN_REQ: 65490 28 30 M 30  Tiempo:90.19250000000001
Node:24 <== QUIT_NOT: 65490 26 M 26  Tiempo:90.203460000000007
Node:10 <== QUIT_NOT: 65490 15 M 15  Tiempo:90.203460000000007
Node:10 <== QUIT_NOT: 65490 30 M 30  Tiempo:90.203460000000007
Node:35 <== FLSH_TRE: -1 M 26  Tiempo:90.204420000000013
Node:35 ==>FLSH_TRE: -1 M 35  Tiempo:90.204420000000013
Node:35 ==>QUIT_NOT: 65490 35 M 35  Tiempo:90.204420000000013
Node:2 <== FLSH_TRE: -1 M 15  Tiempo:90.204420000000013
Node:2 ==>FLSH_TRE: -1 M 2  Tiempo:90.204420000000013
Node:2 ==>QUIT_NOT: 65490 2 M 2  Tiempo:90.204420000000013
Node:10 <== JOIN_REQ: 65490 28 30 M 30  Tiempo:90.204420000000013
Node:10 ==>JOIN_REQ: 65490 28 10 M 10  Tiempo:90.204420000000013
Node:26 <== QUIT_NOT: 65490 35 M 35  Tiempo:90.21538000000001
Node:15 <== QUIT_NOT: 65490 2 M 2  Tiempo:90.21538000000001
Node:34 <== FLSH_TRE: -1 M 35  Tiempo:90.216340000000017
Node:34 ==>FLSH_TRE: -1 M 34  Tiempo:90.216340000000017
Node:34 ==>QUIT_NOT: 65490 34 M 34  Tiempo:90.216340000000017
Node:34 ==>JOIN_REQ: 65490 28 34 M 34  Tiempo:90.216340000000017
Node:43 <== FLSH_TRE: -1 M 2  Tiempo:90.216340000000017
Node:43 ==>FLSH_TRE: -1 M 43  Tiempo:90.216340000000017
Node:43 ==>QUIT_NOT: 65490 43 M 43  Tiempo:90.216340000000017
Node:13 <== JOIN_REQ: 65490 28 10 M 10  Tiempo:90.217300000000023
Node:13 ==>JOIN_REQ: 65490 28 13 M 13  Tiempo:90.217300000000023
Node:35 <== QUIT_NOT: 65490 34 M 34  Tiempo:90.227300000000014
Node:2 <== QUIT_NOT: 65490 43 M 43  Tiempo:90.227300000000014
Node:47 <== FLSH_TRE: -1 M 34  Tiempo:90.22826000000002
Node:47 ==>QUIT_NOT: 65490 47 M 47  Tiempo:90.22826000000002
Node:47 ==>JOIN_REQ: 65490 28 47 M 47  Tiempo:90.22826000000002
Node:35 <== JOIN_REQ: 65490 28 34 M 34  Tiempo:90.22826000000002
Node:35 ==>JOIN_REQ: 65490 28 35 M 35  Tiempo:90.22826000000002
Node:31 <== FLSH_TRE: -1 M 43  Tiempo:90.22826000000002
Node:31 ==>QUIT_NOT: 65490 31 M 31  Tiempo:90.22826000000002
```

```
Node:31 ==>JOIN_REQ: 65490 28 31 M 31  Tiempo:90.22826000000002
Node:3 <== JOIN_REQ: 65490 28 13 M 13  Tiempo:90.229220000000026
Node:3 ==>JOIN_REQ: 65490 28 3 M 3  Tiempo:90.229220000000026
Node:34 <== QUIT_NOT: 65490 47 M 47  Tiempo:90.239220000000017
Node:43 <== QUIT_NOT: 65490 31 M 31  Tiempo:90.239220000000017
Node:34 <== JOIN_REQ: 65490 28 47 M 47  Tiempo:90.240180000000024
Node:43 <== JOIN_REQ: 65490 28 31 M 31  Tiempo:90.240180000000024
Node:43 ==>JOIN_REQ: 65490 28 43 M 43  Tiempo:90.240180000000024
Node:26 <== JOIN_REQ: 65490 28 35 M 35  Tiempo:90.240180000000024
Node:26 ==>JOIN_REQ: 65490 28 26 M 26  Tiempo:90.240180000000024
Node:32 <== JOIN_REQ: 65490 28 3 M 3  Tiempo:90.24114000000003
Node:32 ==>JOIN_ACK: 65490 28 M 32  Tiempo:90.24114000000003
Node:3 <== JOIN_ACK: 65490 28 M 32  Tiempo:90.252100000000027
Node:3 ==>JOIN_ACK: 65490 28 M 3  Tiempo:90.252100000000027
Node:25 <== JOIN_REQ: 65490 28 26 M 26  Tiempo:90.252100000000027
Node:25 ==>JOIN_ACK: 65490 28 M 25  Tiempo:90.252100000000027
Node:2 <== JOIN_REQ: 65490 28 43 M 43  Tiempo:90.252100000000027
Node:2 ==>JOIN_REQ: 65490 28 2 M 2  Tiempo:90.252100000000027
Node:26 <== JOIN_ACK: 65490 28 M 25  Tiempo:90.263060000000024
Node:26 ==>JOIN_ACK: 65490 28 M 26  Tiempo:90.263060000000024
Node:13 <== JOIN_ACK: 65490 28 M 3  Tiempo:90.264020000000031
Node:13 ==>JOIN_ACK: 65490 28 M 13  Tiempo:90.264020000000031
Node:14 <== JOIN_REQ: 65490 28 2 M 2  Tiempo:90.264020000000031
Node:14 ==>JOIN_REQ: 65490 28 14 M 14  Tiempo:90.264020000000031
Node:35 <== JOIN_ACK: 65490 28 M 26  Tiempo:90.274980000000028
Node:35 ==>JOIN_ACK: 65490 28 M 35  Tiempo:90.274980000000028
Node:10 <== JOIN_ACK: 65490 28 M 13  Tiempo:90.275940000000034
Node:10 ==>JOIN_ACK: 65490 28 M 10  Tiempo:90.275940000000034
Node:41 <== JOIN_REQ: 65490 28 14 M 14  Tiempo:90.275940000000034
Node:41 ==>JOIN_REQ: 65490 28 41 M 41  Tiempo:90.275940000000034
Node:34 <== JOIN_ACK: 65490 28 M 35  Tiempo:90.286900000000031
Node:34 ==>JOIN_ACK: 65490 28 M 34  Tiempo:90.286900000000031
Node:30 <== JOIN_ACK: 65490 28 M 10  Tiempo:90.287860000000038
Node:8 <== JOIN_REQ: 65490 28 41 M 41  Tiempo:90.287860000000038
Node:8 ==>JOIN_ACK: 65490 28 M 8  Tiempo:90.287860000000038
Node:41 <== JOIN_ACK: 65490 28 M 8  Tiempo:90.298820000000035
Node:41 ==>JOIN_ACK: 65490 28 M 41  Tiempo:90.298820000000035
Node:47 <== JOIN_ACK: 65490 28 M 34  Tiempo:90.298820000000035
Node:14 <== JOIN_ACK: 65490 28 M 41  Tiempo:90.310740000000038
Node:14 ==>JOIN_ACK: 65490 28 M 14  Tiempo:90.310740000000038
Node:2 <== JOIN_ACK: 65490 28 M 14  Tiempo:90.322660000000042
Node:2 ==>JOIN_ACK: 65490 28 M 2  Tiempo:90.322660000000042
Node:43 <== JOIN_ACK: 65490 28 M 2  Tiempo:90.334580000000045
Node:43 ==>JOIN_ACK: 65490 28 M 43  Tiempo:90.334580000000045
Node:31 <== JOIN_ACK: 65490 28 M 43  Tiempo:90.346500000000049
Node:3 ==>QUIT_NOT: 65490 3 M 3  Tiempo:93.157700000000006
Node:17 ==>QUIT_NOT: 65490 17 M 17  Tiempo:93.168700000000001
Node:3 <== QUIT_NOT: 65490 17 M 17  Tiempo:93.179659999999998
Node:24 ==>QUIT_NOT: 65490 24 M 24  Tiempo:93.180599999999998
Node:17 <== QUIT_NOT: 65490 24 M 24  Tiempo:93.191559999999996
Node:26 ==>QUIT_NOT: 65490 26 M 26  Tiempo:93.192499999999995
Node:15 ==>QUIT_NOT: 65490 15 M 15  Tiempo:93.192499999999995
Node:24 <== QUIT_NOT: 65490 26 M 26  Tiempo:93.203459999999993
Node:10 <== QUIT_NOT: 65490 15 M 15  Tiempo:93.203459999999993
Node:2 ==>QUIT_NOT: 65490 2 M 2  Tiempo:93.204400000000007
Node:15 <== QUIT_NOT: 65490 2 M 2  Tiempo:93.215360000000004
Node:3 ==>QUIT_NOT: 65490 3 M 3  Tiempo:96.157700000000006
Node:17 ==>QUIT_NOT: 65490 17 M 17  Tiempo:96.168700000000001
Node:3 <== QUIT_NOT: 65490 17 M 17  Tiempo:96.179659999999998
Node:24 ==>QUIT_NOT: 65490 24 M 24  Tiempo:96.180599999999998
Node:17 <== QUIT_NOT: 65490 24 M 24  Tiempo:96.191559999999996
```

```
Node:26 ==>QUIT_NOT: 65490 26 M 26  Tiempo:96.192499999999995
Node:15 ==>QUIT_NOT: 65490 15 M 15  Tiempo:96.192499999999995
Node:24 <== QUIT_NOT: 65490 26 M 26  Tiempo:96.203459999999993
Node:10 <== QUIT_NOT: 65490 15 M 15  Tiempo:96.203459999999993
Node:2 ==>QUIT_NOT: 65490 2 M 2  Tiempo:96.204400000000007
Node:15 <== QUIT_NOT: 65490 2 M 2  Tiempo:96.215360000000004
Node:3 ==>QUIT_NOT: 65490 3 M 3  Tiempo:99.157700000000006
Node:17 ==>QUIT_NOT: 65490 17 M 17  Tiempo:99.168700000000001
Node:3 <== QUIT_NOT: 65490 17 M 17  Tiempo:99.179659999999998
Node:24 ==>QUIT_NOT: 65490 24 M 24  Tiempo:99.180599999999998
Node:17 <== QUIT_NOT: 65490 24 M 24  Tiempo:99.191559999999996
Node:26 ==>QUIT_NOT: 65490 26 M 26  Tiempo:99.192499999999995
Node:15 ==>QUIT_NOT: 65490 15 M 15  Tiempo:99.192499999999995
Node:24 <== QUIT_NOT: 65490 26 M 26  Tiempo:99.203459999999993
Node:10 <== QUIT_NOT: 65490 15 M 15  Tiempo:99.203459999999993
Node:2 ==>QUIT_NOT: 65490 2 M 2  Tiempo:99.204400000000007
Node:15 <== QUIT_NOT: 65490 2 M 2  Tiempo:99.215360000000004
Node 28  drop.  replicator at 118.02213333333333: _o2204 source: 7
dest: 65490
Node 28  drop.  replicator at 118.04426666666666: _o2252 source: 25
dest: 65490
Node 28  drop.  replicator at 118.05533333333332: _o2273 source: 1
dest: 65490
Node 28  drop.  replicator at 118.06639999999999: _o2307 source: 30
dest: 65490
Node 28  drop.  replicator at 118.07746666666665: _o2325 source: 11
dest: 65490
Node 28  drop.  replicator at 118.07853333333333: _o2347 source: 34
dest: 65490
Node 28  drop.  replicator at 118.08853333333332: _o2376 source: 12
dest: 65490
Node 28  drop.  replicator at 118.08959999999999: _o2392 source: 31
dest: 65490
Node 28  drop.  replicator at 118.09066666666666: _o2399 source: 47
dest: 65490
Node 28  drop.  replicator at 118.09173333333334: _o2402 source: 45
dest: 65490
Nodo: 28, child: 4
Nodo: 4, child: 7 32 3
Nodo: 7, child:
Nodo: 32, child: 8 16 3
Nodo: 3, child: 13
Nodo: 8, child: 25 41
Nodo: 16, child: 22 42
Nodo: 13, child: 10
Nodo: 25, child: 26
Nodo: 41, child: 14
Nodo: 22, child: 1
Nodo: 42, child: 46
Nodo: 10, child: 30
Nodo: 26, child: 35
Nodo: 14, child: 2
Nodo: 1, child:
Nodo: 46, child: 33
Nodo: 30, child:
Nodo: 35, child: 34
Nodo: 2, child: 43
Nodo: 33, child: 11 44
Nodo: 34, child: 47
Nodo: 43, child: 31
Nodo: 11, child:
```

```
Nodo: 44, child: 45 12
Nodo: 47, child:
Nodo: 31, child:
Nodo: 45, child:
Nodo: 12, child:
```

C.5 Resultados obtenidos: costos y demoras máximas promedio

Para todos los nodos actuando como cores, agrupados por número de vínculos pertenecientes al subárbol de distribución desconectado

#vínculos →	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Demora	**0,2398**	**0,2675**	**0,2820**	**0,2947**	**0,3036**	**0,3105**	**0,3250**	**0,3112**
Costo	**8,77**	**14,58**	**19,97**	**28,32**	**38,30**	**50,03**	**67,50**	**88,18**

Tabla C.1. Costos y demoras máximas promedio agrupados por cantidad de vínculos del árbol de distribución conectado

C.6 Resultados obtenidos: costos y demoras agrupadas por core

Core	Costo máx	Costo mín	Costo prom	Dem. Máx.	Dem. Mín.	Dem. Prom
1	57	6	21,71	0,3345	0,2044	0,2699
2	92	8	27,52	0,3226	0,2391	0,2864
3	82	6	22,46	0,3365	0,1915	0,2669
4	71	6	24,03	0,3465	0,1915	0,2879
5	76	6	24,14	0,3345	0,1438	0,2763
6	103	8	24,40	0,3107	0,2153	0,2631
7	71	6	24,03	0,3584	0,2153	0,3054
8	61	6	20,77	0,3703	0,2034	0,2617
9	95	8	31,81	0,3465	0,2154	0,2983
10	62	8	20,90	0,3107	0,2153	0,2594
11	77	8	23,10	0,3346	0,1795	0,2755
12	87	6	24,45	0,3346	0,1677	0,2841
13	93	6	23,68	0,3355	0,2034	0,2522
14	71	6	24,12	0,3246	0,1915	0,2692
15	45	8	20,72	0,2997	0,2154	0,2578
16	58	6	19,95	0,3345	0,1677	0,2594
17	55	6	20,42	0,3107	0,1677	0,2570
18	89	6	26,92	0,3465	0,1915	0,2822
19	89	6	27,80	0,3703	0,1914	0,2853
20	98	6	27,51	0,3703	0,2153	0,3004
21	73	6	25,66	0,3355	0,1677	0,2867
22	50	6	20,38	0,3345	0,1915	0,2613
23	134	6	27,76	0,3464	0,2272	0,2816
24	75	6	22,92	0,3465	0,1438	0,2725
25	78	6	24,37	0,3584	0,2034	0,2902
26	82	6	22,76	0,3107	0,1676	0,2634
27	113	6	24,04	0,3345	0,2034	0,2699
28	71	6	28,48	0,3465	0,1915	0,2857
29	103	6	22,50	0,2988	0,1915	0,2532
30	77	8	24,40	0,3584	0,2153	0,2857
31	96	6	26,79	0,3474	0,2392	0,3062
32	58	6	20,40	0,3107	0,1915	0,2566
33	74	8	22,95	0,3117	0,1915	0,2646
34	79	6	21,90	0,3345	0,1438	0,2676
35	54	6	20,28	0,3107	0,1557	0,2511
36	54	6	21,54	0,3345	0,1557	0,2652
37	84	6	23,18	0,3593	0,1677	0,2616
38	57	8	23,27	0,3584	0,1677	0,2820
39	76	8	24,81	0,3703	0,1915	0,2763
40	113	8	26,92	0,3942	0,2153	0,2984
41	101	6	24,74	0,4061	0,2153	0,2833
42	60	8	21,90	0,3346	0,1915	0,2002
43	81	8	29,09	0,3355	0,1915	0,3038
44	87	6	24,18	0,3226	0,1915	0,2782
45	65	6	24,57	0,3354	0,1677	0,2927
46	56	8	23,40	0,3345	0,2034	0,2735
47	61	6	23,90	0,3346	0,1677	0,2830

Tabla C.2. Costos y demoras agrupados por core