UNIVERSIDAD NACIONAL DE LA PLATA
FACULTAD DE CIENCIAS EXACTAS
DEPARTAMENTO DE INFORMÁTICA

TRABAJO DE GRADO

Simulación del protocolo de ruteo OSPF

Directores:
Ing. Luis Armando Marrone
Lic. Francisco Javier Díaz

Carina Cecilia Correa
Axel Comper

1997

Agradecemos a:

Nuestros padres.
Cecilia y Fernando.
Miguel Luengo.
Cristian Aldama.

En el día de la fecha, 10 de noviembre de 1998, se reúne la Comisión Evaluadora y presencia la exposición del trabajo. Considerando que corresponde aprobar el mismo con nota 10 (diez).

P. Borzi

H. Villagarcia

Contenidos.

Capítulo 1

Introducción

Para llegar a nuestro objetivo, la simulación del protocolo de ruteo OSPF, incursionamos en distintas etapas.

Iniciamos con un análisis de la simulación en general, detallando en particular el caso específico de simular una red de computadoras. Para ello utilizamos la información encontrada en Internet.

La búsqueda de un simulador existente en el ámbito académico nos permitió conocer varios simuladores, como el MaRS, Real, Atlas, NS, todos recolectados a través de Internet y elegimos el simulador de ruteo de la Universidad de Maryland, llamado MaRS, por ser el más completo, flexible y con mayor documentación encontrado.
El MaRS de acuerdo al análisis conceptual es un simulador orientado a eventos o sucesos con avance de tiempo a intervalos fijos que simula algoritmos de ruteo, no protocolos. Por lo tanto, el agregado de un nuevo protocolo sería un objetivo aún mayor.

Siguiendo con el análisis teórico del tema, derivamos en los distintos protocolos de ruteo (OSPF, RIP, IS-IS, IGRP), eligiendo al protocolo OSPF porque es uno de los más estándares ya que ha desplazado en el mercado a RIP, interesante y práctico para simular.

Una vez concretado el tema debimos analizar en detalle el funcionamiento del MaRS, no solo en el aspecto del ruteo, sino en el manejo de cualquier evento en particular.

Por otro lado se tomaron distintas muestras del comportamiento del OSPF en un PC-Router ante distintas circunstancias (fallas y reparaciones de enlaces, tráfico normal) para su análisis y así alcanzar un conocimiento más detallado del funcionamiento del protocolo.

Con toda esta información se implementó el OSPF en el MaRS creando una nueva componente de ruteo en el simulador, teniendo en cuenta no alterar el funcionamiento de las componentes ya existentes, ni el comportamiento del resto de los algortimos. Se reprodujeron las mismas circunstancias que cuando se tomaron las muestras y se analizaron los resultados para comprobar su comportamiento con el real.

Al final el MaRS nos permitió simular el protocolo OSPF, volcar sobre él distintas configuraciones tomadas de la realidad y poder así analizar nada menos que su comportamiento, tomar muestras de distintas situaciones y compararlas con el escenario real.

Capítulo 2

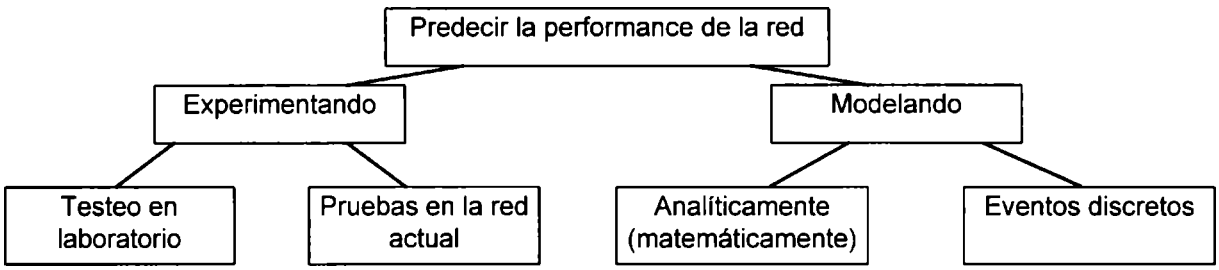
Conceptos de Simulación.

2.1. Simulación.

Para describir adecuadamente las operaciones en distintas condiciones de una red hay que desarrollar un programa de simulación basado en un modelo definido.

Se puede predecir la perfomance de una red:

- **Experimentando:** se construye un testeador que controla la perfomance de la red o bien se investiga con nuevas tecnologías y aplicaciones sobre una red existente.
- **Modelando:**
  - Un **modelo analítico** de ecuaciones que describen la perfomance y solucionan problemas de latencia y utilización. Útil para el planeamiento de una red en la toma de decisiones o configuraciones (ancho de banda, espacio para almacenar tráfico, etc.).
  - Un **modelo de eventos discretos** representa al tráfico como una secuencia de mensajes, paquetes, o frames, vía un reloj centralizado se envía tráfico en ciertos momentos a distintos puntos de la red, generándolo en tiempos random y los recursos de la red como por ejemplo: bridge, routers, servers, switches, simulan encolar y procesar paquetes



La modelización es la característica más importante de la simulación. Se deben definir:

- **Topología:** se puede detallar servidores, estaciones de trabajo, dispositivos de interconexión (bridges, routers), plataformas (Ethernet, Token Ring, etc.).
- **Tráfico:** se detallan los elementos que lo generan, como los protocolos de ruteo, políticas para manejar las solicitudes de los clientes y las respuestas de los servidores, como también la forma de trabajar de las aplicaciones.

[SCHA,1996]

2.2. Utilidad de la simulación.

En muchos casos es importante predecir el comportamiento de una red:

- si se tiene pensado comprar un nuevo servidor con mayor tiempo de respuesta.
- por el agregado de estaciones de trabajo.
- por la forma de segmentación de la red.
- probar protocolos de ruteo para optimizar la red, etc.

En todos estos casos la respuesta se obtiene de:

- experiencias anteriores, propias o ajenas, que seguramente serán diferentes.
- por consejo de algún proveedor, con compromisos comerciales.
- por propia intuición.

En estos casos hay mucha inversión de tiempo y dinero, por lo tanto no hay mucho margen para prueba y error.

La simulación debe permitir establecer con el menor error posible las condiciones de trabajo en una red, real o proyectada. [SCHA,1996]

2.3. Simulación segura.

Se deberán considerar estos conceptos:

- Confrontar el modelo contra la red. No importa cuan seguro sea el simulador si el tráfico y la topología no representa a la red. La simulación siempre es una aproximación por lo que se debe determinar que detalles son suficientes para validar o refutar los resultados producto de la simulación.
- Capturar tráfico. Se puede tener la posibilidad de recolectar datos estadísticos del tráfico usual de la red. Esta información será utilizada por la simulación para generar un tráfico similar.
- Hacer una extensa prueba de simulación. Es un grave error no probar lo suficientemente la simulación, por lo que los resultados no son seguros. Está suficientemente probado cuando la simulación llega a un estado seguro y se simule un tráfico representativo luego de alcanzar ese estado. Alcanza un estado seguro cuando las colas llegan a un punto operacional normal.
- Simular distintos escenarios. Ya que la actividad del usuario es azarosa probablemente se requiere modelar varios casos. Ejemplos: casos promedios, el peor de los casos, la carga mas pesada a mediados o fin de mes. Dependiendo de las funciones que debe soportar la red y el riesgo que puede tolerar.
- La limitación de recursos en la red debe ser bien modelada y de acuerdo a ese modelo se podrán obtener algunos resultados que en otros modelos no.[SCHA,1996]

2.4. Simulando una red.

2.4.1. Simulando el tráfico.

El tráfico es lo más difícil de simular por sus características aleatorias.
Hay varios principios:

- Para el caso de una red LAN el acceso a los recursos es básicamente secuencial, por lo tanto se puede analizar como una cola. En estos casos alguien pide un servicio , si los servidores están ocupados esperan en una cola. Esto matemáticamente se resuelve utilizando la **teoría de colas**, por lo general con distribución de Poisson.
- En lo que respecta al tráfico de ráfagas se necesitan distintos modelos para diferentes ocasiones y existe una dependencia con el evento anterior, por lo que no respeta el proceso poissiano. Esto se puede procesar con una modulación **Markoviana**, que produce "nubes" dispersas aleatoriamente en el tiempo representando las ráfagas.
- En los casos mas complejos en el cual la matemática se vuelve imposible o no práctica, la simulación **Monte Carlo** permite una salida.

2.4.1.1. Teoría de colas.

El sistema de colas puede ser usado para el siguiente modelo: clientes que llegan, solicitan un servicio, son servidos y parten.

Se caracterizan por estos 5 componentes:

- Existe una función probabilística de tiempo de arribos que describe el intervalo entre arribos consecutivos.
- Existe una función probabilística de tiempos de servicio que varia de cliente en cliente, y que tipo de servicio sea.
- Existe una cantidad de servicios que pueden ser atendidos por una única cola, o bien varias colas independientes, o un sistema multiserver de colas.
- Existe una disciplina que determina el orden en el que los clientes serán atendidos (con o sin prioridades).
- Puede existir buffers con espacio limitados o no en las colas.

Ejemplificamos este modelo:

Un buffer infinito, con un simple servicio y la metodología FIFO.

Una notación que lo representa en la literatura de las colas es A/B/m donde A es una función sobre el tiempo de arribo, B es una función sobre el tiempo del servicio y m la cantidad de servicios. Las funciones probabilísticas (A y B) se pueden sacar de los siguientes conjuntos:

M = funciones exponenciales.

D = todos los clientes tienen el mismo valor determinístico.

G = general (funciones probabilísticas arbitrarias).

Aún no se conoce solución analítica para el modelo General (G/G/n), por lo que todas las pruebas se aplican sobre el modelo M/M/1, lo cual es razonable para cualquier sistema de clientes independientes asignar una función exponencial, donde la probabilidad de arribo para n clientes en un intervalo de tiempo t, esta dado por Poisson:

$$P_n(t) = \frac{(\lambda t)^n e^{-\lambda t}}{n!}$$

donde λ es el promedio de arribo.

Los resultados derivados de este modelo encaja perfectamente en el modelo de colas de paquetes en espera en una red. Las diferencias en una red son:

- Los canales de comunicación no son aislados, la salida de uno puede ser la entrada del otro. Por lo que una entrada no es mayor que un proceso Poisson pero las salidas de los servicios de la forma M/M/1 caen en la cola de entrada de otro servicio, por lo que resulta un proceso Poisson.
- El tamaño de los paquetes se mantiene durante la simulación, por lo que se soluciona perdiendo la identidad del paquete y cambiando su longitud al azar, esto se conoce como una "recepción independiente". [TANE,1996]

2.4.1.2. Simulación Monte Carlo.

Es una técnica de reducción. Reduce el problema original en subproblemas sobre una red mas pequeña representada por un grafo, construida en base a la original y condicionada por unos de los caminos disponibles del grafo. Consiste en repetir independientemente K veces el siguiente experimento:

Para cada link l, se ejecuta la prueba de Bernoulli con una distribución (r_l , $1-r_l$) que es un seudo número random generado. DFS es usado para determinar si el nodo s y t están conectados en el grafo parcial. [CANC,1995]

2.4.2. Simulando el ruteo.

El ruteo simulado debe garantizar: ser seguro por lo tanto debe estar libre de deadlock y en lo posible ser adaptativo y minimal, correcto, simple, robusto, estable, equitativo y optimo.

Se dividen en 2 grandes clases:

- Adaptativos: cambian las decisiones de ruteo frente a los cambios en la red. Este modela un patrón de comunicación no estructurado presente en muchas aplicaciones.
- No adaptativos: se basan en medidas o estimaciones del tráfico y de las topologías computarizadas anticipadamente, lo que hace un ruteo estático. [TANE,1996]

Algoritmos de ruteo adaptativos o dinámicos:

- El algoritmo de **hipercubo**, nivela cada nodo de la red representada en un grafo acíclico, con un peso determinado, y divide el ruteo de paquetes en 2 fases:
  La primera mueve el mensaje del nodo S a un nodo D vecino encontrado en un camino minimal pudiendo bajar o subir de nivel lo que representa movimientos estáticos o dinámicos respectivamente.
  Cuando no hay más pasos que incrementan el nivel empieza la segunda fase: en la que todos los pasos decrementan un nivel, por lo que son movimientos estáticos. [FELP,1991]
- La **grilla** nivela los nodos, en vez de pesarlos determina el nivel del nodo (x,y) como x+y.
  Asignan un movimiento de las siguientes formas:
  Por transposición: el nodo (x,y) le envía mensajes al nodo(y,x).
  Por complemento en la grilla de n x n ,el nodo (x,y) le envía al $(n-1-x,n-1-y)^2$ y en el hipercubo se aplica a la dirección binaria del nodo.
  Por permutaciones niveladas asignando un peso o nivel a los nodos.

Ambos son totalmente adapativos porque dependen de la congestión local de la red, minimales y libres de deadlock. [FELP,1991]

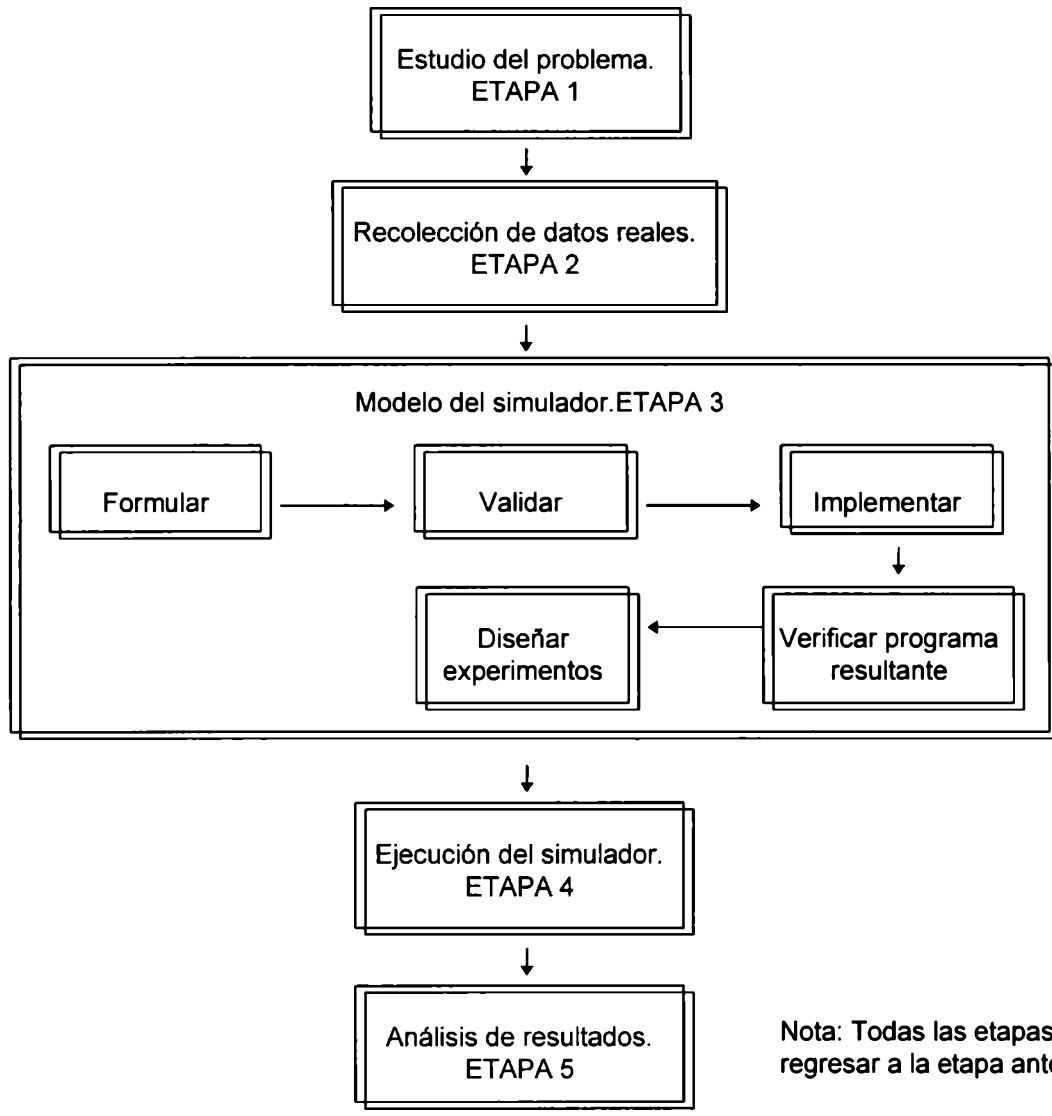
- **Vector de distancia**: Cada router se basa en una tabla con las mejores distancias a cada destino y la línea a usar, dicha tabla se actualiza intercambiando información con sus routers vecinos. Originalmente fue usado en ARPANET y es también usado ahora en Internet bajo el nombre de RIP. La métrica puede ser el número de saltos, el retardo en milisegundos, el número de paquetes encolados. [TANE,1996]
- **Estados de enlace**: Cada router debe descubrir sus vecinos y aprender su direccionamiento, medir para cada uno de ellos los retardos y costos, construir paquetes que lleven esta información, enviarlos a sus routers vecinos y computar el camino más corto para cada uno de ellos, usando Dijkstra por ejemplo. [TANE,1996]
- **Jerárquico**: Las tablas de ruteo crecen proporcionalmente a la red, por lo que no solo ocupan memoria sino también tiempo de búsqueda, este modelo es usado por lo general en redes telefónicas. El número óptimo de niveles para una red de N routers es ln(N) con un total de $e^{ln(N)}$ de entradas por router. [TANE,1996]
- **Hosts móviles**: Para enviar un paquete a un host móvil, primero se debe encontrar al host, que tiene una dirección propia. El mundo se divide en áreas, en las que hay **agentes externos**, que rastrean los usuarios móviles que visitan el área y **agentes propios** que rastrean los usuarios de esa área y están visitando otra, el host móvil envía un paquete a la dirección del host casa, este envía encapsulado el paquete al agente externo y su dirección al host móvil para iniciar una sesión de red entre ambos. [TANE,1996]
- **Broadcast**: Consiste en enviar paquetes simultáneamente a todos los destinos, solo los que estén interesados lo leerán. Existen varios métodos: enviar un **paquete distintivo** a cada destino por lo que requiere una lista de todos los destinos; **flooding** el cual comprende un significativo consumo de ancho de banda; **multidestino** que adiciona a cada paquete la lista de destinos por los que desea pasar y cuando llega al router este genera una copia del paquete y lo deriva a las líneas de salida, etc. [TANE,1996]

- **Multicast**: Consiste en enviar mensajes a un grupo de la red, por lo que se necesita crear y eliminar grupos lo cual no concierne al ruteo, pero si debe conocerlo propagando esta información a sus vecinos. [TANE,1996]

Algoritmos de ruteo no adaptativos o estáticos:

- **Camino más corto**: Usando Dijkstra computa el camino más corto en números de saltos o distancia geográfica. [TANE,1996]
- **Flooding**: Envía el paquete por cada línea existente, mediante un contador inicializado con la longitud del máximo camino a recorrer, el paquete será distribuido hasta llegar a destino o su contador llegue a 0. Se puede mejorar haciendo una selección de líneas a enviar. [TANE,1996]
- **Basado en flujo o tráfico**: Como se usa en redes estables y predecibles, se puede calcular el retardo de los paquetes, reduciendo el problema a encontrar un algoritmo que obtenga el retardo mínimo. [TANE,1996]

2.5. Diseño de simuladores.



Nota: Todas las etapas pueden regresar a la etapa anterior.

[VAZQ,1997]

2.6. Modelo de Simuladores.

Los modelos de simulación representan:

- los **estados** de los elementos que componen el sistema: procesos, recursos, colas, etc., detallando sus atributos y relaciones.
- los **sucesos** o **eventos** que alteran los estados del sistema a lo largo del tiempo.

Según la evolución temporal del sistema se pueden nombrar dos modelos:

- **El modelo continuo** resuelve el problema analíticamente variando el tiempo en forma casi continua.
- **El modelo discreto** varía los estados del sistema de acuerdo a un conjunto de relaciones lógicas, hay un número finito de cambios y ocurren en ciertos instantes de tiempo y no de forma continua.

Para los basados en el modelo discreto existen dos enfoques:

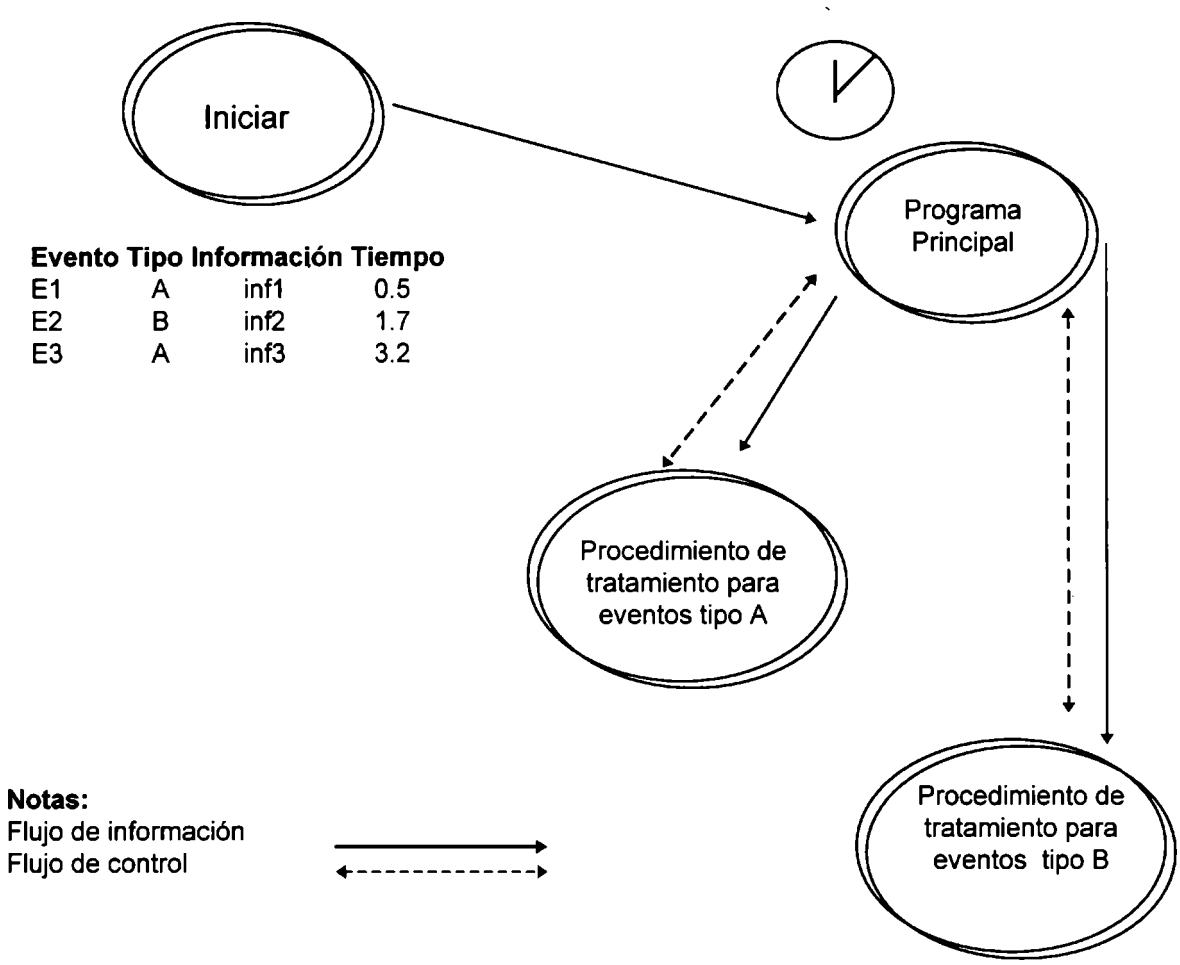
- **Orientado a eventos.**
  Enfoque de bajo nivel.
  Tiene un único elemento básico: el suceso o evento.
  Tiene un conjunto mínimo de primitivas.
- **Orientado a procesos.**
  Enfoque de mayor nivel de abstracción que el anterior.
  Tiene como elementos básicos a los procesos y los recursos.
  Tiene un conjunto de primitivas más rico. [VAZQ,1997]

2.6.1. Simulación orientada a eventos.

Cada evento elemental que pueda cambiar el estado del sistema se considera por separado y tiene asociado un procedimiento de tratamiento que agrupa todas las acciones posibles para tal evento.

Existe un programa principal que tiene una lista de eventos en orden cronológico y llama a los procedimientos de tratamiento asociados al evento para atenderlo. [VAZQ,1997]

Esquema de gestión de eventos.



2.6.2. Simulación orientada a procesos.

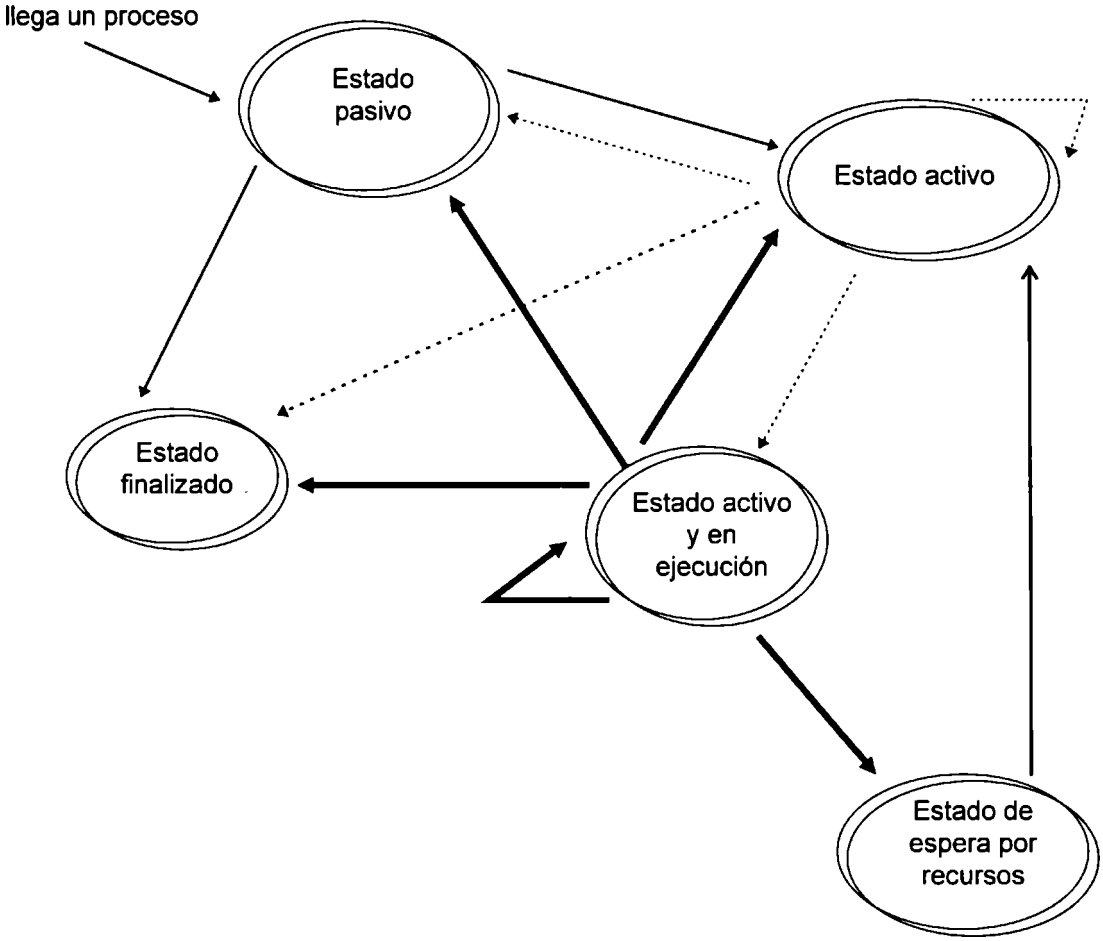
Los sucesos elementales se consideran como flujo de control de los componentes activos del sistema que son los procesos: proceso de llegada, de servicio ,de salida.

Existe un programa principal que coordina el avance de los diferentes procesos los que se van ejecutando sucesiva y no concurrentemente. Cada proceso que se está ejecutando puede pasar a los siguientes estados: finalizado, pasivo o activo, y en ese instante el programa principal decidirá que proceso ejecutar de la cola de activos.

Cuando ya no hay más procesos listos o activos, se avanza el tiempo simulado hasta el instante en que otro proceso esté activo.

Los recursos son elementos pasivos por los que compiten los procesos y tienen asociado una cola de procesos en espera para usarlo. [VAZQ,1997]

Esquema de gestión de procesos y recursos.



2.7.Implementación de Simuladores.

Independientemente del lenguaje en que se implemente un simulador éste se caracteriza por :

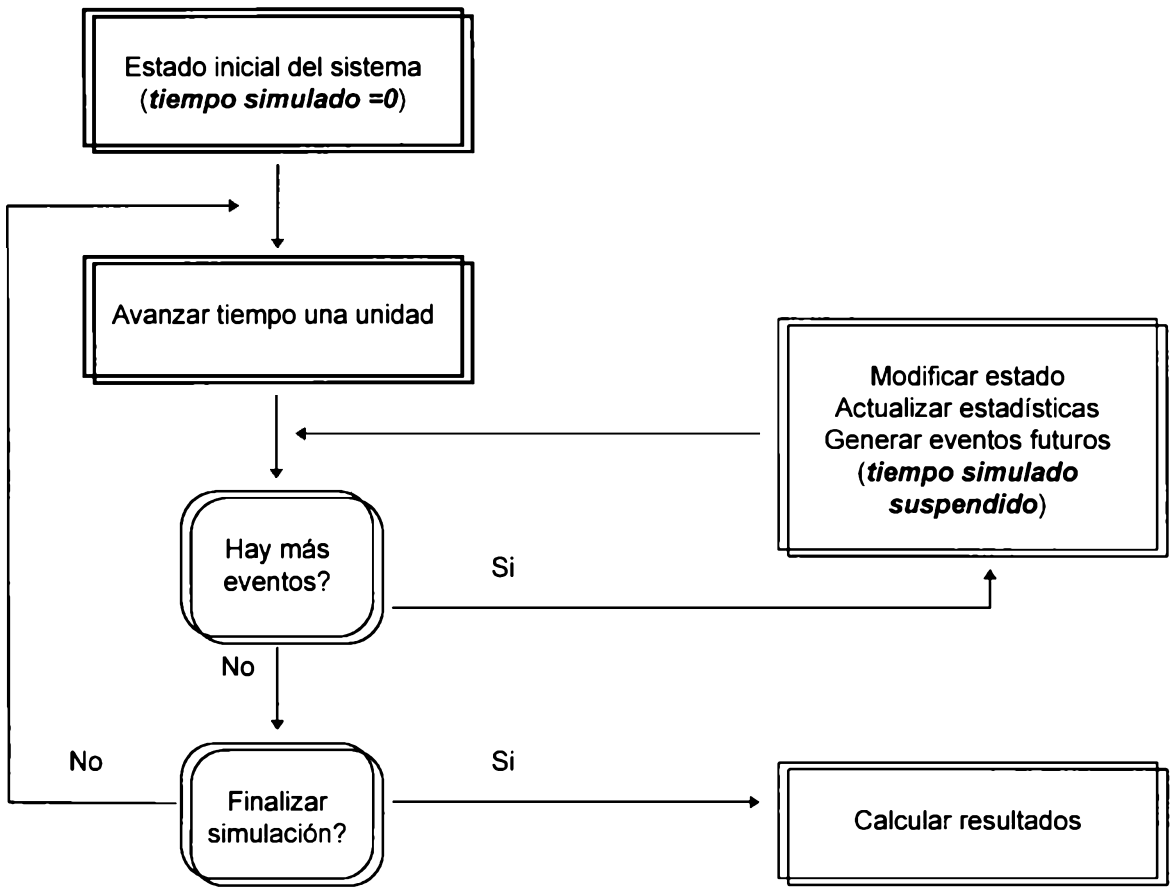
- Los objetos o elementos básicos que intervienen.
- Las primitivas u operaciones provistas por el lenguaje consideradas indivisibles y que manipulan los objetos anteriores.

Las opciones para simular son:

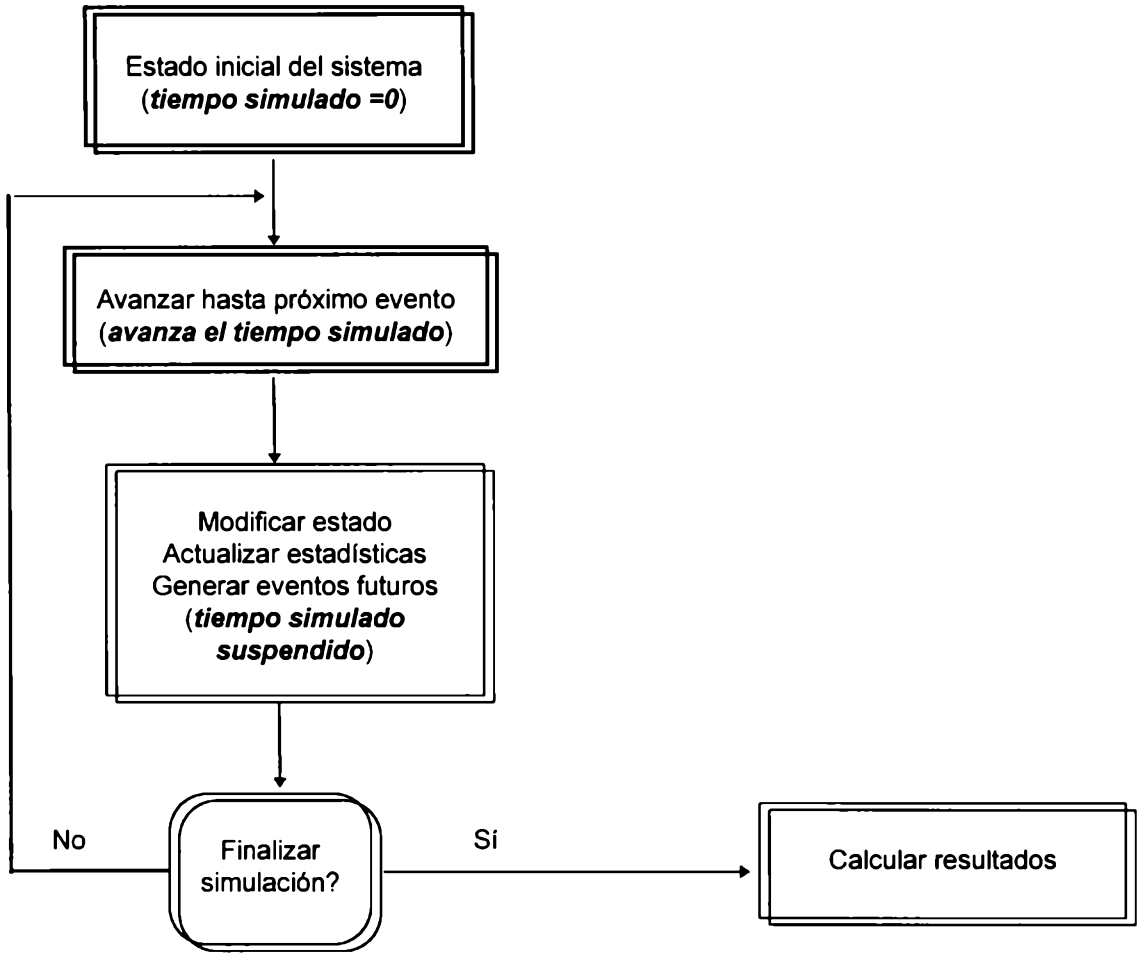
- Usar un paquete específico en el que se describe el modelo sin programar, pero resulta relativamente rígido. Ejemplos: Network, Comnet.
- Usar lenguajes de simulación en el que hay que programar el modelo ya que las funciones de simulación están hechas. Ejemplos: Simscript.
- Usar lenguajes de propósito general en el que hay que programar el modelo y las funciones de simulación. Ejemplo: C, Ada.
- Usar técnicas de descripción formal, que es un lenguaje para especificar, probar y evaluar. Ejemplo: Lotos, Estelle.

Para un simulador orientado a eventos se puede utilizar uno de estos dos esquemas:

- Esquema considerando avance de tiempo a intervalos fijos.



- Esquema considerando avance de tiempo evento a evento.



[VAZQ,1997]

Capítulo 3

MaRS (Maryland Routing Simulator).

3.1.Introducción.

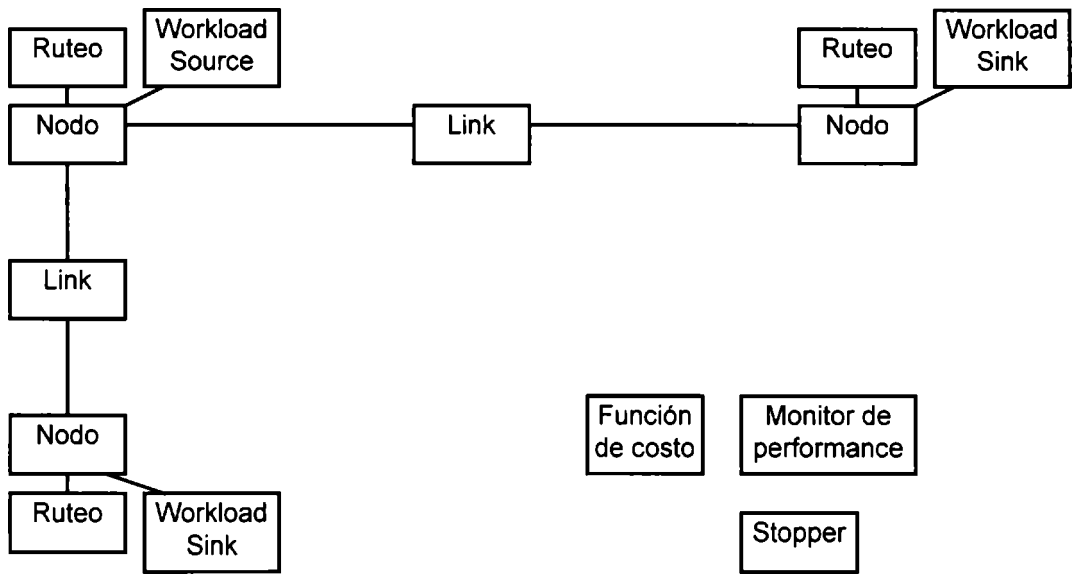
Esta diseñado para proveer una plataforma flexible para la evaluación y comparación de algoritmos de ruteo de redes.

Permite que el usuario defina una configuración de red, controle la simulación, mantenga los valores de los parámetros seleccionados, grabar, cargar y modificar configuraciones de red. [MARSU,1991]

3.2.Target system.

Es la red a simular. Se define por un conjunto de estados variables y de eventos. Cada evento tiene una rutina asociada que especifica los estados y su actualización. El tiempo de simulación es aquel en el que el target system ha sido simulado.

MaRS considera al target system una **red física**, un **algoritmo de ruteo** y un **workload**.



3.3. Red física.

Está compuesta de **componentes nodos** y **componentes links**. Cada componente tiene parámetros de entrada (cambiados por el usuario) y de salida.

3.3.1. Componente nodo.

Modeliza el aspecto "físico" de una entidad de almacenamiento y envío. Tiene una cola de paquetes por cada link de salida y una cola común para todos los paquetes recibidos por los link de entrada. El tamaño total de todas las colas no debe exceder un número especificado de bytes. En el link de salida los paquetes de ruteo tienen prioridad.

Si el nodo esta en estado de falla todo paquete que recibe es eliminado. Si el paquete es workload este será retransmitido por el componente workload que lo produjo.

Si el nodo esta en buen estado, cuando recibe un paquete workload es pasado al componente workload conectado a éste, o lo agrega a la cola de un link de salida.

Si el próximo salto no esta definido en el componente de ruteo el paquete es eliminado.

También es eliminado por limitación del espacio en el buffer.

Cuando recibe un paquete de ruteo del componente de ruteo conectado a éste y hay espacio en el buffer, el paquete es insertado en la cola de salida al frente de todos los paquetes workload que están en ese momento. Si no hay espacio en el buffer se trata de hacerlo eliminando un número predeterminado de paquetes workload cuyo tamaño sea mayor o igual al paquete de ruteo. En caso de intento exitoso, el paquete de ruteo es insertado como anteriormente, caso contrario es eliminado.

Cuando recibe un paquete de ruteo desde un link de entrada y hay espacio en el buffer, el paquete es agregado a la cola común para ser procesados por el componente de ruteo conectado al nodo. Si no hay espacio en el buffer se utiliza la política anterior.

Nota: Los paquetes de ruteo son los generados por componentes de ruteo. Los paquetes workload son los generados por componentes workload.

Un nodo al fallar puede ser reparado de acuerdo a los parámetros especificados.

Parámetros de entrada:

- Demora en procesar un paquete (ms).
- Espacio libre en buffer (bytes). -1 = buffer ilimitado.
- Distribución de los tiempos entre-fallas. Exponencial (media en seg.) o uniforme (media y desviación estándar en seg.).
- Distribución de los tiempos de reparación (como anterior).

Parámetros de salida:

- Estado del nodo (up / down).
- Cantidad de espacio utilizado en el buffer del nodo (bytes).
- Máximo espacio ocupado del buffer del nodo durante la simulación (bytes).
- Número de paquetes eliminados por nodo en la simulación.
- Promedio de paquetes eliminados. Total de paquetes eliminados por el nodo durante el último período de actualización dividido por la longitud del período.
- Utilización del buffer.
- Longitud de las colas (paquetes).

3.3.2. Componente link.

Modeliza un canal de transmisión entre dos nodos. Representa dos vías de canales, uno desde el nodo A al B y el otro desde el B al A. Cada una de las vías es modelada con una cola

de paquetes. Si un paquete está en la cola quiere decir que está en tránsito en el link. Un link puede fallar y ser reparado de acuerdo a los siguientes parámetros.

Parámetros de entrada:

- Demora de propagación (ms).
- Ancho de banda (bytes/seg.).
- Distribución de los tiempos entre-fallas. Exponencial (media en seg.) o uniforme (media y desviación estándar en seg.).
- Distribución de los tiempos de reparación (como anterior).

Parámetros de salida:

- Estado del link (up / down).
- Utilización de workload por cada dirección. La fracción de la capacidad del link que es usado por paquetes workload en el último período de actualización, dónde la capacidad del link es definida como el ancho de banda multiplicado por el período de actualización.
- Utilización de ruteo por cada dirección. La fracción de la capacidad del link que es usado por paquetes de ruteo en el último período de actualización.

3.4. Algoritmo de ruteo.

Mantiene en cada nodo información de ruteo que permite el ruteo de paquetes a los distintos destinos. Para realizarlo un **componente de ruteo** es conectado a todos los componentes nodo.

3.4.1. Componente de ruteo.

Contiene la estructura de datos para representar la información de ruteo y las funciones para actualizar esta estructura y propagar la información.

Algoritmos implementados:

- **SPF** (Shortest Path First): es un algoritmo de estados de enlace, cada nodo mantiene una base de datos describiendo la topología de la red y los costos de enlaces. Periódicamente por broadcast se actualiza dicha base. Cada nodo independientemente calcula el camino más corto para saber el próximo paso hacia el nodo destino.

Parámetro de entrada:

- Media y desviación estándar para la distribución uniforme del período de broadcast.

Parámetros de salida:

- Tabla global de topología. Indica el costo de cada link en la red.
- Tabla local de topología. Indica el costo y estado de los links de salida.
- Tabla de ruteo.

- **SEGALL** y **ExBF** son algoritmos de distancia vectorial, periódicamente cada nodo envía un sumario informando a sus vecinos el mejor costo actual para uno o más destinos. Cuando un nodo inicia una actualización se dice que inicia una computación, y el tiempo entre dos computaciones seguidas se llama tiempo intercomputación.

Parámetro de entrada:

- Media y desviación estándar para la distribución uniforme de los tiempos intercomputación.

Parámetros de salida:

- Tabla con información de distancia por cada destino.
- Tabla local de topología. Indica el costo y estado de los links de salida.
- Tabla de ruteo.

No es responsabilidad de estos componentes el cálculo de los costos entre enlaces.

3.5. Función costo-enlace.

Computa los costos por enlace.

Es invocada por el componente de ruteo a t_i instancias de tiempo, llamado período de actualización de cost-link, por lo general coincide con los períodos de broadcast o intercomputación.

Se pueden utilizar las siguientes funciones:

- **Hop-count:** 1 si el link es up, 0 si no.
- **Utilización:** fracción de el tiempo que la cola de link de salida está llena durante el último período de actualización de cost-link.
- **Espera:** promedio de espera sufrido por un paquete en la cola de link de salida y en transmisión (incluyendo el tiempo de proceso en el nodo y el retardo de propagación de el link), para todos los paquetes transmitidos durante el último período de actualización cost-link.
- **Espera hop-normalized:** calculando el promedio de espera por paquete encolado y el promedio de retardo de transmisión por paquete en el último período de actualización cost-link, se calcula una utilización a partir de éstos promedios y asumiendo un modelo de cola M/M/1

3.6. Workload.

Es definido en términos de **pares source-sink**. Cada par consiste de un **componente source** y un **componente sink**. Estos son conectados al componente nodo (por lo general a distintos componentes nodos).

Componente source: produce paquetes y los pasa al componente nodo. Los paquetes son enviados por los componentes nodo y por los componentes link hasta que alcanzan el nodo de destino.

Componente sink: consume los paquetes recibidos en el nodo destino.

Se pueden elegir entre tres diferentes tipos de pares source-sink:

- FTP (transferencia de archivos).
- TELNET (login remoto).
- Simple Traffic (workload simple).

3.6.1. FTP-Source y FTP-Sink.

Representa una transferencia de archivos. El FTP-Source produce y envía paquetes de datos, FTP-Sink consume éstos paquetes y envía paquetes de ack de respuesta. También se pueden intercambiar paquetes token, utilizados para estimar el tiempo de roundtrip y así poder utilizar un período de time-out para retransmitir paquetes perdidos.

FTP provee un esquema de control de flujo estático como las ventanas. Se definen dos ventanas, ventana de producción (limita la cantidad de paquetes producidos pero no enviados) y ventana de envío (limita los paquetes enviados pero sin ack).

Parámetros de entrada:

- Estado inicial de source-sink (on / off). El estado puede estar on en el inicio de la simulación o puede estar off al inicio y activada después de una espera entre conexiones.
- Número de conexiones entre FTP-Source y FTP-Sink.
- Promedio de paquetes por conexión.
- Longitud de paquete (bytes).

- Promedio de espera entre conexiones (ms). Distribuido exponencialmente, es la diferencia de tiempo entre la producción del último paquete de una conexión y la producción del primer paquete en la próxima conexión.
- Retardo entre paquetes (ms).
- Tamaño de la ventana de producción (paquetes).
- Tamaño de la ventana de envío (paquetes).

Parámetros de salida:

- Estado del source-sink (on / off). Es on si todos los paquetes han sido aceptados (ack), off en caso contrario.
- Número de la conexión actual (la primera es 1).
- Número de paquetes en la conexión actual.
- Número del paquete actual en la conexión actual (el primero es 1).
- Número de paquetes producidos.
- Número de paquetes enviados.
- Número de paquetes aceptados.
- Número de paquetes recibidos.
- Número de paquetes retransmitidos.
- Tiempo de roundtrip estimado (ms).
- Tasa de envíos. La cantidad de bytes enviados en el último período de actualización dividido por la longitud del período.
- Tasa de aceptados. La cantidad de bytes aceptados en el último período de actualización dividido por la longitud del período.
- Tasa de retransmisiones. La cantidad de bytes retransmitidos en el último período de actualización dividido por la longitud del período.
- Retardo por paquete (ms). Incluye las retransmisiones.

3.6.2.TELNET-Source y TELNET-Sink.

Representa un tráfico interactivo. El TELNET-Source produce y envía paquetes de datos, TELNET-Sink consume éstos paquetes y envía paquetes de ack de respuesta y viceversa.

Para el control de flujo, retransmisión de paquetes, tiempo estimado de roundtrip utiliza paquetes token y las mismas estrategias que FTP.

Parámetros de entrada:

- Estado inicial de source-sink (on / off).
- Número de conexiones entre TELNET-Source y TELNET-Sink.
- Promedio de paquetes por conexión.
- Longitud de paquete (bytes).
- Longitud de paquete de réplica (bytes). Se refiere a los paquetes producidos por el TELNET-Sink.
- Retardo entre conexiones (ms). Retardo entre paquetes (ms). Retardo entre paquetes de réplica(ms). Distribución exponencial o uniforme.
- Tamaño de la ventana de producción (paquetes).
- Tamaño de la ventana de envío (paquetes).

Parámetros de salida:

- Estado del source-sink (on / off).
- Número de la conexión actual (la primera es 1).
- Número de paquetes en la conexión actual.
- Número del paquete actual en la conexión actual (el primero es 1).
- Número de paquetes producidos.
- Número de paquetes enviados.
- Número de paquetes aceptados.
- Número de paquetes recibidos.
- Número de paquetes retransmitidos.
- Tiempo de roundtrip estimado (ms).
- Tasa de envíos.

- Tasa de aceptados y de retransmisiones.
- Retardo por paquete (ms).

3.6.3.Simple-Traffic-Source y Simple-Traffic-Sink.

Representa un modelo muy simple de workload. El Simple-Traffic-Source produce y envía paquetes de datos, el Simple-Traffic-Sink consume éstos paquetes. No hay paquetes ack, ni token. No hay retransmisiones.

Provee un esquema de control de flujo estático usando una ventana de producción (limita la cantidad de paquetes producidos pero no enviados).

Parámetros de entrada:

- Longitud de paquete (bytes).
- Distribución del tiempo entre paquetes (exponencial o uniforme).
- Tamaño de la ventana de producción (paquetes).

Parámetros de salida:

- Número de paquetes enviados.
- Número de paquetes aceptados.
- Número de paquetes recibidos (igual al paquetes aceptados).
- Número de paquetes perdidos.

3.7.Componente monitor de performance.

Provee medidas de performance para evaluar y comparar distintos algoritmos de ruteo. Hay dos medidas de performance:

- **Actualizada periódicamente:** son actualizadas por un tiempo determinado dentro de un intervalo fijo de tiempo llamado período de actualización.

Por ejemplo:

- Throughput promedio o en un instante dado.
- Espera promedio por paquete o en un instante dado.
- Número de paquetes de ruteo usados.
- Número de paquetes eliminados.
- Carga promedio o en un instante dado de datos o de ruteo.

- **Actualizada por un evento:** son actualizadas cada vez que un evento ocurre.

Por ejemplo:

- Número de conexiones actuales.
- Número promedio de paquetes en una conexión FTP o TELNET.
- Promedio de tiempo de vida de una conexión FTP o TELNET.
- Máxima espera de paquetes.
- Número actual de links fallados.
- Se calculan dos tipos de valores para medir performance:

Valores instantáneos: se calculan basándose solamente en estadísticas recolectadas durante el último período de actualización.

Valores promedio: se calculan basándose en estadísticas recolectadas después de un período de tiempo especificado, llamado intervalo startup, desde el inicio de la simulación. El período desde el fin del intervalo startup hasta el fin de la simulación se llama intervalo de medición. El uso del intervalo startup es porque al iniciar la simulación la red está vacía.

3.8.Componente stopper.

Permite definir una condición de terminación para la simulación, basada en los parámetros de entrada y en las estadísticas recolectadas por el componente monitor de performance durante la simulación.

Se define el throughput de la red en un intervalo de tiempo como el total de bytes de datos aceptados desde el inicio al final del intervalo dividido la longitud del intervalo.

La condición de terminación está definida como: durante un número especificado de intervalos de longitud especificada, el throughput de la red en cada intervalo está con un especificado porcentaje de error límite relativo al throughput promedio del total de los intervalos.

Parámetros de entrada:

- Número de intervalos.
- Longitud del intervalo (ms).
- Límite de errores (porcentaje).

3.9.Eventos.

MARS es un simulador de modelo discreto y con avance de tiempo a intervalos fijos, los eventos que ocurren se dividen en 4 diferentes clases:

- Eventos tipo comandos: para establecer las operaciones básicas, crear borrar componentes. Todas las componentes proveen rutinas para estos eventos.
- Eventos tipo eventos: son los que se envían de componente a componente. Se definen globalmente en **eventdefs.h** solo las componentes que la reciben deben proveer alguna acción. Por ejemplo enviar un paquete del nodo componente al source.
- Eventos tipo privado: son especificados y conocidos por la componente son invisible para el resto de las componentes. Se definen en **components.h** y no en **eventdefs.h**.
- Eventos tipo registración : son eventos que pueden ser grabados a un archivo durante la simulación. Se definen globalmente en **eventdefs.h** . Se puede usar este archivo para repetir una secuencia de eventos en distintas ejecuciones.

Como las componentes se envían eventos para comunicarse o paquetes por la red existe un manejador de eventos que los organiza usando un timer cuya unidad es un tick. [MARSP,1991]

3.10.Creación de componentes.

- Se crean dos archivos, el header (.h) para la estructura de datos y otro (.c) para las rutinas.
- Definir la estructura de datos en el archivo header.
- Definir los eventos privados en el archivo header
- Escribir las rutinas .
- Definir el tipo de la nueva componente y clase en el archivo comptypes.h .
- Agregar una entrada para la nueva componente en components_array en el archivo comptypes.c .
- Modificar el makefile y recompilar.

[MARSP,1991]

3.10.1.Creación de una componente de ruteo.

- La estructura de datos de la componente de ruteo debe ser una extensión de la definida en route.h cuyos campos son:
  processing_time: es un puntero a una función que retoma un entero llamada con un paquete para estimar su tiempo de procesamiento en microsegundos . Es útil para organizar los eventos de ruteo.
  no_of_nodes: es un entero que representa el número conocido de nodos en la red. Útil para evitar los loops en la tabla de ruteo.
  link_cost_time: almacena el tiempo simulado en ticks , es de tipo TICK_T.
  node: puntero a la componente de tipo NODE al cual la componente de ruteo está conectada.
  tabla de ruteo: es un puntero a la estructura de tipo ROUTINGTABLE definida en route.h , la cual almacena el costo y próximo salto de cada destino.
  local_top: puntero a la estructura de tipo LOCALTOPOLOGYTABLE definida en route.h , la cual para cada link saliente almacena el costo de link y si es necesario lo calcula; es creada, inicializada y mayormente actualizada por la componente nodo.
- La estructura del paquete contiene un campo pk_type que es seteado para saber que es de ruteo. La información de ruteo está en el campo rt_pk de un rt_packet definido en packet.h cuyos campos son:
  rt_type: es un entero que representa el subtipo del paquete, definidos como:
  rt_node_shutdown: el nodo al cual está unido ha fallado.
  rt_node_wakeup: el nodo al cual está unido está operando.
  rt_link_shutdown: el link ha fallado.
  rt_link_wakeup: el link está operando.
  cc: es un paquete de intercambio entre componentes.
  rt: es la unión de diferentes paquetes de ruteo. Almacena la información de intercambio entre componentes de ruteo.
- Se debe agregar las rutinas de acción que sean relevantes para la nueva componente de ruteo incluyendo eventos tipo comando.
- Las rutinas para el costo de enlace son llamadas periódicamente por la componente de ruteo y su uso depende del algoritmo de ruteo.

[MARSP,1991]

Capítulo 4

Implementación del MaRS.

Se analiza al simulador en su versión 2.1 no motif enfocado en el comportamiento del algoritmo de ruteo SPF que es el que más relacionado con el protocolo OSPF ya que ambos se basan en la tecnología de estados de enlace. [MARS,1991]

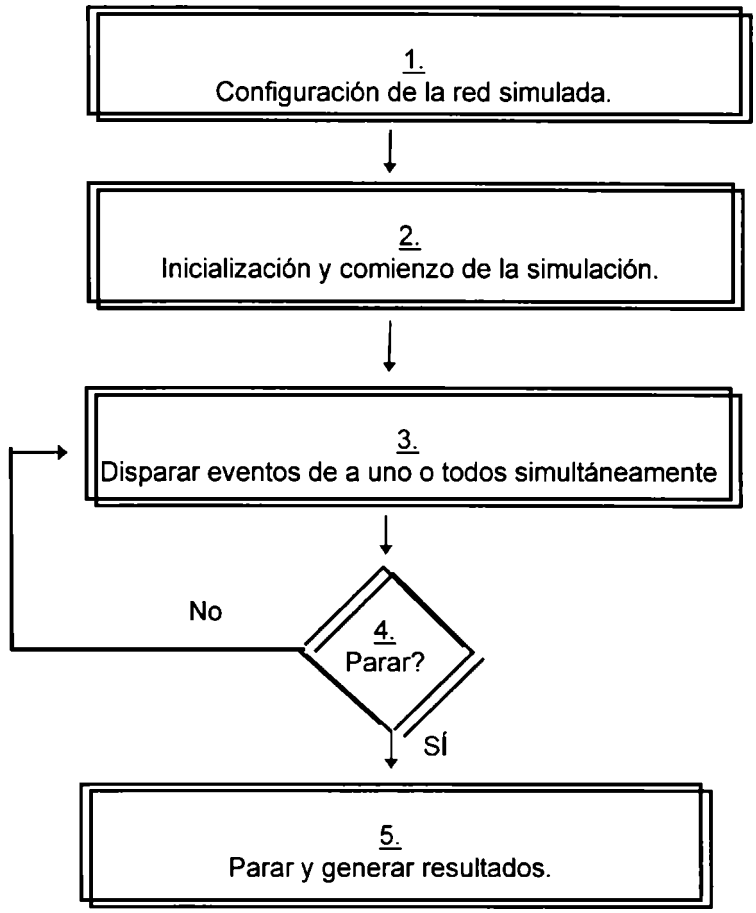
4.1.Análisis de la generación de la simulación.

- Creación de la red:
  A partir del archivo de configuración de la red se crea una lista con todas las componentes de la red y sus respectivas relaciones para su uso durante la simulación.

- Comienzo de la simulación:
  Se inicializa la simulación, lo cual consiste en ejecutar para cada componente el evento **event_reset** que inicializa variables de cada componente.
  Se inicia la simulación, lo cual consiste en ejecutar para cada componente el evento **event_start** que empieza a encolar los eventos asociados a cada componente.

- Generación de la simulación:
  Según seteos de variables globales se dispara el evento encolado de menor tiempo **ev_fire_first()** o todos con el mismo menor tiempo simultáneamente con **ev_fire_all_first()**, se detiene cuando venció el tiempo de simulación o bien se controle ese tiempo a través de la componente stopper.
  Los eventos que se encolaron a través de **event_start** son los encargados de encolar otros eventos que necesariamente se dispararán para generar tráfico de datos, paquetes de ruteo, calcular costos, conectar o no los enlaces, detectar errores, etc.
  Cuando el tiempo simulado finalice se ejecutará el **ev_stop** de cada componente y se generan los resultados en un archivo de salida.

Esquema de los pasos de simulación en Mars.



4.2.Análisis del manejo de eventos.

Inicialmente se ejecuta el **event_start** de cada componente la cual encola los eventos correspondientes a cada una. Luego se disparan los eventos encolados, el o los de menor tiempo simultáneamente.

Podemos decir que el MaRS es un simulador de eventos discretos orientado a sucesos con avance de tiempo a intervalos fijos. Ver figuras de las secciones 2.6.1. y 2.7.

Cuando el tiempo de simulación finaliza se ejecuta el **ev_stop** de cada componente.

Analizaremos los eventos tratados por los componentes node, link y spf, porque son los involucrados en el tráfico de ruteo.

- **Eventos tratados por la componente Node. Codificado : en Node.c, función node_action().**

Eventos	¿Qué hace?	Eventos que encola (•) o ejecuta (∈) :	Eventos contenido en :
ev_reset nd_reset()	Limpia las colas de paquetes de E/S, setea parámetros y llena la tabla topológica local.		

ev_create nd_create()	Asigna memoria para esta estructura de nodo ,crea lista de vecinos, inicializa parámetros (delay, velocidad, tiempos de falla, etc.).		
ev_del nd_delete()	Borra al nodo y sus colas E/S.		
ev_neighbor nd_neighbor()	Se asigna como vecino de un nodo a : una componente de ruteo, o enlace, o un workload previo varios chequeos y además si se avecina una componente enlace se alloca las colas de /S de paquetes y si es de ruteo obtiene punteros a las tablas de ruteo, a la función que estima los tiempos de proceso de ruteo y a la componente de ruteo.		
ev_uneighbor nd_uneighbor()	Se desasigna un vecino al nodo.		
ev_start nd_start()	Si el promedio de fallas es >0 encola el evento de falla para un tiempo de falla estimado y encola el evento de cálculos estadísticos con tiempo inmediato.	• ev_failure • ev_instant_rate	• Node.c • Node.c
ev_node_receive nd_receive()	Si el nodo no está caído y hay espacio en las colas: si el paquete recibido es de ruteo encola su procesamiento con el tiempo de ruteo calculado : ev_route_processing , si el paquete es de transporte y el destino es justamente el mismo ejecuta su recepción inmediata : ev_aptr_receive y si el destino es otro encola el paquete calculando próximo salto para su envío con tiempo de arribo + espera : ev_node_send ; luego actualiza la tabla topológica local : LTT, si el nodo está caído o hay espacio u otro problema , se descartan los paquetes.	• ev_route_processing • ev_node_send ∈ ev_aptr_receive	• Spf.c , Segal.c . • Node.c. • Ftp.c,Telnet.c
ev_node_send nd_send()	Ejecuta el ev_link_send para el paquete del la cola de su vecino (router o workload) y organiza el próximo envío.	∈ ev_link_send • ev_node-send	• Link.c . • Node.c .
ev_node_produce nd_produce()	Si el nodo está levantado, si hay espacio en las colas el destino del paquete ∈ a la tabla de ruteo RT --> da prioridad a los paquetes de ruteo, encola su envío y actualiza tablas, o descarta el paquete.	• ev_node_send	• Node.c .

ev_node_failure nd_failure()	Si el nodo está caído encola su reparación con tiempo de reparación, asume que los paquetes de datos transmitidos no se perdieron por lo que vacía las colas y ejecuta el ev_link_failure por cada enlace. Genera un paquete de ruteo rt_node_shutdown y lo procesa con ev_route_processing.	• ev_node_repair ∈ ev_link_failure ∈ ev_route_processing	• Node.c . • Link.c . • Spf.c, Segal.c
ev_node_repair nd_repair()	Se recupera, encola su próxima falla con ev_node-failure, luego mediante paquetes rt_node_wakeup se procede a su reparación mediante ev_route_processing y una vez reparado lo hacen sus enlaces vía ev_link_repair.	• ev_node_failure ∈ ev_route_processing ∈ ev_link_repair	• Node.c . • Spf.c, Segal.c • Link.c .
ev_instant_rate instant_rate()	Calcula el promedio de caídas y se encola a si misma para procesarse en tiempo constante.	• ev_instant_rate	• Node.c .
ev_stop	Devuelve el puntero de la componente nodo.		

- **Eventos tratados por la componente Link. Codificado en: Link.c, función link_action().**

Eventos	¿Qué hace?	Eventos que encola (•) o ejecuta (∈) :	Eventos contenido en :
ev_reset link_reset()	Setea el estado del enlace como up y otros datos.		
ev_create link_create()	Alloca memoria para la estructura de link e inicializa parámetros con los resultados del enlace.		
ev_del link_delete()	Vacía sus colas, se borra a sí mismo y ejecuta ev_uneighbor por cada vecino o sea los dos nodos que conecta .	∈ ev_uneighbor	• Node.c .
ev_neighbor link_neighbor()	Establece como vecino a 2 componentes nodo e inicializa parámetros.		
ev_uneighbor link_uneighbor()	Borra el vecino de la lista de vecinos del link que siempre son dos nodos.		
ev_start link_start()	Encola la próxima falla con tiempo de falla.	• ev_link_failure	• Link.c .
ev_link_receive link_receive()	Envía paquetes a uno u otro vecino con ev_node_receive , si hay más paquetes en las colas encola ev_link_receive con un tiempo t.	• ev_link_receive ∈ ev_node_receive	• Link.c . • Node.c .

ev_link_send link_send()	Si el link está levantado encola el ev_link_receive para uno u otro vecino con tiempo de propagación t. Si está down y el paquete es de transporte ejecuta ev_aptr_retransmit.	• ev_link_receive ∈ ev_aptr_retransmit	• Link.c . • Telnet.c, Ftp.c
ev_link_failure link_failure()	El link cae, vacía las colas retransmitiendo los paquetes de transporte ejecutando ev_aptr_retransmit; produce un paquete rt_link_shutdown para ambas colas y ejecuta el ev_link_receive; encola su reparación con ev_link_repair con tiempo de reparación.	• ev_link_repair ∈ ev_link_receive ∈ ev_aptr_retransmit	• Link.c . • Link.c . • Telnet.c ,Ftp.c
ev_link_repair link_repair ()	Encola la próxima falla con tiempo de falla t, y se repara o levanta, o lo repara el nodo vecino que estaba fallando y se restableció. Genera un paquete de ruteo rt_link_wakeup y ejecuta ev_link_receive por ambas colas.	• ev-link_failure ∈ ev_link_receive	• Link.c . • Link.c .
ev_stop	Devuelve el puntero del link.		

- **Eventos tratados por la componente de ruteo Spf. Codificado en: Spf.c , función spf_action().**

Eventos	¿Qué hace?	Eventos que encola (•) o ejecuta (∈) :	Eventos contenido en :
ev_reset spf_reset()	Inicializa número de nodos vecinos , número de secuencia y costo de enlace.		
ev_create spf_create()	Inicializa parámetros.		
ev_del spf_delete()	Borra la componente de ruteo.		
ev_neighbor spf_neighbor()	Solo permite tener como vecino a una componente nodo y actualiza bases (RT,GTT,SNT).		
ev_uneighbor spf_uneighbor()	Se remueve el vecino y se actualiza la tabla topológica local.		
ev_start spf_start()	Encola el evento spf_broadcast para ejecutarse inmediatamente.	• ev_spf_broadcast	• Spf.c .

ev_route_processing spf_processing()	Trata los cinco tipos de paquetes rt_link up y down que se generan para levantar o caer un enlace y rt_node up y down cuando se levanta y cae el nodo, rt_ls que llevan los cambios de la red a otros routers x broadcast a través del ev_node_produce llega a los nodos para ser distribuidos ; se actualizan las tablas; obtiene el próximo paquete a rutear encolándolo con el evento ev_route_processing.	• ev_route_processing ∈ ev_node_produce	• Spf.c . • Node.c .
ev_spf_broadcast spf_broadcast()	Encola ev_spf_broadcast para un tiempo t de broadcast calcula métricas , genera paquetes rt_ls para ser procesados por broadcast encolando este evento y enviándoselos a sus nodos para ser distribuidos.	• ev_spf_broadcast ∈ ev_node_produce	• Spf.c . • Node.c .
ev_mk_peer	Devuelve el puntero de la componente de ruteo.		
ev_stop	Devuelve el puntero de la componente de ruteo.		

4.3.Análisis de la generación de paquetes de ruteo.

Inicialmente cuando se ejecuta el evento ev_start de cada componente de **ruteo** se genera un paquete rt_ls que es el que lleva los cambios producidos en ese router a sus vecinos y por broadcast se actualizarán los cambios recalculando las tablas de ruteo, global y local.

Otros paquetes de ruteo son generados por la componente **Nodo** cuando falla o se recupera o la componente **Link** cuando uno o varios enlaces fallan o se recuperan.

Además cuando un enlace se repara se genera un paquete rt_ls para la actualización de las tablas por broadcast.

Los tipos de paquetes de ruteo son para dormir o levantar un enlace o nodo o para informar los cambios a otros routers x broadcast.

Por lo tanto la componente de **ruteo** o más especificamente **spf** o **segal** sirven para rutear los paquetes con su modalidad y solo genera los paquetes que llevan los cambios efectuados al resto de la red , la componente nodo y enlace determinan cuando es necesario generar paquetes de ruteo, por fallas y recuperación.

4.4.Tabla de tópicos de interés.

Tema.	**módulo.c - función invocada.**	**módulo.c - función invocante.**	**Breve explicación del tema.**
¿ Cómo se crea la red simulada ?	File.c read_world()	Main()	A partir del archivo parametrizado se configura la red; cada componente se registra en una lista de componentes.
¿ Cómo ver el resultado de la	File.c save_snapshot()	Main()	En un archivo sim_snap se generan

simulación?			los resultados almacenados durante la simulación y se ven al finalizar editando el archivo.
¿ Cómo comienza la simulación?	Main.c sim_reset() Main.c sim_start()	Main() Main.c	Resetea y libera memoria ocupada por simulaciones previas. Vía eventos de start de cada componente se encolan los eventos necesarios para cada uno y luego se van disparando hasta que se decida terminar y se ejecuta el evento de stop de cada uno.
¿ Cómo se genera la simulación ?	Event.c ev_fire_first() o ev_fire_all()	Main()	Según seteos de variables de entorno se ejecuta el o los eventos con menor tiempo en ser disparados respectivamente.
¿ Cuándo termina la simulación?	Main.c sim_stop()	Main()	Cuando venció el tiempo de simulación se ejecuta el ev_stop de cada componente de la lista.
¿ Cuándo se genera un paquete de ruteo?	Ruteo.c spf.c ev_start()	Node.c node_failure() node_repair() Link.c link_failure() link_repair()	Cuando un nodo **falla**: genera un paquete rt_node_shutdown para informar a sus vecinos a través del proceso de ruteo (ev_route_processing ERP) y sus enlaces también caen, cuando el nodo se **recupera** genera el paquete rt-node_wakeup para ERP y hace que sus enlaces se restauren. Cuando un enlace o su nodo **falla**: el enlace genera un paquete rt_link_shutdown para sus dos nodos vecinos y cuando se repara por el mismo o porque su nodo se reparó genera un paquete rt_link_wakeup y lo envía a sus vecinos para su distribución a partir del

			nodo que lo envía a su router y este realiza lo necesario para efectuar los cambios generando para ello un paquete rt_ls que lleva por broadcast los cambios al resto de la red y así se actualizan las tablas.

Capítulo 5

Análisis del OSPF.

5.1.Conceptos básicos.

El protocolo de ruteo OSPF (Open Shortest Path First) está basado en la tecnología de estados de enlace SPF y fue diseñado por IETF, grupo de trabajo de Internet Engineering Task Force, para un ambiente Internet incluyendo subredes IP, TOS, marca de información de ruteo externa, autenticación de paquetes y multicast IP en el envió y recepción de mensajes, clasificado como un protocolo de acceso interno IGP (Interior Gateway Protocol), por lo que la información se distribuye entre routers pertenecientes al mismo AS (Sistema Autónomo) y además responde a los cambios de la topología reduciendo el tráfico de ruteo.

Para el OSPF el sistema autónomo es esencialmente un grupo de routers que intercambian información de ruta vía un protocolo de ruteo en común.

Un AS comprende una o más redes que pueden o no tener subredes. Todos los routers del AS deben ser configurados con el mismo número asignado al AS por NIC (Network Information Center). Además los routers se agrupan en áreas y para las redes se elige un router como router designado que lleva el control. [MOY,1996-1997]

5.2.Funcionamiento.

Una copia separada del algoritmo de ruteo corre en cada área. Routers de múltiples áreas corren múltiples copias.

Cuando un router se levanta:

- Inicializa las estructuras de datos del protocolo de ruteo .
- Espera las indicaciones de los protocolos de bajo nivel de las interfaces que funcionan.
- Obtiene los vecinos usando el protocolo Hello, sincroniza y actualiza las bases de datos entre los adyacentes obtenidos. Si la red es broadcast el router dinámicamente detecta sus vecinos enviando hello por multicast y en las redes multiacceso el protocolo hello además elige el DR de la red.
- Un router periódicamente reconoce su estado llamado estado de enlace y lo advierte frente a un cambio . Los adyacentes del router se reflejan en el contenido de un aviso de estado de enlace; por lo que esta relación permite detectar routers muertos. Estos avisos fluyen en el área para que el resto tengan exactamente la misma base topológica la cual consiste de los avisos de estado de enlace recibidos por cada router del área y de la cual cada router calcula el camino más corto con el como raíz y así generar la tabla de ruteo.

5.3.Clasificación de áreas.

Permite redes contiguas y host agrupados, tales grupos más los grupos conectándose a las redes forman un área en la que corre un algoritmo distinto del SPF con una base topológica propia, invisible para el resto y reduce considerablemente el tráfico.

Cada router tiene una tabla topológica local para cada área a la que pertenece y los de la misma área la comparten.

Area backbone: cada dominio del AS para un ruteo OSPF pasa por un área backbone, es la troncal identificada como 0.0.0.0 , no es específicamente un área sino una conexión de routers pertenecientes a múltiples áreas . Por lo general están en áreas contiguas caso contrario se debe recuperar la conectividad con enlaces virtuales. El backbone distribuye información a sus áreas y es invisible a cada una.

Areas stub: es un área que no permite avisos de routers externos.

5.4.Clasificación de routers.

Cuando un AS se divide en áreas OSPF, los routers se dividen de acuerdo a su función en estas 4 categorías solapadas:

- Router interno: conecta redes pertenecientes a la misma área.
- Router de borde: conecta múltiples áreas corre múltiple copias del SPF y se adiciona al backbone.
- Router de backbone: tienen interface al backbone ya que tienen interface a más de un área y los que tienen todas sus interfaces al backbone son considerados internos.
- Router AS boundary: son los que intercambian información con routers de otros AS. Pueden ser internos , de borde , participar o no del backbone.
- Router intra-area: cuando los routers que originan la información lo envían a un destino perteneciente a la misma área.
- Router inter-area: cuando los routers que originan la información lo envían a un destino perteneciente a otra área.

5.5.Vecinos y adyacentes.

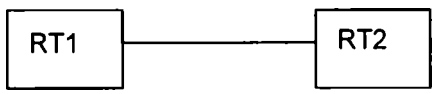
Para facilitar el intercambio de información se crean adyacencias entre routers vecinos.

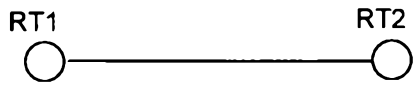
Dos routers son vecinos porque tienen una interface en común y se descubren por el protocolo Hello.

Dos routers son adyacentes cuando se estableció una relación de intercambio entre vecinos.

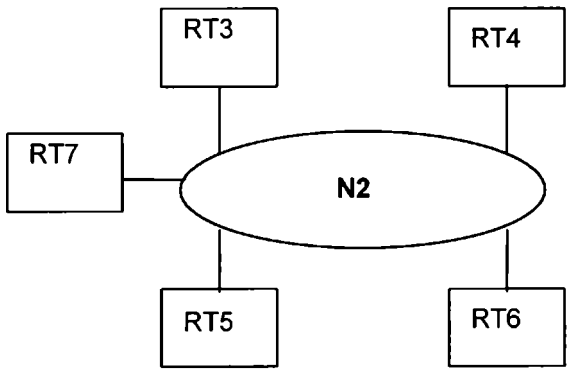
Grafos de adyacencias:

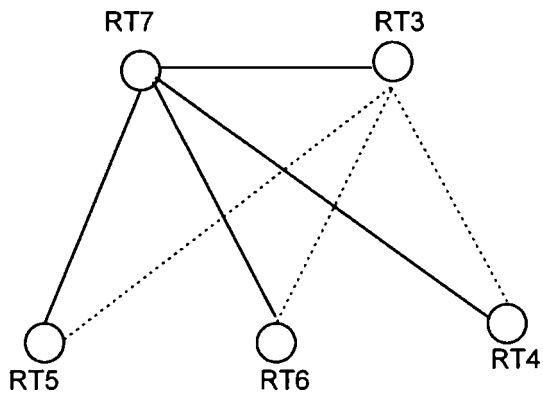
Para una conexión point-to-point.





Para una conexión de red.





5.6.Routers designados (DR) y Router backup designados (BDR).

Vía un algoritmo de elección del router designado por prioridades y con el protocolo hello se escoge un router de una red multiacceso para centralizar el tráfico de esa red y sincronizar la actualización de la información, a través de 2 funciones importantes: el envío de los avisos de red y la adyacencia con todos los routers de la misma.

El BDR, será el DR cuando el previo DR falle si no hay uno elegido será necesaria la elección entre los routers conectados a la red vía el protocolo hello.

5.7.Enlaces virtuales.

El área 0 o backbone, no puede desconectarse de ningún área y para ello usa enlaces virtuales configurados por ambos routers y los puntos finales son routers de borde. Se consideran como una red punto a punto sin numeración , solo los routers finales detectan el enlace determinando si es o no viable y su costo.

5.8.El camino más corto.

Describe la mejor ruta a cualquier destino de red o host verificando cantidad de saltos.

Para la información externa se considera 2 tipos de métricas: Tipo 1 es comparada con la métrica del enlace y para el tipo 2 se asume que el costo es mayor que por cualquier camino intra-AS.

5.9.Estructuras.

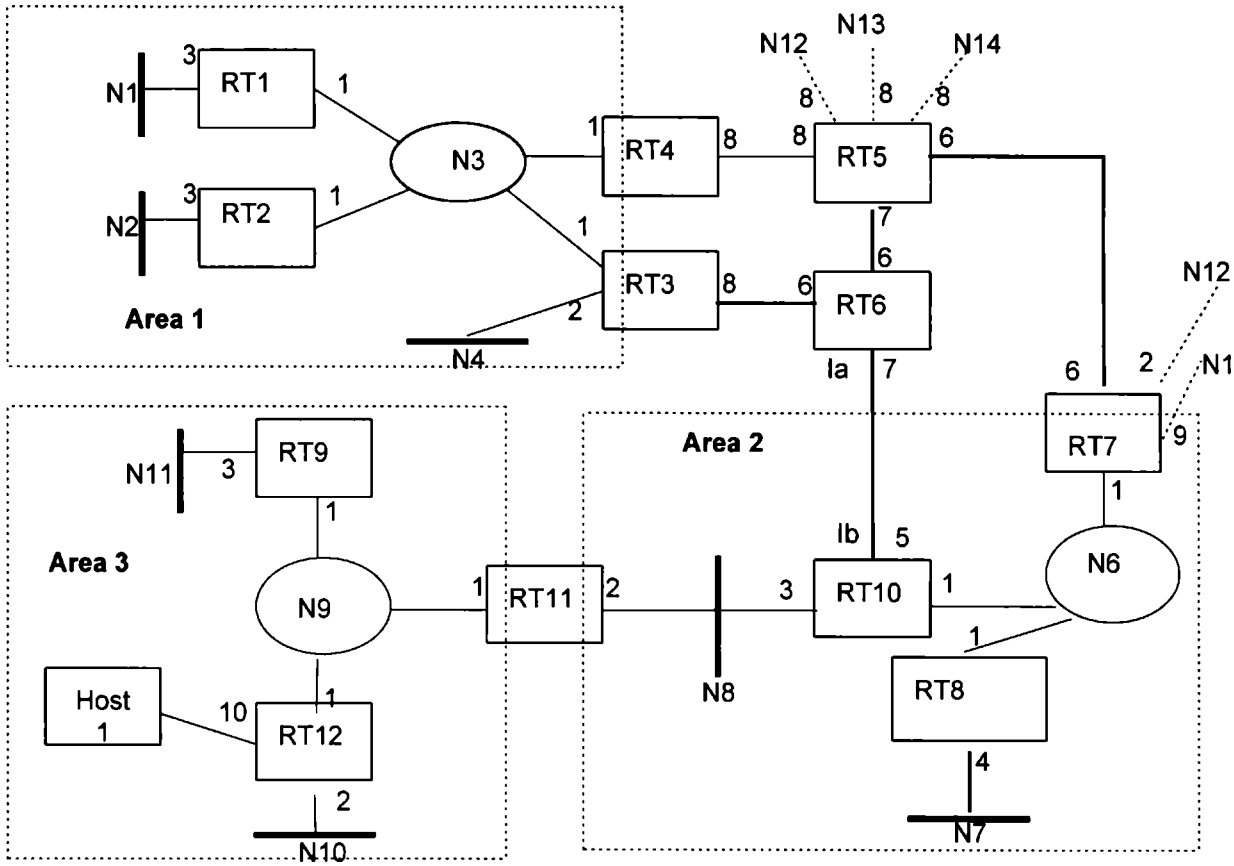
5.9.1.Tabla topológica.

Describe en un grafo dirigido la topología del SA. Los vértices consisten de routers y redes. La conexión final del grafo es entre dos routers unidos físicamente punto a punto.

Los vértices del grafo pueden ser clasificados de acuerdo a su función y solo en algunos de ellos hay tráfico y están marcados como de entrada y salida.

Tipo de vértice	Nombre	Tránsito
1	router	sí
2	red	sí
3	red stub	no

Ejemplo de un Sistema Autónomo :



El sistema autónomo está dividido en áreas:

El área 1 consiste de las redes N1 a N4 y los routers RT1-RT4.

El área 2 consiste de las redes N6 a N8 y los routers RT7, RT8, RT10 y RT11.

El área 3 consiste de las redes N9 a N11, el host H1 y los routers RT9, RT11 y RT12.

La clasificación de sus routers es:

RT1, RT2, RT5, RT6, RT8, RT9 y RT12 son routers internos.

RT3, RT4, RT7, RT10, RT11 son routers de borde.

RT5 Y RT7 son routers de borde de otro SA.

Ejemplo de la tabla topológica para el Area 1 redes y routers se representan por vértices.:

Desde / Hacia	RT1	RT2	RT3	RT4	RT5	RT7	N3
RT1							0
RT2							0
RT3							0
RT4							0
RT5			14	8			
RT7			20	14			
N1	3						
N2		3					
N3	1	1	1	1			
N4			2				
Ia,Ib			15	22			
N6			16	15			
N7			20	19			

N8			18	18			
N9-N11,H1			19	26			
N12					8	2	
N13					8		
N14					8		
N15						9	

5.9.2.Tabla de ruteo.

Contiene toda la información necesaria para transmitir un paquete IP hacia su destino.

Cada entrada en la tabla describe la colección de los mejores caminos a un destino en particular; indicando el próximo salto y proveyendo rutas default.

Hay una simple tabla en cada router, construido por los avisos de estado de enlace de la tabla topológica de cada área a la cual se conecta. Luego construye el árbol con el camino más corto con el como raíz.

Tipo de destino:

Red (N): incluye redes IP, subredes IP, Host IP, routers.

Routers de borde (BR): conectado a múltiples áreas, origina avisos de enlaces sumariales y con ella se calcularía rutas inter-area.

Router de borde de otro AS (ASBR): originan avisos de enlaces externos y se usa para calcular rutas externas.

ID Destino: Identificación o nombre del destino, para una red es la dirección IP, caso contrario es el Id del router OSPF.

Máscara: solo definidas para redes.

Tipo de servicio: Puede haber un conjunto de caminos para cada tipo de servicio.

Área: es la identificación de área para la entrada en la tabla. Para caminos externos no se define área.

Tipo de caminos: intra-area (1), inter-area (2) o AS externo (3).

Costo: el costo para llegar a destino, para intra-area e inter-area el costo está en término de la métrica del estado de enlace. Para caminos AS externos, para métricas de tipo 1 indica el costo total del camino y para métricas de tipo 2 el costo al router del otro AS.

Próximo salto: la interface del router a usar cuando se envíe el tráfico al destino. Para redes multiacceso la dirección IP del próximo router hacia el destino.

Advertising router: válido para caminos inter-area y AS externos e indica el router ID del router que originó el aviso sumarial o externo.

Tabla de ruteo para el router RT4 del sistema autónomo de la sección anterior:

Tipo de destino	Destino	Área	Tipo de camino	Costo	Próximo Salto	Router que advierte
N	N1	1	1	4	RT1	*
N	N2	1	1	4	RT2	*
N	N3	1	1	1	*	*
N	N4	1	1	3	RT3	*
BR	RT3	1	1	1	*	*
N	Ib	0	1	22	RT5	*
N	Ia	0	1	27	RT5	*
BR	RT3	0	1	21	RT5	*
BR	RT7	0	1	14	RT5	*
BR	RT10	0	1	22	RT5	*
BR	RT11	0	1	25	RT5	*
ASBR	RT5	0	1	8	*	*
ASBR	RT7	0	1	14	RT5	*

N	N6	0	2	15	RT5	RT7
N	N7	0	2	19	RT5	RT7
N	N8	0	2	18	RT5	RT7
N	N9-N11, H1	0	2	26	RT5	RT11
N	N12	*	3	16	RT5	RT5, RT7
N	N13	*	3	16	RT5	RT5
N	N14	*	3	16	RT5	RT5
N	N15	*	3	23	RT5	RT7

5.9.3.Tabla de estados de enlace.

Cada router tiene una base de estados de enlace por cada área, referenciada como la base topológica que es la misma para todos los routers de una misma área.

Cada parte de esta base forman un aviso de router, de red, sumarial o externo por lo que OSPF debe acceder a estas partes de la base basándose en el tipo de Ls, link state id y advertising router como acceso directo, así con esta función el router puede determinar si es un aviso que el generó y asignarle número de secuencia.

5.9.4.Paquetes.

Los paquetes de ruteo OSPF corren directamente sobre IP (código 89). No provee explícitamente soporte de fragmentación y reensamble, si lo requiere lo usa directamente de IP.

Además tienen IP Tos en 0 por lo que tendrán preferencia sobre el tráfico regular de IP, comparten el mismo encabezado y existen 5 tipos de paquetes : Helio, Descripción de la base de datos, Pedido de estado de enlace, Actualización de estado de enlace y Reconocimiento del estado de enlace.

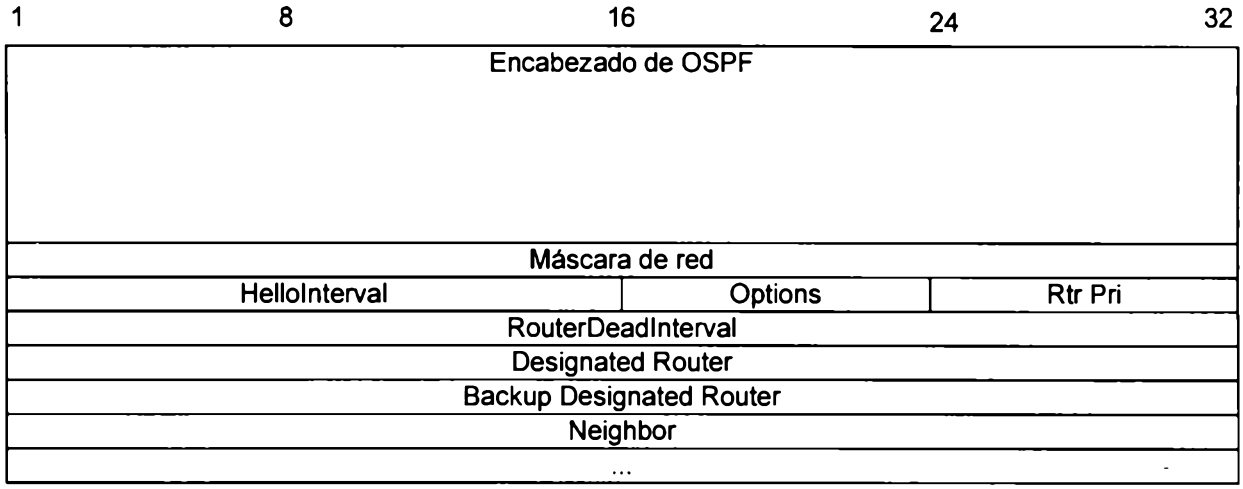
5.9.4.1.Encabezado OSPF.

1 8 16 24 32

Versión	Tipo	Longitud del paquete OSPF
Router ID que origino el paquete		
Área ID en la que se originó el paquete		
Checksum de IP		Tipo de autenticación
Autenticación Autenticación		

5.9.4.2.Paquete Hello.

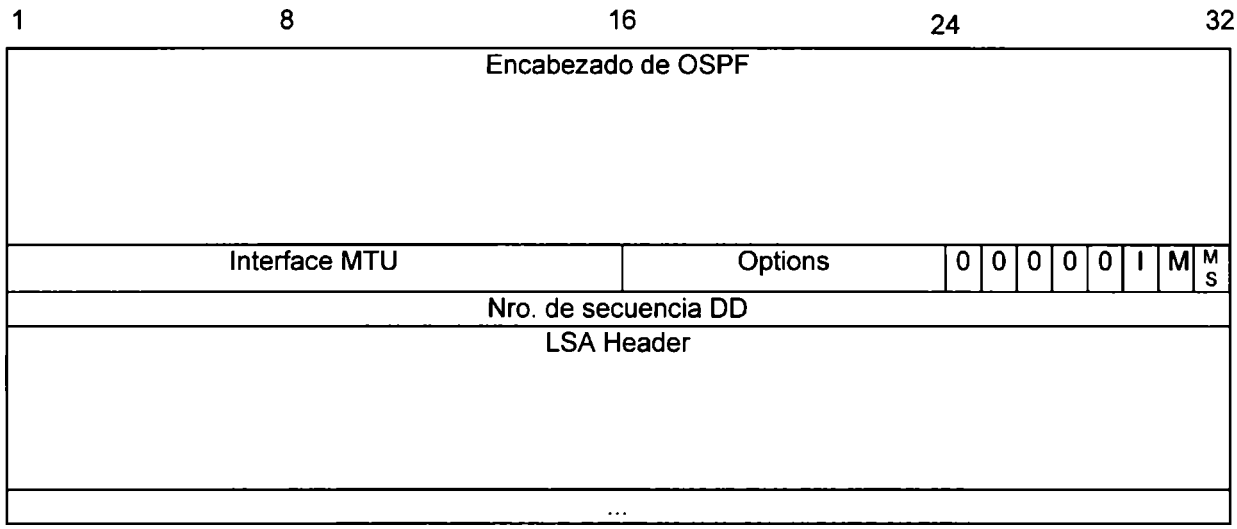
Es el paquete de tipo 1 y es enviado periódicamente por multicast a todas las interfaces incluyendo enlaces virtuales para mantener y establecer la conexión con sus vecinos.



Máscara de red: de la red asociada a la interface.
HelloInterval: segundos entre paquetes hello.
Options: capacidades opcionales soportadas por el router.
Rtr Pri: prioridad usada en la elección del DR , en 0 significa que nunca podrá ser elegido DR.
RouterDeadInterval: segundos antes de silenciar o morir el router.
Designated Router: ID del DR o sea de su dirección IP.
Backup Designated Router: ID del BDR o sea de su dirección IP.
Neighbor: IDs de los routers a los que se le envió paquetes hello en los últimos segundos (RouterDeadInterval).

5.9.4.3.Paquete de Descripción de la base de datos.

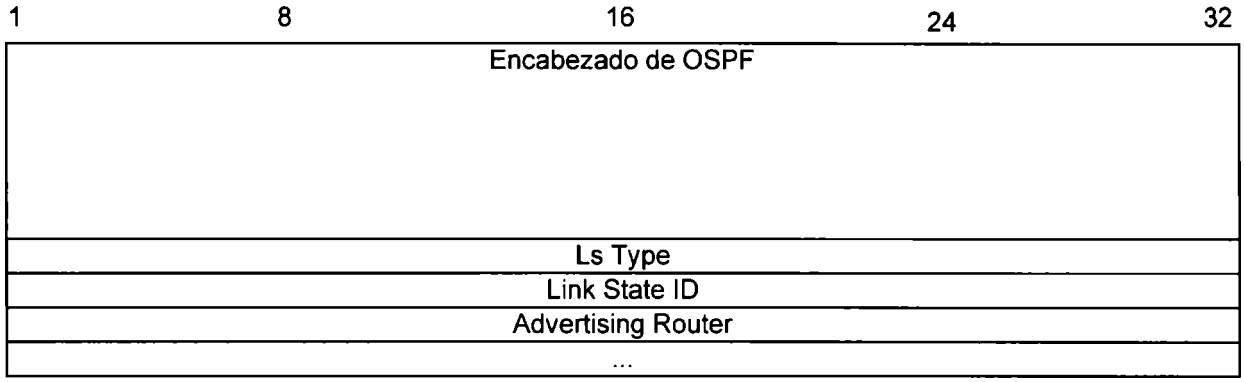
Es el paquete de tipo 2. Cuando dos routers son adyacentes pueden intercambiar paquetes DD, que describen la base topológica de cada uno, previa negociación del master y slave en este intercambio de información.



I : bit de inicio.Igual a 1 si es el inicio del intercambio, 0 en caso contrario.
M: cuando es igual a 1 indica que hay más paquetes por enviar, 0 en caso contrario.
MS: si es igual a 1 indica que es el master de la conversación, 0 slave.
Número de secuencia DD: se incrementa en cada envío hasta que la descripción de la base de datos se haya completado.

5.9.4.4.Paquete de Pedido de estado de enlace.

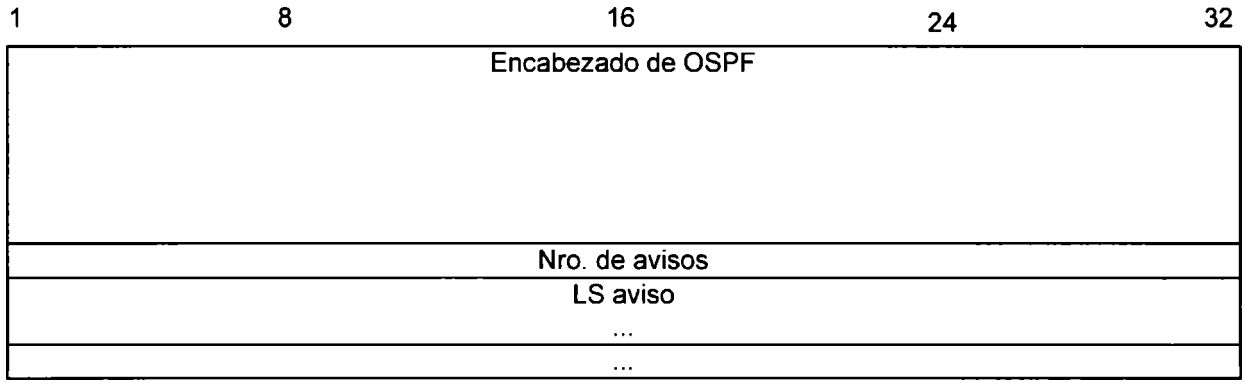
Es el paquete de tipo 3 y se usa para pedir a sus vecinos la parte de la base de datos a actualizar.



Utiliza los tres datos que identifican unívocamente a un aviso:
Ls Type: tipo de aviso (ver 5.9.4.7).
Link State ID: identificación de la porción de la red que describe el aviso.
Advertising Router: la identificación del router que origina el aviso.

5.9.4.5.Paquete de Actualización de estado de enlace.

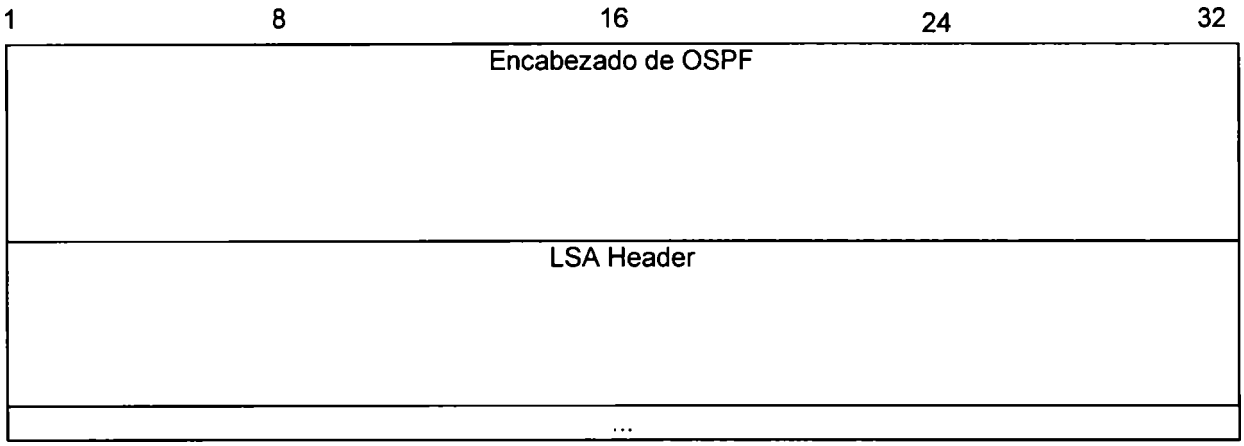
Es el paquete de tipo 4 e implementan la distribución de los avisos vía multicast con retransmisión. Los avisos de un paquete tipo 4 pueden ser de routers , de red y sumariales.



Nro. de avisos: cantidad de avisos en el paquete.
LS aviso: el o los avisos enviados.

5.9.4.6.Paquete de Reconocimiento de estado de enlace.

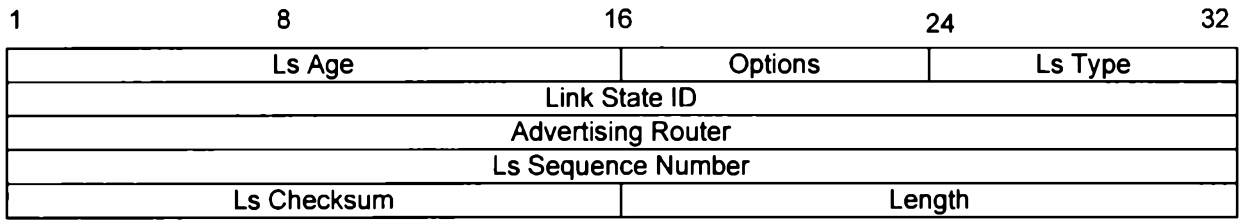
Es el paquete de tipo 5, se utilizan para hacer el flooding de paquetes de actualización confiable. Los paquetes de actualización que se envían por flooding deben ser reconocidos con un paquete de éste tipo.



LSA Header: el encabezado del aviso a reconocer.

5.9.4.7.Formato de los avisos.

Header de un aviso (LSA Header).



Ls age: segundos de creación del aviso usado para saber cual es el más reciente.
Options: capacidades opcionales para describir una porción del dominio de ruteo.
Ls Type: indica si es un aviso de router (1), de red (2), sumarial con destino a una red IP (3), sumarial con destino a un router de borde con otro AS (4) o externo al SA (5).
Ls Id: si el LsType es 1 ó 4 es el router Id, si es 3 ó 5 es el Id de la red y para el 2 es la dirección IP del DR.
Advertising router: Id del router que originó el aviso. Para LsType igual a 1 es el LS Id, para el 2 el Id del DR, y para el 3, 4 y 5 el Id del router de borde.
Ls sequence number: número de secuencia del aviso.

5.9.4.7.1.Clasificación de tipos de avisos.

- **Avisos de router:** Son avisos de tipo 1. Describen los estados de los enlaces del router en el área.

1	8	16	24	32
LS Header				
0	|B|E	0	Nro. de links	
Link ID				
Link Data				
Tipo de link		Nro. de TOS	Métrica	
TOS		0	TOS métrica	
...				

bit B: si está seteado el router es un router de borde de área.
Bit E: si está seteado el router es externo.
Nro. de links: cantidad de enlaces descriptos. Debe ser el total de links del router para el área.
Link Id: Id del objeto al cual se conecta, depende del tipo de link:

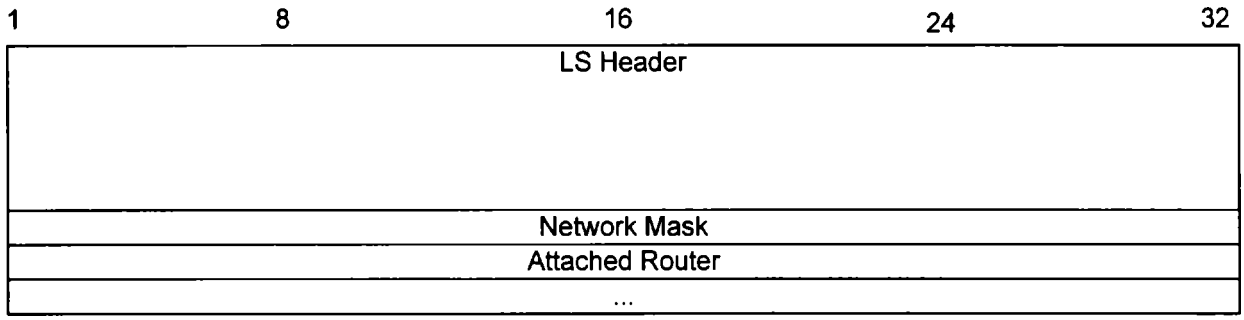
tipo	**link Id**
1	Id del router vecino.
2	Dirección IP o Id del DR.
3	Dirección IP .

Link Data: para redes stub es la máscara de la red IP, para el resto, la dirección IP del router. Es necesaria durante el proceso de construcción de la tabla de ruteo, para calcular la dirección IP del próximo salto.
Tipo de link: Tipo de la conexión con otro router:

tipo	**link Id**
1	Conexión point-to-point a otro router.
2	Conexión a una red de tránsito.
3	Conexión a una red stub.

Nro. de TOS: número de diferentes tipos de servicio (por lo general igual a cero).
Métrica: el costo de usar éste enlace.
TOS: tipo de servicio.
TOS Métrica: información específica para el TOS.

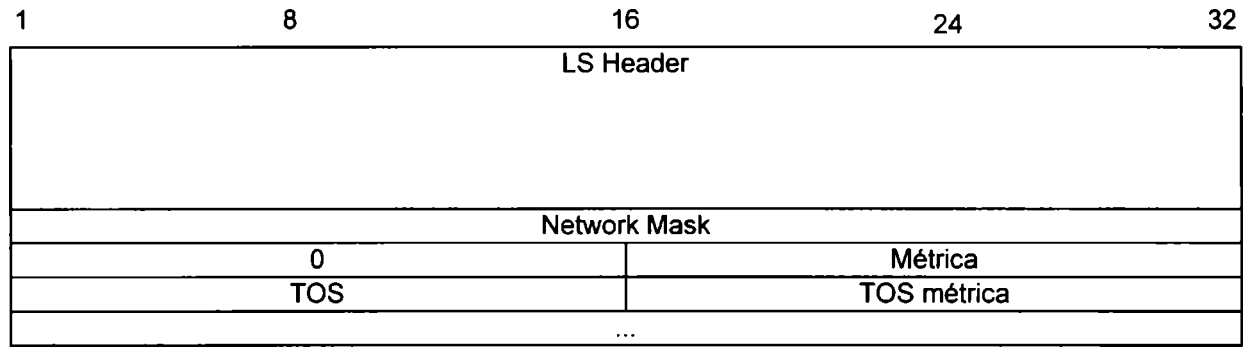
- **Avisos de red:** Son avisos de tipo 2. Son originados por el DR de una red de tránsito, describiendo todos los routers conectados a la red, incluyendo el DR. La distancia de la red a sus routers es 0, por lo que no es necesario el servicio TOS y su métrica en los avisos.



Network Mask: la máscara de la red IP. Por ejemplo una red de clase A tiene la máscara 0xff000000 .

Attached Router: El ID de cada router conectado a la red y que están en estado full adyacentes del DR, incluyendo el DR. La cantidad se deduce de la longitud del aviso.

- **Aviso sumarial:** Son avisos de tipo 3 cuando el destino es una red IP y 4 cuando el destino es un router de borde con otro AS y corre en OSPF, pero su formato es el mismo. Originados por routers de borde.



Network Mask: Para avisos tipo 3 indica la dirección máscara de la red IP destino, 0 para tipo 4 (no se usa).

Métrica: costo del router, expresado con la misma unidad que el costo de las interfaces en el aviso de router.

TOS: el tipo del servicio, por lo general igual a cero.

TOS métrica: información métrica específica para el tipo de servicio.

5.10.Algoritmos para el tratamiento de casos puntuales.

5.10.1.Elección del DR y BDR.

El algoritmo es invocado por la máquina de estados de interface.

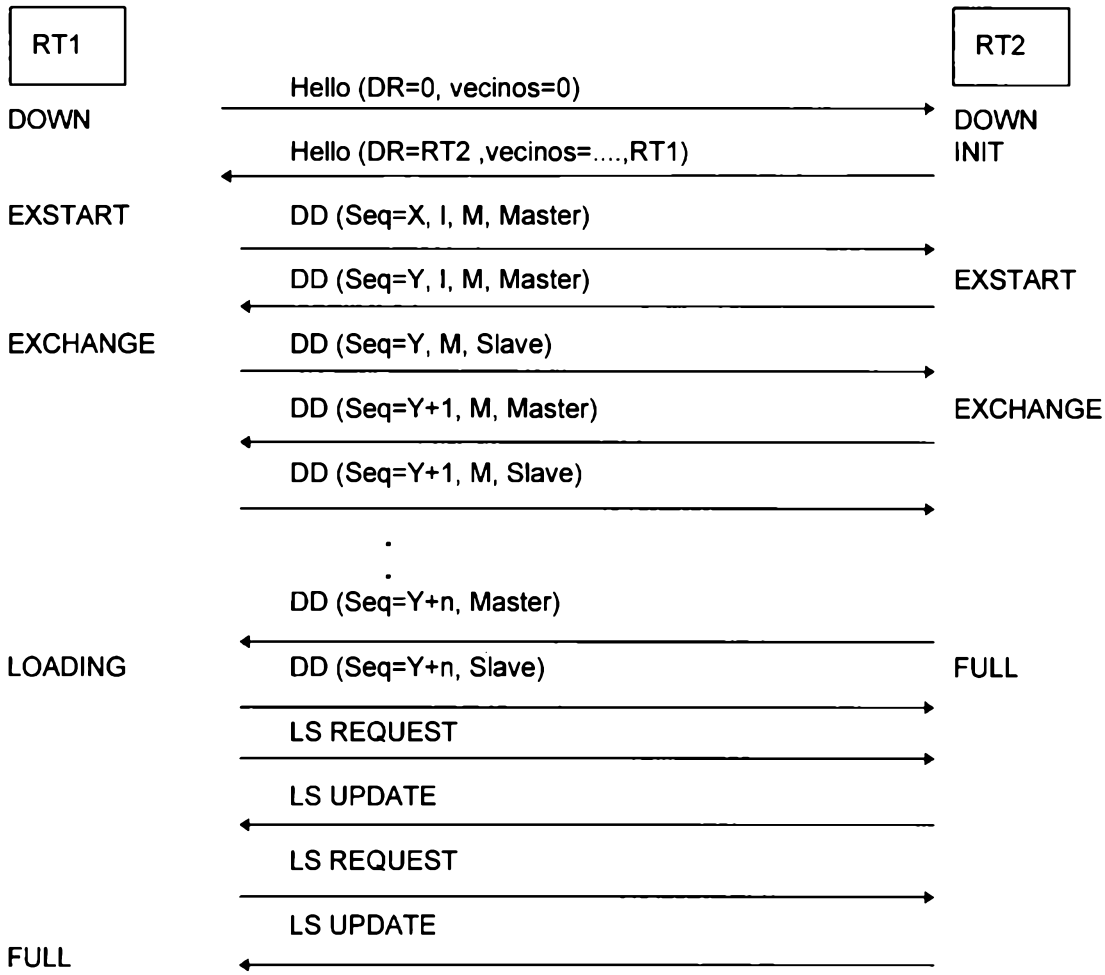
Vamos a denominar al router que realiza el cálculo, router X .El router X procederá a examinar la lista de vecinos conectados a la red y en comunicación bidireccional con él por lo que el estado de todos es 2-way o mayor incluyendo al router X. De esta lista quedan descartados los que tienen prioridad 0 y se procederá de acuerdo a los siguientes pasos:

- Considerar los valores actuales del DR y BDR.
- Calcular el BDR: Si ninguno o más de un router se declara BDR en el paquete helio se considerará al que tenga mayor prioridad, en caso de empate se elegirá al de mayor router ID. En el caso que ninguno se declare se debe excluir a los que se declaran DR.
- Calcular el DR: Si más de un router se declara DR en el paquete helio se considerará al que tenga mayor prioridad, en caso de empate se elegirá al de mayor router ID. En el caso que ninguno se declare se elije al BDR como DR.
- Si el router X es el nuevo DR o BDR se repite el 2do y 3er paso para asegurar que el router X no sea elegido DR y BDR a la vez.
- Como resultado de este cálculo el router X es ahora el DR, BDR o nada y el estado de su interface cambia a DR, Backup o DROthers respectivamente.
- En caso de ser elegido DR o BDR, debe enviar paquetes hello al resto de los vecinos DROthers invocando el evento Start y también formar nuevos adyacentes invocando el evento AdjOK? para todos los vecinos con estado 2-way o superior.

5.10.2.Protocolo Hello.

Es el algoritmo necesario para establecer la adyacencia entre vecinos y se realiza cuando una interface comienza a ser operacional.

El siguiente ejemplo muestra el estado de cambios de vecindad por cada router, RT2 es el DR de la red y RT1 empieza a enviarle hello sin saber que su vecino es el DR y está totalmente actualizado.



5.10.3.Eventos generados por cambios en la tabla de ruteo.

Un cambio en la tabla de ruteo hace que routers de borde tomen ciertas acciones. El costo o camino ha cambiado en una entrada de la tabla. Si el destino es una red o un router externo, no es solo un simple cambio de router externo pues se debe generar un nuevo aviso para cada área conectada incluyendo el backbone y si es eliminado y no advertido para un área en particular debe ser avisado con costo LSInfinity.

Una entrada configurada con enlaces virtuales ha cambiado por lo que su destino es un router de borde y si se recupera debe generar el evento InterfaceUp para formar la adyacencia a través del protocolo hello caso contrario deberá generar el evento InterfaceDown para destruir las adyacencias. Si solo ha cambiado el costo y está totalmente conectado a sus adyacentes deberá enviar avisos de router y posiblemente causará cambios en otras tablas.

5.10.4.Procedimiento de flooding.

Primera parte:

Determinar si el aviso es más reciente y actualizarlo en la base topológica.

Un paquete de actualización puede contener distintos avisos y fluye un salto desde su punto de origen, éste paquete debe ser aceptado por un paquete de reconocimiento.

El flooding comienza cuando un paquete de actualización es recibido. Entre los chequeos para ver si se procesa o no están las siguientes consideraciones:

El estado de su conversación debe estar en estado mayor o igual que Exchange.

Para cada uno de los avisos del paquete verifica:

- Que exista el tipo del aviso.
- Si es un aviso externo y es un área stub se descarta el aviso y se procede con otro.
- Si la edad llegó a Maxage, el aviso no existe en la base topológica y los vecinos del router están en estado Exchange o Loading se envía un paquete de reconocimiento.
- Si el aviso existe en la base topológica y el aviso es más reciente:
  - Si ha sido actualizado por flooding en menos de MinLSArrival, se descarta el nuevo aviso sin aceptarlo.
  - Caso contrario se fluye por sus interfaces de acuerdo a su estado de interface o el de su conversación.
  - Se remueve la vieja copia y se guarda el nuevo aviso pudiéndose recalcular la tabla de ruteo.
  - Se acepta el aviso enviando un paquete de reconocimiento.
  - Se verifica si él mismo ha originado el aviso tomando la acciones pertinentes.
  - Si el aviso está pedido en la lista de Pedidos, ha ocurrido un error en el proceso de intercambio por lo que deberá parar el flooding y reiniciar el intercambio.
  - Si el aviso es la misma instancia y está en la lista de retransmisiones, el router está esperando que sea aceptado por lo que lo remueve de la lista de retransmisiones simulando la aceptación.
  - En este punto acepta el aviso enviando un paquete de reconocimiento.

- El aviso de la base topológica es más reciente que el enviado, por lo que lo descarta, envía en paquete de actualización a su vecino, lo copia de la base porque es el más reciente. No incorpora el aviso que le enviaron a su lista de retransmisiones ni lo acepta.

Selección del enlace más reciente.

Cuando un router encuentra dos instancias de un aviso debe determinar cual es la más reciente, comparándolo contra la base de enlaces durante el intercambio de información.

Un aviso de enlace se identifica por su LS type, Link State ID y Advertising Router y para determinar el más reciente se chequea los campos LS sequence number, Ls age y LS checksum de la siguiente manera:

- El aviso más reciente es aquel que posea el LS sequence number más nuevo. Si ambas instancias tienen el mismo número de secuencia continúa.
- Se considera más reciente al que tenga el LS checksum mayor. Si son iguales continúa.
- Se considera más reciente al que tenga Ls age igual a MaxAge. Si ninguno lo tiene continúa.
- Si las edades de ambos difieren en más de MaxAgeDiff, se considera más reciente al menor o de edad más joven. Caso contrario los avisos tienen la misma instancia.

Agregar un aviso a la base topológica.

Agregar un aviso como resultado del flooding o porque el mismo router lo ha originado, puede causar el recálculo de la tabla de ruteo, siempre que la instancia sea más reciente, comparándose: el campo opción del aviso, una de las instancias tiene edad Maxage y la otra no, la longitud del aviso ha cambiado, el cuerpo del aviso ha cambiado sin tener en cuenta al número de secuencia y el checksum.

Para los avisos de tipo router y de red, se recalcula toda la tabla de ruteo por cada área; para los avisos sumariales y externos se recalcula solo la mejor ruta para tal destino y cualquier vieja instancia del aviso es removida de la base y de todas las listas de retransmisiones de sus vecinos.

Segunda Parte:

Selección de las interfaces por la cual distribuir el aviso recibido, actualizarlo en la base y lista de retransmisiones de los vecinos y mantener la lista de pedidos .

Cuando el aviso recibido es más reciente, el router debe distribuirlo por sus interfaces, dependiendo de la interface y el tipo del aviso.

Los avisos se fluyen por todo el sistema autónomo y por ende todas las interfaces excluyendo los enlaces virtuales y conexiones a redes stub.

El resto de los avisos se distribuyen solo por las interfaces del área, si el área es el backbone incluye enlaces virtuales.

1. Por cada vecino conectado a la interface se verifica:
   - Si el estado es menor a Exchange no participa del flooding y tomamos el siguiente enlace.
   - Si el enlace no está totalmente adyacente se debe examinar la lista de pedidos:
     - Si el aviso es menos reciente, examinar el siguiente enlace.
     - Si son la misma instancia o es más reciente eliminarlo de la lista de pedidos y examinar el siguiente enlace.
     - Si el aviso es recibido del vecino examinar el siguiente.
     - En este punto el aviso es la última actualización, por lo que se agrega a la lista de retransmisiones.
2. En este punto el router no tiene que fluir el aviso y sigue por la siguiente interface.
3. Si el aviso recibido fue enviado por el DR o el BDR seguramente el resto de los routers han recibido el aviso por lo que no se envía por ninguna interface.
4. Si la interface es BDR seguramente el DR le envió el aviso por lo que se descarta el envío por esta interface.
5. En este punto el aviso debe fluirse vía paquete de actualización.

Recibiendo avisos generados por si mismo.

Es común que el router reciba un aviso originado por si mismo en el flooding. Esto ocurre si el aviso tiene el mismo Router Id que el router y si es un aviso de red debe tener el ID del enlace igual a la dirección IP de algunos de los routers de la red.

Si la instancia es más nueva deberá incrementar el número de secuencia y originar una nueva instancia del aviso.

Envío de paquetes de reconocimiento.

Cada aviso recibido debe ser aceptado, usualmente enviando paquetes de reconocimiento o bien confirmados con paquetes de actualización como ocurre en el protocolo Hello.

Se pueden aceptar varios avisos en un simple paquete, y se puede enviar de 2 formas: a intervalos de tiempo o directamente ni bien lo recibe en calidad de respuesta. La primera forma facilita la aceptación de varios avisos y a varios vecinos por multicasting y también permite el ruteo a varios routers de la red.

El proceso de envío de paquetes de reconocimiento depende del estado de las interfaces y las circunstancias siguientes:

Circunstancias	Acción tomada en estado Backup	Acción tomada en el resto de los estados
Los LSAs se fluyen por las interfaces	No envía paquetes de reconocimiento.	No envía paquetes de reconocimiento.
El LSA es más reciente pero no ha sido fluido.	Si ha sido enviado por el DR, enviará un paquete de reconocimiento en un intervalo de tiempo, no envía en caso contrario.	Enviará un paquete de reconocimiento en un intervalo de tiempo.
El LSA es duplicado y se lo considera como una aceptación implícita.	Si ha sido enviado por el DR, enviará un paquete de reconocimiento en un intervalo de tiempo, nada en caso contrario.	Enviará un paquete de reconocimiento en un intervalo de tiempo.
El LSA es duplicado y no se lo considera como una aceptación implícita.	Se envía un paquete de reconocimiento directamente.	Se envía un paquete de reconocimiento directamente.
La edad del LSA es igual a Maxage y no hay una ocurrencia del aviso y ninguno de sus vecinos están en estado Exchange o Loading.	Se envía un paquete de reconocimiento directamente.	Se envía un paquete de reconocimiento directamente.

Recepción de paquetes de reconocimiento.

Si el vecino está en estado menor que Exchange el paquete se descarta.

Si existe una instancia del aviso en la lista de retransmisiones y es una aceptación para la misma instancia se elimina de la lista.

5.10.5.Generación de avisos.

Un router puede originar varios tipos de avisos y los distribuye para cada área a la que pertenece describiendo los estados de sus enlaces.

Un DR origina avisos para la red.

Un router de borde origina avisos sumariales a cada destino inter-area que conozca.

Un router externo avisa a cada destino externo que conozca.

Se originan con estos eventos:

- Disparo del timer de refresco de LS: hay uno por cada aviso de estado de enlace que un router ha generado. El período de actualización de avisos es necesario para mantener espacio, los avisos sumariales y externos no se refrescan.
- Cambios en la interface.
- El DR de una red ha cambiado por lo que el nuevo DR origina avisos de red.

- Uno de los routers vecinos ha cambiado desde o hacia estado FULL.

Para routers de borde:

- Un route intra-area de tipo 1 ha sido agregado, borrado, modificado en la tabla de ruteo por lo que se genera un aviso sumarial para esta ruta enviándolo a cada área conectada incluyendo el backbone.
- Un router inter-area de tipo 2 ha sido agregado, borrado, modificado en la tabla de ruteo por lo que se genera un aviso sumarial para esta ruta enviándolo al área.

Para routers externos:

- Se genera un protocolo de ruteo externo EGP por el cual se generan los avisos externos.

Construcción de la lista de avisos de enlaces de ruteo: se consideran los siguientes pasos para cada interface de un router en el área A:

- Si la red no pertenece al área A, el enlace no se adicionan en los avisos y se sigue con la siguiente interface.
- Elseif el estado de la interface es Down, el enlace no se adiciona.
- Elseif el estado de la interface es Point to Point adicionar el enlace de acuerdo a :
  - Si el router vecino es un adyacente y su estado es Full , agregar el siguiente aviso:
    Tipo :1.
    Link ID: Router ID del vecino.
    Link Data : Dirección IP de la interface.
  - Si se conoce la dirección IP del vecino, agregar el siguiente aviso:
    Tipo: 3 (red stub).
    Link ID: dirección IP del vecino.
    Link Data: la máscara 0xffffffff (host route).
    Costo : costo de la interface
- Elseif si el estado de la interface es Loopback, agregar el siguiente aviso:
  Tipo : 3 (red stub).
  Link ID : dirección IP de la interface.
  Link Data : la máscara 0xffffffff (host route).
  Costo : 0.
- Elseif si el estado es Waiting , agregar el siguiente aviso:
  Tipo: 3 (red stub).
  Link ID : dirección IP de la red.
  Link data : máscara de la red.
- Elseif hay un DR para la red y el router está Full y es adyacente del DR, o el router es el DR y está Full y es adyacente de al menos otro router, agregar el siguiente aviso:
  Tipo: 2 (red de tránsito).
  Link ID: dirección IP de la interface del DR en la red.
  Link Data: dirección IP de la interface.
- Otherwise agregar un aviso como si estuviese en estado de Waiting.

Construcción de la lista de avisos de red:

El DR de la red origina los aviso en estado Full y es adyacente de al menos otro router de la red y los avisos fluyen solo en el área donde está contenida la red. Los routers listados en los avisos son los que eran full y adyacentes con el DR y se identifican con su router ID y el costo es 0 para todos los routers conectados a la red.

Construcción de la lista de avisos sumariales:

Describe la ruta a un simple destino que es externo al área pero pertenece al AS y fluyen solo en una simple área y son generados por routers de borde.

Para saber a quien advertir los avisos en un área A, se procesa cada entrada a la tabla:

- Solo destinos de red y de routers de borde con otros AS son advertidos. Los routers de borde no son analizados.
- Routers externos nunca son advertidos por lo que esta entrada no es analizada.

- Elseif el camino está en el área A, no se generan avisos sumariales para esta ruta.
- Elseif el destino es un router de borde de un AS, se agrega el siguiente aviso:
  Tipo: 4.
  Link ID : ID del router de borde del AS.
  Métrica : costo en la tabla de ruteo para esa entrada.
- Else el destino es una red.
  Para un router inter-area:
  Tipo : 3.
  Link ID : dirección de la red.
  Métrica : costo en la tabla de ruteo para esa entrada.

  Para un router intra-area la red está contenida en una de las áreas en que el router se conecta directamente por lo que se agrega el siguiente aviso:
  Tipo: 3.
  Link ID: rango (En un área se define una lista de rangos direccionales [dirección, máscara]).
  Costo: el menor costo a cualquier componente de la red.

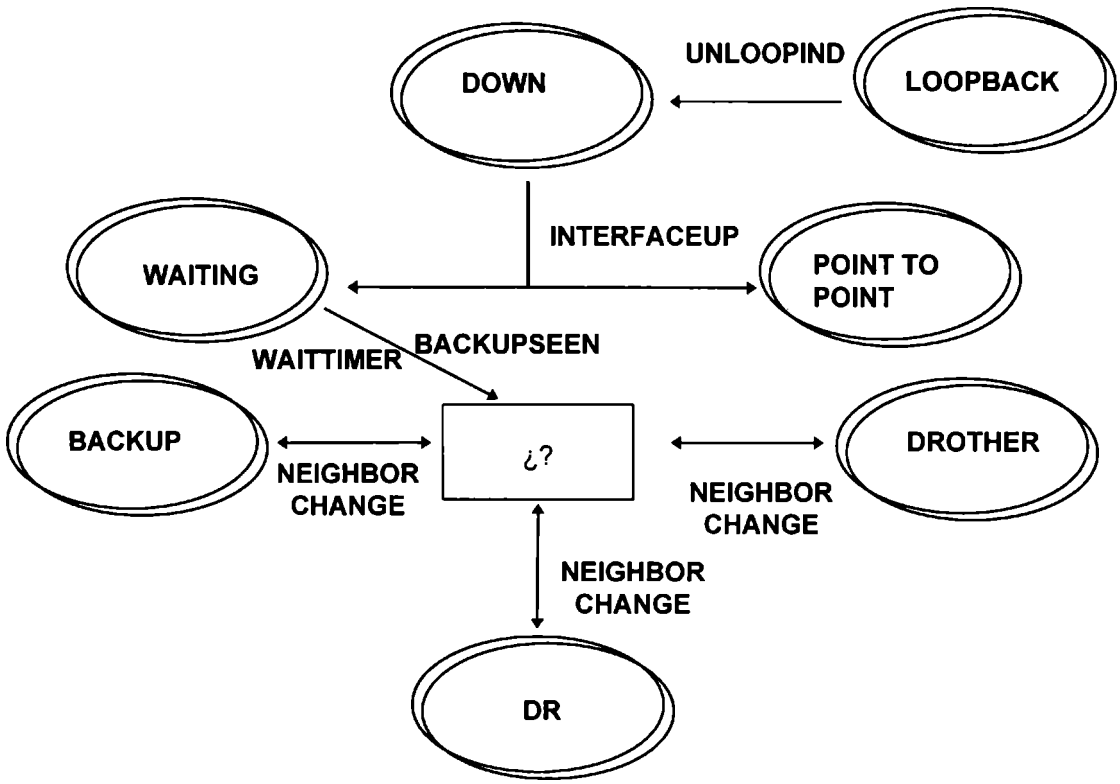
Nota: Si un router advierte un aviso sumarial, para un destino inalcanzable, genera un nuevo aviso sumarial con métrica LSInfinty.

Construcción de la lista de avisos sumariales:

- Cada AS describe enlaces externos para destinos externos al AS, fluyen por todo el AS y son generados por routers de borde del AS para cada ruta que sabe externa aunque tenga otro protocolo de ruteo como EGP. La métrica para el tipo 1 es comparada con la métrica del enlace y para el tipo 2 se asume que el costo es mayor que por cualquier camino intra-AS.

5.10.6.Diagrama de estados de la interface y eventos que lo causan.

Los arcos del grafo son rotulados con el evento que causa el cambio de estado.



Nota 1 : Los eventos **INTERFACE DOWN y LOOPIND** suceden desde cualquier estado por eso no figuran en el gráfico.

ESTADOS:

DOWN: Es el estado de interface inicial donde los protocolos de bajo nivel indican su inutilidad y los protocolos de tráfico no pueden operar.

LOOPBACK: La interface del router está en loop por hardware o software.

WAITING: El router está tratando de determinar la identidad del router DR, monitoreando los paquetes helios.

POINT TO POINT: La interface está operando y conectada por lo que el router intentará formar sus adyacentes y los paquetes hello son enviados a sus vecinos cada hellointerval .

DROTHER: La interface es sobre una red broadcast NBMA en la que existe otro router que ha sido elegido DR y puede existir o no el BDR.

BACKUP: El router que está en éste estado es el BDR y empieza a operar porque el DR falla.

DR: Es el DR de la red. Las adyacencias están establecidas con todos los routers de la red.

EVENTOS:

INTERFACEUP: Puede salir del estado Down porque ya está operable, en un enlace virtual se determina a través del cálculo del SPF.

WAIT TIMER: Se dispara cuando ha finalizado la espera requerida para la elección del BDR.

BACKUP SEEN: El router ha detectado la existencia o no del DR y BDR, pudiendo ser de 2 formas: 1) se recibe un paquete por un vecino clamando ser el DR ó 2) se recibe un paquete por un vecino clamando ser el DR e indicándole que el no es el BDR. Caso contrario el router aparece en la lista de vecinos del paquete hello y sale del estado de waiting como DROTHER.

NEIGHBOR CHANGE: Hay cambios en los vecinos asociados a la interface, por lo que el DR o BDR necesitan ser recalculados. Los siguientes eventos de un vecino produce cambios:

- Establecer comunicación con un vecino, seguramente el vecino ya pasó por el estado 2-way o más.
- Dejar de comunicarse con el vecino, seguramente el vecino pasó al estado de inicio o uno menor.

Alguno de los vecinos se declara o deja de declararse DR o BDR, detectándose en la lista de vecinos del hello. La prioridad del router ha cambiado, detectándose en la lista de vecinos del hello.

UNLOOPIND: Se indica que la interface no está en loop detectado por los protocolos de bajo nivel o administración de red.

LOOPIND: Desde cualquier estado se indica que la interface está en loop detectado por los protocolos de bajo nivel o administración de red y pasa al estado Loopback.

INTERFACEDOWN: Desde cualquier estado ,los protocolos de bajo nivel indican que la interface no funciona, pasa al estado Down, destruyendo la comunicación con sus vecinos a través del evento KillNbr.

5.10.7.Diagrama de estados de los vecinos y eventos que lo causan.

Un router OSPF conversa con sus vecinos. Cada conversación se describe como una estructura de vecindad, se monta en una interface y se identifica por el router ID o dirección IP del vecino y contiene toda la información necesaria para formar adyacencia con sus vecinos.

ESTADO: Nivel que da funcionalidad a la conversación entre vecinos.

TIEMPO DE INACTIVIDAD: Su disparo indica que no se ha visto un paquete hello recientemente.

MASTER/SLAVE: Se negocia la relación master / slave para saber quien es el master que envía los paquetes de descripción de la base y el slave que le responde.

NUMERO DE SECUENCIA: Identifica individualmente un paquete DD, se setea en el envío con un valor por ejemplo la hora y el master lo incrementa en cada nuevo DD enviado. Este número en el slave significa el último paquete recibido de sus master.

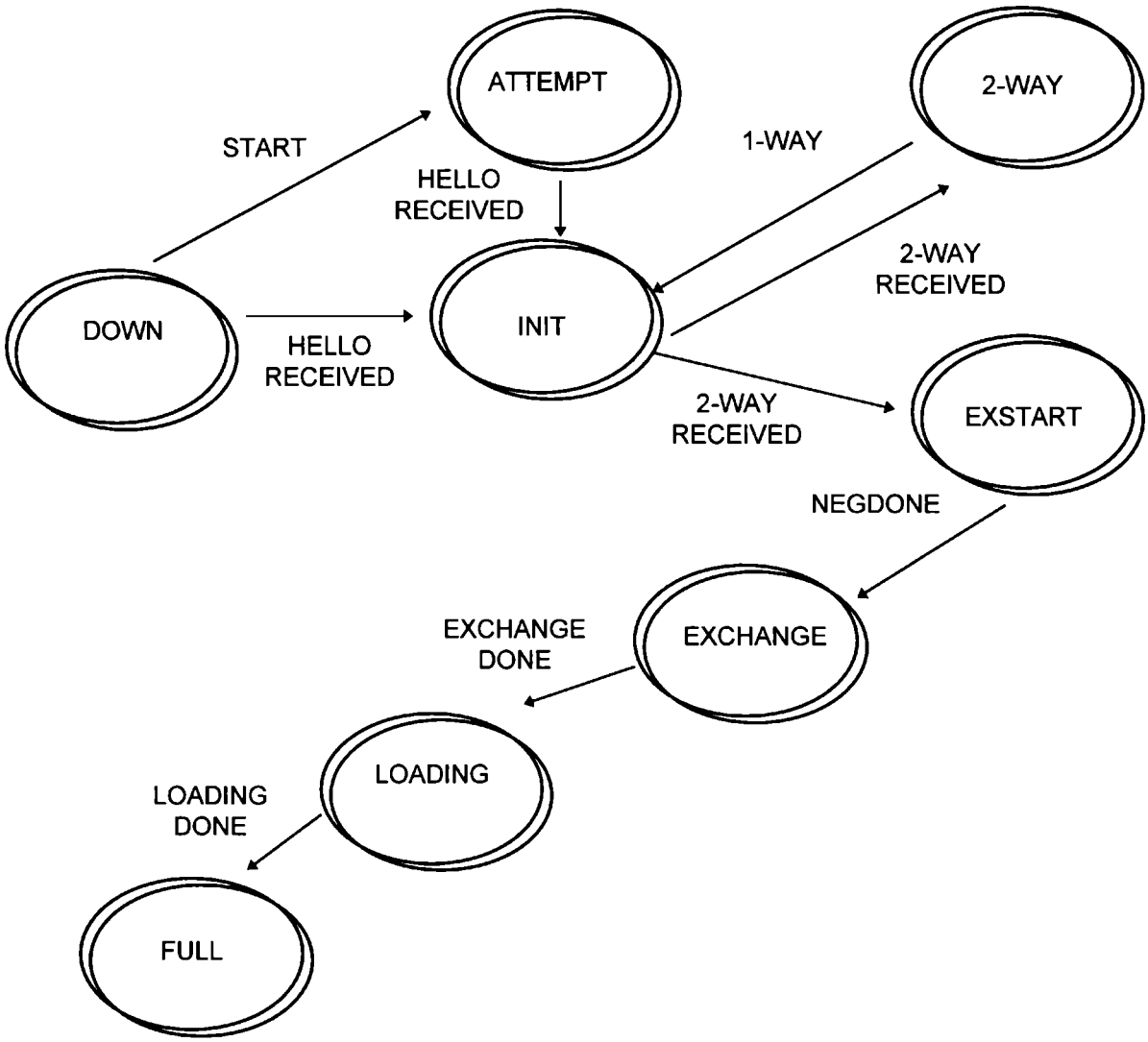
ID DEL VECINO
PRIORIDAD DEL VECINO
DIRECCIÓN IP DEL VECINO
DR DEL VECINO
BDR DEL VECINO

LISTA DE AVISOS RETRANSMITIDOS: Es la lista de avisos obtenida sin tener su reconocimiento por lo que son retransmitidos hasta ser reconocidos o se destruya la adyacencia.

LISTA DE AVISOS SUMARIALES: Se obtiene de la base topológica en el momento que el vecino pase a estado de intercambio enviando paquetes DD.

LISTA DE AVISOS PEDIDOS: La lista se crea a partir de los paquetes DD recibidos y enviará paquetes LS Request al vecino hasta que quede vacía.

El siguiente gráfico muestra los estados de una comunicación entre vecinos, desde el estado más bajo , los intermedios y el estado en el que logran ser adyacentes. Los arcos del gráfico indican los eventos que producen los cambios de estado por el protocolo hello.



ESTADOS:

DOWN: Estado inicial de una conversación e indica que no hubo información reciente de su vecino. En redes no broadcast, los paquetes hello se envían en forma más reducida en estado Down.

ATTEMPT: Indica que si bien no se ha recibido información reciente del vecino, se intenta contactarlo a través de hello enviados en intervalos determinados de tiempo.

INIT: Se ha visto un paquete hello del vecino, pero todavía no se ha intentado una comunicación bidireccional por lo que el router no aparece en el paquete hello. Todos los vecinos en este estado o mayor, se listan en el paquete hello enviado por la interface asociada.

2-WAY: La comunicación es bidireccional. En este estado o mayor se eligen el DR y BDR.

EXSTART: Su objetivo es negociar que router será el master y slave de la comunicación decidiendo el número inicial de la secuencia. A partir de este estado los vecinos se llaman adyacentes.

EXCHANGE: Los routers adyacentes se intercambian la base de estados de enlace a través de paquetes de intercambio de la base de avisos .

LOADING: Los routers adyacentes solicitan a través de paquetes Link State Request, los avisos más recientes descubiertos en el estado anterior Exchange.

FULL: Los routers adyacentes que intercambiaron sus avisos están completos. Estos adyacentes no aparecerán en avisos de red o router.

EVENTOS:

HELLO RECEIVED: Se ha recibido un paquete hello de su vecino.

START: Indica que un paquete hello puede enviarse a su vecino cada hello intervalos. Usado solo por vecinos no asociados a una red broadcast.

2-WAY RECEIVED: La comunicación es bidireccional. Se notifica porque el router se ha visto en el paquete hello del otro router.

NEGOTIATION DONE: Se negoció la relación Master Slave intercambiándose un número de secuencia para los paquetes de intercambio de la base de avisos.

EXCHANGE DOWN: Ambos routers se intercambiaron toda la secuencia de paquete de intercambio y cada uno conoce que parte de su base de avisos está desactualizada.

SEQ NUMBER MISMATCH: Se ha perdido el número de secuencia o reiniciado el envío por algún error.

BAD LSREQ: Se ha solicitado un aviso no contenido en la tabla. Indica un error en el proceso de sincronización.

LOADING DONE: Se ha transmitido toda la base de avisos, entonces el pedido de avisos vendrá vacío, luego de procesar los paquetes de actualización.

ADYOK?: Se decide la adyacencia entre vecinos, siguiendo con unos y destruyéndola con otros.

EVENTOS QUE REVIERTEN EL ESTADO:

1-WAY: Se recibe del vecino un paquete hello en el que no es mencionado, por lo que ya no existe una comunicación bidireccional.

KILLNBR: Indica que toda comunicación con el vecino es imposible, forzando al vecino al estado Down.

INACTIVITY TIMER: Se disparó este timer indicando que no se ha visto un paquete hello de sus vecinos, por lo tanto el router revierte al estado Down.

LLDOWN: Es una indicación de bajo nivel de que el vecino es irrecuperable, por lo tanto el router revierte su estado a Down.

Capítulo 6

Análisis del OSPF en el MaRS.

6.1.Introducción.

El simulador MaRS es un simulador de ruteo que tiene implementado 3 algoritmos: SPF, Segal y ExBF, el primero es un algoritmo de estados de enlace y el resto algoritmos de distancia vectorial, todos son dinámicos o adaptativos.

Con la nueva componente del protocolo de ruteo OSPF que desarrollamos, la modalidad de trabajo es distinta, es un protocolo que utiliza el algoritmo SPF, tiene una organización estructural y por ende un ruteo más preciso. Su comportamiento va cambiando según los estados de los routers, ya sea por cada una de sus interfaces como también por sus conversaciones. Está organizado a nivel áreas, tipos de routers y tipos de conexiones.

El escenario OSPF que elegimos para simular está formado por una sola área de routers, por lo tanto todos los routers son de tipo internos, pudiendo utilizar todos los tipos de conexiones de routers: a red de tránsito, a red stub y point to point.

El comportamiento que implementamos comprende la elección del DR y BDR, la realización del protocolo Hello al levantarse cada enlace y el procedimiento de flooding informando los cambios o avisos vía paquetes de actualización.

Definimos las estructuras de conversación y de interfaces haciendo variar sus estados de acuerdo a los esquemas presentados en el capítulo 5.

Para la implementación, además de las nuevas estructuras y funciones del OSPF, realizamos modificaciones en estructuras de datos y código ya existentes en el MaRS.

6.2.La inserción de la componente OSPF.

Creamos dos archivos, el header para la estructura de datos, llamado ospf.h y otro para las rutinas, llamado ospf.c.

En el archivo ospf.h definimos la estructura de un router OSPF y estructuras secundarias y declaramos distintas constantes. [Apéndice A1]

En el archivo ospf.c definimos las distintas funciones que caracterizan el funcionamiento del protocolo. [Apéndice A2]

Definimos el tipo de la nueva componente en el archivo comptypes.h . [Apéndice A3]

Agregamos una entrada para la nueva componente en el arreglo component_types en el archivo comptypes.c . [Apéndice A4]

En el archivo packet.h agregamos las nuevas estructuras de los paquetes que maneja OSPF. [Apéndice A5]

En el archivo link.h agregamos a la estructura de link el tipo de conexión del enlace y el nombre de la red si el enlace estuviese conectado a una. [Apéndice A6]

Modificamos en los archivos node.c y link.c el tratamiento de la falla y reparación para tratar el caso particular del OSPF. [Apéndice A7 y A8 respectivamente].

Modificamos en el archivo sim.h las distintas funciones de conversión de unidad de tiempo a ticks (unidad básica de tiempo del MaRS) y viceversa. [Apéndice A9]

Modificamos el Makefile para incluir la nueva componente.

6.3.Funciones y eventos manejados en el OSPF.

Algunas funciones las rehusamos, como por ejemplo las definidas en los routers SPF que implementa el MaRS : el recálculo de la tabla de ruteo, el camino más corto, algunas búsquedas en tablas.

Las funciones principales que implementamos y hacen a su comportamiento son:

Elegir el DR.

Procesar cada uno de los paquetes de acuerdo al estado de la conversación.

Actualizar la base topológica o de estados de enlace.

Generar los avisos.

Fluir los avisos.

Mostrar el flujo de paquetes, el estado de la base topológica o de enlaces y la tabla de ruteo.

Entre los eventos propios del OSPF que implementamos, citamos:

Ev_route_processing: Se invoca cada vez que el router recibe un paquete de ruteo y lo procesa según el tipo de paquete y el estado en que se encuentre.

Ev_ospf_broadcast: Se invoca cada cierto intervalo de tiempo para enviar paquetes Hello a sus vecinos.

Ev_ospf_down_interf: Se invoca desde componentes link cuando se produce la falla del enlace.

Ev_ospf_up_interf: Se invoca desde componentes link cuando se repara un enlace y desde la propia componente de ruteo cuando se inicia la simulación y debe levantar todos sus enlaces.

El resto de los eventos son comunes a todas las componentes de ruteo creadas por el MaRS, y solo algunas han sufrido modificaciones, como por ejemplo la creación, comienzo e inicialización de una componente OSPF.

- **Eventos tratados por la componente de ruteo OSPF. Codificado en: ospf.c , función ospf_action().**

Eventos	¿Qué hace?	Eventos que encola (•) o ejecuta (∈) :	Eventos contenido en :
Ev_reset ospf_reset()	Inicializa número de nodos vecinos , número de secuencia , costo de enlace y la conversación.		
Ev_create ospf_create()	Crea la componente e inicializa los parámetros de configuración.		
Ev_del ospf_delete()	Borra la componente de ruteo.		
Ev_neighbor ospf_neighbor()	Solo permite tener como vecino a una componente nodo y actualiza bases (RT,GTT,SNT).		
Ev_uneighbor ospf_uneighbor()	Se remueve el vecino y se actualiza la tabla topológica local.		

Ev_start ospf_start()	Elige el DR si es un router conectado a una red y encola los eventos ospf_broadcast y ospf_up_interface para ejecutarlos inmediatamente con el fin de levantar sus enlaces e iniciar el protocolo hello.	• ev_ospf_broadcast • ev_ospf_up_interf	• Ospf.c .
Ev_route_processing ospf_processing()	Según los estados del router y los tipos de paquetes a procesar se enviarán distintos tipos de paquetes de acuerdo al **estado y requerimientos del router**, al destino correspondiente. Se actualizan las tablas. Obtiene el próximo paquete a rutear encolándolo con el evento ev_route_processing.	• ev_route_processing ∈ ev_node_produce	• Ospf.c . • Node.c .
Ev_ospf_broadcast ospf_broadcast()	Encola ev_ospf_broadcast para un. tiempo t de broadcast calcula métricas , genera paquetes hello para ser procesados por broadcast encolando este evento y los envía a sus nodos para ser distribuidos, respetando los estados de la conversación establecida; al cambiar de estado empezará a realizar el protocolo hello con el resto a medida que se reciban, y luego procesen los paquetes hello	• ev_ospf_broadcast	• Ospf.c
Ev_mk_peer	Devuelve el puntero de la componente de ruteo.		
Ev_stop	Devuelve el puntero de la componente de ruteo.		

6.4.Modificaciones en node.c y link.c.

En el componente node modificamos las siguientes funciones:

- node_failure(): si la componente de ruteo conectada al nodo es OSPF el envío del paquete de falla del nodo no debe realizarse. El paquete RT_NODE_SHUTDOWN no existe en OSPF. [Apéndice A7]
- node_repair(): como en el caso anterior, si la componente de ruteo conectada al nodo es OSPF el envío del paquete de reparación del nodo no debe realizarse. El paquete RT_NODE_WAKEUP no existe en OSPF. [Apéndice A7]

En el componente link modificamos las siguientes funciones:

- link_failure(): si las componentes de ruteo conectadas a los nodos correspondientes al link son OSPF se ejecuta el evento EV_OSPF_DOWN_INTERF por cada uno de estos nodos y no se realiza el envío del paquete de falla del link. El paquete RT_LINK_SHUTDOWN no existe en OSPF. [Apéndice A8] También agregamos el tratamiento del nuevo parámetro de falla. (Ver en 5.5).
- link_repair():si las componentes de ruteo conectadas a los nodos correspondientes al link son OSPF se ejecuta el evento EV_OSPF_UP_INTERF por cada uno de estos nodos y no se realiza el envío del paquete de falla del link. El paquete

RT_LINK_WAKEUP no existe en OSPF. [Apéndice A8] También agregamos el tratamiento del nuevo parámetro de falla. (Ver en 6.5).

- link_start(): agregamos el tratamiento del nuevo parámetro de falla. (Ver en 6.5).

6.5.Nuevos parámetros de inicialización.

Agregamos para el componente link los siguientes parámetros:

- **Conexión**: indica el tipo de enlace, puede ser PTOP, TORED o TOSTUB, para enlace point-to-point, a una red de tránsito o a una red stub, respectivamente.
- **IP Red**: es la dirección de red si el tipo de enlace es TORED.
- **TFalla**: es el tiempo en segundos en que el link va a fallar. Si se indica este parámetro el link falla en el tiempo indicado una sola vez, sin tener en cuenta el parámetro ya existente "Mean time btw failures" .
- **TRepara**: es el tiempo en segundos en que el link se va a reparar después de su falla. Si se indica este parámetro el link se repara en el tiempo indicado una sola vez, sin tener en cuenta el parámetro ya existente "Mean time to repair" .

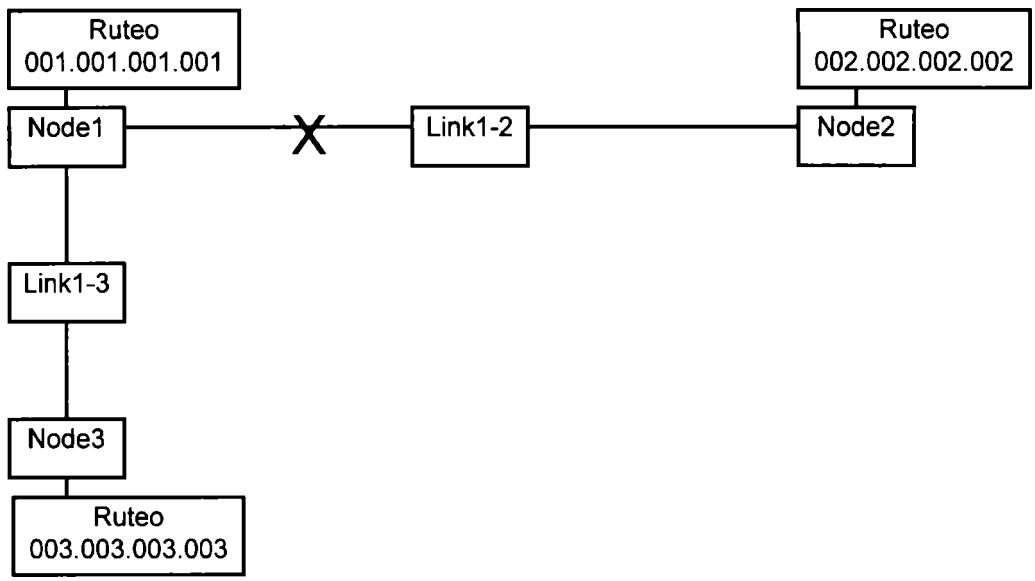
Agregamos para el componente de ruteo OSPF los siguientes parámetros:

- **Prioridad**: indica la prioridad del router usada en la elección del DR.
- **TraceTR**: Si es igual a uno arma el archivo **salidatr** con el estado de su tabla de ruteo cada vez que ésta se actualiza. Si es igual a cero no genera el archivo.
- **Trace**: Si es igual a uno arma el archivo **salida** con el seguimiento de los paquetes que envía y recibe. Si es igual a cero no genera el archivo.

6.6. Muestras realizadas.

6.6.1.Conexión point-to-point.

El siguiente target system configura una conexión de tres routers con enlaces de tipo point-to-point.



La siguiente muestra es el tráfico de paquetes OSPF en el router 001.001.001.001 (Node1) desde la inicialización del escenario simulado, con la caída de la interface Link1-2 y su posterior reparación.

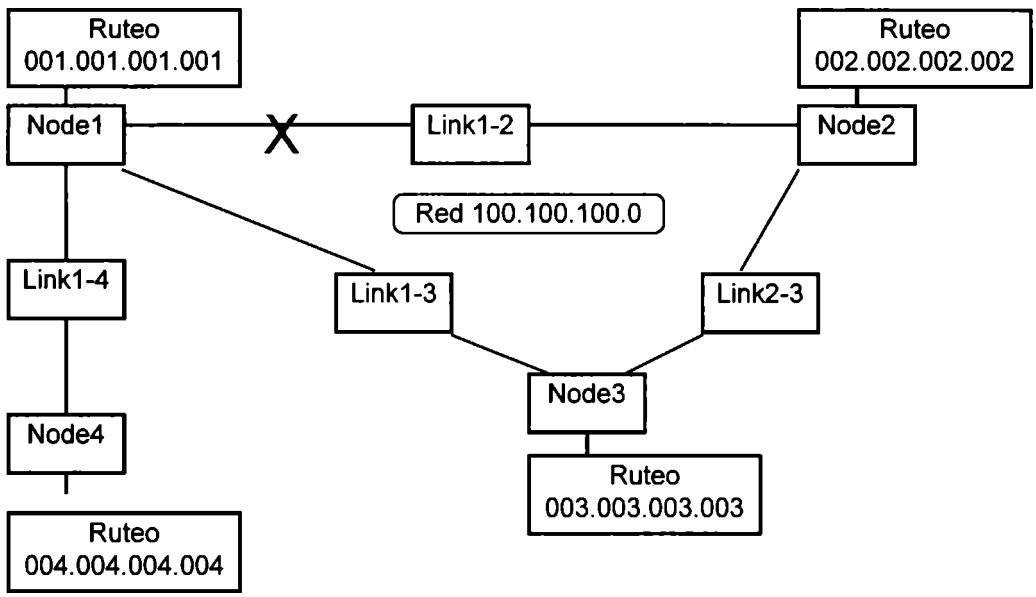
```
18:48:42 HELLO RID:001.001.001.001 Dest:node2 Long:128 DR:001.001.001.001 Neigrs:
18:48:42 HELLO RID:001.001.001.001 Dest:node3 Long:128 DR:001.001.001.001 Neigrs:
18:48:42 HELLO RID:002.002.002.002 Dest:node1 Long:128 DR:002.002.002.002 Neigrs:
18:48:42 HELLO RID:003.003.003.003 Dest:node1 Long:128 DR:003.003.003.003 Neigrs:
18:48:51 HELLO RID:001.001.001.001 Dest:node2 Long:136 DR:001.001.001.001
         Neigrs:003.003.003.003 002.002.002.002
18:48:51 HELLO RID:001.001.001.001 Dest:node3 Long:136 DR:001.001.001.001
         Neigrs:003.003.003.003 002.002.002.002
18:48:51 HELLO RID:003.003.003.003 Dest:node1 Long:132 DR:003.003.003.003
         Neigrs:001.001.001.001
18:48:51 HELLO RID:002.002.002.002 Dest:node1 Long:132 DR:002.002.002.002
         Neigrs:001.001.001.001
18:48:51 DD RID:001.001.001.001 Dest:node3 Long:32 I:1 M:1 MS:1 Seq:1
18:48:51 DD RID:001.001.001.001 Dest:node2 Long:32 I:1 M:1 MS:1 Seq:1
18:48:51 DD RID:003.003.003.003 Dest:node1 Long:32 I:1 M:1 MS:1 Seq:1
18:48:51 DD RID:001.001.001.001 Dest:node3 Long:32 I:1 M:1 MS:0 Seq:1
18:48:51 DD RID:002.002.002.002 Dest:node1 Long:32 I:1 M:1 MS:1 Seq:1
18:48:51 DD RID:001.001.001.001 Dest:node2 Long:32 I:1 M:1 MS:0 Seq:1
18:48:51 DD RID:003.003.003.003 Dest:node1 Long:52 I:0 M:0 MS:1 Seq:2
--> LStype:1 LSID:003.003.003.003 AdvRouting:003.003.003.003 LSAge:0 LSSeq:0
18:48:51 DD RID:001.001.001.001 Dest:node3 Long:52 I:0 M:0 MS:0 Seq:2
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001 LSAge:0 LSSeq:0
18:48:51 LSREQ RID:001.001.001.001 Dest:node3 Long:34
--> LStype:1 LSID:003.003.003.003 AdvRouting:003.003.003.003
18:48:51 DD RID:002.002.002.002 Dest:node1 Long:52 I:0 M:0 MS:1 Seq:2
--> LStype:1 LSID:002.002.002.002 AdvRouting:002.002.002.002 LSAge:0 LSSeq:0
18:48:51 DD RID:001.001.001.001 Dest:node2 Long:52 I:0 M:0 MS:0 Seq:2
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001 LSAge:0 LSSeq:0
18:48:51 LSREQ RID:001.001.001.001 Dest:node2 Long:34
--> LStype:1 LSID:002.002.002.002 AdvRouting:002.002.002.002
18:48:51 LSREQ RID:003.003.003.003 Dest:node1 Long:34
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001
18:48:51 LSUPD RID:001.001.001.001 Dest:node3 Long:76 NroLsa:1
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001 NroLinks:2
------> LinkID:003.003.003.003 LinkData:lk1-3 TypeConexion:1 Tos0metric:1
------> LinkID:002.002.002.002 LinkData:lk1-2 TypeConexion:1 Tos0metric:3
18:48:51 LSUPD RID:003.003.003.003 Dest:node1 Long:64 NroLsa:1
--> LStype:1 LSID:003.003.003.003 AdvRouting:003.003.003.003 NroLinks:1
------> LinkID:001.001.001.001 LinkData:lk1-3 TypeConexion:1 Tos0metric:1
18:48:51 LSREQ RID:002.002.002.002 Dest:node1 Long:34
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001
18:48:51 LSUPD RID:001.001.001.001 Dest:node2 Long:76 NroLsa:1
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001 NroLinks:2
------> LinkID:003.003.003.003 LinkData:lk1-3 TypeConexion:1 Tos0metric:1
------> LinkID:002.002.002.002 LinkData:lk1-2 TypeConexion:1 Tos0metric:3
18:48:51 LSUPD RID:002.002.002.002 Dest:node1 Long:64 NroLsa:1
--> LStype:1 LSID:002.002.002.002 AdvRouting:002.002.002.002 NroLinks:1
------> LinkID:001.001.001.001 LinkData:lk1-2 TypeConexion:1 Tos0metric:3
18:49:00 HELLO RID:001.001.001.001 Dest:node2 Long:136 DR:001.001.001.001
         Neigrs:002.002.002.002 003.003.003.003
18:49:00 HELLO RID:001.001.001.001 Dest:node3 Long:136 DR:001.001.001.001
         Neigrs:002.002.002.002 003.003.003.003
18:49:00 HELLO RID:002.002.002.002 Dest:node1 Long:132 DR:002.002.002.002
         Neigrs:001.001.001.001
```

18:49:00 HELLO RID:003.003.003.003 Dest:node1 Long:132 DR:003.003.003.003
Neigrs:001.001.001.001
18:49:09 HELLO RID:001.001.001.001 Dest:node2 Long:136 DR:001.001.001.001
Neigrs:003.003.003.003 002.002.002.002
18:49:09 HELLO RID:001.001.001.001 Dest:node3 Long:136 DR:001.001.001.001
Neigrs:003.003.003.003 002.002.002.002
18:49:09 HELLO RID:002.002.002.002 Dest:node1 Long:132 DR:002.002.002.002
Neigrs:001.001.001.001
18:49:09 HELLO RID:003.003.003.003 Dest:node1 Long:132 DR:003.003.003.003
Neigrs:001.001.001.001
18:49:17 ------- CAIDA DE LA INTERFACE: lk1-2 -------
18:49:17 HELLO RID:001.001.001.001 Dest:node3 Long:136 DR:001.001.001.001
Neigrs:003.003.003.003 002.002.002.002
18:49:17 HELLO RID:003.003.003.003 Dest:node1 Long:132 DR:003.003.003.003
Neigrs:001.001.001.001
18:49:20 HELLO RID:001.001.001.001 Dest:node3 Long:132 DR:001.001.001.001
Neigrs:003.003.003.003
18:49:20 HELLO RID:003.003.003.003 Dest:node1 Long:132 DR:003.003.003.003
Neigrs:001.001.001.001
18:49:22 HELLO RID:001.001.001.001 Dest:node3 Long:132 DR:001.001.001.001
Neigrs:003.003.003.003
18:49:22 HELLO RID:003.003.003.003 Dest:node1 Long:132 DR:003.003.003.003
Neigrs:001.001.001.001
18:49:25 HELLO RID:001.001.001.001 Dest:node3 Long:132 DR:001.001.001.001
Neigrs:003.003.003.003
18:49:25 HELLO RID:003.003.003.003 Dest:node1 Long:132 DR:003.003.003.003
Neigrs:001.001.001.001
18:49:27 HELLO RID:001.001.001.001 Dest:node3 Long:132 DR:001.001.001.001
Neigrs:003.003.003.003
18:49:47 ------- REPARACION DE LA INTERFACE: lk1-2 -------
18:49:47 HELLO RID:002.002.002.002 Dest:node1 Long:128 DR:002.002.002.002 Neigrs:
18:49:47 HELLO RID:003.003.003.003 Dest:node1 Long:132 DR:003.003.003.003
Neigrs:001.001.001.001
18:49:47 HELLO RID:001.001.001.001 Dest:node2 Long:136 DR:001.001.001.001
Neigrs:003.003.003.003
18:49:47 HELLO RID:001.001.001.001 Dest:node3 Long:136 DR:001.001.001.001
Neigrs:003.003.003.003
18:49:56 HELLO RID:001.001.001.001 Dest:node2 Long:136 DR:001.001.001.001
Neigrs:002.002.002.002 003.003.003.003
18:49:56 HELLO RID:001.001.001.001 Dest:node3 Long:136 DR:001.001.001.001
Neigrs:002.002.002.002 003.003.003.003
18:49:56 HELLO RID:003.003.003.003 Dest:node1 Long:132 DR:003.003.003.003
Neigrs:001.001.001.001
18:49:56 HELLO RID:002.002.002.002 Dest:node1 Long:132 DR:002.002.002.002
Neigrs:001.001.001.001
18:49:56 DD RID:001.001.001.001 Dest:node2 Long:32 I:1 M:1 MS:1 Seq:3
18:49:56 DD RID:002.002.002.002 Dest:node1 Long:32 I:1 M:1 MS:1 Seq:3
18:49:56 DD RID:001.001.001.001 Dest:node2 Long:32 I:1 M:1 MS:0 Seq:3
18:49:56 DD RID:002.002.002.002 Dest:node1 Long:72 I:0 M:0 MS:1 Seq:4
--> LStype:1 LSID:002.002.002.002 AdvRouting:002.002.002.002 LSAge:0 LSSeq:0
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001 LSAge:0 LSSeq:0
18:49:56 DD RID:001.001.001.001 Dest:node2 Long:92 I:0 M:0 MS:0 Seq:4
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001 LSAge:0 LSSeq:0
--> LStype:1 LSID:002.002.002.002 AdvRouting:002.002.002.002 LSAge:0 LSSeq:0
--> LStype:1 LSID:003.003.003.003 AdvRouting:003.003.003.003 LSAge:0 LSSeq:0
18:49:56 LSREQ RID:002.002.002.002 Dest:node1 Long:34
--> LStype:1 LSID:003.003.003.003 AdvRouting:003.003.003.003
18:49:56 LSUPD RID:001.001.001.001 Dest:node2 Long:64 NroLsa:1

--> LStype:1 LSID:003.003.003.003 AdvRouting:003.003.003.003 NroLinks:1
------> LinkID:001.001.001.001 LinkData:lk1-3 TypeConexion:1 Tos0metric:1
18:50:05 HELLO RID:001.001.001.001 Dest:node2 Long:136 DR:001.001.001.001
Neigrs:002.002.002.002 003.003.003.003
18:50:05 HELLO RID:001.001.001.001 Dest:node3 Long:136 DR:001.001.001.001
Neigrs:002.002.002.002 003.003.003.003
18:50:05 HELLO RID:003.003.003.003 Dest:node1 Long:132 DR:003.003.003.003
Neigrs:001.001.001.001
18:50:05 HELLO RID:002.002.002.002 Dest:node1 Long:132 DR:002.002.002.002
Neigrs:001.001.001.001
18:50:14 HELLO RID:001.001.001.001 Dest:node2 Long:136 DR:001.001.001.001
Neigrs:002.002.002.002 003.003.003.003
18:50:14 HELLO RID:001.001.001.001 Dest:node3 Long:136 DR:001.001.001.001
Neigrs:002.002.002.002 003.003.003.003
18:50:14 HELLO RID:003.003.003.003 Dest:node1 Long:132 DR:003.003.003.003
Neigrs:001.001.001.001
18:50:14 HELLO RID:002.002.002.002 Dest:node1 Long:132 DR:002.002.002.002
Neigrs:001.001.001.001

6.6.2.Conexión red de tránsito.

El siguiente target system configura una conexión de tres routers con enlaces a una red de tránsito y un cuarto router conectado point-to-point a uno de ellos.



La siguiente muestra es el tráfico de paquetes OSPF en el router 001.001.001.001 (Node1) desde la inicialización del escenario simulado, con la caída de la interface Link1-2 y su posterior reparación.

18:32:43 HELLO RID:001.001.001.001 Dest:node2 Long:128 DR:001.001.001.001 Neigrs:
18:32:43 HELLO RID:001.001.001.001 Dest:node3 Long:128 DR:001.001.001.001 Neigrs:
18:32:43 HELLO RID:001.001.001.001 Dest:node4 Long:128 DR:001.001.001.001 Neigrs:
18:32:43 HELLO RID:002.002.002.002 Dest:node1 Long:128 DR:001.001.001.001 Neigrs:
18:32:43 HELLO RID:003.003.003.003 Dest:node1 Long:128 DR:001.001.001.001 Neigrs:
18:32:43 HELLO RID:004.004.004.004 Dest:node1 Long:128 DR:004.004.004.004 Neigrs:

18:32:58 HELLO RID:001.001.001.001 Dest:node2 Long:140 DR:001.001.001.001
Neigrs:004.004.004.004 003.003.003.003 002.002.002.002
18:32:58 HELLO RID:001.001.001.001 Dest:node3 Long:140 DR:001.001.001.001
Neigrs:004.004.004.004 003.003.003.003 002.002.002.002
18:32:58 HELLO RID:001.001.001.001 Dest:node4 Long:140 DR:001.001.001.001
Neigrs:004.004.004.004 003.003.003.003 002.002.002.002
18:32:58 HELLO RID:004.004.004.004 Dest:node1 Long:132 DR:004.004.004.004
Neigrs:001.001.001.001
18:32:58 DD RID:001.001.001.001 Dest:node4 Long:32 I:1 M:1 MS:1 Seq:1
18:32:58 HELLO RID:003.003.003.003 Dest:node1 Long:136 DR:001.001.001.001
Neigrs:001.001.001.001 002.002.002.002
18:32:58 DD RID:001.001.001.001 Dest:node3 Long:32 I:1 M:1 MS:1 Seq:1
18:32:58 HELLO RID:002.002.002.002 Dest:node1 Long:136 DR:001.001.001.001
Neigrs:001.001.001.001 003.003.003.003
18:32:58 DD RID:001.001.001.001 Dest:node2 Long:32 I:1 M:1 MS:1 Seq:1
18:32:58 DD RID:003.003.003.003 Dest:node1 Long:32 I:1 M:1 MS:1 Seq:1
18:32:58 DD RID:001.001.001.001 Dest:node3 Long:32 I:1 M:1 MS:0 Seq:1
18:32:58 DD RID:004.004.004.004 Dest:node1 Long:32 I:1 M:1 MS:1 Seq:1
18:32:58 DD RID:001.001.001.001 Dest:node4 Long:32 I:1 M:1 MS:0 Seq:1
18:32:58 DD RID:002.002.002.002 Dest:node1 Long:32 I:1 M:1 MS:1 Seq:1
18:32:58 DD RID:001.001.001.001 Dest:node2 Long:32 I:1 M:1 MS:0 Seq:1
18:32:58 DD RID:004.004.004.004 Dest:node1 Long:52 I:0 M:0 MS:1 Seq:2
--> LStype:1 LSID:004.004.004.004 AdvRouting:004.004.004.004 LSAge:0 LSSeq:0
18:32:58 DD RID:001.001.001.001 Dest:node4 Long:72 I:0 M:0 MS:0 Seq:2
--> LStype:2 LSID:001.001.001.001 AdvRouting:001.001.001.001 LSAge:0 LSSeq:0
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001 LSAge:0 LSSeq:0
18:32:58 LSREQ RID:001.001.001.001 Dest:node4 Long:34
--> LStype:1 LSID:004.004.004.004 AdvRouting:004.004.004.004
18:32:58 DD RID:003.003.003.003 Dest:node1 Long:52 I:0 M:0 MS:1 Seq:2
--> LStype:1 LSID:003.003.003.003 AdvRouting:003.003.003.003 LSAge:0 LSSeq:0
18:32:58 DD RID:001.001.001.001 Dest:node3 Long:72 I:0 M:0 MS:0 Seq:2
--> LStype:2 LSID:001.001.001.001 AdvRouting:001.001.001.001 LSAge:0 LSSeq:0
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001 LSAge:0 LSSeq:0
18:32:58 LSREQ RID:001.001.001.001 Dest:node3 Long:34
--> LStype:1 LSID:003.003.003.003 AdvRouting:003.003.003.003
18:32:58 DD RID:002.002.002.002 Dest:node1 Long:52 I:0 M:0 MS:1 Seq:2
--> LStype:1 LSID:002.002.002.002 AdvRouting:002.002.002.002 LSAge:0 LSSeq:0
18:32:58 DD RID:001.001.001.001 Dest:node2 Long:72 I:0 M:0 MS:0 Seq:2
--> LStype:2 LSID:001.001.001.001 AdvRouting:001.001.001.001 LSAge:0 LSSeq:0
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001 LSAge:0 LSSeq:0
18:32:58 LSREQ RID:001.001.001.001 Dest:node2 Long:34
--> LStype:1 LSID:002.002.002.002 AdvRouting:002.002.002.002
18:32:58 LSREQ RID:004.004.004.004 Dest:node1 Long:44
--> LStype:2 LSID:001.001.001.001 AdvRouting:001.001.001.001
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001
18:32:58 LSUPD RID:001.001.001.001 Dest:node4 Long:120 NroLsa:2
--> LStype:2 LSID:001.001.001.001 AdvRouting:001.001.001.001
NetworkMask:255.255.255.255 AtachedRouters:003.003.003.003 002.002.002.002
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001 NroLinks:3
------> LinkID:004.004.004.004 LinkData:lk1-4 TypeConexion:1 Tos0metric:1
------> LinkID:001.001.001.001 LinkData:lk1-3 TypeConexion:2 Tos0metric:3
------> LinkID:001.001.001.001 LinkData:lk1-2 TypeConexion:2 Tos0metric:3
18:32:58 LSUPD RID:004.004.004.004 Dest:node1 Long:64 NroLsa:1
--> LStype:1 LSID:004.004.004.004 AdvRouting:004.004.004.004 NroLinks:1
------> LinkID:001.001.001.001 LinkData:lk1-4 TypeConexion:1 Tos0metric:1
18:32:58 LSREQ RID:003.003.003.003 Dest:node1 Long:44
--> LStype:2 LSID:001.001.001.001 AdvRouting:001.001.001.001
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001

```
18:32:58 LSUPD RID:001.001.001.001 Dest:node3 Long:120 NroLsa:2
--> LStype:2 LSID:001.001.001.001 AdvRouting:001.001.001.001
        NetworkMask:255.255.255.255 AtachedRouters:003.003.003.003 002.002.002.002
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001 NroLinks:3
------> LinkID:004.004.004.004 LinkData:lk1-4 TypeConexion:1 Tos0metric:1
------> LinkID:001.001.001.001 LinkData:lk1-3 TypeConexion:2 Tos0metric:3
------> LinkID:001.001.001.001 LinkData:lk1-2 TypeConexion:2 Tos0metric:3
18:32:58 LSREQ RID:002.002.002.002 Dest:node1 Long:44
--> LStype:2 LSID:001.001.001.001 AdvRouting:001.001.001.001
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001
18:32:58 LSUPD RID:001.001.001.001 Dest:node2 Long:120 NroLsa:2
--> LStype:2 LSID:001.001.001.001 AdvRouting:001.001.001.001
        NetworkMask:255.255.255.255 AtachedRouters:003.003.003.003 002.002.002.002
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001 NroLinks:3
------> LinkID:004.004.004.004 LinkData:lk1-4 TypeConexion:1 Tos0metric:1
------> LinkID:001.001.001.001 LinkData:lk1-3 TypeConexion:2 Tos0metric:3
------> LinkID:001.001.001.001 LinkData:lk1-2 TypeConexion:2 Tos0metric:3
18:32:58 LSUPD RID:003.003.003.003 Dest:node1 Long:76 NroLsa:1
--> LStype:1 LSID:003.003.003.003 AdvRouting:003.003.003.003 NroLinks:2
------> LinkID:001.001.001.001 LinkData:lk2-3 TypeConexion:2 Tos0metric:1
------> LinkID:001.001.001.001 LinkData:lk1-3 TypeConexion:2 Tos0metric:3
18:32:58 LSUPD RID:002.002.002.002 Dest:node1 Long:76 NroLsa:1
--> LStype:1 LSID:002.002.002.002 AdvRouting:002.002.002.002 NroLinks:2
------> LinkID:001.001.001.001 LinkData:lk2-3 TypeConexion:2 Tos0metric:1
------> LinkID:001.001.001.001 LinkData:lk1-2 TypeConexion:2 Tos0metric:3
18:33:14 HELLO RID:001.001.001.001 Dest:node2 Long:140 DR:001.001.001.001
        Neigrs:002.002.002.002 003.003.003.003 004.004.004.004
18:33:14 HELLO RID:001.001.001.001 Dest:node3 Long:140 DR:001.001.001.001
        Neigrs:002.002.002.002 003.003.003.003 004.004.004.004
18:33:14 HELLO RID:001.001.001.001 Dest:node4 Long:140 DR:001.001.001.001
        Neigrs:002.002.002.002 003.003.003.003 004.004.004.004
18:33:14 HELLO RID:002.002.002.002 Dest:node1 Long:136 DR:001.001.001.001
        Neigrs:001.001.001.001 003.003.003.003
18:33:14 HELLO RID:003.003.003.003 Dest:node1 Long:136 DR:001.001.001.001
        Neigrs:001.001.001.001 002.002.002.002
18:33:14 HELLO RID:004.004.004.004 Dest:node1 Long:132 DR:004.004.004.004
        Neigrs:001.001.001.001
18:33:30 HELLO RID:001.001.001.001 Dest:node2 Long:140 DR:001.001.001.001
        Neigrs:004.004.004.004 003.003.003.003 002.002.002.002
18:33:30 HELLO RID:001.001.001.001 Dest:node3 Long:140 DR:001.001.001.001
        Neigrs:004.004.004.004 003.003.003.003 002.002.002.002
18:33:30 HELLO RID:001.001.001.001 Dest:node4 Long:140 DR:001.001.001.001
        Neigrs:004.004.004.004 003.003.003.003 002.002.002.002
18:33:30 HELLO RID:002.002.002.002 Dest:node1 Long:136 DR:001.001.001.001
        Neigrs:003.003.003.003 001.001.001.001
18:33:30 HELLO RID:003.003.003.003 Dest:node1 Long:136 DR:001.001.001.001
        Neigrs:001.001.001.001 002.002.002.002
18:33:30 HELLO RID:004.004.004.004 Dest:node1 Long:132 DR:004.004.004.004
        Neigrs:001.001.001.001
18:33:45 HELLO RID:001.001.001.001 Dest:node2 Long:140 DR:001.001.001.001
        Neigrs:004.004.004.004 003.003.003.003 002.002.002.002
18:33:45 HELLO RID:001.001.001.001 Dest:node3 Long:140 DR:001.001.001.001
        Neigrs:004.004.004.004 003.003.003.003 002.002.002.002
18:33:45 HELLO RID:001.001.001.001 Dest:node4 Long:140 DR:001.001.001.001
        Neigrs:004.004.004.004 003.003.003.003 002.002.002.002
18:33:45 ------- CAIDA DE LA INTERFACE: lk1-2 -------
18:33:45 HELLO RID:004.004.004.004 Dest:node1 Long:132 DR:004.004.004.004
        Neigrs:001.001.001.001
```

18:33:45 HELLO RID:001.001.001.001 Dest:node4 Long:132 DR:001.001.001.001
Neigrs:004.004.004.004
18:34:00 HELLO RID:004.004.004.004 Dest:node1 Long:132 DR:004.004.004.004
Neigrs:001.001.001.001
18:34:00 HELLO RID:001.001.001.001 Dest:node4 Long:132 DR:001.001.001.001
Neigrs:004.004.004.004
18:34:01 ------- REPARACION DE LA INTERFACE: lk1-2 -------
18:34:01 HELLO RID:004.004.004.004 Dest:node1 Long:132 DR:004.004.004.004
Neigrs:001.001.001.001
18:34:01 HELLO RID:001.001.001.001 Dest:node4 Long:132 DR:001.001.001.001
Neigrs:004.004.004.004
18:34:26 HELLO RID:001.001.001.001 Dest:node2 Long:132 DR:001.001.001.001
Neigrs:004.004.004.004
18:34:26 HELLO RID:001.001.001.001 Dest:node3 Long:132 DR:001.001.001.001
Neigrs:004.004.004.004
18:34:26 HELLO RID:001.001.001.001 Dest:node4 Long:132 DR:001.001.001.001
Neigrs:004.004.004.004
18:34:26 HELLO RID:003.003.003.003 Dest:node1 Long:132 DR:001.001.001.001
Neigrs:002.002.002.002
18:34:26 HELLO RID:004.004.004.004 Dest:node1 Long:132 DR:004.004.004.004
Neigrs:001.001.001.001
18:34:26 HELLO RID:002.002.002.002 Dest:node1 Long:132 DR:001.001.001.001
Neigrs:003.003.003.003
18:34:38 HELLO RID:001.001.001.001 Dest:node2 Long:140 DR:001.001.001.001
Neigrs:002.002.002.002 004.004.004.004 003.003.003.003
18:34:38 HELLO RID:001.001.001.001 Dest:node3 Long:140 DR:001.001.001.001
Neigrs:002.002.002.002 004.004.004.004 003.003.003.003
18:34:38 HELLO RID:001.001.001.001 Dest:node4 Long:140 DR:001.001.001.001
Neigrs:002.002.002.002 004.004.004.004 003.003.003.003
18:34:38 HELLO RID:004.004.004.004 Dest:node1 Long:132 DR:004.004.004.004
Neigrs:001.001.001.001
18:34:38 HELLO RID:003.003.003.003 Dest:node1 Long:136 DR:001.001.001.001
Neigrs:001.001.001.001 002.002.002.002
18:34:38 HELLO RID:002.002.002.002 Dest:node1 Long:136 DR:001.001.001.001
Neigrs:001.001.001.001 003.003.003.003
18:34:38 DD RID:001.001.001.001 Dest:node2 Long:32 I:1 M:1 MS:1 Seq:3
18:34:38 DD RID:002.002.002.002 Dest:node1 Long:32 I:1 M:1 MS:1 Seq:3
18:34:38 DD RID:001.001.001.001 Dest:node3 Long:32 I:1 M:1 MS:1 Seq:3
18:34:38 DD RID:001.001.001.001 Dest:node2 Long:32 I:1 M:1 MS:0 Seq:3
18:34:38 DD RID:003.003.003.003 Dest:node1 Long:32 I:1 M:1 MS:1 Seq:3
18:34:38 DD RID:002.002.002.002 Dest:node1 Long:112 I:0 M:0 MS:1 Seq:4
--> LStype:1 LSID:002.002.002.002 AdvRouting:002.002.002.002 LSAge:0 LSSeq:0
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001 LSAge:0 LSSeq:0
--> LStype:2 LSID:001.001.001.001 AdvRouting:001.001.001.001 LSAge:0 LSSeq:0
--> LStype:1 LSID:003.003.003.003 AdvRouting:003.003.003.003 LSAge:0 LSSeq:0
18:34:38 DD RID:001.001.001.001 Dest:node3 Long:32 I:1 M:1 MS:0 Seq:3
18:34:38 DD RID:001.001.001.001 Dest:node2 Long:132 I:0 M:0 MS:0 Seq:4
--> LStype:2 LSID:001.001.001.001 AdvRouting:001.001.001.001 LSAge:0 LSSeq:0
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001 LSAge:0 LSSeq:0
--> LStype:1 LSID:002.002.002.002 AdvRouting:002.002.002.002 LSAge:0 LSSeq:0
--> LStype:1 LSID:003.003.003.003 AdvRouting:003.003.003.003 LSAge:0 LSSeq:0
--> LStype:1 LSID:004.004.004.004 AdvRouting:004.004.004.004 LSAge:0 LSSeq:0
18:34:38 DD RID:003.003.003.003 Dest:node1 Long:112 I:0 M:0 MS:1 Seq:4
--> LStype:1 LSID:002.002.002.002 AdvRouting:002.002.002.002 LSAge:0 LSSeq:0
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001 LSAge:0 LSSeq:0
--> LStype:2 LSID:001.001.001.001 AdvRouting:001.001.001.001 LSAge:0 LSSeq:0
--> LStype:1 LSID:003.003.003.003 AdvRouting:003.003.003.003 LSAge:0 LSSeq:0
18:34:38 LSREQ RID:002.002.002.002 Dest:node1 Long:34

--> LStype:1 LSID:004.004.004.004 AdvRouting:004.004.004.004
18:34:38 DD RID:001.001.001.001 Dest:node3 Long:132 I:0 M:0 MS:0 Seq:4
--> LStype:2 LSID:001.001.001.001 AdvRouting:001.001.001.001 LSAge:0 LSSeq:0
--> LStype:1 LSID:001.001.001.001 AdvRouting:001.001.001.001 LSAge:0 LSSeq:0
--> LStype:1 LSID:002.002.002.002 AdvRouting:002.002.002.002 LSAge:0 LSSeq:0
--> LStype:1 LSID:003.003.003.003 AdvRouting:003.003.003.003 LSAge:0 LSSeq:0
--> LStype:1 LSID:004.004.004.004 AdvRouting:004.004.004.004 LSAge:0 LSSeq:0
18:34:38 LSUPD RID:001.001.001.001 Dest:node2 Long:64 NroLsa:1
--> LStype:1 LSID:004.004.004.004 AdvRouting:004.004.004.004 NroLinks:1
------> LinkID:001.001.001.001 LinkData:lk1-4 TypeConexion:1 Tos0metric:1
18:34:38 LSREQ RID:003.003.003.003 Dest:node1 Long:34
--> LStype:1 LSID:004.004.004.004 AdvRouting:004.004.004.004
18:34:38 LSUPD RID:001.001.001.001 Dest:node3 Long:64 NroLsa:1
--> LStype:1 LSID:004.004.004.004 AdvRouting:004.004.004.004 NroLinks:1
------> LinkID:001.001.001.001 LinkData:lk1-4 TypeConexion:1 Tos0metric:1
18:34:51 HELLO RID:001.001.001.001 Dest:node2 Long:140 DR:001.001.001.001
Neigrs:002.002.002.002 003.003.003.003 004.004.004.004
18:34:51 HELLO RID:001.001.001.001 Dest:node3 Long:140 DR:001.001.001.001
Neigrs:002.002.002.002 003.003.003.003 004.004.004.004
18:34:51 HELLO RID:001.001.001.001 Dest:node4 Long:140 DR:001.001.001.001
Neigrs:002.002.002.002 003.003.003.003 004.004.004.004
18:34:51 HELLO RID:004.004.004.004 Dest:node1 Long:132 DR:004.004.004.004
Neigrs:001.001.001.001
18:34:51 HELLO RID:003.003.003.003 Dest:node1 Long:136 DR:001.001.001.001
Neigrs:001.001.001.001 003.003.003.003
18:34:51 HELLO RID:002.002.002.002 Dest:node1 Long:136 DR:001.001.001.001
Neigrs:001.001.001.001 003.003.003.003
18:35:06 HELLO RID:001.001.001.001 Dest:node2 Long:140 DR:001.001.001.001
Neigrs:002.002.002.002 003.003.003.003 004.004.004.004
18:35:06 HELLO RID:001.001.001.001 Dest:node3 Long:140 DR:001.001.001.001
Neigrs:002.002.002.002 003.003.003.003 004.004.004.004
18:35:06 HELLO RID:001.001.001.001 Dest:node4 Long:140 DR:001.001.001.001
Neigrs:002.002.002.002 003.003.003.003 004.004.004.004
18:35:06 HELLO RID:004.004.004.004 Dest:node1 Long:132 DR:004.004.004.004
Neigrs:001.001.001.001
18:35:06 HELLO RID:003.003.003.003 Dest:node1 Long:136 DR:001.001.001.001
Neigrs:001.001.001.001 003.003.003.003
18:35:06 HELLO RID:002.002.002.002 Dest:node1 Long:136 DR:001.001.001.001
Neigrs:001.001.001.001 003.003.003.003

Capítulo 7

Conclusión.

El MaRS es un sistema de simulación estructurado y preparado para agregar nuevas componentes sin grandes problemas.

Si bien la inserción de la componente OSPF no tuvo mayores inconvenientes, implementar su comportamiento integral se vio obstaculizado por la estructura general del MaRS.

El mayor inconveniente a resolver fue la configuración de la red de tránsito que no existía en el MaRS original, en dónde los routers se conectan de forma point-to-point. Para resolverlo decidimos representar esto conectando los routers pertenecientes a la misma red todos contra todos, identificando el tipo de conexión y la dirección de la red.

Desarrollamos el comportamiento básico y más destacado del OSPF en el MaRS, el sistema queda abierto para implementar otras características como configurar distintas áreas, y con esto agregar la identificación de distintos tipos de routers, como interno y de borde. También se generaría el área backbone y los enlaces virtuales.

Avanzando un paso más se puede extender la representación con la inclusión de sistemas autónomos, quedando así simulado todo el ambiente para el cual el protocolo OSPF fue diseñado.

Una aproximación para poder implementar lo anterior sería crear la componente de área, donde se relacionarían los distintos routers pertenecientes a ella y el componente sistema autónomo, donde se relacionarían las distintas áreas pertenecientes a él.

Agregar el protocolo OSPF al simulador MaRS hace que la simulación sea útil para analizar el comportamiento de este protocolo en distintas configuraciones, compararlo con el resto de los algoritmos de ruteo incluídos en el MaRS, tomar muestras de distintos parámetros sobre el tráfico de datos y ruteo. Con todo esto se podrá obtener conclusiones útiles sobre el OSPF.

Por otro lado este simulador nos permitió con lo implementado, reflejar el comportamiento general de este protocolo, para el estudio en profundidad de su funcionamiento y estructuras.

Apéndices.

A1.Archivo ospf.h.

```
#ifndef OSPF_H
#define OSPF_H

#include "sim.h"
#include "q.h"
#include "list.h"
#include "component.h"
#include "route.h"

#define EV_OSPF_BROADCAST      (EV_CLASS_PRIVATE | 1)
#define EV_OSPF_DOWN_INTERF    (EV_CLASS_PRIVATE | 2)
#define EV_OSPF_UP_INTERF      (EV_CLASS_PRIVATE | 3)
#define GTT(g) (*((GlobalTopTable *)(g->global_top->u.p)))

/* Tipo de Conexion */
#define PTOP   1
#define TORED  2
#define TOSTUB 3

/* Estado en la Conversacion */
#define NULO    0  /* se usara para saber el fin de la conversacion */
#define DOWN    1
#define ATTEMPT 2
#define INIT    3
#define WAY2    4
#define EXSTART 5
#define EXCHANGE 6
#define LOADING 7
#define FULL    8

/* Age de los LSAs en minutos VER DE TRANSFORMARLOS!!! */
#define MAXAGE    60
#define MAXAGEDIFF 15
#define REFRESHLSA 30

#define HELLOINTERVAL 10     /* Segundos. Intervalo entre paquetes hello */

caddr_t ospf_action();

typedef struct {
  Component   *dest;
  int         no;
  int         no_unprocessed;
  char        mark;
} SeqNoTable[MAX_NO_OF_NODES];

typedef unsigned int GlobalTopTable[MAX_NO_OF_NODES][MAX_NO_OF_NODES];

typedef struct {
  int         Lstype;
```

```
  char      *LsID;
  char      *AdvRouting;
} ClaveLSA;

typedef struct {
  char      Estado;
  short     Inactividad;
  char      Master_Slave;
  int       NroSecuencia;
  char      *NeighborID;
  int       NeighborPrior;
  char      *NeighborIP;     /* nombre del link */
  struct _Ospft *NeighborDR;
  struct _Ospft *NeighborBDR;
  list      *RetransmiteLSA;  /* Lista de LSA */
  list      *BaseLSA;         /* Lista de LSA */
  list      *PedidosLS;       /* Lista de ClaveLSA */
  int       MarcaDd;          /* Cantidad de LS_Header enviados en un paq.*/
} Conversacion[MAX_LINKS_PER_NODE];

typedef struct _Ospft {
  struct _Ospft *ospf_next,       /* Links a otros componentes en la lista */
                *ospf_prev;

  short       ospf_class;       /*  */
  short       ospf_type;        /*  */
  char        ospf_name[40];    /* Name of component (appears on screen) */
  PFP         ospf_action;      /* Main function of component.  */
  COMP_OBJECT  ospf_picture;    /* Graphics object displays this thing */
  list        *ospf_neighbors;  /* List of neighbors of this thing */

  /* Parameters-- data that will be displayed on screen */

  short       ospf_menu_up;     /* If true, then the text window is up */
  queue       *ospf_params;     /* Variable-length queue of parameters */

  PFI         processing_time;  /* returns an estimate of proc time */
  int         no_of_nodes;      /* seen no of nodes in the net */
  tick_t      link_cost_time;   /* time at which link costs are calculated*/
  Component   *node;            /* my neighbour node */

  Param       *routing_table;   /* routing table */
  Param       *local_top;       /* local topology table */
  Param       *global_top;      /* global topology table */
  Param       *seq_no_table;    /* last sequence no of a node */
  Param       *brdcast_period;  /* time btw broadcast events */
  Param       *sd;
  Param       *seq_no;          /* sequence no of this node */

  char        *RouterID;
  int         AreaID;
  struct _Ospft *DR;
  struct _Ospft *BDR;
  Param       *Prioridad;
  Param       *TraceTR;
  Param       *Trace;
  int         HelloInterval;
  int         DeadInterval;
```

```
  Conversacion  *Conversa;
  list        *Vecinos;         /* Lista de RouterID */
  list        *ListaLSA;        /* Lista de LSA */

} Ospft;
#endif  /* OSPF_H */
```

A2.Archivo ospf.c.

```
#include <sys/types.h>
#include <stdio.h>
#include <math.h>
#include "sim.h"
#include "q.h"
#include "list.h"
#include "component.h"
#include "log.h"
#include "comptypes.h"
#include "packet.h"
#include "eventdefs.h"
#include "event.h"
#include "node.h"
#include "ospf.h"
#include "perf.h"
#include "link.h"
#include "time.h"

/* #define DEBUG */

#ifdef DEBUG
extern Log debug_log;
#endif

extern FILE *salida;
extern FILE *salidatr;

#define SNT(g) (*((SeqNoTable *)(g->seq_no_table->u.p)))

static caddr_t ospf_create(), ospf_delete(),
   ospf_neighbor(), ospf_uneighbor(), ospf_start(),
   ospf_reset(), ospf_broadcast(), ospf_processing(),
   ospf_generalsa();

static int routing_processing_time2();
static int add_new_node2();
void fr_space2();
char *make_gtt_text2();
static LSA *AvisoEnLista();
static ClaveLSA *ClaveEnLista();
static int VinodelVecino();
static l_elt *AgregaListaRet();
static l_elt *EliminaListaRet();
static char  *LinkAlNodo();
static int Link_LTT();
static OSPF_Packet *Ospfpk_alloc();
```

```
/*********************************************************/
caddr_t ospf_action(src, g, type, pkt, arg)
    Component *src;
    register Ospft *g;
    int type;
    Packet *pkt;
    caddr_t arg;
{
 int k;
 caddr_t result = NULL;

 dbg_set_level(DBG_ERR);

 /* Just a big switch on type of event */

 switch (type) {

   case EV_RESET:
#ifdef DEBUG
    dbg_write(debug_log, DBG_INFO, (Component *)g, "reset");
#endif
    result = ospf_reset(g);
    break;

   case EV_CREATE:
    /* Minor sanity check first-- g should be NULL when initializing. */
#ifdef DEBUG
    if (g)
      dbg_write(debug_log, DBG_INFO, (Component *)NULL,
            "OSPF Generator initialization called with non-null pointer.");
#endif
    result = ospf_create((char *)arg);
    break;

   case EV_DEL:
    result = ospf_delete(g);
    break;

   case EV_NEIGHBOR:
    result = ospf_neighbor(g, (Component *)arg);
    break;

   case EV_UNEIGHBOR:
    result = ospf_uneighbor(g, (Component *)arg);
    break;

   case EV_MK_PEER:
    result = (caddr_t) g;
    break;

   case EV_START:
    result = ospf_start(g);
    break;

   case EV_STOP:
    result = (caddr_t) g;
```

```
    break;

  /* The preceding were the commands.  Now the actual events */

  case EV_ROUTE_PROCESSING:
    result = ospf_processing(g, pkt);
    break;

  case EV_OSPF_BROADCAST:
    result = ospf_broadcast(g);
    break;

  case EV_OSPF_DOWN_INTERF:
    ConversaDown(g,src);
    result = ospf_generalsa(g);
    ospf(g);
    Trace_txt(g,1,src->co_name);
    break;

  case EV_OSPF_UP_INTERF:
    if ( src->co_type == LINK ) /* Se reparo un link */
    {
      ConversaUp(g,src);
      Trace_txt(g,2,src->co_name);
    }
    else                        /* Se inicia el ospf */
    {
      Trace_txt(g,3," ");
    }
    result = ospf_generalsa(g);
    ospf(g);
    break;

  default:
#ifdef DEBUG
    dbg_write(debug_log, DBG_ERR, (Component *)g,
          "got unexpected event of type %x", type);
#endif
    break;
  }

 return(result);
}


/*************************************************************/
static caddr_t ospf_create(name)
   register char *name;
{
 Ospft *newg;

 /* Memory for the component structure. */
 newg = (Ospft *)sim_malloc(sizeof(Ospft));

 newg->RouterID=(char *)sim_malloc(sizeof(char));

 /* First things first-- copy name into the new structure. */
 strncpy(newg->ospf_name, name, 40);
```

```
/* have to create a neighbor list */
newg->ospf_neighbors = l_create();
newg->ospf_params = q_create();

strcpy(newg->RouterID,name);

newg->AreaID=0;
newg->DR=newg;
newg->BDR=newg;
newg->HelloInterval=HELLOINTERVAL;
newg->DeadInterval=0;
newg->Conversa=NULL;
newg->ListaLSA = l_create();
newg->Vecinos = l_create();

newg->ospf_class = ROUTE_CLASS;
newg->ospf_type = OSPF;
newg->ospf_action = ospf_action;
newg->ospf_menu_up = FALSE;

/* Initialize the parameters */
(void)param_init((Component *)newg, "Name",
        (PFD)NULL, make_name_text, make_short_name_text,
        param_input_name,
        0, DisplayMask | InputMask, 0.0);

newg->brdcast_period = param_init((Component *)newg,
        "Time btw topology broad (msec)",
        int_calc, make_int_text, make_short_int_text,
        param_input_int,
        0, DisplayMask | ModifyMask, 0.0);
pval(newg, brdcast_period)->u.i = 10000;

newg->sd = param_init((Component *)newg,
        "Standard deviation",
        (PFD)NULL, make_int_text, make_short_int_text,
        param_input_int,
        0, DisplayMask | ModifyMask, 0.0);
pval(newg, sd)->u.i = 1000;

newg->seq_no = param_init((Component *)newg,
        "Sequence number",
        int_calc, make_int_text, make_short_int_text,
        param_input_int,
        TIME_HISTORY, 0, 0.0);
pval(newg, seq_no)->u.i = 0;

newg->global_top = param_init((Component *)newg,
        "Global topology table",
        (PFD)NULL, make_text, make_gtt_text2,
        (PFI)NULL,
        TEXT_METER, DisplayMask | CanHaveLogMask | CanHaveMeterMask, 0.0);
pval(newg, global_top)->u.p = sim_malloc(sizeof(GlobalTopTable));

newg->local_top = param_init((Component *)newg,
        "Local topology table",
        (PFD)NULL, make_text, make_ltt_text,
```

```
        (PFI)NULL,
        TEXT_METER, DisplayMask | CanHaveLogMask | CanHaveMeterMask, 0.0);
 pval(newg, local_top)->u.p = (caddr_t) NULL;

 newg->routing_table = param_init((Component *)newg,
        "Routing table",
        (PFD)NULL, make_text, make_rt_text,
        (PFI)NULL,
        TEXT_METER, DisplayMask | CanHaveLogMask | CanHaveMeterMask, 0.0);
 pval(newg, routing_table)->u.p = sim_malloc(sizeof(RoutingTable));

 newg->seq_no_table = param_init((Component *)newg,
        "Last sequence no table",
        (PFD)NULL, make_text, make_text,
        (PFI)NULL,
        0, 0, 0.0);
 pval(newg, seq_no_table)->u.p = sim_malloc(sizeof(SeqNoTable));


 newg->Prioridad = param_init((Component *)newg,
        "Prioridad",
        (PFD)NULL, make_int_text, make_short_int_text,
        param_input_int,
        0, DisplayMask | ModifyMask, 0.0);
 pval(newg, Prioridad)->u.i = 0;

 newg->TraceTR = param_init((Component *)newg,
        "TraceTR",
        (PFD)NULL, make_int_text, make_short_int_text,
        param_input_int,
        0, DisplayMask | ModifyMask, 0.0);
 pval(newg, TraceTR)->u.i = 0;

 newg->Trace = param_init((Component *)newg,
        "Trace",
        (PFD)NULL, make_int_text, make_short_int_text,
        param_input_int,
        0, DisplayMask | ModifyMask, 0.0);
 pval(newg, Trace)->u.i = 0;


 newg->processing_time = routing_processing_time2;
 newg->no_of_nodes    = 0;
 newg->node           = (Component *) NULL;

#ifdef DEBUG
 dbg_write(debug_log, DBG_INFO, (Component *)newg,
        "ospf generator initialized");
#endif

 return((caddr_t)newg);
}


/*************************************************************/
static caddr_t ospf_delete(g)
   register Ospft *g;
{
```

```
  free(g->routing_table->u.p);
  free(g->global_top->u.p);
  free(g->seq_no_table->u.p);
  comp_delete((Component *) g);
  return (caddr_t) g;
}


/*************************************************************/
static caddr_t ospf_reset(g)
Ospft *g;
{
 g->no_of_nodes= g->ospf_neighbors->l_len;
 g->seq_no->u.i = 0;
 g->link_cost_time = 0;
 ResetConversacion(g);
 log_param((Component *)g, g->seq_no);
 return (caddr_t) g;

}


/*************************************************************/
ResetConversacion (g)
register Ospft  *g;
{
int    i;
Ospft  *x;

g->Conversa=(Conversacion *)sim_malloc(sizeof(Conversacion)*MAX_LINKS_PER_NODE);

for (i = 0; LTT(g)[i].n_nd != (Component *) NULL; i++)
{
   x=(Ospft *)sim_malloc(sizeof(Ospft));
   x=( (Ospft *) (((Nodee *)LTT(g)[i].n_nd)->nd_ptr_routing_mod) );

   g->Conversa[i]->Estado=DOWN;
    /*g->Conversa[i].Inactividad=0;*/
   g->Conversa[i]->Master_Slave='M';
   g->Conversa[i]->NroSecuencia=0;

   g->Conversa[i]->NeighborID=(char *)sim_malloc(sizeof(char)*15);
   strcpy(g->Conversa[i]->NeighborID,x->RouterID);
   g->Conversa[i]->NeighborIP=(char *)sim_malloc(sizeof(char)*15);
   strcpy(g->Conversa[i]->NeighborIP,((Link *)(LTT(g)[i].n_link))->link_name);
   g->Conversa[i]->NeighborPrior=x->Prioridad->u.i;
   g->Conversa[i]->NeighborDR=x->DR;
   g->Conversa[i]->NeighborBDR=x->BDR;
   g->Conversa[i]->RetransmiteLSA=l_create();
   g->Conversa[i]->BaseLSA=l_create();
   g->Conversa[i]->PedidosLS=l_create();
   g->Conversa[i]->MarcaDd=0;
}
g->Conversa[i]->Estado=NULO;

Trace_Conv(g);

}
```

```
/*************************************************************/
static caddr_t ospf_neighbor(g, c)
    register Ospft *g;
    register Component *c;
{
  caddr_t result;

  /* Put the passed neighbor into my neighbor list, but only if it is
    a legal neighbor (a node). Also can only have one neighbor. */

  result = (caddr_t)add_neighbor((Component *)g, c, 1, 1, NODE_CLASS);
  if (result != (caddr_t)NULL) {
    g->local_top->u.p = (caddr_t)((Nodee *)c)->loc_topology_table;
    g->node     = c;
    add_new_node2(g, c);
  }
  return result;
}


/*************************************************************/
static caddr_t ospf_uneighbor(g, c)
    register Ospft *g;
    register Component *c;
{
  caddr_t result;

  result = (caddr_t)remove_neighbor((Component *)g, c);
  if (result != (caddr_t)NULL) {
    g->local_top->u.p        = (caddr_t) NULL;
    g->node                  = (Component *) NULL;
    g->no_of_nodes           = 0;
  }
  return result;
}


/*************************************************************/
static caddr_t ospf_start(g)
register Ospft *g;
{

 if (g->ospf_neighbors->l_len != 1) {
#ifdef DEBUG
   dbg_write(debug_log, DBG_ERR, (Component *)g,
         "Spf Module not connected to a node");
#endif
   return((caddr_t)NULL);
 }

 if (!lcostfcn_adr) {
   printf("LINK_COST_FUNC typed component not found...\n");
   return((caddr_t)NULL);
 }
```

```
 EleccionDR(g);


 /* start broadcasting */

 ev_enqueue(EV_OSPF_BROADCAST, (Component *)g, (Component *)g, ev_now(),
       g->ospf_action, (Packet *)NULL, (caddr_t)NULL);

 /* Comienza la generacion de avisos, cada router con los propios aunque no
 fluyan ya que los estados de la conversacion seguro estan por debajo de EXCHANGE*/

 ev_enqueue(EV_OSPF_UP_INTERF, (Component *)g, (Component *)g, ev_now(),
       g->ospf_action, (Packet *)NULL, (caddr_t)NULL);

 /* Something non-NULL to return */
 return((caddr_t)g);
}


/*************************************************************/
static void inf_global_topology_table(g, ind)
Ospft *g;
int      ind;
{
  int i;
  for (i=0; i<g->no_of_nodes; i++)
    GTT(g)[ind][i] = INFINITY;

}


/*************************************************************/
static int add_new_node2(g, c)
register Ospft *g;
Component      *c;
{
  int ind;
  int i;

  ind  = g->no_of_nodes;
  g->no_of_nodes++;
  RT(g) [ind].dest    = c;
  RT(g) [ind].hop     = (Component *) NULL;
  RT(g) [ind].cost    = INFINITY;
  SNT(g)[ind].no      = 0;

  for (i=0; i<g->no_of_nodes; i++) {
    GTT(g)[ind][i] = INFINITY;
    GTT(g)[i][ind] = INFINITY;
  }

  return ind;
}


/*************************************************************/
static int index_of(g, c)
register Ospft *g;
```

```
register Component *c;
{
  register int ind;

  for (ind = 0; ind < g->no_of_nodes; ind++)
    if (RT(g)[ind].dest == c)
      return ind;

  return -1;
}


/*********************************************************/
ospf(g)
register Ospft *g;
{
  int i, ind, min_ind;
  unsigned int min_cost;

  for (i = 0; i < g->no_of_nodes; i++) {
    RT(g)[i].hop  = (Component *) NULL;
    RT(g)[i].cost = INFINITY;
    SNT(g)[i].mark = 1;
  }

  ind = index_of(g, g->node);
  RT(g)[ind].cost = 0;
  SNT(g)[ind].mark = 0;

  for (i = 0; LTT(g)[i].n_nd != (Component *) NULL; i++)
    if (*LTT(g)[i].l_status == 'U') {
      ind = index_of(g, LTT(g)[i].n_nd);
      if (ind == -1)
        ind = add_new_node2(g, LTT(g)[i].n_nd);
      RT(g)[ind].hop  = LTT(g)[i].n_link;
      RT(g)[ind].cost = LTT(g)[i].l_cost;
    }

  while (1) {
    min_ind = -1;
    min_cost= INFINITY;
    for  (i=0; i < g->no_of_nodes ; i++)
      if (RT(g)[i].cost < min_cost && SNT(g)[i].mark == 1) {
        min_ind  = i;
        min_cost = RT(g)[i].cost;
      }
    if (min_cost == INFINITY)
      return;

    SNT(g)[min_ind].mark = 0;

    for  (i=0; i < g->no_of_nodes ; i++)
      if (RT(g)[i].cost > min_cost + GTT(g)[min_ind][i]
          && SNT(g)[i].mark == 1
          && GTT(g)[min_ind][i] != INFINITY) {
        RT(g)[i].cost = min_cost + GTT(g)[min_ind][i];
        RT(g)[i].hop  = RT(g)[min_ind].hop;
      }
```

```
  }

}


/*********************************************************/
static int link_costs(g)
register Ospft *g;
{
  int so_ind, neig_ind, i;

  so_ind = index_of(g, g->node);
  inf_global_topology_table(g, so_ind);

  for (i = 0; LTT(g)[i].n_nd != (Component *) NULL; i++) {
    neig_ind    = index_of(g, LTT(g)[i].n_nd);
    if (neig_ind == -1)
      neig_ind = add_new_node2(g, LTT(g)[i].n_nd);
    LTT(g)[i].l_cost = link_cost((Routet *)g, i);
    GTT(g)[so_ind][neig_ind] = LTT(g)[i].l_cost;
  }

  return i;
}


/*********************************************************/
static void broadcast_pkt(g, pkt)
register Ospft *g;
register Packet *pkt;
{
 Packet *p;
 int i, rt_pkts_count;

   rt_pkts_count = 0;

   for (i = 0; LTT(g)[i].n_nd != (Component *) NULL; i++)
   {
    if (*LTT(g)[i].l_status == 'U')
    {
     if ( g->Conversa[i]->Estado <= WAY2 ||
          g->Conversa[i]->Estado == FULL )
     {
      p = pk_alloc();
      p->Ospf_pk=Ospfpk_alloc(p,HELLO);

      memcpy(p, pkt, sizeof(Packet));

      if (((Link *)(LTT(g)[i].n_link))->Conexion->u.i == PTOP)
      {
       p->pk_dest_socket.so_host = LTT(g)[i].n_nd;
      }
      else
      {
       if (((Link *)(LTT(g)[i].n_link))->Conexion->u.i == TORED)
       {
        p->pk_dest_socket.so_host = (Component *)(g->Conversa[i]->NeighborDR->node);
       }
```

```
        else continue;  /* red stub */
       }

       p->pk_dest_socket.so_port   = (Component *) NULL;
       p->pk_source_socket.so_host = (Component *) NULL;
       p->pk_source_socket.so_port = (Component *) NULL;

       p->Ospf_pk->Hello.Vecinos = pkt->Ospf_pk->Hello.Vecinos;
       rt_pkts_count ++;

          Trace_Pkt(g,p);
       ev_call(EV_NODE_PRODUCE, (Component *)g, g->node, g->node->co_action,
            p, LTT(g)[i].n_link);
     }
    }
   }
   pm((Component *)g, OSPF, ROUTING_PKT, rt_pkts_count, 0, 0, 0);
}


/*********************************************************/
OSPF_Packet *Ospfpk_alloc(p,tipo)
register Packet *p;
register int tipo;
{

p->Ospf_pk=(OSPF_Packet *)sim_malloc(sizeof(OSPF_Packet));
p->Ospf_pk->RouterID=(char *)sim_malloc(sizeof(char));

switch (tipo)
{
case HELLO:
      p->Ospf_pk->Hello.DR=(Ospft *)NULL;
      p->Ospf_pk->Hello.BDR=(Ospft *)NULL;
      p->Ospf_pk->Hello.Vecinos=l_create();
         break;

case DD:
      p->Ospf_pk->Dd.HLSAs=l_create();
         break;

case LSREQ:
      p->Ospf_pk->Lsreq.LReq=l_create();
         break;

case LSUPD:
      p->Ospf_pk->Lsupd.Avisos=l_create();
         break;

case LSACK:
      p->Ospf_pk->Lsack.Acks=l_create();
         break;

default: ;
}

return p->Ospf_pk;
```

```
}


/*************************************************************/
static void hello_broadcast(g)
register Ospft *g;
{
  Packet   *pkt;

  g->seq_no->u.i++;
  pkt = pk_alloc();
  pkt->Ospf_pk=Ospfpk_alloc(pkt,HELLO);

  pkt->pk_length      = HELLO_PKT_SIZE + (g->Vecinos->l_len * 4);
  pkt->pk_type              = ROUTE_PACKET;
  pkt->Ospf_pk->Type        = HELLO;
  pkt->Ospf_pk->RouterID=(char *)sim_malloc(sizeof(char)*15);
  strcpy(pkt->Ospf_pk->RouterID,g->RouterID);
  pkt->Ospf_pk->AreaID      = g->AreaID;
  pkt->Ospf_pk->Hello.Prioridad = g->Prioridad->u.i;

  pkt->Ospf_pk->Hello.DR=(Ospft *)sim_malloc(sizeof(Ospft));
  memcpy(pkt->Ospf_pk->Hello.DR,g->DR,sizeof(Ospft));

  pkt->Ospf_pk->Hello.BDR=(Ospft *)sim_malloc(sizeof(Ospft));
  memcpy(pkt->Ospf_pk->Hello.BDR,g->BDR,sizeof(Ospft));

  pkt->Ospf_pk->Long        = HELLO_PKT_SIZE + (g->Vecinos->l_len * 4);

  pkt->Ospf_pk->Hello.Vecinos  = l_duplicate(g->Vecinos);

  broadcast_pkt(g, pkt);

  pk_free(pkt);
}


/*************************************************************/
static caddr_t ospf_broadcast(g)
register Ospft *g;
{
  unsigned int time_now, ticks;

  time_now = ev_now();
  ticks   = 220000; /* SECSD_TO_TICKS(g->HelloInterval); */

  ev_enqueue(EV_OSPF_BROADCAST, (Component *)g, (Component *)g,
        time_now + ticks,
        g->ospf_action, (Packet *)NULL, (caddr_t)NULL);

  if (*(char *)(((Nodee *)g->node)->nd_status->u.p) == 'D')
    return((caddr_t)g);

  /* node is UP */
  compute_link_cost_metrics((Routet *)g);

  hello_broadcast(g);
  ospf(g);
```

```
  Trace_TR(g);

  lq_clear(g->Vecinos);

  log_param((Component *)g, g->routing_table);
  log_param((Component *)g, g->global_top);
  log_param((Component *)g, g->seq_no);

  return (caddr_t) g;
}


/*********************************************************/
static caddr_t ospf_processing(g, pkt)
register Ospft *g;
register Packet *pkt;
{
  int i;
  char *linea;

  free_routing_queue((Routet *)g, pkt);       /* standard */


  i = indice_routerid_ltt(g,pkt->Ospf_pk->RouterID);

  if ( i == -1 ) {
   #ifdef DEBUG
    dbg_write(debug_log, DBG_ERR, (Component *)g,"RouterId no es vecino");
   #endif
    return((caddr_t)NULL);
  }

  Trace_Pkt(g,pkt);

  switch (pkt->Ospf_pk->Type) {

      case HELLO :

          AgregaEnListaVecinos(g,pkt->Ospf_pk->RouterID);

          if ( ! EstaEnVecinos(g->RouterID,pkt) ||
              g->Conversa[i]->Estado == DOWN )
          {
           g->Conversa[i]->Estado=INIT;
           g->Conversa[i]->RetransmiteLSA=l_create();
           g->Conversa[i]->BaseLSA=l_create();
           g->Conversa[i]->PedidosLS=l_create();
           g->Conversa[i]->MarcaDd=0;
           CopiarLD(g,i);              /*Inicializa BaseLSA*/
             }
          else
          {
           if ( g->Conversa[i]->Estado == INIT )
           {
             g->Conversa[i]->Estado=EXSTART;
             g->Conversa[i]->NroSecuencia += 1;
             g->Conversa[i]->Master_Slave = 'M';
```

```
            EnviarDD(g,1,1,1,i);
          }
          else
            if ( g->Conversa[i]->Estado == FULL )
              g->Conversa[i]->Estado=WAY2;
              ospf_generalsa(g);
        }

        break;

    case DD :
        if ( g->Conversa[i]->Estado == EXSTART )
        {

            if ( pkt->Ospf_pk->Dd.Ibit == 1 &&
                pkt->Ospf_pk->Dd.Mbit == 1 &&
                pkt->Ospf_pk->Dd.MSbit == 1 &&
                strcmp(pkt->Ospf_pk->RouterID,g->RouterID) == 1 )
            {
                g->Conversa[i]->Master_Slave = 'S';
                g->Conversa[i]->NroSecuencia=pkt->Ospf_pk->Dd.Seq;
                g->Conversa[i]->Estado=EXCHANGE;
                EnviarDD(g,1,1,0,i);
            }

            if ( pkt->Ospf_pk->Dd.MSbit == 0 &&
                pkt->Ospf_pk->Dd.Seq == g->Conversa[i]->NroSecuencia &&
                strcmp(pkt->Ospf_pk->RouterID,g->RouterID) == -1 )
            {
                g->Conversa[i]->Master_Slave = 'M';
                g->Conversa[i]->NroSecuencia += 1;
                g->Conversa[i]->Estado=EXCHANGE;
                EnviarDD(g,0,1,1,i);
            }
        }

        if ( g->Conversa[i]->Estado == EXCHANGE )
        {

            if ( g->Conversa[i]->Master_Slave == 'M' &&
                g->Conversa[i]->NroSecuencia==pkt->Ospf_pk->Dd.Seq )
            {
                ProcesoDD(g,pkt,i);

                if ( pkt->Ospf_pk->Dd.Mbit == 0 &&
                    g->Conversa[i]->BaseLSA->l_len == 0 )
                {
                  if ( ! HayPedidosLS(g,i) )
                  {
                    g->Conversa[i]->Estado=FULL;
                    ospf_generalsa(g);
                  }
                  else
                  {
                    g->Conversa[i]->Estado=LOADING;
                    EnviarLSReq(g,i);
                  }
                }
```

```
                else
                {
                   g->Conversa[i]->NroSecuencia += 1;
                   EnviarDD(g,0,1,1,i);
                }
            }

            if ( g->Conversa[i]->Master_Slave == 'S' &&
                g->Conversa[i]->NroSecuencia+1 == pkt->Ospf_pk->Dd.Seq )
            {
                ProcesoDD(g,pkt,i);
                g->Conversa[i]->NroSecuencia=pkt->Ospf_pk->Dd.Seq ;
                EnviarDD(g,0,1,0,i);

                if ( pkt->Ospf_pk->Dd.Mbit == 0 &&
                    g->Conversa[i]->BaseLSA->l_len == 0)
                {
                   if ( ! HayPedidosLS(g,i) )
                   {
                      g->Conversa[i]->Estado=FULL;
                      ospf_generalsa(g);
                   }
                   else
                   {
                      g->Conversa[i]->Estado=LOADING;
                      EnviarLSReq(g,i);
                   }
                }
            }

        }

        break;

    case LSREQ :

        if ( g->Conversa[i]->Estado == FULL )
        {
            EnviarLSUpd(g,i,pkt->Ospf_pk->Lsreq.LReq);
        }

        if ( g->Conversa[i]->Estado == LOADING )
        {
            EnviarLSUpd(g,i,pkt->Ospf_pk->Lsreq.LReq);

            if ( HayPedidosLS(g,i) )
                EnviarLSReq(g,i);
            else
            {
                g->Conversa[i]->Estado = FULL;
                ospf_generalsa(g);
            }
        }

        break;

    case LSUPD :
```

```
            if ( g->Conversa[i]->Estado == LOADING ||
                 g->Conversa[i]->Estado == EXCHANGE ||
                 g->Conversa[i]->Estado == WAY2 )
            {
                 ProcesarLSUpd(g,pkt);

                 if ( HayPedidosLS(g,i) )
                      EnviarLSReq(g,i);
                 else
                 {
                      g->Conversa[i]->Estado = FULL;
                      ospf_generalsa(g);
                 }
            }

            break;

       case LSACK :
            if ( g->Conversa[i]->Estado >= EXCHANGE )
            {
                 ProcesarLSAck(g,pkt,i);
            }

            break;


     default : ;
   }

   Trace_Conv(g);

   pk_free(pkt);

   log_param((Component *)g, g->routing_table);
   log_param((Component *)g, g->global_top);
   log_param((Component *)g, g->seq_no);
   log_param((Component *)g, g->seq_no_table);

   schedule_next_EV_ROUTE_PROCESSING((Routet *)g); /* standard */

   return((caddr_t)g);
}


/*************************************************************/
static int routing_processing_time2(g, pkt)
register Ospft *g;
register Packet *pkt;
{
   int src_ind;

   switch (pkt->rt_pk.rt_type) {
    case HELLO :
      return 54000;
    case DD :
      return 54000;
    case LSREQ :
      return 54000;
```

```
    case LSUPD :
      return 54000;
    case LSACK :
      return 54000;
    default : ;
    }
   return 270;
}


/*************************************************************/
char *make_gtt_text2(g, gtt)
Ospft *g;
Param  *gtt;
{
  int i, j;
  extern char text[];
  extern char line[800];

  sprintf(text, "Topology Table of %s$          ",
      g->ospf_name);

  if (!g->routing_table->u.p)
    return text;

  for (i= 0; i < g->no_of_nodes; i++) {
    sprintf(line, "%-11s ", RT(g)[i].dest->co_name);
    strncat(text, line, sizeof(line));
  }
  strcat(text, "$");

  for (i = 0; i < g->no_of_nodes; i++){
       sprintf(line, "%-11s ", RT(g)[i].dest->co_name);
       strncat(text, line, sizeof(line));
       for (j=0; j < g->no_of_nodes; j++){
         sprintf(line, "%-11d ", GTT(g)[i][j]);
         strncat(text, line, sizeof(line));
       }
    strcat(text, "$");
  }

  return text;
}


/*************************************************************/
int indice_routerid_ltt(g,routerid)
register Ospft *g;
register char *routerid;
{
register int i;

  for ( i=0 ; g->node != LTT(g)[i].n_nd &&
          LTT(g)[i].n_nd != (Component *)NULL &&
          strcmp( ((Ospft *)((Nodee *) LTT(g)[i].n_nd)->nd_ptr_routing_mod)->RouterID,routerid )
!= 0 ;
       i++ ) ;
```

```
 if ( LTT(g)[i].n_nd == (Component *)NULL )
 {
   i = -1;
 }

return i;
}


/*************************************************************/
AgregaEnListaVecinos(g,routerid)
register Ospft *g;
register char *routerid;
{

        l_addt(g->Vecinos,routerid);

}


/*************************************************************/
int EstaEnVecinos(routerid,pkt)
register char *routerid;
register Packet *pkt;
{
l_elt *elem;
char  *erid;
int sal;

elem=(l_elt *)sim_malloc(sizeof(l_elt));

sal=0;

for ( elem=pkt->Ospf_pk->Hello.Vecinos->l_head;
     elem != NULL && sal ==  0; elem=elem->le_next )
{

  erid=(char *)sim_malloc(sizeof(char));

  erid = (char *) elem->le_data;
  if ( strcmp(erid,routerid) == 0 )
    sal=1;
}

erid=NULL;
elem=NULL;

return sal;

}


/*************************************************************/
EnviarDD(g,ibit,mbit,msbit,inx)
register Ospft *g;
register int ibit,mbit,msbit,inx;
{
Packet    *pkt,*p;
```

```
LS_Header *hls;
l_elt    *ele;
int       k;
LSA       *lsa;

pkt = pk_alloc();
pkt->Ospf_pk = Ospfpk_alloc(pkt,DD);

pkt->pk_dest_socket.so_host = LTT(g)[inx].n_nd;
pkt->pk_type = ROUTE_PACKET;
pkt->Ospf_pk->Type = DD;
pkt->Ospf_pk->RouterID=(char *)sim_malloc(sizeof(char)*15);
strcpy(pkt->Ospf_pk->RouterID,g->RouterID);
pkt->Ospf_pk->AreaID = g->AreaID;
pkt->Ospf_pk->Dd.Ibit = ibit;
pkt->Ospf_pk->Dd.Mbit = mbit;
pkt->Ospf_pk->Dd.MSbit = msbit;
pkt->Ospf_pk->Dd.Seq = g->Conversa[inx]->NroSecuencia;

pkt->Ospf_pk->Dd.HLSAs = l_create();
k=0;

if ( ibit == 0 )
{
 ele=(l_elt *)sim_malloc(sizeof(l_elt));

 ele=g->Conversa[inx]->BaseLSA->l_head;

 while ( k <= 1400 && ele != NULL )
 {
  hls=(LS_Header *)sim_malloc(sizeof(LS_Header));
  lsa=(LSA *)sim_malloc(sizeof(LSA));

  lsa=(LSA *) ele->le_data;

  hls->Lsage = lsa->Header.Lsage;
  hls->Lstype = lsa->Header.Lstype;
  hls->LsID=(char *)sim_malloc(sizeof(char)*15);
  strcpy(hls->LsID,lsa->Header.LsID);
  hls->AdvRouting=(char *)sim_malloc(sizeof(char)*15);
  strcpy(hls->AdvRouting,lsa->Header.AdvRouting);
  hls->Lsseq = lsa->Header.Lsseq;
  hls->Lslong = lsa->Header.Lslong;

  l_addt(pkt->Ospf_pk->Dd.HLSAs, hls);

  g->Conversa[inx]->MarcaDd+=1;

  if ( msbit == 0 )
          SacarLsa(g,inx);

  ele=ele->le_next;
  k+=20;

 }

 if ( ele == NULL )
 {
```

```
   pkt->Ospf_pk->Dd.Mbit=0;  /* No hay mas lsas para mandar */
  }
  else
  {
   pkt->Ospf_pk->Dd.Mbit=1;
  }

  hls=NULL;
  lsa=NULL;
  ele=NULL;

}

pkt->Ospf_pk->Long = 24 + 8 + k;
pkt->pk_length = 24 + 8 + k;

p = pk_alloc();
p->Ospf_pk = Ospfpk_alloc(p,DD);

memcpy(p, pkt, sizeof(Packet));

if (((Link *)(LTT(g)[inx].n_link))->Conexion->u.i == PTOP)
{
  p->pk_dest_socket.so_host = LTT(g)[inx].n_nd;
}
else
{
  if (((Link *)(LTT(g)[inx].n_link))->Conexion->u.i == TORED)
  {
    p->pk_dest_socket.so_host = (Component *)(g->Conversa[inx]->NeighborDR->node);
  }
}
p->pk_dest_socket.so_port   = (Component *) NULL;
p->pk_source_socket.so_host = (Component *) NULL;
p->pk_source_socket.so_port = (Component *) NULL;

Trace_Pkt(g,p);

ev_call(EV_NODE_PRODUCE, (Component *)g, g->node, g->node->co_action,
      p, LTT(g)[inx].n_link);
pm((Component *)g, OSPF, ROUTING_PKT, 1, 0, 0, 0);

pk_free(pkt);

}


/*************************************************************/
ProcesoDD(g,pkt,inx)
register Ospft *g;
register Packet *pkt;
register int inx;
{
l_elt    *ele;
int      seclsa,agelsa;
LS_Header *lsa;
ClaveLSA  *cla;
```

```
ele=(l_elt *)sim_malloc(sizeof(l_elt));

if ( pkt->Ospf_pk->Dd.Ibit == 0 )
{
 for ( ele=pkt->Ospf_pk->Dd.HLSAs->l_head; ele != NULL; ele=ele->le_next )
 {
  cla=(ClaveLSA *)sim_malloc(sizeof(ClaveLSA));
  lsa=(LS_Header *)sim_malloc(sizeof(LS_Header));

  lsa = (LS_Header *) ele->le_data;
  cla->Lstype=lsa->Lstype;
  cla->LsID=(char *)sim_malloc(sizeof(char)*15);
  strcpy(cla->LsID,lsa->LsID);
  cla->AdvRouting=(char *)sim_malloc(sizeof(char)*15);
  strcpy(cla->AdvRouting,lsa->AdvRouting);

  SecAgeLsa(g,lsa->Lstype,lsa->LsID,lsa->AdvRouting,&seclsa,&agelsa);

  if ( lsa->Lsseq > seclsa )
    AgregaListaReq(g,inx,cla);

  if ( lsa->Lsseq == seclsa && lsa->Lsage < agelsa )
    AgregaListaReq(g,inx,cla);

 }

 if ( g->Conversa[inx]->Master_Slave == 'M' )
 {
  if ( pkt->Ospf_pk->Dd.Seq == g->Conversa[inx]->NroSecuencia )
    SacarLsa(g,inx);
 }

}

cla=NULL;
lsa=NULL;
ele=NULL;

}


/*************************************************************/
SecAgeLsa(g,tipo,id,advr,sec,age)
register Ospft *g;
register int   tipo;
register char  *id,*advr;
register int   *sec,*age;
{
LSA  *lsa;
l_elt *ele;

 ele=(l_elt *)sim_malloc(sizeof(l_elt));

 for ( ele=g->ListaLSA->l_head;ele != NULL;ele=ele->le_next)
 {
  lsa=(LSA *)sim_malloc(sizeof(LSA));

  lsa = (LSA *)ele->le_data;
```

```
   if ( lsa->Header.Lstype == tipo &&
        strcmp(lsa->Header.LsID,id) == 0 &&
        strcmp(lsa->Header.AdvRouting,advr) == 0 )
     break;
 }


 if ( ele == NULL )
 {
  *sec = -1;
  *age = 32767;
 }
 else
 {
  *sec = lsa->Header.Lsseq;
  *age = lsa->Header.Lsage;
 }

lsa=NULL;
ele=NULL;

}


/*************************************************************/
AgregaListaReq(g,inx,cla)
register Ospft    *g;
register int      inx;
register ClaveLSA *cla;
{

 l_addt(g->Conversa[inx]->PedidosLS,cla);

}


/*************************************************************/
SacarLsa(g,inx)
register Ospft *g;
register int   inx;
{
l_elt *ele;
int i;

for (i=1; i <= g->Conversa[inx]->MarcaDd; i++)
 {
   ele=(l_elt *)sim_malloc(sizeof(l_elt));

   ele = le_remh(g->Conversa[inx]->BaseLSA);
 }

g->Conversa[inx]->MarcaDd=0;

ele=NULL;

}

/*************************************************************/
```

```
int HayPedidosLS(g,inx)
register Ospft *g;
register int   inx;
{

 if ( g->Conversa[inx]->PedidosLS->l_len > 0 )
 {
  return 1;
 }
 else
  {
  return 0;
 }

}


/*************************************************************/
EnviarLSReq(g,inx)
register Ospft *g;
register int   inx;
{
Packet   *pkt,*p;
ClaveLSA  *cla;
int      k;
l_elt       *ele;

ele=(l_elt *)sim_malloc(sizeof(l_elt));

  k=0;
  pkt = pk_alloc();
  pkt->Ospf_pk = Ospfpk_alloc(pkt,LSREQ);

  pkt->pk_dest_socket.so_host = LTT(g)[inx].n_nd;
  pkt->pk_type = ROUTE_PACKET;
  pkt->Ospf_pk->Type = LSREQ;
  pkt->Ospf_pk->RouterID=(char *)sim_malloc(sizeof(char)*15);
  strcpy(pkt->Ospf_pk->RouterID,g->RouterID);
  pkt->Ospf_pk->AreaID = g->AreaID;
  pkt->Ospf_pk->Lsreq.LReq = l_create();

  ele=g->Conversa[inx]->PedidosLS->l_head;

  while (k <= 1418 && ele != NULL )
  {
   cla=(ClaveLSA *)sim_malloc(sizeof(ClaveLSA));
   cla = (ClaveLSA *) ele->le_data;

   l_addt(pkt->Ospf_pk->Lsreq.LReq,cla);
   ele=ele->le_next;
   k+=10;
  }

  ele=NULL;
  cla=NULL;

  pkt->Ospf_pk->Long = 24 + k;
  pkt->pk_length = 24 + k;
```

```
  p = pk_alloc();
  p->Ospf_pk = Ospfpk_alloc(p,LSREQ);

  memcpy(p, pkt, sizeof(Packet));

  if (((Link *)(LTT(g)[inx].n_link))->Conexion->u.i == PTOP)
  {
    p->pk_dest_socket.so_host = LTT(g)[inx].n_nd;
  }
  else
  {
    if (((Link *)(LTT(g)[inx].n_link))->Conexion->u.i == TORED)
    {
      p->pk_dest_socket.so_host = (Component *)(g->Conversa[inx]->NeighborDR->node);
    }
  }
  p->pk_dest_socket.so_port   = (Component *) NULL;
  p->pk_source_socket.so_host = (Component *) NULL;
  p->pk_source_socket.so_port = (Component *) NULL;

  Trace_Pkt(g,p);

  ev_call(EV_NODE_PRODUCE, (Component *)g, g->node, g->node->co_action,
        p, LTT(g)[inx].n_link);
  pm((Component *)g, OSPF, ROUTING_PKT, 1, 0, 0, 0);

  pk_free(pkt);

}


/*********************************************************/
EnviarLSUpd(g,inx,avisos)
register Ospft  *g;
register int    inx;
register list   *avisos;
{
Packet    *pupd,*p;
ClaveLSA  *cla;
LSA       *lsa, *avisored, *avisorouter;
int       i,AgregaRouter,AgregaRed,tam;
l_elt     *ele;

ele=(l_elt *)sim_malloc(sizeof(l_elt));

  pupd = pk_alloc();
  pupd->Ospf_pk = Ospfpk_alloc(pupd,LSUPD);
  pupd->pk_dest_socket.so_host = LTT(g)[inx].n_nd;
  pupd->pk_type = ROUTE_PACKET;
  pupd->Ospf_pk->Type = LSUPD;

  pupd->Ospf_pk->RouterID=(char *)sim_malloc(sizeof(char)*15);
  strcpy(pupd->Ospf_pk->RouterID,g->RouterID);
  pupd->Ospf_pk->AreaID = g->AreaID;
  pupd->Ospf_pk->Lsupd.NroLSA=0;
  tam = 24 + 4;    /* Header Ospf + Nro.Adver.*/
```

```
for (ele=avisos->l_head; ele != NULL; ele=ele->le_next)
{
 cla=(ClaveLSA *)sim_malloc(sizeof(ClaveLSA));
 lsa=(LSA *)sim_malloc(sizeof(LSA));
 avisored=(LSA *)sim_malloc(sizeof(LSA));
 avisorouter=(LSA *)sim_malloc(sizeof(LSA));

 cla = (ClaveLSA *) ele->le_data;

 lsa = AvisoEnLista(g->ListaLSA,cla);

 if ( lsa != (LSA *)NULL && lsa->Header.Lstype == 1 )
 { /* es un aviso de router */
  avisorouter->Header.Lsage=lsa->Header.Lsage;
  avisorouter->Header.Lstype=lsa->Header.Lstype;
  avisorouter->Header.LsID=(char *)sim_malloc(sizeof(char)*15);
  strcpy(avisorouter->Header.LsID,lsa->Header.LsID);
  avisorouter->Header.AdvRouting=(char *)sim_malloc(sizeof(char)*15);
  strcpy(avisorouter->Header.AdvRouting,lsa->Header.AdvRouting);
  avisorouter->Header.Lsseq=lsa->Header.Lsseq;
  avisorouter->Header.Lslong=lsa->Header.Lslong;
  avisorouter->Lsrou.Ebit=lsa->Lsrou.Ebit;
  avisorouter->Lsrou.Bbit=lsa->Lsrou.Bbit;
  avisorouter->Lsrou.Nrolinks=lsa->Lsrou.Nrolinks;
  avisorouter->Lsrou.Links=l_duplicate(lsa->Lsrou.Links);

  tam = tam + 20 + 4 + ( 12 * lsa->Lsrou.Links->l_len );
  if ( tam <= 1428 )
  {
   l_addt(pupd->Ospf_pk->Lsupd.Avisos,avisorouter);
   pupd->Ospf_pk->Lsupd.NroLSA+=1;
   AgregaRouter=0;
  }
  else
  {
   AgregaRouter=1;
   pupd->Ospf_pk->Long = tam - (20+4+(12 * lsa->Lsrou.Links->l_len));
   pupd->pk_length = pupd->Ospf_pk->Long;
  }

 }
 else  /* no es aviso de router */
 {
  if ( lsa != (LSA *) NULL && lsa->Header.Lstype == 2 )
  {  /* es un aviso de red */
   avisored->Header.Lsage=lsa->Header.Lsage;
   avisored->Header.Lstype=lsa->Header.Lstype;
   avisored->Header.LsID=(char *)sim_malloc(sizeof(char)*15);
   strcpy(avisored->Header.LsID,lsa->Header.LsID);
   avisored->Header.AdvRouting=(char *)sim_malloc(sizeof(char)*15);
   strcpy(avisored->Header.AdvRouting,lsa->Header.AdvRouting);
   avisored->Header.Lsseq=lsa->Header.Lsseq;
   avisored->Header.Lslong=lsa->Header.Lslong;
   avisored->Lsred.NetworkMask=(char *)sim_malloc(sizeof(char)*15);
   strcpy(avisored->Lsred.NetworkMask,lsa->Lsred.NetworkMask);
   avisored->Lsred.AtachedRouter=l_duplicate(lsa->Lsred.AtachedRouter);

   tam = tam + 20 + 4 + ( 4 * lsa->Lsred.AtachedRouter->l_len );
```

```
    if ( tam <= 1428 )
    {
     l_addt(pupd->Ospf_pk->Lsupd.Avisos,avisored);
     pupd->Ospf_pk->Lsupd.NroLSA+=1;
     AgregaRed=0;
    }
    else
    {
     AgregaRed=1;
     pupd->Ospf_pk->Long = tam - (20+4+(4 * lsa->Lsred.AtachedRouter->l_len));
     pupd->pk_length = pupd->Ospf_pk->Long;
    }

  }
}

if ( tam >= 1428 )
{

  p = pk_alloc();
  p->Ospf_pk = Ospfpk_alloc(p,LSUPD);

  memcpy(p, pupd, sizeof(Packet));

  if (((Link *)(LTT(g)[inx].n_link))->Conexion->u.i == PTOP)
  {
   p->pk_dest_socket.so_host = LTT(g)[inx].n_nd;
  }
  else
  {
   if (((Link *)(LTT(g)[inx].n_link))->Conexion->u.i == TORED)
   {
    p->pk_dest_socket.so_host = (Component *)(g->Conversa[inx]->NeighborDR->node);
   }
  }
  p->pk_dest_socket.so_port   = (Component *) NULL;
  p->pk_source_socket.so_host = (Component *) NULL;
  p->pk_source_socket.so_port = (Component *) NULL;

  Trace_Pkt(g,p);

  ev_call(EV_NODE_PRODUCE, (Component *)g, g->node, g->node->co_action,
       p, LTT(g)[inx].n_link);
  pm((Component *)g, OSPF, ROUTING_PKT, 1, 0, 0, 0);

  pk_free(pupd);

  pupd = pk_alloc();
  pupd->Ospf_pk = Ospfpk_alloc(pupd,LSUPD);

  pupd->pk_dest_socket.so_host = LTT(g)[inx].n_nd;
  pupd->pk_type = ROUTE_PACKET;
  pupd->Ospf_pk->Type = LSUPD;
  pupd->Ospf_pk->RouterID=(char *)sim_malloc(sizeof(char)*15);
  strcpy(pupd->Ospf_pk->RouterID,g->RouterID);
  pupd->Ospf_pk->AreaID = g->AreaID;
  pupd->Ospf_pk->Lsupd.Avisos = l_create();
  pupd->Ospf_pk->Lsupd.NroLSA=0;
```

```
    tam = 24 + 4;

    if ( AgregaRouter )
    {
      l_addt(pupd->Ospf_pk->Lsupd.Avisos,avisorouter);
      pupd->Ospf_pk->Lsupd.NroLSA+=1;
      tam = tam + 20 + 4 + ( 12 * lsa->Lsrou.Links->l_len );
    }
    if ( AgregaRed )
    {
      l_addt(pupd->Ospf_pk->Lsupd.Avisos,avisored);
      pupd->Ospf_pk->Lsupd.NroLSA+=1;
      tam = tam + 20 + 4 + ( 4 * lsa->Lsred.AtachedRouter->l_len );
    }
  }

} /* for */

cla=NULL;
ele=NULL;
lsa=NULL;
avisored=NULL;
avisorouter=NULL;

if ( ele == NULL && tam > 28 )
{
  pupd->Ospf_pk->Long = tam;
  pupd->pk_length = tam;


  p = pk_alloc();
  p->Ospf_pk = Ospfpk_alloc(p,LSUPD);

  memcpy(p, pupd, sizeof(Packet));

  if (((Link *)(LTT(g)[inx].n_link))->Conexion->u.i == PTOP)
  {
    p->pk_dest_socket.so_host = LTT(g)[inx].n_nd;
  }
  else
  {
    if (((Link *)(LTT(g)[inx].n_link))->Conexion->u.i == TORED)
    {
      p->pk_dest_socket.so_host = (Component *)(g->Conversa[inx]->NeighborDR->node);
    }
  }
  p->pk_dest_socket.so_port   = (Component *) NULL;
  p->pk_source_socket.so_host = (Component *) NULL;
  p->pk_source_socket.so_port = (Component *) NULL;

  Trace_Pkt(g,p);

  ev_call(EV_NODE_PRODUCE, (Component *)g, g->node, g->node->co_action,
        p, LTT(g)[inx].n_link);
  pm((Component *)g, OSPF, ROUTING_PKT, 1, 0, 0, 0);

  pk_free(pupd);
```

```
  }

}


/*********************************************************/
ProcesarLSUpd(g,pkt)
register Ospft *g;
register Packet *pkt;
{
l_elt *ele;
LSA   *lsa;
ClaveLSA *cla;

 ele=(l_elt *)sim_malloc(sizeof(l_elt));

 for (ele=pkt->Ospf_pk->Lsupd.Avisos->l_head;ele != NULL; ele=ele->le_next)
 {

 lsa=(LSA *)sim_malloc(sizeof(LSA));
 cla=(ClaveLSA *)sim_malloc(sizeof(ClaveLSA));

 lsa=(LSA *) ele->le_data;
 cla->Lstype=lsa->Header.Lstype;
 cla->LsID=(char *)sim_malloc(sizeof(char)*15);
 strcpy(cla->LsID,lsa->Header.LsID);
 cla->AdvRouting=(char *)sim_malloc(sizeof(char)*15);
 strcpy(cla->AdvRouting,lsa->Header.AdvRouting);

 if ( ! AvisoEnLista(g->ListaLSA,cla) || avisomasrec(g,lsa->Header) == 1 )
 {
   FluirAviso(g,lsa);
   /*RemoverAviso(g,lsa);*/
   ActualizarLD(g,lsa);

   Trace_ListaLSA(g);

   AceptarAviso(g,lsa);
   SelfAviso(g,lsa);
 }
 else
 {
   if ( AvisoEnLista(g->ListaLSA,cla) && avisomasrec(g,lsa->Header) == 2 )
   {
     AceptarAviso(g,lsa);
   }
 }

 }

ele=NULL;
lsa=NULL;
cla=NULL;

}

/*********************************************************/
CopiarLD(g,inx)
```

```
register Ospft *g;
register int   inx;
{
l_elt *ele;
LSA *lsa;

 ele=(l_elt *)sim_malloc(sizeof(l_elt));

 g->Conversa[inx]->BaseLSA=l_duplicate(g->ListaLSA);

 for ( ele=g->Conversa[inx]->BaseLSA->l_head; ele != NULL; ele=ele->le_next )
 {
  lsa=(LSA *)sim_malloc(sizeof(LSA));

  lsa=(LSA *)ele->le_data;
  if ( lsa->Header.Lsage == MAXAGE )
   l_del(g->Conversa[inx]->BaseLSA,ele);
 }

lsa=NULL;

}


/*************************************************************************/
LSA *AvisoEnLista(listax,cla)
register list *listax;  /* Lista de LSA */
register ClaveLSA *cla;
{

/*RETORNA UN PUNTERO A UN LSA */
l_elt *ele;
LSA   *lsax;
int   salt;

salt=0;
ele=(l_elt *)sim_malloc(sizeof(l_elt));
lsax=(LSA *)sim_malloc(sizeof(LSA));

for (ele=listax->l_head; (salt !=1 && ele != NULL); ele=ele->le_next)
{
 lsax=(LSA *)sim_malloc(sizeof(LSA));

 lsax = (LSA *) ele->le_data;

 if (cla->Lstype == lsax->Header.Lstype &&
    strcmp(cla->LsID,lsax->Header.LsID) == 0 &&
    strcmp(cla->AdvRouting,lsax->Header.AdvRouting) == 0 )
 {
  salt=1;
 }
}

ele=NULL;

if (salt==0)
  lsax=((LSA *) NULL);
```

```
return(lsax);

}


/*************************************************************************/
ClaveLSA *ClaveEnLista(listax,cla)
register list *listax;  /* Lista de LSA */
register ClaveLSA *cla;
{

/*RETORNA UN PUNTERO A UN ClaveLSA */
l_elt *ele;
ClaveLSA   *lsa;

ele=(l_elt *)sim_malloc(sizeof(l_elt));

for (ele=listax->l_head; ele != NULL; ele=ele->le_next)
{
  lsa=(ClaveLSA *)sim_malloc(sizeof(ClaveLSA));

  lsa = (ClaveLSA *) ele->le_data;

  if (cla->Lstype == lsa->Lstype &&
      strcmp(cla->LsID,lsa->LsID) == 0 &&
      strcmp(cla->AdvRouting,lsa->AdvRouting) == 0 )
  {
    ele=NULL;
    return lsa;
  }
}
return ((ClaveLSA *)NULL);

}
/*********************************************************/
ProcesarLSAck(g,pkt,inx)
register Ospft *g;
register Packet *pkt;
int inx;
{
l_elt     *ele;
LS_Header  *lsh;
LSA        *lsa;
ClaveLSA   *cla;

ele=(l_elt *)sim_malloc(sizeof(l_elt));

for (ele=pkt->Ospf_pk->Lsack.Acks->l_head;ele != NULL; ele=ele->le_next)
{
  lsh=(LS_Header *)sim_malloc(sizeof(LS_Header));
  lsa=(LSA *)sim_malloc(sizeof(LSA));
  cla=(ClaveLSA *)sim_malloc(sizeof(ClaveLSA));

  lsh = (LS_Header *) ele->le_data;

  cla->Lstype=lsh->Lstype;
  cla->LsID=(char *)sim_malloc(sizeof(char)*15);
  strcpy(cla->LsID,lsh->LsID);
```

```
 cla->AdvRouting=(char *)sim_malloc(sizeof(char)*15);
 strcpy(cla->AdvRouting,lsh->AdvRouting);

 lsa = AvisoEnLista(g->Conversa[inx]->RetransmiteLSA,cla);

 if ( lsa != (LSA *) NULL )
 {
  l_del(g->Conversa[inx]->RetransmiteLSA,lsa);
 }
}

lsh=NULL;
lsa=NULL;
cla=NULL;

}


/*************************************************************************/
int avisomasrec(g,hlsa)
register Ospft *g;
register LS_Header hlsa;
{
ClaveLSA  *cla;
LS_Header *hlsa1;
LSA       *lsa;
int       sal;
/*RETORNA EN SAL: 0 SI EL AVISO ES MENOS RECIENTE
                  1 ES MAS RECIENTE
                  2 SI ES LA MISMA INSTACIA */

cla=(ClaveLSA *)sim_malloc(sizeof(ClaveLSA));
hlsa1=(LS_Header *)sim_malloc(sizeof(LS_Header));
lsa=(LSA *)sim_malloc(sizeof(LSA));

sal = 1;
cla->Lstype=hlsa.Lstype;
cla->LsID=(char *)sim_malloc(sizeof(char)*15);
strcpy(cla->LsID,hlsa.LsID);
cla->AdvRouting=(char *)sim_malloc(sizeof(char)*15);
strcpy(cla->AdvRouting,hlsa.AdvRouting);

lsa = AvisoEnLista(g->ListaLSA,cla);
if (lsa != (LSA *) NULL )
{
  sal=Reciente(lsa->Header,hlsa);
}

cla=NULL;
hlsa1=NULL;
lsa=NULL;

return sal;

}

/*************************************************************************/
int Reciente(hlsa1,hlsa)
```

```
register LS_Header hlsa1,hlsa;
{

/*RETORNA
 0 SI EL 2D0 PARAMETRO ES MENOS RECIENTE,
 1 SI EL 2DO PARAMETRO ES MAS RECIENTE,
 2 SI SON LA MISMA INSTANCIA
*/
int sal;

if ( hlsa1.Lsseq > hlsa.Lsseq ||
    (hlsa1.Lsage == MAXAGE && hlsa.Lsage < MAXAGE) )
{
  sal=0;
}
else
{
  if ( hlsa1.Lsage < MAXAGE && hlsa.Lsage == MAXAGE )
  {
    sal=1;
  }
}

if ( sal == 0 && hlsa1.Lsseq < hlsa.Lsseq )
{
  if ( hlsa1.Lsage > MAXAGEDIFF || hlsa.Lsage > MAXAGEDIFF )
  {
    if ( hlsa1.Lsage < hlsa.Lsage )
    {
      sal=1;
    }
    else
    {
     if ( hlsa1.Lsage == hlsa.Lsage )
     {
       sal=2;
     }
    }
  }
}

return sal;

}


/*****************************************************************************/
FluirAviso(g,lsa)
register Ospft *g;
register LSA *lsa;

{
ClaveLSA   *cla;
list       *avisos;
int        i;


for (i = 0; LTT(g)[i].n_nd != (Component *) NULL; i++)
```

```
{
 if (*LTT(g)[i].l_status == 'U')
 {
  cla=(ClaveLSA *)sim_malloc(sizeof(ClaveLSA));
  avisos=(list *)sim_malloc(sizeof(list));

  cla->Lstype=lsa->Header.Lstype;
  cla->LsID=(char *)sim_malloc(sizeof(char)*15);
  strcpy(cla->LsID,lsa->Header.LsID);
  cla->AdvRouting=(char *)sim_malloc(sizeof(char)*15);
  strcpy(cla->AdvRouting,lsa->Header.AdvRouting);

  EliminaListaReq(g,cla,i);

  if ( ((((Link *)(LTT(g)[i].n_link))->Conexion->u.i == TORED &&
      strcmp(g->RouterID,g->DR->RouterID) == 0 ) ||
      ((Link *)(LTT(g)[i].n_link))->Conexion->u.i == PTOP) &&
      ! VinodelVecino(g,i,lsa) )
  {

    if ( g->Conversa[i]->Estado == EXCHANGE ||
       g->Conversa[i]->Estado == LOADING ||
       g->Conversa[i]->Estado == WAY2 )
    {

      if ( avisomasrec(g,lsa->Header) == 1 )
      {                    /* Si es mas reciente Fluir aviso */
        AgregaListaRet(g,i,lsa);
        avisos=l_create();
        l_addt(avisos,lsa);
        EnviarLSUpd(g,i,avisos);
      }
    }

  }
 }
}

cla=NULL;
avisos=NULL;

}


/*****************************************************************************/
ActualizarLD(g,lsa)
register Ospft *g;
register LSA *lsa;
{
LSA   *lsa1;
int   i;
ClaveLSA *cla;

lsa1=(LSA *)sim_malloc(sizeof(LSA));
cla=(ClaveLSA *)sim_malloc(sizeof(ClaveLSA));

if ( (lsa->Header.Lstype == 1 || lsa->Header.Lstype == 2) &&
   lsa->Header.Lsage != MAXAGE )
```

```
{
  ospf(g);
  Trace_TR(g);
}
cla->Lstype=lsa->Header.Lstype;
cla->LsID=(char *)sim_malloc(sizeof(char)*15);
strcpy(cla->LsID,lsa->Header.LsID);
cla->AdvRouting=(char *)sim_malloc(sizeof(char)*15);
strcpy(cla->AdvRouting,lsa->Header.AdvRouting);

lsa1 = AvisoEnLista(g->ListaLSA,cla);
if (lsa1 != (LSA *) NULL)
{
  l_del(g->ListaLSA,lsa1);
}
l_addt(g->ListaLSA,lsa);

/*BORRAR LA LISTA DE RETRANSMICIONES DE CADA CONVERSACION */
for (i=1; g->Conversa[i]->Estado != NULO ; i++)
{
   cla=(ClaveLSA *)sim_malloc(sizeof(ClaveLSA));
   lsa1=(LSA *)sim_malloc(sizeof(LSA));

   /* seguro que el aviso esta una sola vez en la lista de retransmiciones */
   cla->Lstype=lsa->Header.Lstype;
   cla->LsID=(char *)sim_malloc(sizeof(char)*15);
   strcpy(cla->LsID,lsa->Header.LsID);
   cla->AdvRouting=(char *)sim_malloc(sizeof(char)*15);
   strcpy(cla->AdvRouting,lsa->Header.AdvRouting);

   if ( lsa1 = AvisoEnLista(g->Conversa[i]->RetransmiteLSA,cla) )
   {
     if ( Reciente(lsa1->Header,lsa->Header) == 1 )
     {
       EliminaListaRet(g,lsa1,i);
     }
   }
}

lsa1=NULL;
cla=NULL;

}


/*****************************************************************************/
AceptarAviso(g,lsa)
register Ospft *g;
register LSA *lsa;
{
int i;
l_elt *ele;

for (i=1; g->Conversa[i]->Estado != NULO ; i++)
{
 ele=(l_elt *)sim_malloc(sizeof(l_elt));

 ele=EliminaListaRet(g,lsa,i);
```

```
 if ( g->Conversa[i]->Estado == WAY2 && ele == NULL ) /* Proceso de flooding */
 {
   EnviarACK(g,i,lsa);
 }
}

ele=NULL;

}

/*****************************************************************************/
SelfAviso(g,lsa)
register Ospft *g;
register LSA   *lsa;
{
LSA   *lsa1;
ClaveLSA *cla;

lsa1=(LSA *)sim_malloc(sizeof(LSA));
cla=(ClaveLSA *)sim_malloc(sizeof(ClaveLSA));

if ( strcmp(lsa->Header.AdvRouting,g->RouterID) == 0 )
{
  cla->Lstype=lsa->Header.Lstype;
  cla->LsID=(char *)sim_malloc(sizeof(char)*15);
  strcpy(cla->LsID,lsa->Header.LsID);
  cla->AdvRouting=(char *)sim_malloc(sizeof(char)*15);
  strcpy(cla->AdvRouting,lsa->Header.AdvRouting);

  if ( lsa1 = AvisoEnLista(g->ListaLSA,cla) )
  {
    l_del(g->ListaLSA,lsa1);
  }

/*  if ( lsa->Header.Lsage == REFRESHLSA )
    lsa->Header.Lsage=0;
*/

  lsa->Header.Lsseq++;
  l_addt(g->ListaLSA,lsa);
  FluirAviso(g,lsa);
}

lsa1=NULL;
cla=NULL;

}


/*****************************************************************************/
EliminaListaReq(g,cla,inx)
register Ospft    *g;
register ClaveLSA *cla;
register int      inx;
{
ClaveLSA *ele;
```

```
ele=ClaveEnLista(g->Conversa[inx]->PedidosLS,cla);

l_del(g->Conversa[inx]->PedidosLS,ele);

ele=NULL;

}


/*************************************************************************/
l_elt *EliminaListaRet(g,lsa,inx)
register Ospft    *g;
register LSA      *lsa;
register int      inx;
{
LSA *ele;

ele=AvisoEnLista(g->Conversa[inx]->RetransmiteLSA,lsa);

if (ele == NULL)
        return (l_elt *)NULL;
else
        return ((l_elt *) l_del(g->Conversa[inx]->RetransmiteLSA,ele));

}


/*************************************************************************/
l_elt *AgregaListaRet(g,inx,lsa)
register Ospft *g;
register int   inx;
register LSA   *lsa;
{

  return l_addt(g->Conversa[inx]->RetransmiteLSA,lsa);

}


/*************************************************************************/
int VinodelVecino(g,inx,lsa)
register Ospft *g;
register int   inx;
register LSA   *lsa;
{

  return (strcmp(lsa->Header.AdvRouting,g->Conversa[inx]->NeighborID)==0 ||
        strcmp(lsa->Header.AdvRouting,g->Conversa[inx]->NeighborDR->RouterID)==0 ||
        strcmp(lsa->Header.AdvRouting,g->Conversa[inx]->NeighborBDR->RouterID)==0 );

}


/*************************************************************************/
EnviarACK(g,inx,lsa)
register Ospft *g;
register int   inx;
register LSA   *lsa;
```

```
{
Packet *pkt,*p;
LS_Header *hlsa;

hlsa=(LS_Header *)sim_malloc(sizeof(LS_Header));

pkt = pk_alloc();
pkt->Ospf_pk = Ospfpk_alloc(pkt,LSACK);

pkt->pk_dest_socket.so_host = LTT(g)[inx].n_nd;
pkt->pk_type = ROUTE_PACKET;
pkt->Ospf_pk->Type = LSACK;
pkt->Ospf_pk->RouterID=(char *)sim_malloc(sizeof(char)*15);
strcpy(pkt->Ospf_pk->RouterID,g->RouterID);
pkt->Ospf_pk->AreaID = g->AreaID;
pkt->Ospf_pk->Long= 24 + 20;    /* Header Ospf + Header LSA */
pkt->pk_length= 24 + 20;

pkt->Ospf_pk->Lsack.Acks = l_create();

hlsa->Lsage=lsa->Header.Lsage;
hlsa->Lstype=lsa->Header.Lstype;
hlsa->LsID=(char *)sim_malloc(sizeof(char)*15);
strcpy(hlsa->LsID,lsa->Header.LsID);
hlsa->AdvRouting=(char *)sim_malloc(sizeof(char)*15);
strcpy(hlsa->AdvRouting,lsa->Header.AdvRouting);
hlsa->Lsseq=lsa->Header.Lsseq;
hlsa->Lslong=lsa->Header.Lslong;

l_addt(pkt->Ospf_pk->Lsack.Acks, hlsa);

p = pk_alloc();
p->Ospf_pk = Ospfpk_alloc(p,LSACK);

memcpy(p, pkt, sizeof(Packet));

if (((Link *)(LTT(g)[inx].n_link))->Conexion->u.i == PTOP)
{
  p->pk_dest_socket.so_host = LTT(g)[inx].n_nd;
}
else
{
 if (((Link *)(LTT(g)[inx].n_link))->Conexion->u.i == TORED)
 {
   p->pk_dest_socket.so_host = (Component *)(g->Conversa[inx]->NeighborDR->node);
 }
}
p->pk_dest_socket.so_port   = (Component *) NULL;
p->pk_source_socket.so_host = (Component *) NULL;
p->pk_source_socket.so_port = (Component *) NULL;

Trace_Pkt(g,p);

ev_call(EV_NODE_PRODUCE, (Component *)g, g->node, g->node->co_action,
      p, LTT(g)[inx].n_link);
pm((Component *)g, OSPF, ROUTING_PKT, 1, 0, 0, 0);

pk_free(pkt);
```

```
}

/*************************************************************************/

static caddr_t ospf_generalsa(g)
register Ospft *g;
{
LSA *lsa,*lsa1;
LinkRouter *lr;
ClaveLSA *cla;
int i;
char *nid;
unsigned int time_now;
unsigned long ticks;

l_elt *ele,*ele2;

/*
time_now = ev_now();
ticks    = SECSD_TO_TICKS(60) * REFRESHLSA;
ev_enqueue(EV_OSPF_UP_INTERF, (Component *)g, (Component *)g,time_now + ticks,
        g->ospf_action, (Packet *)NULL, (caddr_t)NULL);
*/

lsa=(LSA *)sim_malloc(sizeof(LSA));
lsa1=(LSA *)sim_malloc(sizeof(LSA));
cla=(ClaveLSA *)sim_malloc(sizeof(ClaveLSA));
lr=(LinkRouter *)sim_malloc(sizeof(LinkRouter));
nid=(char *)sim_malloc(sizeof(char));

/********Genera Aviso de Router********/
lsa->Header.Lsage=0;
lsa->Header.Lstype=1;
lsa->Header.LsID=(char *)sim_malloc(sizeof(char)*15);
strcpy(lsa->Header.LsID,g->RouterID);
lsa->Header.AdvRouting=(char *)sim_malloc(sizeof(char)*15);
strcpy(lsa->Header.AdvRouting,g->RouterID);
lsa->Header.Lsseq=0;
lsa->Header.Lslong=20+4;
lsa->Lsrou.Ebit=0;
lsa->Lsrou.Bbit=0;
lsa->Lsrou.Nrolinks=0;
lsa->Lsrou.Links=l_create();

for ( i=0; LTT(g)[i].n_link != (Component *)NULL; i++ )    /* Por cada interface */
{
 if ( *LTT(g)[i].l_status == 'U' )
 {
  lr=(LinkRouter *)sim_malloc(sizeof(LinkRouter));

  switch ( ((Link *)(LTT(g)[i].n_link))->Conexion->u.i )
  {

    case PTOP:
     lr->LinkID=(char *)sim_malloc(sizeof(char)*15);
        strcpy(lr->LinkID,g->Conversa[i]->NeighborID);
```

```
        lr->LinkData=(char *)sim_malloc(sizeof(char)*15);
        strcpy(lr->LinkData,g->Conversa[i]->NeighborIP);
        lr->Typeconexion=1;
        lr->Tos0metric=LTT(g)[i].l_cost;
        lsa->Lsrou.Nrolinks++;
        l_addt(lsa->Lsrou.Links,lr);
        lsa->Header.Lslong+=12;
      break;

    case TORED:
        lr->LinkID=(char *)sim_malloc(sizeof(char)*15);
        strcpy(lr->LinkID,g->DR->ospf_name);  /* LinkAINodo(g,(Nodee *)g->DR->node); */
        lr->LinkData=(char *)sim_malloc(sizeof(char)*15);
        strcpy(lr->LinkData,g->Conversa[i]->NeighborIP);
        lr->Typeconexion=2;
        lr->Tos0metric=LTT(g)[i].l_cost;
        lsa->Lsrou.Nrolinks++;
        l_addt(lsa->Lsrou.Links,lr);
        lsa->Header.Lslong+=12;
      break;

    case TOSTUB:
      lr->LinkID=(char *)sim_malloc(sizeof(char)*15);
      strcpy(lr->LinkID,g->Conversa[i]->NeighborID);
        lr->LinkData="0xffffff00";
        lr->Typeconexion=3;
        lr->Tos0metric=LTT(g)[i].l_cost;
        lsa->Lsrou.Nrolinks++;
        l_addt(lsa->Lsrou.Links,lr);
        lsa->Header.Lslong+=12;
      break;

    default: ;
   }
  }

}

cla->Lstype=lsa->Header.Lstype;
cla->LsID=(char *)sim_malloc(sizeof(char)*15);
strcpy(cla->LsID,lsa->Header.LsID);
cla->AdvRouting=(char *)sim_malloc(sizeof(char)*15);
strcpy(cla->AdvRouting,lsa->Header.AdvRouting);

if ( lsa1 = AvisoEnLista(g->ListaLSA,cla) )
{
  l_del(g->ListaLSA,lsa1);
}

l_addt(g->ListaLSA,lsa);
FluirAviso(g,lsa);

lsa=(LSA *)sim_malloc(sizeof(LSA));
lsa1=(LSA *)sim_malloc(sizeof(LSA));
cla=(ClaveLSA *)sim_malloc(sizeof(ClaveLSA));
lr=(LinkRouter *)sim_malloc(sizeof(LinkRouter));
nid=(char *)sim_malloc(sizeof(char));
```

```
/********Solo el DR Genera un Aviso de Red********/
if ( strcmp(g->DR->RouterID,g->RouterID) == 0 && g->Prioridad->u.i > 0)
{
  lsa->Header.Lsage=0;
  lsa->Header.Lstype=2;
  lsa->Header.LsID=(char *)sim_malloc(sizeof(char)*15);
  strcpy(lsa->Header.LsID,g->RouterID);
  lsa->Header.AdvRouting=(char *)sim_malloc(sizeof(char)*15);
  strcpy(lsa->Header.AdvRouting,g->RouterID);
  lsa->Header.Lsseq=0;
  lsa->Header.Lslong=20+4;
  lsa->Lsred.NetworkMask="255.255.255.255";
  lsa->Lsred.AtachedRouter=l_create();

  for ( i=0; LTT(g)[i].n_link != (Component *)NULL; i++ )  /* Por cada interface */
  {
    if ( *LTT(g)[i].l_status == 'U' &&
         ((Link *)(LTT(g)[i].n_link))->Conexion->u.i == TORED &&
         g->Conversa[i]->Estado >= WAY2 )
    {
      nid=(char *)sim_malloc(sizeof(char)*15);
      strcpy(nid,g->Conversa[i]->NeighborID);
      l_addt(lsa->Lsred.AtachedRouter,nid);
      lsa->Header.Lslong+=4;
    }
  }

/*
if ( lsa->Header.Lslong > 24 )
{
*/

  cla->Lstype=lsa->Header.Lstype;
  cla->LsID=(char *)sim_malloc(sizeof(char)*15);
  strcpy(cla->LsID,lsa->Header.LsID);
  cla->AdvRouting=(char *)sim_malloc(sizeof(char)*15);
  strcpy(cla->AdvRouting,lsa->Header.AdvRouting);

  if ( lsa1 = AvisoEnLista(g->ListaLSA,cla) )
  {
    l_del(g->ListaLSA,lsa1);
  }

  l_addt(g->ListaLSA,lsa);
  FluirAviso(g,lsa);

/*
}
*/

} /*del DR*/

Trace_ListaLSA(g);

lsa=NULL;
lsa1=NULL;
cla=NULL;
lr=NULL;
```

```
nid=NULL;

return (caddr_t)g;

}


/*************************************************************************/
ConversaDown(g,l)
register Ospft *g;
register Component *l;
{
int k,i;
char *ipred;

k = Link_LTT((Nodee *) g->node,l);

if ( k >= 0 )
{
  g->Conversa[k]->Estado=DOWN;
  g->Conversa[k]->RetransmiteLSA=l_create();
  g->Conversa[k]->BaseLSA=l_create();
  g->Conversa[k]->PedidosLS=l_create();
  g->Conversa[k]->MarcaDd=0;

  link_failure_init_LTT((Routet *)g, k);
}

printf("XXXX estado: %s \n",LTT(g)[k].l_status);

/* Down todos los links de la misma red */
if ( ((Link *)l)->Conexion->u.i == TORED )
{
  ipred=(char *)sim_malloc(sizeof(char)*15);
  strcpy(ipred,((Link *)l)->IpRed->u.p);

  for (i = 0; LTT(g)[i].n_nd != (Component *) NULL; i++)
  {
    if ( strcmp( ((Link *)LTT(g)[i].n_link)->IpRed->u.p, ipred) == 0 )
    {
      g->Conversa[i]->Estado=DOWN;
      g->Conversa[i]->RetransmiteLSA=l_create();
      g->Conversa[i]->BaseLSA=l_create();
      g->Conversa[i]->PedidosLS=l_create();
      g->Conversa[i]->MarcaDd=0;

      link_failure_init_LTT((Routet *)g, i);
    }
  }
}

ipred=NULL;
Trace_Conv(g);

}


/*************************************************************************/
```

```
ConversaUp(g,l)
register Ospft *g;
register Component *l;
{
int k,i;
char *ipred;

k = Link_LTT((Nodee *) g->node,l);

if ( k >= 0 )
{
 link_repair_init_LTT((Routet *)g, k);
}

/* Up todos los links de la misma red */
if ( ((Link *)l)->Conexion->u.i == TORED )
{
 ipred=(char *)sim_malloc(sizeof(char)*15);
 strcpy(ipred,((Link *)l)->IpRed->u.p);

 for (i = 0; LTT(g)[i].n_nd != (Component *) NULL; i++)
 {
  if ( strcmp( ((Link *)LTT(g)[i].n_link)->IpRed->u.p, ipred) == 0 )
  {
   link_repair_init_LTT((Routet *)g, k);
  }
 }
}

ipred=NULL;
Trace_Conv(g);

}


/*************************************************************************/
char *LinkAlNodo(g,nd)
register Ospft *g;
register Nodee *nd;
{
int i;

for (i = 0; LTT(g)[i].n_nd != (Component *) NULL; i++)
{
 if ( LTT(g)[i].n_nd == (Component *)nd )
  return ((Link *)LTT(g)[i].n_link)->link_name;
}
return (char *)NULL;

}


/*************************************************************************/
int Link_LTT(nd,l)
register Nodee *nd;
register Component *l;
{
int i;
```

```
 i=0;
 while ( nd->loc_topology_table[i].n_link != NULL &&
       nd->loc_topology_table[i].n_link != l )
  i++;

 if ( nd->loc_topology_table[i].n_link == NULL )
  return -1;
 else
  return i;

}


/*****************************************************************************/
EleccionDR(g)
register Ospft *g;
{
int i,p,pmax;

pmax = g->Prioridad->u.i;

for ( i=0; LTT(g)[i].n_link != (Component *)NULL; i++ )    /* Por cada interface */
{
 if ( *LTT(g)[i].l_status == 'U' &&
     ((Link *)(LTT(g)[i].n_link))->Conexion->u.i == TORED )
 {
  p=( (Ospft *)(((Nodee *)LTT(g)[i].n_nd)->nd_ptr_routing_mod))->Prioridad->u.i;

  if ( p > pmax )
  {
   g->DR=( (Ospft *)(((Nodee *)LTT(g)[i].n_nd)->nd_ptr_routing_mod));
   pmax = p;
  }
  else
  {
   if ( p == pmax &&
       strcmp( ( (Ospft *)(((Nodee *)LTT(g)[i].n_nd)->nd_ptr_routing_mod))->RouterID,g->DR-
>RouterID) > 0 )

    {
     g->DR=( (Ospft *)(((Nodee *)LTT(g)[i].n_nd)->nd_ptr_routing_mod));

    }
  }
 }
}
}


/*****************************************************************************/
Trace_Pkt(g,p)
register Ospft *g;
register Packet *p;

{
char *dest;
l_elt *ele,*ele2;
```

```
char *h;
LS_Header *hlsa;
ClaveLSA *cla;
LSA *lsa;
LinkRouter *lr;

if ( g->Trace->u.i == 0 )
        return;

 dest=(char *)sim_malloc(sizeof(char));
 ele=(l_elt *)sim_malloc(sizeof(l_elt));
 ele2=(l_elt *)sim_malloc(sizeof(l_elt));
 h=(char *)sim_malloc(sizeof(char));
 hlsa=(LS_Header *)sim_malloc(sizeof(LS_Header));
 cla=(ClaveLSA *)sim_malloc(sizeof(ClaveLSA));
 lsa=(LSA *)sim_malloc(sizeof(LSA));
 lr=(LinkRouter *)sim_malloc(sizeof(LinkRouter));

 hora(h);

 if ( p->pk_dest_socket.so_host != NULL )
  dest=p->pk_dest_socket.so_host->co_name;
 else
  dest=" ";

 switch ( p->Ospf_pk->Type )
 {
 case HELLO: fprintf(salida,"%s HELLO RID:%s Dest:%s Long:%d DR:%s Neigrs:",h,
             p->Ospf_pk->RouterID,dest,p->Ospf_pk->Long,p->Ospf_pk->Hello.DR->RouterID);
          for (ele=p->Ospf_pk->Hello.Vecinos->l_head; ele != NULL; ele=ele->le_next)
          {
            fprintf(salida,"%s ",(char *)ele->le_data);
          }
          fprintf(salida,"\n");

          break;
 case DD: fprintf(salida,"%s DD RID:%s Dest:%s Long:%d I:%d M:%d MS:%d Seq:%d \n",
              h, p->Ospf_pk->RouterID,dest,p->Ospf_pk->Long,
              p->Ospf_pk->Dd.Ibit,p->Ospf_pk->Dd.Mbit,
              p->Ospf_pk->Dd.MSbit,p->Ospf_pk->Dd.Seq);

           for ( ele=p->Ospf_pk->Dd.HLSAs->l_head; ele != NULL; ele=ele->le_next )
           {
            hlsa=(LS_Header *)sim_malloc(sizeof(LS_Header));
            hlsa = (LS_Header *) ele->le_data;
            fprintf(salida,"--> LStype:%d LSID:%s AdvRouting:%s LSAge:%d LSSeq:%d \n",
                    hlsa->Lstype,hlsa->LsID,hlsa->AdvRouting,hlsa->Lsage,hlsa->Lsseq);
           }
       break;
 case LSREQ: fprintf(salida,"%s LSREQ RID:%s Dest:%s Long:%d \n",
               h, p->Ospf_pk->RouterID,dest,p->Ospf_pk->Long);

             for ( ele=p->Ospf_pk->Lsreq.LReq->l_head; ele != NULL; ele=ele->le_next )
             {
              cla=(ClaveLSA *)sim_malloc(sizeof(ClaveLSA));
              cla = (ClaveLSA *) ele->le_data;
              fprintf(salida,"--> LStype:%d LSID:%s AdvRouting:%s \n",
                      cla->Lstype,cla->LsID,cla->AdvRouting);
```

```
            }
          break;
 case LSUPD: fprintf(salida,"%s LSUPD RID:%s Dest:%s Long:%d NroLsa:%d \n",
                h, p->Ospf_pk->RouterID,dest,
                p->Ospf_pk->Long,p->Ospf_pk->Lsupd.NroLSA);

              for ( ele=p->Ospf_pk->Lsupd.Avisos->l_head; ele != NULL; ele=ele->le_next )
              {
               lsa=(LSA *)sim_malloc(sizeof(LSA));
               lsa = (LSA *) ele->le_data;
               fprintf(salida,"--> LStype:%d LSID:%s AdvRouting:%s ",
                        lsa->Header.Lstype,lsa->Header.LsID,lsa->Header.AdvRouting);

               if ( lsa->Header.Lstype == 1 )
               {
                 fprintf(salida,"NroLinks:%d \n",lsa->Lsrou.Nrolinks);
                 for ( ele2=lsa->Lsrou.Links->l_head; ele2 != NULL; ele2=ele2->le_next )
                 {
                  lr=(LinkRouter *)sim_malloc(sizeof(LinkRouter));
                  lr = (LinkRouter *) ele2->le_data;
                  fprintf(salida,"------> LinkID:%s LinkData:%s TypeConexion:%d Tos0metric:%d
\n",
                              lr->LinkID, lr->LinkData, lr->Typeconexion, lr->Tos0metric);

                 }
               }
               else
               {
                if ( lsa->Header.Lstype == 2 )
                {
                 fprintf(salida,"NetworkMask:%s AtachedRouters:",
                          lsa->Lsred.NetworkMask);
                 for ( ele2=lsa->Lsred.AtachedRouter->l_head; ele2 != NULL; ele2=ele2->le_next
)
                 {
                  fprintf(salida,"%s ",ele2->le_data);
                 }
                 fprintf(salida,"\n");
                }
               }
              }
          break;
 case LSACK: fprintf(salida,"%s LSACK RID:%s Dest:%s Long:%d \n",
                h, p->Ospf_pk->RouterID,dest,p->Ospf_pk->Long);

              for ( ele=p->Ospf_pk->Lsack.Acks->l_head; ele != NULL; ele=ele->le_next )
              {
               hlsa=(LS_Header *)sim_malloc(sizeof(LS_Header));
               hlsa = (LS_Header *) ele->le_data;
               fprintf(salida,"--> LStype:%d LSID:%s AdvRouting:%s LSAge:%d LSSeq:%d \n",
                      hlsa->Lstype,hlsa->LsID,hlsa->AdvRouting,hlsa->Lsage,hlsa->Lsseq);
              }
          break;
 default: fprintf(salida,"X \n");
        break;
 }
 fprintf(salida," \n");
 fflush(salida);
```

```
  dest= NULL;
  ele = NULL;
  ele2= NULL;
  h   = NULL;
  hlsa= NULL;
  cla = NULL;
  lsa = NULL;
  lr  = NULL;

}


/*************************************************************************/
Trace_txt(g,p,c)
register Ospft *g;
register int p;
register char *c;
{

char *h;

if ( g->Trace->u.i == 0 )
        return;

h=(char *)sim_malloc(sizeof(char));

hora(h);

switch ( p )
{
case 1:
        fprintf(salida,"%s ------- CAIDA DE LA INTERFACE: %s ------- \n",h,c);
        break;
case 2:
        fprintf(salida,"%s ------- REPARACION DE LA INTERFACE: %s ------- \n",h,c);
        break;
case 3:
        fprintf(salida,"%s ------- GENERACION DE LSA ------- \n",h);
        break;
default:
        break;
}

fflush(salida);

h = NULL;

}


/*************************************************************************/
hora(h)
char *h;
{
int i;
char *fecha;
time_t fec,*ti;
```

```
  fec = time(ti);
  fecha = ctime(&fec);

  fecha+=11;

  for (i=0; i<8 ; i++)
    *h++ = *fecha++;

}


/*************************************************************************/
Trace_TR2(g)
register Ospft *g;
{
char *h;
int  i;

if ( g->TraceTR->u.i == 0 )
          return;

h=(char *)sim_malloc(sizeof(char));

hora(h);

fprintf(salidatr,"%s ------- TABLA DE RUTEO DE: %s ------- \n",h,g->ospf_name);
fprintf(salidatr,"  Destino              Link              Costo \n");

for ( i=0; i < g->no_of_nodes ; i++ )
{
  if ( RT(g)[i].hop != (Component *)NULL )
    fprintf(salidatr,"  %s                    %s                    %d \n",
         RT(g)[i].dest->co_name, RT(g)[i].hop->co_name, RT(g)[i].cost);
}
fprintf(salidatr,"------------------------------------------------------ \n");
fflush(salidatr);

h = NULL;

}


/*************************************************************************/
Trace_TR(g)
register Ospft *g;
{
char *h;
int  i;

if ( g->TraceTR->u.i == 0 )
          return;

h=(char *)sim_malloc(sizeof(char));

hora(h);

fprintf(salidatr,"%s ------- TABLA DE RUTEO DE: %s ------- \n",h,g->ospf_name);
```

```
fprintf(salidatr,"  Destino          Link              Costo \n");

for ( i=0; LTT(g)[i].n_nd != (Component *)NULL ; i++ )
{
  fprintf(salidatr,"  %s               %s                %d \n",
        LTT(g)[i].n_nd->co_name, LTT(g)[i].n_link->co_name, LTT(g)[i].l_cost);
}
fprintf(salidatr,"------------------------------------------------------------ \n");
fflush(salidatr);

h = NULL;

}


/*****************************************************************************/
Trace_Conv(g)
register Ospft *g;
{
/*
char *h;
int  i;

if ( g->Trace->u.i == 0 )
          return;

h=(char *)sim_malloc(sizeof(char));

hora(h);

fprintf(salida,"%s ------- CONVERSACION DE: %s ------- \n",h,g->ospf_name);
for ( i=0; g->Conversa[i]->Estado != NULO ; i++ )
{
  fprintf(salida,"  %s       %d \n",LTT(g)[i].n_link->co_name, g->Conversa[i]->Estado);
}
fprintf(salida,"------------------------------------------------------------ \n");
fflush(salida);

h = NULL;
*/
}


/*****************************************************************************/
Trace_ListaLSA(g)
register Ospft *g;
{
l_elt *ele,*ele2;
LinkRouter *lr;
LSA *lsa;

if ( g->Trace->u.i == 0 )
          return;

ele=(l_elt *)sim_malloc(sizeof(l_elt));
ele2=(l_elt *)sim_malloc(sizeof(l_elt));
lr=(LinkRouter *)sim_malloc(sizeof(LinkRouter));
```

```
fprintf(salida,"ListaLSA de %s ---------- \n",g->ospf_name);
for ( ele=g->ListaLSA->l_head; ele != NULL; ele=ele->le_next )
{
 lsa=(LSA *)sim_malloc(sizeof(LSA));
 lsa = (LSA *) ele->le_data;
 fprintf(salida,"--> LStype:%d LSID:%s AdvRouting:%s ",
 lsa->Header.Lstype,lsa->Header.LsID,lsa->Header.AdvRouting);

 if ( lsa->Header.Lstype == 1 )
 {
  fprintf(salida,"NroLinks:%d \n",lsa->Lsrou.Nrolinks);
  for ( ele2=lsa->Lsrou.Links->l_head; ele2 != NULL; ele2=ele2->le_next )
  {
   lr=(LinkRouter *)sim_malloc(sizeof(LinkRouter));
   lr = (LinkRouter *) ele2->le_data;
   fprintf(salida,"------> LinkID:%s LinkData:%s TypeConexion:%d Tos0metric:%d \n",
   lr->LinkID, lr->LinkData, lr->Typeconexion, lr->Tos0metric);

  }
 }
 else
 {
  if ( lsa->Header.Lstype == 2 )
  {
   fprintf(salida,"NetworkMask:%s AtachedRouters:",
        lsa->Lsred.NetworkMask);
   for ( ele2=lsa->Lsred.AtachedRouter->l_head; ele2 != NULL; ele2=ele2->le_next )
   {
    fprintf(salida,"%s ",ele2->le_data);
   }
   fprintf(salida,"\n");
  }
 }
}

fflush(salida);

}
```

A3.Archivo comptypes.h.

```
#ifndef COMPTYPES_H
#define COMPTYPES_H
/* File that defines the different types of components in the particular
   model being implemented.  Will vary with different implementations.  */

#include "sim.h"

caddr_t link_action(), node_action(), spf_action(), segal_action(),
      exBF_ack_action(),exBF_action(), ftp_source_action(),
      ftp_sink_action(), pm_action(),
      SmplTrfc_source_action(), SmplTrfc_sink_action(), stopper_action(),
      telnet_source_action(), telnet_sink_action(), lcostfcn_action(),
      ospf_action();
```

```
typedef struct {
  char          typename[40];
  short         class;          /* What class this component is */
  PFP           action;
  int           *parent_types; /* eric added */
} Component_type;

/* Classes of components */
#define OTHER_CLASS       0
#define LINK_CLASS   1
#define NODE_CLASS 2
#define ROUTE_CLASS       3
#define APTR_CLASS  4
#define AUXILIARY_CLASS    5

/* Types of components.  These numbers must be indexes into
   the component_types[] array.  They are also supposed to be
   constants.  (They are not #define's only so that the nubmers
   can be changed without recompiling everything that uses them.) */

/* CONST is defined in sim.h. */

#define LINK                    0
#define NODE                    1
#define SPF                     2
#define SEGAL                   3
#define EXBF                4
#define EXBF_ACK            5
#define FTP_SOURCE          6
#define FTP_SINK      7
#define TELNET_SOURCE     8
#define TELNET_SINK         9
#define SMPLTRFC_SOURCE 10
#define SMPLTRFC_SINK       11
#define PERF_MONITOR        12
#define STOPPER                     13
#define LCOSTFCN            14
#define OSPF                15

extern Component_type component_types[];
#endif  /* COMPTYPES_H  */
```

A4.Archivo comptypes.c.

```
#include "sim.h"
#include "comptypes.h"

/*****************************************************************************
  The following array is used by the simulator to map the strings
  contained in the world file to component types (integers) and to
  the action routines used by each component.  The strings *must* be
  uppercase, and the strings in any world file must match the strings
  here exactly (except for case).  The integer component type is the
  index into the array of the component type name.  */
```

```
/* The following are array assignments of the form:
 <comp_type>_p[]={<parent_type_list>,-1}

The assignment means: Component's of type <comp_type> have parents which are
            members of <parent_list>.
For example, when an FTP_SOURCE is connected to a NODE, the FTP_SOURCE would
become a child of the NODE because `NODE' is in the parent list of FTP_SOURCE.

If a LINK were connected to a NODE, then the LINK would become a sister of the
NODE, because 'NODE' is not in the parent list of LINK. */

static int              link_p[]={-1};
static int              node_p[]={-1};
static int               spf_p[]={NODE,-1};
static int              ospf_p[]={NODE,-1};
static int             segal_p[]={NODE,-1};
static int              exbf_p[]={NODE,-1};
static int         exbf_ack_p[]={NODE,-1};
static int        ftp_source_p[]={NODE,-1};
static int         ftp_sink_p[]={NODE,-1};
static int      telnet_source_p[]={NODE,-1};
static int       telnet_sink_p[]={NODE,-1};
static int      simple_source_p[]={NODE,-1};
static int       simple_sink_p[]={NODE,-1};
static int         perf_mon_p[]={-1};
static int            stopper_p[]={-1};
static int    link_cost_func_p[]={-1};

Component_type component_types[] = {
        {"LINK",        LINK_CLASS,  link_action,
          link_p},
        {"NODE",              NODE_CLASS, node_action,
          node_p},
        {"SPF",               ROUTE_CLASS,      spf_action,
          spf_p},
        {"SEGAL",             ROUTE_CLASS,      segal_action,
          segal_p},
        {"EXBF",              ROUTE_CLASS,      exBF_action,
          exbf_p},
        {"EXBF_ACK",          ROUTE_CLASS,      exBF_ack_action,
          exbf_ack_p},
        {"FTP_SOURCE",              APTR_CLASS, ftp_source_action,
          ftp_source_p},
        {"FTP_SINK",          APTR_CLASS, ftp_sink_action,
          ftp_sink_p},
        {"TELNET_SOURCE",  APTR_CLASS, telnet_source_action,
          telnet_source_p},
        {"TELNET_SINK",             APTR_CLASS, telnet_sink_action,
          telnet_sink_p},
        {"SIMPLE_SOURCE",  APTR_CLASS, SmplTrfc_source_action,
          simple_source_p},
        {"SIMPLE_SINK",             APTR_CLASS, SmplTrfc_sink_action,
          simple_sink_p},
        {"PERF_MON",          AUXILIARY_CLASS,pm_action,
          perf_mon_p},
        {"STOPPER",           AUXILIARY_CLASS,stopper_action,
          stopper_p},
        {"LINK_COST_FUNC", AUXILIARY_CLASS,lcostfcn_action,
```

```
        link_cost_func_p},
    {"OSPF",            ROUTE_CLASS,    ospf_action,
        ospf_p},
    {"", NULL,NULL,NULL} /* This line indicates the end of the list.  */
};
```

A5.Archivo packet.h.

```
#ifndef PACKET_H
#define PACKET_H

#include "sim.h"
#include "component.h"


/* Packet types seen by a node or a link */
#define TR_PACKET      0
#define ROUTE_PACKET   1

/* Packet sub-types for application/transport module */
#define TR_DATA        0
#define TR_ACK         1
#define TR_TOKEN       2

/* Packet sub-types for routing packets */
#define RT_LINK_SHUTDOWN       0
#define RT_LINK_WAKEUP         1
#define RT_NODE_SHUTDOWN       2
#define RT_NODE_WAKEUP         3
#define RT_LS                  4 /* spf */
#define RT_MSG                 5 /* merlin segal 79 */
#define RT_REQ                 6 /* merlin segal 79 */
#define RT_VECTOR              7 /* exBF 89 */
#define RT_VECTOR_ACK          8 /* exBF 89 */
#define HELLO                  9 /* ospf */
#define DD                    10 /* ospf */
#define LSREQ                 11 /* ospf */
#define LSUPD                 12 /* ospf */
#define LSACK                 13 /* ospf */

/* Packet size definitions */
#define ACK_PKT_SIZE          32
#define TOKEN_PKT_SIZE        32
#define REQ_PKT_SIZE          16
#define MSG_PKT_SIZE          20
#define LS_PKT_HEADER_SIZE    16
#define VECTOR_PKT_HEADER_SIZE 24
#define HELLO_PKT_SIZE        128

/* Definition of a packet.
   The first part of the packet structure includes all information
   needed by the simulator itself and layers below.
*/

typedef struct _Socket {
```

```
  Component *so_port, *so_host;
} Socket;

typedef struct _APTR_Packet {   /* transport protocol part of packet */
  int   tr_type;
  int   response;
  int   data_size;
} APTR_Packet;

typedef struct {
   Component    *destination;   /* Destination node */
   unsigned int best_cost;      /* Best cost from my node to destination */
   Component    *head;          /* Head of the current best path to */
                                /* destination */
} rt_vector_triplet;    /* Used by Extended BF */

/* Routing vector packet used by Extended BF */
typedef struct {
   Component          *sender;       /* Sender node */
   int                no_of_triplets; /* no of entries in routing vector */

   int                failure_flag;   /* indicates whether a topological
                                         change triggered the updates */
   Component          *node1;         /* event identity : nodes at both
                                         ends of the failed link */
   Component          *node2;

} RT_VECTOR_Packet;

typedef struct {
   Component    *sink;
   Component    *sender;
   int          cycle_no;
   unsigned int d;
} RT_MSG_Packet;

typedef struct {
   Component    *sink;
   Component    *sender;
   int          cycle_no;
} RT_REQ_Packet;

typedef struct {
   Component    *neighbour;
   unsigned int cost;
} CostPair;

typedef struct  {               /* LS routing packet */
   Component    *node;
   unsigned int seq_no;
   int          no_pairs;
} RT_LS_Packet;

typedef struct _RT_Packet {     /* routing module part of a packet */
  int           rt_type;
  union {
    RT_LS_Packet        ls;
    RT_MSG_Packet       msg;
```

```
    RT_REQ_Packet     req;
    RT_VECTOR_Packet  vector;
  } rt;
} RT_Packet;


/*************************************************************************/
/*NUEVOS PAQUETES PARA OSPF*/

typedef struct {
  int         Prioridad;  /*Prioridad del router*/
  struct _Ospft *DR;        /*DR de la red a la que corresponde el router*/
  struct _Ospft *BDR;       /*BDR de la red a la que corresponde el router*/
  list        *Vecinos;   /*lista de vecinos (RouterID)*/
} HELLO_Packet;

typedef struct {

  int         Ibit;      /* 1=primer paq. 0=c.c. */
  int         Mbit;      /* 1=mas paq. 0=ultimo paq. */
  int         MSbit;     /* 1=master, 0=slave */
  int         Seq;       /* nro. de secuencia del paquete DD*/
  list        *HLSAs;    /* lista de LS_Header */
} DD_Packet;

typedef struct {
  list        *LReq;     /* lista de ClaveLSA */
} LSREQ_Packet;

typedef struct {
  int         Lsage;     /* Edad del aviso */
  int         Lstype;    /* 1=avisos de router, 2=avisos de red */
                         /* 3=avisos sumariales de borde */
                         /* 4=avisos ASBR, 5=avisos externos de otro AS */
  char        *LsID;     /* si el Lstype es 1 y 4 vale el routerid, */
                         /* 3 y 5 vale la direccion de red y para 2 es */
                         /* la interface del DR */
  char        *AdvRouting; /*routerid del router que origina el aviso*/
  int         Lsseq;     /*Nro.Secuencia para detectar nuevos*/
  int         Lslong;
} LS_Header;

typedef struct {
  char        *LinkID;     /*Typeconexion: 1, routerid del vecino, 2 IP del DR, 3 IP de la red*/
  char        *LinkData;   /*Typeconexion: 3, networkmask, cc, interface IP del router*/
  int         Typeconexion; /*1=point to point, 2=red de transito, 3=red stub*/
  int         Tos0metric;  /*el costo en saltos de usar el router para Tos 0*/
} LinkRouter;

typedef struct {
  int         Ebit;       /* Ebit=1 El router es borde de un AS*/
  int         Bbit;       /* Bbit=1 El router es borde de area*/
  int         Nrolinks;   /* Nro de links de router descriptos en el aviso, es el total del area*/
  list        *Links;     /* Lista de LinkRouter */
} LSRouter;

typedef struct {
  char        *NetworkMask;   /*Mascara IP de la red , por ejemplo redes A =0xff000000*/
```

```
  list      *AtachedRouter;  /*RouterIDs de la misma red y que son full adyacentes al DR*/
} LSRed;

typedef struct {
  char      *NetworkMask; /*Mascara IP de la red*/
  int       Tos0metric;   /*el costo en saltos de usar el router para Tos0*/
} LSSum;

typedef struct {
  LS_Header    Header;
  LSRouter     Lsrou;
  LSRed                Lsred;
  LSSum                Lssum;
} LSA;

typedef struct {
  int       NroLSA;     /*nro de avisos en el paquete*/
  list      *Avisos;    /*lista de LSA*/
} LSUPD_Packet;

typedef struct {
  list      *Acks;      /* Lista de LS_Header */
} LSACK_Packet;


typedef struct {
  int       Long;
  int       Type;       /* 1=HELLO, 2=DD,3=LSREQ,4=LSUPD,5=LSACK */
  char      *RouterID;  /*Router id que origina el paquete*/
  int       AreaID;     /*Area a la que pertenece el paquete*/
  HELLO_Packet  Hello;      /*Distintos datos de un paquete OSPF*/
  DD_Packet    Dd;
  LSREQ_Packet  Lsreq;
  LSUPD_Packet  Lsupd;
  LSACK_Packet  Lsack;
} OSPF_Packet;

/*****************************************************************
  Main packet structure.
*/

typedef struct _Packet {
  /* Pointer for use by the queue these things are stored in. */
  struct _Packet *pk_next;

  /* Ultimate source & destination.  Like sockets in TCP. */
  Socket pk_source_socket, pk_dest_socket;

  /* The component that this is going to. */
  Component *pk_next_comp;

  /* Length in bytes.  Note that this length has nothing to do with the
     length of this struture.  It is the length of the imaginary packet that
     is being sent.  */
  int pk_length;

  unsigned int pk_time;       /* Usually departure time */
  unsigned int pk_arrival_time;
```

```
 unsigned int pk_sent_time;   /* Used to compute the average delay of */
                              /* a packet from the time it was sent   */
                              /* to the time it was acked             */

 int pk_color;                /* Color assigned to packet */

 unsigned pk_uid;

 unsigned pk_type;            /* type of packet like : data, routing, etc */

 char    *tail;               /* other information can be following */

 APTR_Packet tr_pk;           /* transport protocol part of packet */
 RT_Packet   rt_pk;           /* routing part of packet */
 OSPF_Packet *Ospf_pk;        /* Parte de un paquete OSPF */

} Packet;


void pk_free(), pk_free_all();
Packet *pk_alloc();
#endif  /* PACKET_H  */
```

A6.Archivo link.h.

```
#ifndef LINK_H
#define LINK_H
/*
 link.h

 */
#include "sim.h"
#include "event.h"
#include "q.h"
#include "list.h"
#include "component.h"

#define EV_LINK_RECEIVE    (EV_CLASS_PRIVATE | 1)

caddr_t link_action();

/* Link - Structure */
typedef struct _Link {
  struct _Link *link_next,                  /* Links to other components in the list */
          *link_prev;
  short       link_class;          /*  */
  short           link_type;               /*  */
  char            link_name[40];           /* Name of component (appears on screen) */
  PFP             link_action;             /* Main function of component.  */
  COMP_OBJECT       link_picture;              /* Graphics object displays this thing */
  list            *link_neighbors;         /* List of neighbors of this thing */

  /* Parameters-- data that will be displayed on screen */
```

```
 short          link_menu_up;        /* If true, then the text window is up */
 queue          *link_params;        /* Variable-length queue of parameters */

 Param          *link_propagation_delay;/* Length of the link */
 Param          *link_bandwidth;      /* Bandwidth of the link in bytes/sec. */
 Param          *link_failure_time;       /* Parameter for the failure model */
 Param          *dist_failure;       /* Parameter to indicate what distribution
                                 to use for times btw failures */
 Param            *sd_failure;              /* Standard deviation for uniform dist for failures */
 Param        *link_repair_time;            /* Parameter for the repair model */
 Param          *dist_repair;        /* Parameter to indicate what distribution
                                 to use for the repair times */
 Param          *sd_repair;               /* Standard deviation for uniform dist for repairs */
 Param        *link_status;        /* Status of the link */
 Param        *failure_status;            /* Indicates who caused the failure */

 Param        *idata_util_1;       /* Utilization of lq1 in terms of data/ack/token packets*/
 Param        *irout_util_1;       /* Utilization of lq1 in terms of routing packets*/

 Param        *idata_util_2;       /* Utilization of lq2 in terms of data/ack/token packets*/
 Param        *irout_util_2;       /* Utilization of lq2 in terms of routing packets*/

 int        data_bytes_1;        /* data/ack/token bytes sent on link 1 since last util update */
 int        rout_bytes_1;        /* routing bytes sent on link 1 since last util update */
 int        data_bytes_2;        /* data/ack/token bytes sent on link 2 since last util update */
 int        rout_bytes_2;        /* routing bytes sent on link 2 since last util update */

 int        bytes_in_dt;         /* max number of bytes transmitted in dt */

 queue        *link_queue_1;       /* Packet queue in one direction */
 queue        *link_queue_2;       /* Packet queue in the second direction */
 Event          *next_rcv_ev_1;                 /* Next recv event for 1st direction */
 Event          *next_rcv_ev_2;                 /* Next recv event for 2st direction */

 Param          *Conexion;          /* 1=PtoP 2=toRed 3=toStub */
 Param          *t_falla;           /* Tiempo de falla en seg */
 Param          *t_repara;          /* Tiempo de reparacion en seg */
 Param          *IpRed;             /* Nombre de la red */

} Link;
#endif
```

A7.Archivo node.c.

```
static caddr_t nd_failure(nd, src)
   register Nodee *nd;
   register Component *src;
{
  register Neighbor *n;
  register Packet *pkt;
  register unsigned int ticks;
  register int k;
  Packet *pkt1;
printf("XXXX nodo falla \n");
  if (play_flag &&  (Nodee *) src == nd)
```

```
  return (caddr_t) nd;

pval(nd, nd_status)->u.p = "Down";
log_param((Component *)nd, pval(nd, nd_status));

/* Schedule repair if node itself caused it */
if (src == (Component *)nd) {
  ticks = ev_now() +
    (unsigned)((double)(time_to_repair(nd)) * 1000000.0 / USECS_PER_TICK);
  ev_enqueue(EV_NODE_REPAIR, (Component *)nd, (Component *)nd,
                ticks, nd->nd_action, (Packet *)NULL, (caddr_t)NULL);
}

/***** Assume when I'm down all outgoing links are down  *****/
for (n = (Neighbor *)nd->nd_neighbors->l_head; n; n = n->n_next) {
   if (n->n_c->co_class == LINK_CLASS) {
     ev_call(EV_LINK_FAILURE, (Component *)nd, (Component *)n->n_c,
          ((Component *)n->n_c)->co_action, (Packet *)NULL, (caddr_t)NULL);
   }
}


pval(nd, nd_curr_buffer_space)->u.i = 0;


/* Assume a packet being transmitted is not lost when the node fails */
/* It will be dropped by the link anyway!                            */
for (n = (Neighbor *)nd->nd_neighbors->l_head; n; n = n->n_next) {
   if (n->n_c->co_class == LINK_CLASS) {  /* Empty outgoing queues */
     pkt1 = (Packet *)q_deq(n->n_pq);
     while (pkt = (Packet *)q_deq(n->n_pq))
          discard_pk(nd, pkt);
        k = index_in_local_top_table(nd, n->n_c);
        nd->loc_topology_table[k].avg_queue_size = 0;
        nd->loc_topology_table[k].last_utilization = 0.0;
        nd->loc_topology_table[k].queue_size = 0;
        nd->loc_topology_table[k].tot_queue_size = 0;
        nd->loc_topology_table[k].last_op_time = ev_now();
        nd->loc_topology_table[k].total_delay = 0.0;
        nd->loc_topology_table[k].total_trans_queuing_delay = 0.0;
        nd->loc_topology_table[k].total_tr_delay = 0.0;
        nd->loc_topology_table[k].ave_delay =nd->loc_topology_table[k].l_delay
         + nd->nd_delay->u.i;
        nd->loc_topology_table[k].no_of_packets = 0;
        nd->loc_topology_table[k].total_util = 0;
        nd->loc_topology_table[k].ave_util = 0.0;
        nd->loc_topology_table[k].l_cost   = lcostfcn_adr->minimum->u.i;
        nd->loc_topology_table[k].delay_cost = lcostfcn_adr->minimum->u.i;
        nd->loc_topology_table[k].util_cost = lcostfcn_adr->minimum->u.i;
        nd->loc_topology_table[k].hndly_cost = lcostfcn_adr->minimum->u.i;
     if (pkt1) {
       q_addh(n->n_pq, (caddr_t)pkt1);
       (pval(nd, nd_curr_buffer_space)->u.i) += pkt1->pk_length;
       k = index_in_local_top_table(nd, n->n_c);
       nd->loc_topology_table[k].queue_size = pkt1->pk_length;
     }
     log_param((Component *)nd, n->n_p);
   }
```

```
 }


/* Assume the routing packet currently being processed is not
   lost when the node fails */
 pkt1 = (Packet *)q_deq((queue *)pval(nd, nd_pq)->u.p);

 /* Empty the routing packet queue */
 while (pkt = (Packet *)(q_deq((queue *)pval(nd, nd_pq)->u.p)))
      discard_pk(nd, pkt);
 if (pkt1) {
   q_addh((queue *)pval(nd, nd_pq)->u.p, (caddr_t)pkt1);
   (pval(nd, nd_curr_buffer_space)->u.i) += pkt1->pk_length;
 }


 if ( nd->nd_ptr_routing_mod->route_type != OSPF )
 {
   pkt = pk_alloc();
   (pkt->rt_pk).rt_type = ROUTE_PACKET;
   pkt->pk_type = RT_NODE_SHUTDOWN;  /*  shutdown packet */
   pkt->pk_length = 20;            /*** ???? ***/
   q_addt((queue *)pval(nd, nd_pq)->u.p, (caddr_t)pkt);
   log_param((Component *)nd, pval(nd, nd_pq));
   occupy_space(nd, pkt);


   if (pkt1)
     pkt->pk_time = pkt1->pk_time;
   else {
     pkt->pk_time = ev_now();
     ev_call(EV_ROUTE_PROCESSING, (Component *)nd,
          (Component *)(nd->nd_ptr_routing_mod),
          ((Component *)(nd->nd_ptr_routing_mod))->co_action,
           pkt, (caddr_t)NULL);
   }
 }

 return((caddr_t)nd);
}


/*************************************/
static caddr_t nd_repair(nd, src)
   register Nodee *nd;
   register Component *src;
{
  register Packet *pkt;
  register unsigned int ticks;
  register Neighbor *n;
printf("XXXX nodo repara \n");
  if (play_flag && (Nodee *) src == nd)
     return (caddr_t) nd;

  pval(nd, nd_status)->u.p = "Up";
  log_param((Component *)nd, pval(nd, nd_status));

  /* Schedule next failure if node itself caused it */
```

```
  if (src == (Component *)nd) {
    ticks = (unsigned)((double)(time_to_fail(nd)) * 1000000.0 /
                                        USECS_PER_TICK);
    ev_enqueue(EV_NODE_FAILURE, (Component *)nd, (Component *)nd,
                    ticks+ev_now(), nd->nd_action, (Packet *)NULL, (caddr_t)NULL);
  }

  if ( nd->nd_ptr_routing_mod->route_type != OSPF )
  {
    pkt = pk_alloc();
    (pkt->rt_pk).rt_type = ROUTE_PACKET;
    pkt->pk_type = RT_NODE_WAKEUP  ; /*  startup packet */
    pkt->pk_length = 20;            /*****  ????  *****/
    pkt->pk_time = ev_now();
    q_addt((queue *)pval(nd, nd_pq)->u.p, (caddr_t)pkt);
    log_param((Component *)nd, pval(nd, nd_pq));
    occupy_space(nd, pkt);
    ev_call(EV_ROUTE_PROCESSING, (Component *)nd,
          (Component *)(nd->nd_ptr_routing_mod),
          ((Component *)(nd->nd_ptr_routing_mod))->co_action,
           pkt, (caddr_t)NULL);
  }

  /*** I'm up, inform all outgoing links ***/
  for (n = (Neighbor *)nd->nd_neighbors->l_head; n; n = n->n_next) {
     if (n->n_c->co_class == LINK_CLASS) {
        ev_call(EV_LINK_REPAIR, (Component *)nd, (Component *)n->n_c,
            ((Component *)n->n_c)->co_action, (Packet *)NULL, (caddr_t)NULL);
     }
  }

  return((caddr_t)nd);
}
```

A8.Archivo link.c.

```
static caddr_t link_start(l)
    register Link *l;
{
  register unsigned int ticks, time;

  /* Check if the link is properly connected */
  if (l->link_neighbors->l_len != 2){
#ifdef  DEBUG
       dbg_write ( debug_log, DBG_ERR, (Component *)l,
              "Link is not properly connected");
#endif
    return((caddr_t)NULL);

  }

  /* Calculate the max number of bytes in one time interval dt */

  l->bytes_in_dt = (int) (( (double)(l->link_bandwidth->u.i) * (double) perf_update_dt_usecs )
                          / 1000000.0 );
```

```
  /* Add the link to the list of components in the perf_module */
  pm((Component *)l, LINK, NEW_COMPONENT, 0, 0, 0, 0);

  /* Schedule first link failure */
  ticks = 0;

  if (l->link_failure_time->u.i > 0)
    ticks = MSECS_TO_TICKS(time_to_fail_l(l));
  else
    if ( l->t_falla->u.i > 0 )
          ticks = SECSD_TO_TICKS(l->t_falla->u.i);

  if ( ticks > 0 )
  {
   time = ev_now() + ticks;
   ev_enqueue (EV_LINK_FAILURE, (Component *)l, (Component *)l, time,
                    l->link_action, (Packet *)NULL, (caddr_t)NULL );
  }

  /* Something non-Null to return */
  return ((caddr_t)l);

}



/*************************************************************/
static caddr_t link_repair(l, src)
    register Link       *l;
    register Component *src;
{
  register Packet    *pkt_1, *pkt_2;
  register unsigned int time, ticks;
  register int queue_flag;
  Component *c,*c3;

  if (play_flag && l == (Link *)src)
    return (caddr_t) l;

  if (l == (Link *)src)
  {

    /* link repair caused by the link */
    l->failure_status->u.i = l->failure_status->u.i - 1;

    if ( l->t_falla->u.i == 0 )
    {
      ticks = MSECS_TO_TICKS(time_to_fail_l(l));
      time = ticks + ev_now();

      /* Schedule next link failure */
      ev_enqueue (EV_LINK_FAILURE, (Component *)l, (Component *)l,
              time, l->link_action, (Packet *)NULL, (caddr_t)NULL);
    }
  }
  else
    /* link repair caused by an adjacent node */
```

```
  l->failure_status->u.i = l->failure_status->u.i - 2;

log_param((Component *)l, l->failure_status);

/** If the link is totally up, schedule wakeup-packets **/
if (l->failure_status->u.i == 0){
  pm((Component *)l, LINK, LINK_WAKEUP, 0, 0, 0, 0);
  /* wakeup-packet for the first neighbor node*/


  /* VER SI ES OSPF */

  c=(Component *)sim_malloc(sizeof(Component));
  c3=(Component *)sim_malloc(sizeof(Component));

  c=((Component *)(((Neighbor *)(l->link_neighbors->l_head))->n_c));

  if ( ((Nodee *)c)->nd_ptr_routing_mod->route_type == OSPF )
  {
   ev_enqueue(EV_OSPF_UP_INTERF, (Component *)l,
          (Component *)((Nodee *)c)->nd_ptr_routing_mod, ev_now(),
          ((Component *)((Nodee *)c)->nd_ptr_routing_mod)->co_action,
          (Packet *)NULL, (caddr_t)NULL);

   c3=((Component *)(((Neighbor *)(l->link_neighbors->l_head->le_next))->n_c));
   ev_enqueue(EV_OSPF_UP_INTERF, (Component *)l,
          (Component *)((Nodee *)c3)->nd_ptr_routing_mod, ev_now(),
          ((Component *)((Nodee *)c3)->nd_ptr_routing_mod)->co_action,
          (Packet *)NULL, (caddr_t)NULL);

  }
  else
  {
   pkt_1 = pk_alloc();
   pkt_1->pk_type = ROUTE_PACKET;
   pkt_1->rt_pk.rt_type = RT_LINK_WAKEUP;
   pkt_1->pk_length = 50;
   pkt_1->pk_source_socket.so_host = (Component *) l;

   q_addt(l->link_queue_1, (caddr_t)pkt_1);

   ev_call (EV_LINK_RECEIVE, (Component *)l, (Component *)l,
        l->link_action, (Packet *)NULL, l->link_queue_1);

   /* wakeup-packet for the second neighbor node*/
   pkt_2 = pk_alloc();
   pkt_2->pk_type = ROUTE_PACKET;
   pkt_2->rt_pk.rt_type = RT_LINK_WAKEUP;
   pkt_2->pk_length = 50;
   pkt_2->pk_source_socket.so_host = (Component *) l;

   q_addt(l->link_queue_2, (caddr_t)pkt_2);
   ev_call (EV_LINK_RECEIVE, (Component *)l, (Component *)l,
        l->link_action, (Packet *)NULL, l->link_queue_2);
  }
  l->link_status->u.p = "Up";
  log_param((Component *)l, l->link_status);
 }
```

```
  return((caddr_t) l);
}

/*************************************************************/

static caddr_t link_failure(l, src)
    register Component *src;
    register Link      *l;
{
  register Packet *pkt_1, *pkt_2, *pkt;
  register Component            *c,*c2,*c3;
  register unsigned int   ticks, time;
  register int queue_flag;

  if (play_flag && l == (Link *) src)
    return (caddr_t) l;

/* link is up and failed now */
  if (*(char *)l->link_status->u.p == 'U'){
    l->link_status->u.p = "Down";
    pm((Component *)l, LINK, LINK_FAILURE, 0, 0, 0, 0);
    log_param((Component *)l, l->link_status);

    queue_flag = l->link_queue_1->q_len;
    /* Empty out link_queue_1 and retransmit APTR packets */
    while ((pkt = (Packet *)q_deq(l->link_queue_1)) != NULL ){
        if (pkt->pk_type == TR_PACKET){
          c = (Component *)(pkt->pk_source_socket.so_port);
          ev_call(EV_APTR_RETRANSMIT, (Component *)l, c,
                c->co_action, pkt, (caddr_t)NULL);
         }
          else pk_free(pkt);
        }
    if (queue_flag) {
      ev_dequeue(l->next_rcv_ev_1);
      l->next_rcv_ev_1 = (Event *) NULL;
    }


    c2=(Component *)sim_malloc(sizeof(Component));
    c3=(Component *)sim_malloc(sizeof(Component));

    c2=((Component *)(((Neighbor *)(l->link_neighbors->l_head))->n_c));

    /*VER SI ES OSPF*/

    if ( ((Nodee *)c2)->nd_ptr_routing_mod->route_type == OSPF )
    {
      ev_enqueue(EV_OSPF_DOWN_INTERF, (Component *)l,
              (Component *)((Nodee *)c2)->nd_ptr_routing_mod, ev_now(),
              ((Component *)((Nodee *)c2)->nd_ptr_routing_mod)->co_action,
              (Packet *)NULL, (caddr_t)NULL);

      c3=((Component *)(((Neighbor *)(l->link_neighbors->l_head->le_next))->n_c));
      ev_enqueue(EV_OSPF_DOWN_INTERF, (Component *)l,
              (Component *)((Nodee *)c3)->nd_ptr_routing_mod, ev_now(),
              ((Component *)((Nodee *)c3)->nd_ptr_routing_mod)->co_action,
              (Packet *)NULL, (caddr_t)NULL);
```

```
}
else
{
 /* Produce shutdown packet and call link_receive */
 pkt_1 = pk_alloc();
 pkt_1->pk_type = ROUTE_PACKET;
 pkt_1->rt_pk.rt_type = RT_LINK_SHUTDOWN;
 pkt_1->pk_length = 50;
 pkt_1->pk_source_socket.so_host = (Component *) l;

 q_addh(l->link_queue_1, (caddr_t)pkt_1);

 ev_call(EV_LINK_RECEIVE, (Component *)l, (Component *)l,
         l->link_action, (Packet *)NULL, l->link_queue_1);
}
queue_flag = l->link_queue_2->q_len;
/* Empty out link_queue_2 and retransmit APTR packets */
while ((pkt = (Packet *)q_deq(l->link_queue_2)) != NULL ){
    if (pkt->pk_type == TR_PACKET){
      c = (Component *)(pkt->pk_source_socket.so_port);
      ev_call(EV_APTR_RETRANSMIT, (Component *)l, c,
           c->co_action, pkt, (caddr_t)NULL);
     }
    else pk_free(pkt);
    }
if (queue_flag) {
  ev_dequeue(l->next_rcv_ev_2);
  l->next_rcv_ev_2 = (Event *) NULL;
}


if ( ((Nodee *)c2)->nd_ptr_routing_mod->route_type != OSPF )
{
 /* Produce shutdown packet and call link_receive */
 pkt_2 = pk_alloc();
 pkt_2->pk_type = ROUTE_PACKET;
 pkt_2->rt_pk.rt_type = RT_LINK_SHUTDOWN;
 pkt_2->pk_length = 50;
 pkt_2->pk_source_socket.so_host = (Component *) l;

 q_addh(l->link_queue_2, (caddr_t)pkt_2);
 ev_call(EV_LINK_RECEIVE, (Component *)l, (Component *)l,
         l->link_action, (Packet *)NULL, l->link_queue_2);
}

/** link fails because of link failure, schedule repair **/
if (l == (Link *)src)
{
  l->failure_status->u.i = l->failure_status->u.i +1;
  ticks = 0;

  if ( l->t_repara->u.i == 0 )
    ticks = MSECS_TO_TICKS(time_to_repair_l(l));
  else
       ticks = SECSD_TO_TICKS(l->t_repara->u.i);

  if ( ticks > 0 )
  {
```

```
        time = ev_now() + ticks;
        ev_enqueue(EV_LINK_REPAIR, (Component *)l, (Component *)l,
             time, l->link_action, (Packet *)NULL, (caddr_t)NULL);
      }
    }

    /** link fails because of node failure, no repair scheduled **/
    else
      l->failure_status->u.i = l->failure_status->u.i + 2;

    return((caddr_t)l);
}

/*
 if (l == (Link *)src){
    l->failure_status->u.i = l->failure_status->u.i +1;
    ticks = MSECS_TO_TICKS(time_to_repair_l(l));
    time = ev_now() + ticks;

    ev_enqueue(EV_LINK_REPAIR, (Component *)l, (Component *)l,
            time, l->link_action, (Packet *)NULL, (caddr_t)NULL);
    }
  else
    l->failure_status->u.i = l->failure_status->u.i + 2;
*/

 /** log the new failure status **/
 log_param((Component *)l, l->failure_status);
  return((caddr_t) l);

}
```

A9.Archivo sim.h.

```
#ifndef SIM_H
#define SIM_H
 /* $Id: sim.h,v 10.1 92/10/06 23:08:59 ca Exp $ */

#include <sys/types.h>
#include <strings.h>
#include <stdio.h>
#include <math.h>

/* Things for all simulator routines */

#define TRUE 1
#define FALSE 0
#ifndef NULL
#  define NULL 0
#endif  /* not NULL */
#define OFF 0
#define ON 1
#define ERROR -1
#define OK 1
```

```
#define DEFAULT_PERF_UPDATE_DT_USECS        25000

#ifdef OFFLINE
#define DEFAULT_SKIP_TIME_IN_TICKS      10000000
#else
#define DEFAULT_SKIP_TIME_IN_TICKS      0
#endif

extern unsigned long int perf_update_dt_usecs;
extern unsigned long int skip_time_in_ticks;

typedef unsigned int tick_t;
extern int  record_flag;
extern int  play_flag;
extern int  profile_flag;
extern char playfilename[];
extern FILE *record_file;
extern FILE *profilefile;

/* Define time elapsed per simulator clock tick either in USECS_PER_TICK
   or NSECS_PER_TICK.  Either USECS_PER_TICK or NSECS_PER_TICK should be
   defined, but not both.
   NSECS_PER_TICK must divide 1000 or else you will get some strange
   roundoff errors. */

/*#define USECS_PER_TICK              10 */
#define USECS_PER_TICK              1000

#ifdef NSECS_PER_TICK
# define  NSECS_TO_TICKS(time) ((tick_t)  ((time) / NSECS_PER_TICK) )
# define  TICKS_TO_NSECS(ticks) ((ticks) * NSECS_PER_TICK)
# define  USECS_TO_TICKS(time) ((tick_t) ((time) * (1000 / NSECS_PER_TICK)) )
# define  TICKS_TO_USECS(ticks) ((ticks) / (1000 / NSECS_PER_TICK))
# define  SECSD_TO_TICKS(time) ((tick_t) (((time) * 1000000000) / NSECS_PER_TICK))
# define  TICKS_TO_SECSD(ticks) ((((double) (ticks)) * NSECS_PER_TICK) / 1000000000)
#else
# ifdef USECS_PER_TICK
/*
#  define NSECS_TO_TICKS(time) ((tick_t) ((time) / (USECS_PER_TICK * 1000)))
#  define TICKS_TO_NSECS(ticks)  ((ticks) * (USECS_PER_TICK * 1000))
#  define USECS_TO_TICKS(time) ((tick_t) ((time) / USECS_PER_TICK))
#  define TICKS_TO_USECS(ticks)  ((ticks) * USECS_PER_TICK)
#  define MSECS_TO_TICKS(time) ((tick_t) (((time) * 1000) / USECS_PER_TICK))
#  define TICKS_TO_MSECS(ticks) ((((double) (ticks)) * USECS_PER_TICK) / 1000)
#  define SECSD_TO_TICKS(time) ((tick_t) (((time) * 1000000) / USECS_PER_TICK))
#  define TICKS_TO_SECSD(ticks) ((((double) (ticks)) * USECS_PER_TICK) / 1000000)
*/
#  define NSECS_TO_TICKS(time) ((tick_t) ((time) / (USECS_PER_TICK * 22000)))
#  define TICKS_TO_NSECS(ticks)  ((ticks) * (USECS_PER_TICK * 22000))
#  define USECS_TO_TICKS(time) ((tick_t) (((time) * 22) / USECS_PER_TICK))
#  define TICKS_TO_USECS(ticks)  ((((double) (ticks)) * USECS_PER_TICK) / 22)
#  define MSECS_TO_TICKS(time) ((tick_t) (((time) * 22000) / USECS_PER_TICK))
#  define TICKS_TO_MSECS(ticks) ((((double) (ticks)) * USECS_PER_TICK) / 22000)
#  define SECSD_TO_TICKS(time) ((tick_t) (((time) * 22000000) / USECS_PER_TICK))
#  define TICKS_TO_SECSD(ticks) ((((double) (ticks)) * USECS_PER_TICK) / 22000000)
# endif
#endif
```

```
#define UNIX_4_2   1
#define UNIX_4_3   2
#define SLOW_START  3


#define DBG_INFO      1
#define DBG_ERR               2

#ifdef __STDC__
extern double   atof( const char *__nptr );
#define CONST const
#else  /* __STDC__ */
double atof();
#define CONST
#endif  /* __STDC__ */

extern char *strncpy(char *, const char *, size_t), *strcpy(char *, const char *), *strcat(char *,
const char *);
/*redefinicion la ignora
*strncat(char *, const char *, size_t);
*/
/*No se pueden redefinir en AIX...
extern char *calloc(), *malloc(), *realloc(); */
char *sim_calloc(), *sim_malloc(), *sim_realloc();
void panic();

/** Largest number returned by random(). (2**31)-1   From RANDOM(3) */
#define MAX_RANDOM_NUMBER 2147483647
extern long random();
double random_range();

#define min(a, b)  ((a) < (b) ? (a) : (b))
#define max(a, b)  ((a) > (b) ? (a) : (b))
/* Some source uses upper, some lower. */
/*#define MAX(a, b)  ((a) > (b) ? (a) : (b))
#define MIN(a, b)  ((a) < (b) ? (a) : (b))
*/
/* Oft-used item:  Pointer to a Function that returns an Integer */
typedef int (*PFI)();
/* Pointer to Function that returns a Double.  */
typedef double (*PFD)();
/* Pointer to Function that returns a Pointer */
typedef caddr_t (*PFP)();

#endif   /* SIM_H   */
```

Bibliografía.

Referencias.

[SCHA,1996] Effective Networks Simulation. Steve Schaffer.
http://www.optimal.com/products/op/sim-wp.html

[TANE,1996] Redes de computadoras 2da. y 3ra. edición. Andrew Tanenbaum.

[CANC,1995] Publication interne N.915 de IRISA. Recursive path cinditioning Monte Carlo simulation of comunication network reliability. Héctor Cancela, Mohamed El Khadiri.

[VAZQ,1997] Programa de Doctorado en Simulación de Redes de Comunicaciones. Departamento de Ingeniería de Sistemas Telemáticos ETSI de Telecomunicaciones de la Universidad Politécnica de Madrid de Enrique Vázquez Gallo.
http://www.dit.upm.es/~enrique/simredes.html

[MaRSU,1991] Manual de usuario del MaRS. Versión 1.0 Alaettinoglu, Dussa-Zieger, Matta, Shankar, Junio de 1991.

[MaRSP,1991] Manual de programador del MaRS. Versión 1.0 Alaettinoglu, Dussa-Zieger, Matta, Shankar, Junio de 1991.

[MaRS,1991] Código fuente del simulador MaRS. Versión 2.1 Alaettinoglu, Dussa-Zieger, Matta, Shankar.

[MOY,1996-1997] RFCs 1131 y 2328. OSPF Versión 2. J. Moy.

[FELP,1991] 20 JAIIO Resultados de simulaciones sobre comportamiento de ruteo completamente adaptativo, minimal y libre de deadlock en grillas e hipercubos. Sergio Felperin y Jorge Sanz.

Bibliografía consultada:

Para conceptos generales de simulación:

Simulación de redes Lic. Luis Alvarez. LAN & WAN Enero de 1996.

Para el análisis de los simuladores:

Real Users Manual.
Real Programmers Manual.
http:// minnie.cs.adfa.oz.au/REAL/index.html

Network Simulator de Lawrence Bekerley National Laboratory.
http:// www-nrg.ee.lbl.gov/ns/

ATLAS Reference Manual 1.0 (12/91): (Analisys Tool for Local Area Network Simulation) por Markus Rümekasten, Universidad de Paderborn, Departamento de Matemáticas y Ciencias de la Computación.
http://www.dit.upm.es/~enrique/simredes.html

Ibrahim Matta, Desarrollador del MaRS y Cristian Aldama actualmente a cargo del simulador Mars. E-mail: mars@cs.umd.edu o matta@ccs.neu.edu y cristian@orion.ls.fi.upm.es respectivamente.

Para evaluar distintos protocolos de ruteo:

Cisco Routing Protocols.
http:// www.cisco.com/warp/public/732/Tech/rtrp-pc.html

Cisco The IP Routing Protocols.
http:// www.cisco.com/univercd/cc/td/doc/product/software/ssr91/csc_r/56294.htm
Cisco. OSPF: Frequently Asked Questions.
http:// www.cisco.com/ warp/public/458/10.html

Cisco. An Introduction to IGRP.
http:// www.cisco.com/ warp/public/459/2.html

Cisco. Introduction to EIGRP.
http:// www.cisco.com/ warp/public/103/1.html

Para analizar el OSPF:

RFC 1245. OSPF Protocol Analysis. J. Moy.
RFC 1246. Experience with the OSPF protocol. J. Moy.
RFC 1253. OSPF Versión 2 Management Information Base. F. Baker y R. Coltun.