MODELADO DE DEPENDENCIA DE UN SISTEMA DISTRIBUIDO INTERCONECTADO USANDO ALGORITMOS DE RUTEO

Tesis presentada

Por

I.S.C. ROBERTO SOLIS ROBLES

Presentada ante la Dirección Académica de la Universidad Virtual del

Instituto Tecnológico y de Estudios Superiores de Monterrey

como requisito parcial para optar

al título de

MAESTRO EN CIENCIAS DE LA COMPUTACION

Mayo de 1999

Maestría en Ciencias Computacionales

MODELADO DE DEPENDENCIA DE UN SISTEMA DISTRIBUIDO

INTERCONECTADO USANDO ALGORITMOS DE RUTEO

Tesis presentada

Por

I.S.C. ROBERTO SOLIS ROBLES

Aprobada en contenido y estilo por:

_____

Dr. Roberto Abraham Valdivia Beutelspacher, asesor.

_____

Dr. Roberto Gómez Cárdenas, Presidente del jurado.

_____

Dr. Jesús Antonio Sánchez Velázquez, Secretario del jurado.

_____

Dr. Martín López Morales.
Director del Programa de Maestría en
Ciencias de la Computación

RESUMEN

MODELADO DE DEPENDENCIA DE UN SISTEMA DISTRIBUIDO INTERCONECTADO USANDO ALGORITMOS DE RUTEO

MAYO DE 1999

ROBERTO SOLIS ROBLES

INGENIERO EN SISTEMAS COMPUTACIONALES

INSTITUTO TECNOLOGICO DE ZACATECAS

POSGRADO EN COMPUTACION

UNIVERSIDAD AUTONOMA DE ZACATECAS

Dirigida por el Dr. Roberto Abraham Valdivia Beutelspacher

La evaluación de la dependencia de los sistemas es un tema de suma importancia en la actualidad puesto que existe una gran cantidad de sistemas de cómputo cuyo funcionamiento es básico para el logro de misiones críticas, tal dependencia tiene como uno de sus factores mas importantes a la confiabilidad, la cual es la probabilidad de que el sistema funcione de manera correcta; para su cálculo existen ya diferentes técnicas de diverso tipo, pero su ejecución requiere de mucho tiempo de cómputo. Por tanto se propone un modelo de confiabilidad de red basado en un trabajo anterior que proporciona una aproximación del 97% al valor real de la confiabilidad de manera eficiente y que es

mejorado haciendo uso de algoritmos de ruteo de tal forma que permite reducir el tiempo de cálculo de tal aproximación a la confiabilidad real hasta en un 50%, por ejemplo en un grafo K^6 el algoritmo anterior calcula una aproximación en 110 ms mientras que el propuesto en este trabajo calcula una aproximación similar en 60 ms. Este tiempo es reducido gracias a que el algoritmo de ruteo utiliza el propio algoritmo y no usa todas las rutas existentes. En nuestro algoritmo se utiliza como métrica el valor de probabilidad de cada enlace lo cual da como resultado las rutas con mayor confiabilidad.

INDICE DE CONTENIDO

INDICE DE FIGURAS

CAPITULO 1

INTRODUCCION

1.1. El papel de la dependencia en la computación y sus factores determinantes

El avance que ha tenido la tecnología computacional ha sido tan grande que se ha ido introducido en todos los ámbitos de la sociedad, de tal forma que encontramos microprocesadores en sistemas tan complejos y críticos como lo son los de control aéreo hasta en sistemas de control de aparatos eléctricos domésticos comunes como lo es un horno de microondas. Tales sistemas, principalmente los de misión crítica, requieren de altos niveles de dependencia, es decir, que se pueda confiar en que ellos proporcionaran un servicio congruente y de acuerdo a las necesidades por las cuales fueron desarrollados. Tal dependencia se cuantifica en varios factores entre los que encontramos la tolerancia a fallas, la seguridad, la disponibilidad y la confiabilidad.

La confiabilidad se entiende como la probabilidad de que el sistema funcione correctamente y continúe haciendo su trabajo, por lo cual es de gran valor el poder conocer de antemano que tan confiable es, pues de acuerdo al sistema de que se trate dependerá la seguridad de diversos aspectos tales como la vida humana o los equipos. Debido a que en los sistemas de cómputo se tiene un conjunto de componentes unidos a través de alguna red de interconexión, el cálculo de su confiabilidad se hace más complejo entre más componentes existan. Debido a esto surge el campo de la confiabilidad de redes el cual abarca una gran variedad de temas relacionados al diseño y análisis de redes sujetas a fallas en sus componentes y proporciona enfoques teóricos para estimar parámetros tales como el tiempo medio entre fallas o la disponibilidad de un componente y de la red en su totalidad.

La confiabilidad es también un tema importante en los sistemas distribuidos ya que estos se construyen sobre una red de interconexión que va a permitir la comunicación entre las distintas partes del mismo; entonces, para determinar si un sistema distribuido es confiable, se requiere primero determinar si la red de interconexión subyacente es confiable.

Lo que se busca entonces es la generación de métodos que permitan determinar que tan probable es que alguno de los componentes de la red de interconexión fallen haciendo que el sistema tenga un mal funcionamiento y deje de prestar el servicio para el cual fue desarrollado con las consecuencias que esto pudiera traer; lo cual, como ya se mencionó es un factor importante para saber que tanto se puede depender del sistema. Para poder llegar a esta dependencia se necesita no solamente saber de antemano el valor de confiabilidad de los componentes (que tan posible es que sigan funcionando) sino también saber que técnicas entran en acción si es que llegaran a fallar algún o algunos componentes; tales técnicas son conocidas de manera genérica como tolerancia a fallas. A un sistema con esta capacidad se le conoce como sistema tolerante a fallas.

1.2. Propósito de esta tesis

El propósito de esta tesis es desarrollar un modelo que permita determinar la dependencia de un sistema distribuido a través de la obtención de sus parámetros de confiabilidad de manera eficiente enfocándose en la red de interconexión subyacente. Para ello se tomará como base un trabajo anterior en la materia al cual se mejorará mediante la utilización de algoritmos de ruteo, los cuales sirven para que la red de interconexión

determine el camino a seguir para un mensaje específico, este varía dependiendo de lo que tales algoritmos usan como guía (distancia, retardo, etc.).

1.3. Contenido de la tesis

El capítulo 2 proporciona la terminología básica relacionada con la tolerancia a fallas, explica cuales son las causas de su existencia, los tipos de fallas que se pueden dar en un sistema, los métodos utilizados para su manejo entre los cuales están la prevención, detección y eliminación; también aclara el papel que juega la redundancia en la tolerancia a las fallas y los tipos de redundancia que existen.

El capítulo 3 describe las medidas de dependencia que se utilizan en las redes explicando de manera detallada los principales parámetros que determinan la confiabilidad de las mismas, los tipos de modelos existentes para el cálculo de la confiabilidad así como las desventajas que tiene cada uno de ellos.

El capítulo 4 describe las funciones del ruteo, las formas de determinación de las rutas a seguir, tipos de algoritmos existentes, así como las diversas formas que existen de ruteo dependiendo de la tecnología de comunicación subyacente, la cual puede ser de conmutación de circuitos, móvil, de alta velocidad o de conmutación de paquetes, a esta última se le da un mayor énfasis pues la mayor parte de los sistemas distribuidos existentes se encuentra construido sobre redes de este tipo (Internet como principal ejemplo).

El capítulo 5 desarrolla el modelo de confiabilidad de la red basado en la teoría de grafos puesto que es la representación natural de una red, analizando su parte estructural (modelo determinista) así como lo relacionado a las probabilidades de funcionamiento de cada uno de los componentes (modelo probabilístico), usando para ello el algoritmo de ruteo seleccionado en el capítulo 4.

El capítulo 6 describe la implantación realizada del modelo en el lenguaje Java, explicando cada una de las partes que conforman la interfaz de usuario y listando las clases utilizadas para el desarrollo del programa.

El capítulo 7 contiene los resultados obtenidos al aplicar el modelo de confiabilidad del capítulo 5, una discusión de los mismos y las conclusiones a las cuales se llegó.

Por último, el Anexo proporciona los conceptos básicos relacionados con la teoría de grafos los cuales son utilizados para la explicación del modelo de confiabilidad.

CAPITULO 2

TOLERANCIA A FALLAS

2.1. Introducción

Desde la invención de los microprocesadores a principios de los 70's el costo de la tecnología basada en computadoras ha caído de manera estable [Storey 96], lo cual ha llevado a un incremento dramático en el uso de la misma. Ahora bien, los microprocesadores no son usados sólo en computadoras de escritorio sino en aparatos domésticos y en sistemas tales como el control de vuelos, apagado de plantas nucleares o frenos antibloqueo, estos sistemas reciben el nombre de inmersos[1]. En tales sistemas un fallo puede resultar en un daño directo y posiblemente muy serio a una o más personas. En el caso de la planta nuclear es claro: la falla del sistema de control puede poner en peligro las vidas de miles de personas si no es que la de millones, y puede tener implicaciones sobre el ambiente global. En una escala más pequeña, la falla de sistemas de control aéreo puede causar potencialmente un accidente aéreo que puede matar a cientos de personas. De manera similar, la falla de un sistema de frenado puede poner en riesgo las vidas de varias personas. También los aparatos domésticos basados en tecnología computacional pueden causar daño aunque de manera menos clara, por ejemplo, un horno de microondas que no se apaga al abrirse la puerta puede ocasionar una explosión. Debido a ello se requieren de técnicas que permitan evitar la ocurrencia de estas fallas así como enfrentarlas en caso de que surjan.

Existen tres términos importantes y que se utilizan para expresar la ocurrencia de eventos no deseados en el ambiente de operación de un sistema [Laprie 92, Hiltunen 96]:

---

[1] Embedded systems

- Una falla es un defecto dentro del sistema.
- Un error es una desviación de la operación requerida del sistema o subsistema.
- Un mal funcionamiento del sistema ocurre cuando el sistema no realiza su función requerida.

Los mal funcionamientos del sistema pueden tomar varias formas. Por ejemplo, el mal funcionamiento de un componente de hardware representa una falla como también lo hace una pieza de software mal codificada[2]. Una equivocación en el diseño del sistema también representa una falla. La presencia de una falla puede llevar a un error, que es el mecanismo por el cual la falla se hace aparente. Por ejemplo, si un dispositivo de memoria dentro de una computadora tuviera una falla el mal funcionamiento no se detectaría mientras no se accediera tal dispositivo. Si la presencia de un error causa que el sistema en su totalidad se desvíe de su operación requerida esto se considera un mal funcionamiento del sistema.

La relación entre estos 3 términos se muestra gráficamente en la figura 2.1:

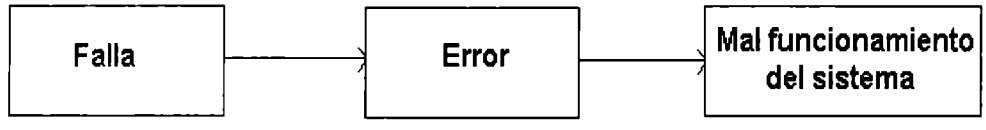

Figura 2.1 Relación entre falla, error y mal funcionamiento del sistema

2.2. Tipos de fallas

Las fallas se pueden caracterizar de diversas formas. Entre las más importantes se encuentra la distinción debido a su naturaleza, duración o alcance.

De acuerdo a su naturaleza las fallas se pueden dividir en:

---

[2] Bug

- Fallas aleatorias. Estas están asociadas con el mal funcionamiento de componentes de hardware. Todos los componentes físicos están sujetos a mal funcionamientos y por lo tanto todos los sistemas están sujetos a fallas aleatorias. Cuando se trabaja dentro de su ambiente operativo los componentes individuales fallan de manera aleatoria. Sin embargo, a este tipo de fallas se le puede hacer un análisis estadístico con el propósito de predecir la probabilidad de falla.
- Fallas sistemáticas. Estas tienen muchas formas tales como fallas en el diseño y equivocaciones dentro de la especificación del sistema. Todas las fallas de software caen dentro de esta categoría pues suceden en el diseño (especificación, codificación, errores lógicos al hacer cálculos, sobreflujos de pila, etc.). Usualmente no se les puede realizar un análisis estadístico.

De acuerdo a su duración:

- Fallas permanentes. Estas son las que existen de manera indefinida o hasta que se toma alguna acción correctiva. Las fallas de diseño, incluyendo las de software son siempre permanentes.
- Fallas transitorias. Son las que aparecen en un momento determinado y desaparecen después de un tiempo. Ejemplos de estas fallas son los efectos causados por una partícula alfa que llega a los chips de memoria. Estos eventos ocurren raramente pero pueden cambiar el estado de uno o más bits dentro del plano de la memoria sin causar un daño duradero al dispositivo. Aunque la falla es transitoria, el error que produce permanece después de que la falla ha desaparecido.

- Fallas intermitentes. Estas aparecen, desaparecen y reaparecen tiempo después. Tales fallas pueden resultar de corrosiones en los contactos de los conectores de tal forma que en ocasiones se hacen conexiones correctas mientras que en otras no. También pueden resultar de interferencias eléctricas. Por su naturaleza, estas fallas son muy difíciles de detectar y remover ya que el procedimiento de detección de fallas debe coincidir con la existencia de la falla.

De acuerdo a su alcance:

- Fallas locales. Solo afectan a un módulo de hardware o de software.
- Fallas globales. Tienen efectos sobre todo el sistema.

2.3. Manejo de fallas

Un sistema libre de fallas operaría de manera perfecta, ya que no tendría fallas ni de diseño ni de componentes. No habría ni errores ni mal funcionamiento del sistema. Desafortunadamente, la eliminación completa de fallas es imposible, primero debido a que todos los componentes están sujetos a fallas aleatorias debido al desgaste, edad u otros efectos y segundo, porque no es posible lograr la perfección en el diseño. Por lo tanto, las fallas son inevitables. Sin embargo, si el sistema fue apropiadamente diseñado y se toman ciertas medidas, puede ser posible que continúe ejecutando sus funciones aun en la presencia de uno o más errores. Estas medidas se dividen en cuatro grupos de técnicas:

- Prevención de fallas. Buscan evitar que se introduzcan fallas en el sistema durante la etapa de diseño. Es la meta principal del proceso de diseño. Para ello se utilizan lenguajes de descripción de diseño [Shahdad 86, Morison 82] y métodos formales tales como los descritos en [Bowen 92, Bowen 93, Fencott 92, Rushby 93].

- Eliminación de fallas. Estas técnicas intentan encontrar fallas dentro de un sistema antes de que entre en servicio. Estas incluyen métodos de prueba de hardware y software [BCS 95, Cullyer 94].
- Detección de fallas. Estas técnicas son usadas durante el servicio para detectar fallas dentro del sistema operacional de tal forma que sus efectos sean minimizados.
- Tolerancia a fallas. Estas técnicas se diseñan para que el sistema opere correctamente en presencia de fallas.

Desafortunadamente, ninguna de estas técnicas es completamente efectiva y en aplicaciones altamente críticas se usa una combinación de las mismas para que las fallas del sistema se mantengan a un nivel aceptable.

2.4. Cobertura de fallas

El éxito de estas técnicas se puede medir por la cobertura de fallas que logra, siendo esta la fracción de posibles fallas que puede ser evitada, eliminada, detectada o tolerada.

En la práctica es muy difícil lograr cualquier estimado numérico del éxito de la prevención de fallas, pero se pueden usar modelos de fallas (descritos en [Johnson 89, Storey 96]) para comparar y diseñar alternativas.

La cobertura de eliminación de fallas es una medida del éxito al encontrar fallas durante la fase de prueba del desarrollo del sistema. Debido a las limitantes de los modelos de fallas para encontrar fallas intermitentes o transitorias, este tipo de cobertura no es completo aumentando la necesidad de otras técnicas para mitigar sus efectos.

La cobertura de detección de fallas es la medida del éxito que se tiene para detectar fallas durante la operación del sistema, la cual puede ser estimada (de manera limitada) durante la etapa de diseño usando modelos de fallas. La habilidad del sistema para tolerar

fallas se describe por su cobertura de tolerancia a fallas.

El uso de pruebas para medir los valores reales logrados por las diversas formas de cobertura de fallas es impráctica para los sistemas reales. Los estimados proporcionados por los modelos de fallas dan una idea de los efectos que las fallas tienen sobre el sistema. El beneficio que se obtiene al analizar la cobertura de fallas es que permite que se comparen diseños alternativos y se investiguen las implicaciones de los cambios. Cabe mencionar que la cobertura de fallas afecta los estimados de la confiabilidad del sistema.

2.5. Redundancia

Todas las formas de tolerancia a fallas son logradas por alguna forma de redundancia, esto es, el uso de elementos adicionales dentro del sistema los cuales no se requerirían en un sistema libre de fallas.

La mayoría de los sistemas tolerantes a fallas usan módulos de hardware replicados de tal forma que el mal funcionamiento de uno no resultara en el mal funcionamiento del sistema. Un ejemplo de tal sistema es el de simple redundancia modular triple (TMR) mostrado en la figura 2.2:

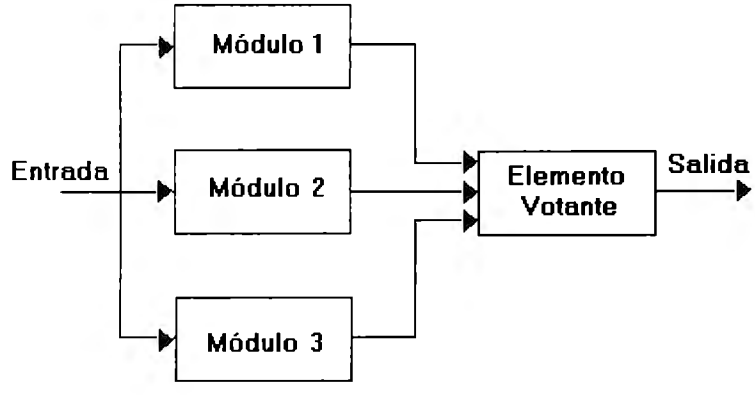

Figura 2.2 Redundancia Modular Triple

El sistema TMR usa 3 módulos idénticos donde cada uno recibe las mismas señales de entrada. Si todos los módulos están funcionando correctamente producirán salidas

idénticas y cualquier diferencia entre sus salidas indicará el mal funcionamiento de un módulo. Una forma de voto es usada para eliminar los efectos de un solo mal funcionamiento al tomar la visión de la mayoría en caso de un desacuerdo y produciendo esta visión como salida. El sistema puede por lo tanto tolerar la falla de un módulo sin afectar la salida del sistema.

El TMR es solo una de las diversas formas de redundancia y es conveniente distinguir entre todos los tipos básicos que son:

- Redundancia de hardware. Es el uso de hardware adicional al requerido para implementar el sistema en ausencia de fallas con la meta de detectar o tolerar fallas. El TMR es un ejemplo de este tipo de redundancia.
- Redundancia de software. Es el uso de software adicional al requerido para implementar el sistema en ausencia de fallas.
- Redundancia de información. Es el uso de información adicional a la requerida para implementar una función dada. Ejemplos de esta incluyen el uso de bits de paridad y códigos de detección de error. La redundancia de información puede ser implementada usando técnicas de hardware o de software y es ampliamente usada en comunicaciones y en dispositivos VLSI tales como memorias y procesadores.
- Redundancia de tiempo. Es el uso de tiempo adicional al requerido para implementar una función dada. Puede involucrar la repetición de cálculos y la comparación de los resultados obtenidos. Esta puede ser usada para detectar fallas transitorias y si se realizan más de 2 cálculos puede permitir que un cálculo fallido sea ignorado proporcionando así tolerancia a fallas. Se puede implementar usando técnicas de hardware o de software.

Los sistemas tolerantes a fallas usan una mezcla de técnicas que proporcionan protección contra un rango de posibles fallas.

Hay que mencionar que se recomienda usar en los sistemas redundantes algo que se denomina diversidad, es decir, que los componentes redundantes provengan de diversos fabricantes, esto con el propósito de reducir la probabilidad de tener fallas idénticas, lo cual haría que la falla no se detectara.

2.6. Técnicas de detección de fallas

Muchas técnicas de tolerancia a fallas se apoyan en técnicas de detección de fallas. En la práctica las fallas son detectadas como resultado de los errores que producen, por lo cual algunos lo denominan como detección de errores [Storey 96]. Se pueden usar técnicas de hardware o software para localizar fallas (ya sean de hardware o de software). Una vez que una falla ha sido detectada esta información puede ser entonces usada por una forma apropiada de tolerancia a fallas por hardware o software. Algunas de las técnicas utilizadas para la detección de fallas son:

- Chequeo de funcionalidad. Se tiene un conjunto de rutinas que verifican que el hardware del sistema este funcionando correctamente. Ejemplos comunes de esta forma de chequeo incluyen los de memoria, procesador y comunicaciones.
- Chequeo de consistencia. Se usa el conocimiento de la naturaleza de la información dentro del sistema para probar su validez. Un ejemplo de esta forma de prueba es el chequeo de rangos. Este compara valores calculados o almacenados de una variable con valores predefinidos para su rango permitido.
- Comparación de señales. En sistemas con redundancia se verifican las señales en puntos similares en los diversos módulos para validarlos. Este proceso es más fácil si los módulos son idénticos y no de diseño diverso.

- Pares de chequeo. Son un caso especial del uso de comparación de señales. Aquí los módulos idénticos son diseñados para permitir una comparación de múltiples señales con el propósito de detectar discrepancias. Si los módulos producen señales idénticas se asume que están libres de fallas.
- Redundancia de información. Existen varios tipos usados para detectar fallas tales como chequeo de paridad, sumas de chequeo o códigos CRC. Estas técnicas se describen en [Guy 91].
- Monitoreo de instrucciones, bus o fuente de poder. Se monitorea constantemente el funcionamiento del procesador para determinar si no genera una falla por haber obtenido operadores o códigos de operación corruptos, si no se acceden direcciones inválidas por un programa determinado o bien que la fuente de poder no reciba voltajes mayores o menores a los que debe tener.

2.7. Tolerancia a fallas en hardware

El método más común de lograr tolerancia a fallas involucra el uso de hardware redundante, esto debido a los bajos costos de los componentes electrónicos así como a su reducido tamaño. La redundancia por hardware tiene tres formas básicas: estática, dinámica e híbrida.

2.7.1. Redundancia estática

Los sistemas estáticos utilizan el enmascaramiento de fallas en vez de la detección de las mismas para lograr la tolerancia a fallas. Están diseñados para tolerar fallas sin requerir ninguna acción específica del sistema o su operador. La redundancia estática se apoya en el uso de alguna forma de mecanismo de voto para comparar las salidas de un número de módulos para enmascarar los efectos de fallas dentro de estas unidades.

La versión más simple es la TMR ya mencionada en la sección 5 de este capítulo. Este simple sistema tiene un punto posible de falla importante, el votante, por lo cual en algunos sistemas críticos se duplica o triplica este elemento.

La TMR se puede generalizar al considerar un arreglo de cualquier número de módulos. Esto se denomina generalmente como redundancia de N módulos (NMR) y en muchos casos un número impar de módulos se usa para permitir el uso de un esquema de mayoría de votos. Conforme el número de módulos se incrementa, la habilidad del sistema para soportar fallas de módulos también se incrementa. En general el fallo de módulos no resultará en una falla del sistema siempre y cuando la mayoría de los módulos operen correctamente; el sistema tolera la falla de (N-1)/2 módulos sin producir un mal funcionamiento del sistema. En la figura 2.3 se muestra el arreglo de un NMR:

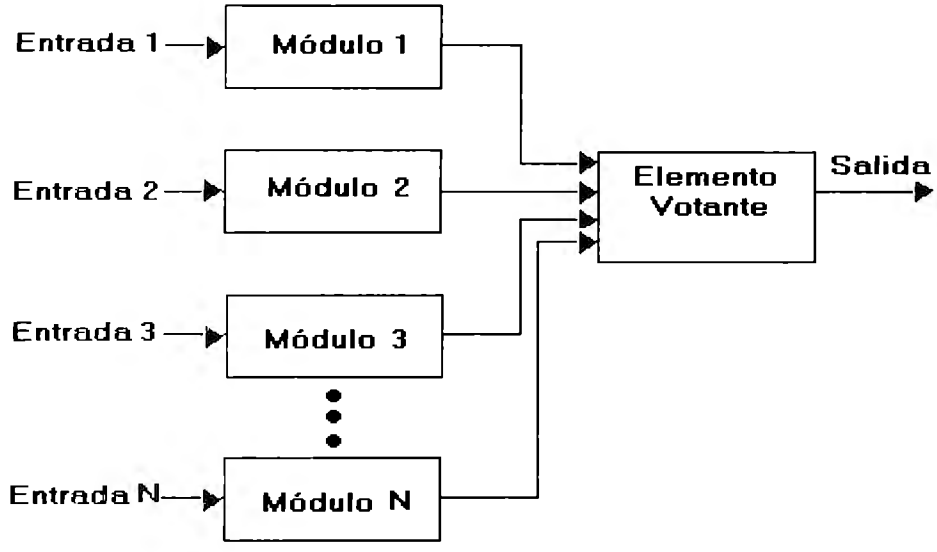

Figura 2.3 Sistema redundante de N módulos

La operación de voto puede ser realizada tanto en hardware (a través del uso de circuitos MSI) o mediante software. La ventaja de la implantación mediante hardware es la velocidad y la de la implantación mediante software es el menor costo.

2.7.2. Redundancia dinámica

Esta se apoya en la detección de fallas y en que el sistema tome una acción apropiada para nulificar sus efectos. Esto involucra la reconfiguración del sistema para remover la influencia del componente con fallas. En sistemas que usan este tipo de redundancia se tiene normalmente una unidad en uso, con uno o más sistemas en espera disponibles en caso de que esta unidad falle. Este enfoque reduce la cantidad de redundancia requerida ya que solo dos módulos son requeridos para enfrentarse a una sola falla y tres para manejar dos unidades con falla. El éxito de este enfoque se determina por la efectividad del proceso de detección de fallas. Los sistemas dinámicos no enmascaran las fallas sino que intentan contenerlas y entonces reconfigurar el sistema para lograr una tolerancia a las fallas. Por esta razón la redundancia dinámica es apropiada para aplicaciones que pueden tolerar errores temporales dentro de su operación.

Uno de los sistemas de redundancia dinámica más comunes es el de "repuesto en espera" mostrado en la figura 2.4.

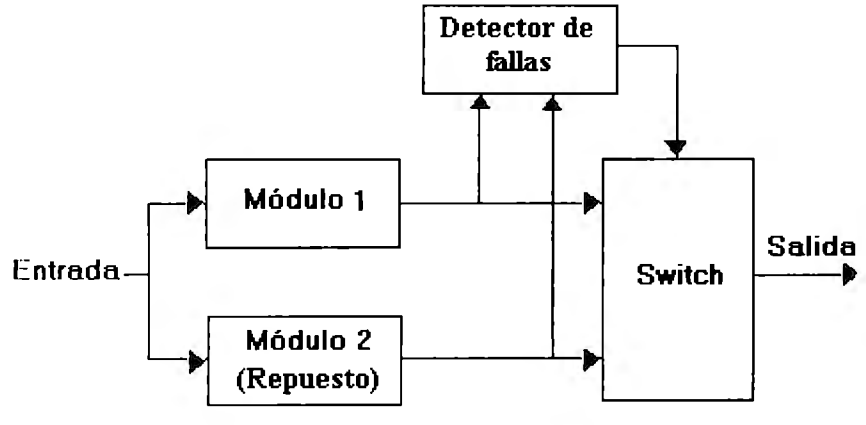

Figura 2.4 Distribución de un repuesto en espera

Aquí un módulo es operado en conexión con alguna forma de esquema de detección de fallas. Mientras no se detecte alguna falla el módulo maneja la salida a través del switch que es controlado por el sistema de detección de fallas. Si se detecta una falla, el switch

reconfigurará el sistema para que las salidas sean tomadas del módulo en espera. La reconfiguración del sistema que sigue a la detección de una falla remueve de manera efectiva el módulo con fallas del sistema y quita sus efectos. El proceso de reconfiguración causara una interrupción breve del sistema mientras las salidas son conmutadas. Esta interrupción puede ser minimizada con el uso de un arreglo en espera en caliente[3], donde el repuesto ejecuta sus funciones continuamente en paralelo con la unidad en activo. Esto permite una rápida transferencia del control con un mínimo de retardo aunque como desventaja se eleva el consumo de energía así como la probabilidad de que el repuesto también falle. La alternativa es usar una espera en frío[4], donde el repuesto está apagado hasta que se llame a servicio. Esto reduce el consumo de energía y el desgaste del módulo de respaldo pero causará una interrupción mayor en el cambio. Este sistema bimodular puede ser extendido para incluir cualquier número de módulos en espera como se muestra en la figura 2.5:

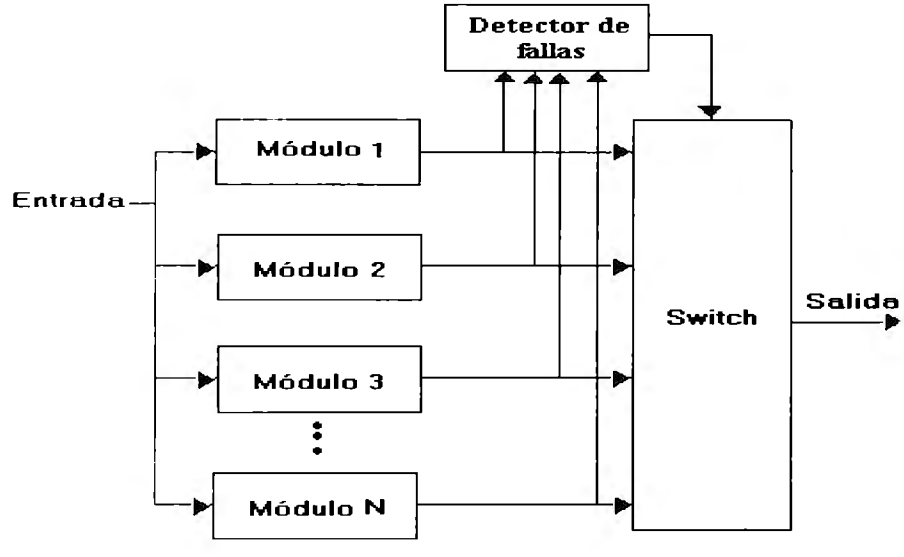

Figura 2.5 Sistema de repuesto en espera con N módulos

[3] Hot standby
[4] Cold standby

Como antes, un sólo módulo determina las salidas en cualquier momento con un circuito de detección de fallas conmutando a módulos sucesivos en el evento de mal funcionamiento de módulos.

La efectividad de los arreglos de repuestos en espera depende grandemente del rendimiento de las técnicas de detección de fallas usadas.

2.7.3. Redundancia híbrida

Las técnicas híbridas usan una combinación de los métodos estáticos y dinámicos. Un enfoque híbrido usa enmascaramiento de fallas para prevenir que los errores sean propagados dentro del sistema y la detección de errores y reconfiguración para remover unidades con fallas del sistema. Se usan muchas técnicas aunque la mayoría puede ser generalizada como alguna forma de redundancia de N módulos con repuestos.

En este enfoque, como su nombre implica, se abarcan elementos derivados de la técnica de redundancia de N módulos y de métodos dinámicos basados en el uso de repuestos en espera. Un arreglo típico se muestra en la siguiente figura:

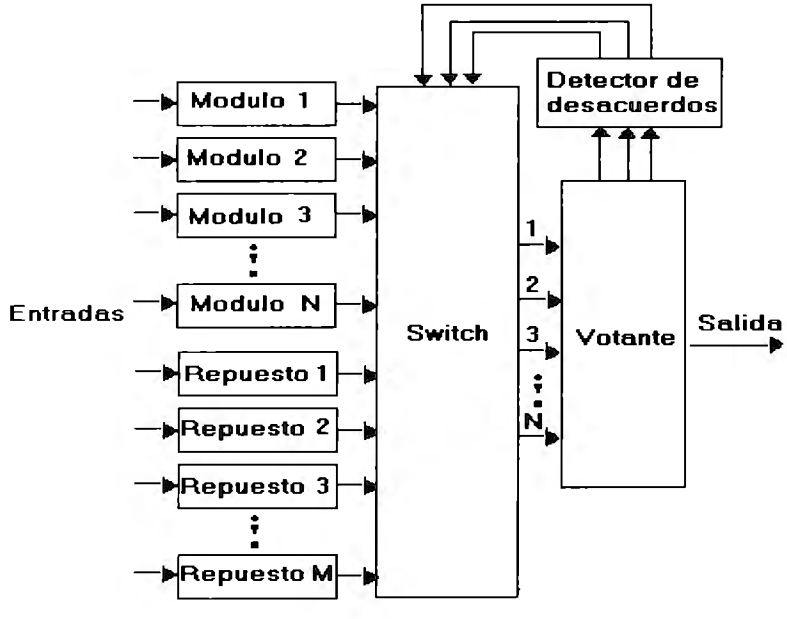

Figura 2.6 Redundancia de N módulos con repuestos

Este arreglo proporciona N módulos activos conectados a través de un conmutador a un votante. En ausencia de fallas el votante simplemente presentará la vista unánime de los módulos como su salida. Si uno de los módulos activos falla, el votante enmascara esta falla adoptando la vista de la mayoría. Sin embargo, el módulo que produjo una salida diferente es identificado por el detector de desacuerdos y este módulo es removido por la unidad de conmutación tomando su lugar uno de los módulos de repuesto. El número de módulos con falla que pueden ser removidos esta determinado por el número de unidades de repuesto proporcionadas (M).

Las técnicas híbridas requieren de un nivel de redundancia entre los asociados a los sistemas estáticos y dinámicos y en muchos casos ofrece una alternativa atractiva aunque se debe tener cuidado en el diseño del conmutador. El uso de mayoría de votos enmascara las fallas al mundo exterior mientras que la identificación de fallas y reconfiguración del sistema mantienen la tolerancia a fallas y sirve de guía para el mantenimiento.

2.8. Tolerancia a fallas por software

Una alternativa al enmascaramiento logrado por el uso de redundancia una vez que se ha detectado una falla es la recuperación de errores mediante software, en ésta el estado erróneo al cual se ha llegado por una falla es reemplazado por un estado válido aceptable. El estado de remplazo se puede construir reparando el estado erróneo o puede ser copiado de algún estado previo que se crea sea previo a la activación de la falla [Rushby 94]. El primero de estos métodos se denomina recuperación de errores hacia adelante y el segundo se denomina recuperación de errores hacia atrás. El manejo de excepciones [Cristian 89] es el medio principal por el cual se organiza la recuperación de errores hacia adelante en software; los puntos de verificación y los "bloques de recuperación" [Randell 75, Anderson 90, Horning 74] proporcionan medios para organizar la recuperación de errores hacia atrás.

La tolerancia a fallas puede ser proporcionada en muchos niveles diferentes en la jerarquía del sistema y en diversas formas de tal manera que se requieren principios de organización [Rushby 94]. En [Cristian 91] se puede encontrar una visión general sobre las alternativas así como aspectos de diseño relacionados con los sistemas tolerantes a fallas.

La importancia relativa de la tolerancia a fallas varía entre los dominios de las aplicaciones, debido principalmente a que las consecuencias de los mal funcionamientos pueden variar desde un simple inconveniente hasta un riesgo financiero significativo o la potencial pérdida de vidas. La tolerancia a fallas es importante en los sistemas distribuidos. Por un lado, debido al gran número de computadoras que se encuentran en algunos de estos sistemas, la probabilidad de que al menos una falle se incrementa y por lo tanto a menudo se requieren de técnicas de tolerancia a fallas para asegurar que estos mal funcionamientos no detengan al sistema. Por otro lado, la autonomía de las computadoras en tales sistemas significa que las fallas son a menudo independientes; y por ello, un servicio tolerante a fallas puede ser construido sobre un sistema distribuido al replicar el servicio en más de una computadora para que permanezca disponible siempre y cuando al menos una máquina esté funcionando [Hiltunen 96].

CAPITULO 3

MEDIDAS DE DEPENDENCIA EN REDES

3.1. Introducción

El uso de sistemas computacionales como los mencionados en la sección 1 del capítulo anterior requiere de altos niveles de dependencia[1], lo cual significa que se puede confiar de manera justificada en el servicio que proporciona [Laprie 92, Storey 96]. La dependencia es cuantificada en términos de varios factores. Uno de esos factores es la tolerancia a fallas, la cual como ya mencionamos significa la habilidad del sistema para continuar proporcionando el servicio especificado aún en presencia de fallas; gracias a la redundancia en hardware y software el sistema es capaz de reconfigurarse a sí mismo cuando ocurren fallas utilizando las técnicas analizadas en el capítulo anterior.

Los otros factores que determinan la dependencia son [Kopetz 93, Hiltunen 96, Storey 96]:

- Confiabilidad: Medida de la entrega de servicio continuo, determinada por la probabilidad R(t) de que el sistema estará proporcionando el servicio correcto después de un intervalo de tiempo t, si es que estaba funcionando correctamente en t=0. El término "funcionando correctamente" significa que opera como se definió dentro de su especificación.
- Seguridad: Probabilidad S(t) de que un sistema no fallará de modo catastrófico dentro de un intervalo t.
- Disponibilidad: Medida de la entrega de servicio correcto con respecto a la alternancia entre servicio correcto e incorrecto, dada por la probabilidad A(t) de

[1] Dependability

que el sistema esté listo para proporcionar el servicio en un punto en el tiempo t. La disponibilidad también se puede ver como la fracción de tiempo en la cual el sistema está funcionando correctamente

- Mantenibilidad: Medida del tiempo para restaurar el sistema desde la última falla experimentada, dada por la probabilidad M(t) de que el sistema sea restaurado en el tiempo t si fallo en el tiempo t=0. La mantenibilidad puede ser tratada de manera cualitativa o cuantitativa. Cuando se usa la última a menudo se expresa en términos del Tiempo Promedio para Reparar (MTTR) el sistema en el evento de una falla.

Examinaremos ahora más a detalle algunos fundamentos matemáticos relacionados con el cálculo de la confiabilidad de componentes individuales.

3.1.1. Confiabilidad

Los componentes que fallan como resultado de fallas no sistemáticas lo harán en tiempos aleatorios. Para un dispositivo dado no es posible predecir el tiempo en el que fallará pero es posible cuantificar la tasa a la cual los miembros de una familia de componentes fallará. De acuerdo a la definición dada para la confiabilidad se puede ver claramente que es una función del tiempo. Si se considera un conjunto de N componentes idénticos, donde todos empiezan a operar al mismo tiempo, entonces en algún tiempo t posterior, el número de componentes funcionando correctamente es n(t), donde:

$$R(t) = \frac{n(t)}{N} \qquad \ldots (3.1)$$

También se puede definir el término no confiabilidad, el cual es la probabilidad de que el sistema no funcionará correctamente en un período dado de tiempo. A este término se le da el símbolo Q(t) y también se denomina probabilidad de falla. Si el número de

componentes que ha fallado durante un tiempo t se representa por $n_f(t)$, entonces:

$$Q(t) = \frac{n_f(t)}{N} \qquad \text{... (3.2)}$$

y, de las definiciones de confiabilidad y no confiabilidad es claro que

$$Q(t) = 1 - R(t) \qquad \text{... (3.3)}$$

3.1.2. Tasa de falla

Muy relacionado con la confiabilidad de un componente está la tasa a la que él mismo falla. La tasa de falla de un dispositivo o sistema es el número de fallas dentro de un período dado de tiempo. Por ejemplo, si un dispositivo falla, en promedio, una vez cada 1000 horas entonces tiene una tasa de falla de 1/1000 fallas por hora. La tasa de falla de un componente normalmente varía con el tiempo y es representada por el símbolo λ durante el tiempo útil de vida del componente el cual es considerado constante.

La experiencia muestra que la tasa de falla de los componentes electrónicos normalmente exhibe diferentes características, como se muestra en la figura 3.1:

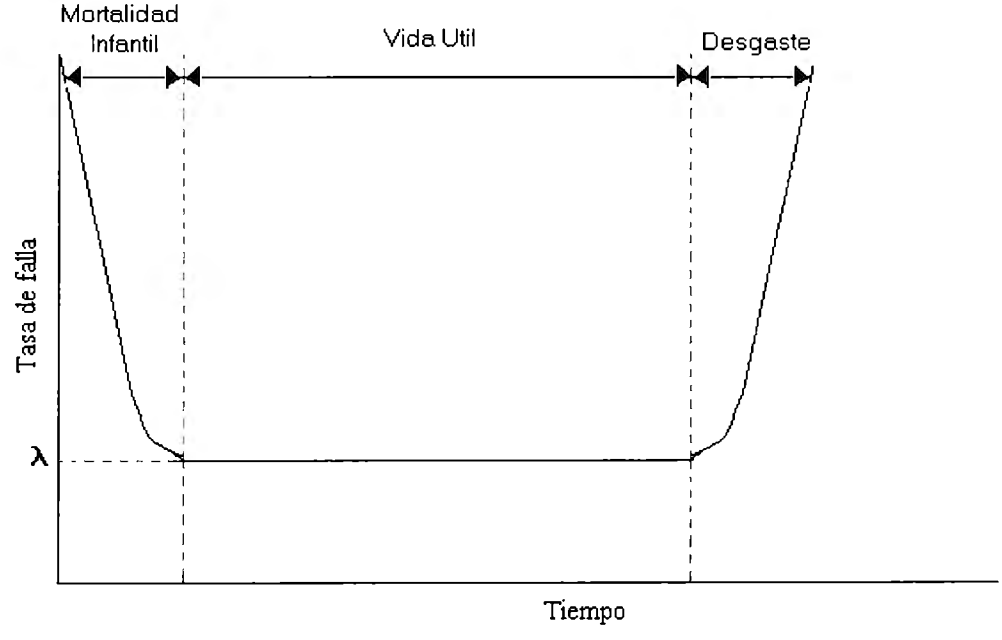

Figura 3.1 Variación típica de la tasa de fallas (Curva Bath-tube)

Por razones obvias, esta característica es normalmente descrita como la curva de "tina de baño"[2]. Inicialmente, los componentes exhiben una alta "mortalidad infantil" debida a la presencia de fallas de manufactura que no fueron detectadas durante la etapa de prueba en su fabricación. Conforme el tiempo pasa el número de componentes que contiene estos defectos disminuye y la tasa de falla cae a un nivel constante. Algún tiempo después los efectos de la edad se hacen aparentes y la tasa de falla nuevamente se eleva conforme los dispositivos se desgastan. Los fabricantes normalmente usan los componentes solo durante la parte relativamente constante de la curva, y a este período normalmente se le denomina "período de vida útil".

Durante la etapa de vida útil la tasa de falla esta relacionada a la confiabilidad del dispositivo por la expresión:

$$R(t) = e^{-\lambda t} \qquad \ldots (3.4)$$

La relación exponencial entre la confiabilidad y el tiempo es conocida como la ley de falla exponencial. Esto indica que para una tasa constante de falla la confiabilidad cae exponencialmente con el tiempo. Así la probabilidad de que un componente trabaje correctamente a través de un período de tiempo dado se decrementa exponencialmente con la longitud del período de tiempo.

3.1.3. Tiempo promedio entre fallas

Otra forma de describir la confiabilidad de un componente es dar un tiempo promedio para fallar (MTTF), esto es, el tiempo esperado que el componente operará antes de que la primera falla ocurra, y se calcula de la siguiente manera:

$$MTTF = \int_0^\infty R(t)dt \qquad \ldots (3.5)$$

---

[2] Bathtube

y por lo tanto, por el período de tiempo donde la falla es constante,

$$MTTF = \int_0^\infty e^{-\lambda t} dt = \frac{1}{\lambda} \qquad \text{... (3.6)}$$

Esto nos lleva que el MTTF es el inverso de la tasa constante de falla del componente.

3.1.4. Tiempo promedio para reparar

El tiempo promedio para reparar (MTTR) es simplemente el tiempo tomado para reparar un componente que ha fallado y volverlo a poner en operación. Esta cifra toma en cuenta el tiempo tomado para detectar la falla, localizarla, efectuar una reparación y reconfigurar el sistema. A menudo el MTTR puede ser estimado durante la etapa de diseño y permite que el rendimiento del sistema sea predecido, pero puede ser necesario determinarlo experimentalmente cuando el sistema esta en operación.

Justo como describimos la confiabilidad de un componente usando la tasa de falla λ, se puede cuantificar la reparabilidad de un componente usando su tasa de reparación μ, que es el tiempo promedio tomado para reparar el sistema. Entonces, y de la misma manera que se calcula el MTTF,

$$MTTR = 1/\mu \qquad \text{... (3.7)}$$

3.1.5. Tiempo promedio entre fallas

Si puede asumirse que una vez que un componente que ha fallado ha sido reparado su rendimiento será equivalente al original, entonces es posible predecir el tiempo promedio entre fallas (MTBF), el cual esta dado por:

$$MTBF = MTTF + MTTR = \frac{\mu + \lambda}{\lambda\mu} \qquad \text{... (3.8)}$$

En la mayoría de los casos el tiempo tomado para reparar un componente (MTTR) será pequeño comparado con el tiempo por el cual opera, así que en la práctica el MTBF será numéricamente similar al MTTF.

3.1.6. Disponibilidad

La disponibilidad es la fracción del tiempo en el cual el componente esta en operación. Esto se puede expresar en términos de MTTF y MTTR:

$$Disponibilidad = \frac{Tiempo\ que\ el\ sistema\ está\ en\ operación}{Tiempo\ total} = \frac{MTTF}{MTTF + MTTR} = \frac{\mu}{\lambda + \mu} \quad ...(3.9)$$

3.2. Modelado de la confiabilidad

Es importante no solo determinar la confiabilidad de componentes individuales sino también poder predecir la confiabilidad que tendrá el sistema en su totalidad. Para ello se usan métodos de modelado de los cuales los más comunes son: modelado combinatorio y modelado de estados de Markov [Valdivia 89].

3.2.1. Modelos combinatorios

Los modelos de confiabilidad combinatoria permiten el cálculo de la confiabilidad de un sistema a partir de la confiabilidad de sus componentes. El modelo distingue entre situaciones donde la falla de uno solo de los componentes causa una falla del sistema y situaciones donde varios componentes deben fallar para causar una falla del sistema. Estas dos situaciones se modelan por los modelos en serie y en paralelo respectivamente. Para estos modelos se usan diagramas de bloque de confiabilidad [NASA 95, Storey 96]. Se pueden también combinar estos dos modelos para representar sistemas más complejos a través del modelo en serie-paralelo.

Se tienen también modelos combinatorios que usan diagramas de bloque de confiabilidad para representar sistemas tolerantes a fallas que usan técnicas tales como TMR, NMR y sistemas de repuestos en espera [Valdivia 89].

3.2.2. Modelos de Markov

Este es un modelo alterno en el cual se asignan varios estados al sistema y se determina la probabilidad de estar en cualquiera de estos estados. Una de las ventajas de este enfoque es que proporciona una manera más poderosa de modelar sistemas que son reparables permitiendo que variables tales como el tiempo tomado para reparar un sistema sean incorporadas. Los modelos de Markov pueden ser discretos o continuos. Se puede encontrar una explicación detallada de estos modelos en [Lewis 96].

3.3. Análisis de confiabilidad de redes

Después de haber visto algunos términos básicos relacionados con el cálculo de la confiabilidad enfoquémonos ahora a las técnicas existentes para determinar la confiabilidad que presenta una red, lo cual está muy relacionado con el propósito básico de esta tesis, el cual es determinar de la mejor manera posible la confiabilidad que presenta la red de interconexión de un sistema distribuido, puesto que éste es el componente básico para que funcione correctamente el sistema. Por tanto, la conectividad entre los diversos componentes es de gran importancia y tiene mucho peso en el cálculo de la confiabilidad.

Consideramos una red como un sistema que involucra el movimiento de algún bien tal como información, productos o personas. Las redes de computadoras, circuitos electrónicos y redes de comunicaciones son algunos ejemplos. En la actualidad la ciencia tiene un gran número de problemas relacionados con el diseño, construcción y operación de tales redes entre los cuales se encuentran el diseño de redes confiables de mínimo costo y el diseño de microprocesadores para la resolución de problemas complejos, entre otros.

Las redes mencionadas se pueden ver como un sistema de componentes que interactúan de alguna forma, y que es convenientemente modelado por un grafo G en donde los vértices (V) representan los componentes y las aristas o enlaces (E) representan la interacción entre los componentes [Colbourn 95]. El grafo resultante G=(V, E) nos da el modelo estructural de la red que se está estudiando y que muestra las interacciones entre los diversos componentes de la red[3].

El rendimiento de las redes que consisten de componentes que no fallan se puede medir en términos de parámetros tales como: diámetro, conectividad, distancia promedio, número máximo de rutas o caminos disjuntos, capacidades de corte, etc.

Cuando los componentes que conforman a la red están sujetos a fallas, las redes pueden ser modeladas por grafos con peso que representan la probabilidad de que un componente funcione correctamente (o en algunos modelos es la probabilidad de que el enlace esté ocupado o con falla). Tal grafo se denomina grafo probabilístico [Colbourn 91].

La mayor parte de la investigación realizada sobre confiabilidad de redes considera al grafo probabilístico G=(V, E) mencionado como un modelo de conectividad de la red en el cual los enlaces tienen asociada una probabilidad de operación p_e, tales probabilidades son estadísticamente independientes y los vértices no fallan. Tal investigación está encaminada a la solución de un problema denominado k-terminal.

Se tiene un conjunto K de vértice tal que $K \subseteq V$ y un vértice $s \in K$ denominado vértice fuente. El problema de la confiabilidad de k terminales de G se define como la probabilidad de que exista una ruta compuesta de enlaces en operación de s a cada vértice de K, y es representado con R(G,s,K) [Colbourn 92]. Los elementos en K-s se denominan vértices destino. Cuando |K|=|V|=n se denomina problema de confiabilidad a todas las

[3] En el anexo se da una breve introducción a los conceptos relacionados con la teoría de grafos.

terminales y se representa con R(G,s,V). Cuando |K|=2 se denomina problema de confiabilidad de 2 terminales, el elemento no fuente de K se representa por t, convirtiéndose pues esto en el problema de determinar la probabilidad de que exista una ruta entre s y t en G y se representa por R(G,s,{s,t}).

El que exista una gran cantidad de investigación referente a esto tiene la siguiente causa: aunque la modelación de una red es simple, el análisis de su confiabilidad es bastante complejo. Se ha comprobado que las técnicas para encontrar valores exactos de los problemas de confiabilidad pertenecen a la clase #P-completa, la cual es una clase aplicada a los problemas de conteo que está en correspondencia a los problemas NP-completos, lo cual significa que no existe un algoritmo que resuelva en un tiempo razonable el problema, la comprobación se puede encontrar en [Colbourn 92]. Esto ha dado como resultado que se utilicen otros métodos que obtengan una estimación del valor de la confiabilidad entre los que se encuentran métodos para obtener límites, métodos de Monte Carlo o métodos de espacios de estado restringidos que debido a la utilización de diversas técnicas combinatorias cuya implantación no es directa aumentan todavía más la complejidad.

3.3.1. Cálculo exacto de la confiabilidad.

Debido a la complejidad que tienen las medidas de confiabilidad de la red, existen algoritmos que resuelven redes generales pero se ejecutan con tiempo exponencial o resuelven clases restringidas de redes y se ejecutan en tiempo polinomial. Ambos tipos de algoritmos se apoyan en transformaciones que se le aplican al grafo que representa la red para simplificarlo. Algunas de esas transformaciones son:

- Eliminación de enlaces irrelevantes.
- Contracciones.
- Reducción en serie.

- Reducción en paralelo.
- Reducción de grado 2

Entre los algoritmos eficientes para clases restringidas se encuentra el realizado por Wood y Satyanarayana [Wood 85], y el de Wald y Colbourn [Wald 83], ambos para la confiabilidad de k terminales. En [Karger 95] se analiza un algoritmo para el problema de todas las terminales. También se pueden encontrar otros algoritmos de este tipo en [Colbourn 95].

3.3.2. Métodos de Monte Carlo

La meta de estos métodos es obtener un estimado de la medida de confiabilidad examinando una pequeña fracción de los estados, seleccionados aleatoriamente [Colbourn 92]. Esto lleva a un estimado puntual de la medida de la confiabilidad, junto con los intervalos de confianza para el estimado. Dicho de otra manera, los métodos de Monte Carlo hacen una simulación. Aunque típicamente son mucho más rápidas que los métodos exactos, las técnicas de Monte Carlo requieren la examinación de un número exponencial de estados para obtener un estimado con la confiabilidad deseada. Una descripción de los métodos de Monte Carlo se puede encontrar en [Cancela 95].

3.3.3. Límites sobre la confiabilidad de la red

Las técnicas para la obtención de límites intentan encontrar estructuras algebraicas o combinatorias en el problema de la confiabilidad, permitiendo la deducción de información estructural al examinar una pequeña fracción de los estados. A diferencia de los métodos de Monte Carlo los estados examinados no son seleccionados aleatoriamente. La meta de las técnicas de obtención de límites es producir límites absolutos superior e inferior sobre la medida de confiabilidad [Colbourn 92], es decir, estos métodos nos arrojan un rango en el

cual se encuentra el valor de la confiabilidad. Cabe mencionar que es un error separar los métodos de Monte Carlo de los de obtención de límites ya que un número de métodos de Monte Carlo emplea técnicas de obtención de límites para restringir el tamaño del espacio de muestra.

Estos métodos se pueden dividir de acuerdo a los casos que manejan:

- Igual probabilidad de falla en los enlaces
- Diferente probabilidad de falla en los enlaces

Algunos de estos algoritmos se pueden encontrar en [Colbourn 95, Colbourn 92, Shier 91]

En [Valdivia 89] se propone un modelo para la obtención de una aproximación del valor de la confiabilidad de una red de interconexión haciendo énfasis en la conectividad como medida de confiabilidad con la característica de que se considera además de la probabilidad de falla en los enlaces, la probabilidad de fallo en los nodos (lo cual no se hace en los métodos a los que se ha hecho referencia en esta sección) aparte de tomar en cuenta también los aspectos deterministicos de la red. Este modelo puede extenderse de tal forma que los resultados obtenidos estén más acordes a la forma en que se seleccionan las rutas en la red específica en la que se esté aplicando y esto se lograría al hacer que se involucre en el modelo el algoritmo de ruteo utilizado en la red de interconexión. Analizaremos pues en el siguiente capítulo los principales aspectos del ruteo y el papel que juega en los distintos tipos de redes para posteriormente aplicar el algoritmo de ruteo que determinemos más conveniente en el modelo de confiabilidad.

CAPITULO 4

RUTEO

4.1. Introducción

La meta del ruteo en una red de comunicaciones es dirigir el tráfico del usuario de la fuente al destino de acuerdo con los requerimientos de servicio del tráfico y las restricciones del servicio de la red [Steenstrup 95]. El ruteo es un sistema complejo que se modela en función de objetivos y restricciones potencialmente conflictivos. Entre los objetivos podemos incluir la maximización del rendimiento de la red y la minimización de los costos de la misma. Entre las restricciones tenemos las impuestas por la tecnología de conmutación de la red y el dinamismo del tráfico de usuario.

4.1.1. Funciones del ruteo

El ruteo ocurre en el nivel 3 (capa de red) del modelo de referencia OSI [Stallings 94, Tanenbaum 97]. El ruteo involucra tres funciones básicas [Steenstrup 95]:

- Recolectar y distribuir información de estado de la red y del tráfico de usuario. Esta información de estado incluye requerimientos de servicio, ubicaciones de usuarios, recursos disponibles en la red y restricciones del uso de estos servicios y recursos.
- Generar y seleccionar rutas factibles e incluso óptimas basadas en la información de estado de la red y del tráfico de usuario. Las rutas factibles son aquellas que satisfacen todas las restricciones de servicio impuestas por la red y los usuarios y las rutas óptimas son rutas factibles que son las mejores respecto a un objetivo de rendimiento específico.

- Transporte de los grupos de información (típicamente denominados paquetes) a través de una interred (lo cual es comúnmente llamado conmutación o switcheo). Existen 2 paradigmas principales para esta actividad: orientada a la conexión y sin conexión. El paradigma orientado a la conexión requiere que se instalen directivas en los conmutadores de una ruta antes de que esta pueda ser usada para transportar tráfico. En cambio, en el paradigma sin conexión el tráfico trae inmersa la información de paso de paquetes en forma de directivas explícitas para cada conmutador o implícitas, en cuyo caso pueden ser interpretadas independientemente por cualquier conmutador en la red.

4.1.2. Determinación de las rutas

El problema general de ruteo en una red (en cuanto a la generación y selección de rutas) consiste en encontrar un protocolo o función de ruteo que para cualquier par fuente - destino, permita que cualquier mensaje generado por la fuente pueda ser ruteado al destino [Gavoille 96]. Para lograr esto se hace uso de algoritmos de ruteo los cuales utilizan una medida estándar llamada métrica para comparar las rutas y obtener la mejor, tal métrica puede ser la distancia, el retardo, etc.

Cuando las redes no tienen ninguna estructura predefinida (tal como la tienen las mallas o hipercubos) se usan por lo general *tablas de ruteo*: cuando un nodo recibe un mensaje consulta su tabla local para determinar el puerto de salida por el cual debe ser enviado el mensaje. Tal proceso es rápido y puede ser aplicado en todas las redes. Estas tablas son inicializadas y mantenidas por los algoritmos de ruteo, y la información en ellas almacenada varía de acuerdo al algoritmo utilizado. Un ejemplo de la información que puede almacenarse en estas tablas es un conjunto de asociaciones destino/siguiente salto las cuales indican que un destino puede ser alcanzado de manera óptima al enviarse un paquete

dirigido a tal destino al nodo que dicta el "siguiente salto" asociado. También se podría almacenar información sobre las características de la vía que permitan seleccionarla de acuerdo a la métrica utilizada.

Los nodos encargados de la función de ruteo (denominados ruteadores) se comunican uno con otro a través de la transmisión de una variedad de mensajes. Uno de ellos es el mensaje *actualización de ruteo*. Estos mensajes consisten generalmente de toda o una parte de la tabla de ruteo. Al analizar las actualizaciones de ruteo de todos los ruteadores, un ruteador puede construir una imagen detallada de la topología de la red. Una vez que la topología de la red es entendida, los ruteadores pueden determinar rutas óptimas a los destinos de la red.

4.1.3. Algoritmos de ruteo

Los algoritmos de ruteo pueden ser diferenciados basados en varias características clave [CISCO 97]:

- Las metas particulares del diseñador del algoritmo, las cuales afectan la operación del protocolo de ruteo resultante.
- El impacto que tiene sobre la red y los recursos del ruteador.
- Las métricas utilizadas para el cálculo de las rutas.

Los algoritmos de ruteo a menudo tienen una o más de las siguientes metas de diseño [CISCO 97]:

- **Optimalidad**. Capacidad de seleccionar la "mejor" ruta, la cual depende de la métrica utilizada para hacer el cálculo.
- **Simplicidad**. Se busca que sean lo más simple posible de forma que sean eficientes y que utilicen un mínimo de recursos (memoria, programación).

- **Robustez**. Deben realizar su trabajo correctamente a pesar de circunstancias inusuales o no previstas tales como fallas de hardware, condiciones de alta carga e implantaciones incorrectas.
- **Convergencia rápida.** La convergencia es el proceso de llegar a un acuerdo de todos los ruteadores sobre rutas óptimas. Esto acuerdos deben tomarse cuando fallan rutas o se crean nuevas y se apoyan en el intercambio de mensajes de actualización. Si se converge de manera lenta se pueden crear ciclos de ruteo.
- **Flexibilidad.** Deben adaptarse de manera rápida y exacta a una variedad de circunstancias de red tales como fallas en enlaces, cambios en el ancho de banda o retardo entre otras.

Los algoritmos pueden ser clasificados de varias maneras [CISCO 97, Scheideler 96]:

- **Estáticos o Dinámicos**. Los algoritmos estáticos son simples y trabajan bien en ambientes donde el tráfico de red es relativamente predecible. Casi no requieren de recursos computacionales ya que no hacen más que pasar paquetes. Las tablas son establecidas de manera manual por los administradores de la red y no cambian automáticamente en respuesta a los cambios en la topología; debido a esto no son adecuados para las redes actuales. Los algoritmos dinámicos en cambios se ajustan en tiempo real a las circunstancias de la red al analizar los mensajes de actualización de la red que se reciben, recalculando las rutas según sea necesario y avisando de ello a los demás.
- **De una sola ruta o Multiruta**. Los algoritmos multiruta permiten la existencia de múltiples vías a un mismo destino y multiplexan los mensajes dando un nivel de rendimiento y confiabilidad. En cambio, los algoritmos de una sola ruta no

permiten esto.

- **Planos o Jerárquicos**. En los algoritmos planos, todos los ruteadores son iguales, mientras que en los jerárquicos algunos ruteadores forman lo que se denomina un troncal de ruteo. Los ruteadores no troncales entregan los paquetes a los ruteadores troncales, los cuales se encargan de llevar los paquetes al área de destino donde son entregados a un ruteador no troncal que se encargara de entregarlos al destino final. Esto permite la reducción de tráfico de actualizaciones.
- **Determinísticos o adaptivos.** Un algoritmo determinístico es aquel en el que la ruta asignada para un determinado mensaje no cambia durante el recorrido del mismo mientras que un en algoritmo adaptivo la ruta para un determinado mensaje puede cambiar durante el recorrido del mensaje debido al tráfico o a que falló un enlace en la ruta [Chien 95]. En este sentido los algoritmos adaptivos hacen un mejor uso de las líneas de comunicación.

4.1.4. Métricas

Las métricas utilizadas para la determinación de la mejor ruta son las siguientes, las cuales son utilizadas por algunos algoritmos de manera única (solo una de ellas) o híbrida (una combinación de ellas) [CISCO 97]:

- **Longitud de la vía**. Esta es la más común y es la suma de los costos asociados con cada enlace atravesado (en caso de que el protocolo permita tal asignación de costos a los enlaces) o bien, es el conteo de saltos (números de ruteadores que se atraviesan) para llegar de la fuente al destino.
- **Confiabilidad**. Que tanta probabilidad de error hay en los enlaces atravesados.
- **Retardo**. Longitud de tiempo requerido para mover un paquete de la fuente al

destino a través de la interred. Este depende de muchos factores tales como el ancho de banda, la congestión en la red y la distancia física. Debido a esta conglomeración de diversas e importantes variables, el retardo es una métrica útil y muy común.

- **Ancho de banda**. La capacidad de tráfico disponible de un enlace. Hay que hacer notar que no siempre el enlace con más ancho de banda es la mejor ruta pues podría estar congestionado.
- **Carga**. Grado con el cual el recurso de la red está ocupado. Se puede calcular de diversas maneras, tales como utilización del CPU o paquetes procesados por segundo.
- **Costo de la comunicación**. El costo económico asociado con el uso del enlace. Algunas compañías están más interesadas en este aspecto que en otros tales como el ancho de banda.

4.1.5. Descentralización del sistema de ruteo

El sistema de ruteo puede ser implantado de manera centralizada o descentralizada. El grado de descentralización depende del dinamismo, robustez y maleabilidad deseados [Steenstrup 95]. Cuando el ruteo es centralizado, una sola entidad realiza las funciones de ruteo lo cual lo hace fácil de administrar y reduce los costos de los recursos requeridos, pero tiene la desventaja de que si esta entidad falla el ruteo deja de estar disponible, aparte de que el tiempo de reacción a los cambios va a depender de la carga y de la distancia a la que se encuentra el lugar donde sucedió alguna modificación de estado. En cambio, en una implantación descentralizada múltiples entidades similares realizan la función de ruteo de manera independiente y distribuida pero en forma parcial y existe una cooperación para proporcionar la funcionalidad completa. Esto hace que se eleve la tolerancia a fallas y

reduce el tiempo de respuesta a las modificaciones de estado, también reduce la cantidad de recursos requerido en cada entidad y permite escalabilidad aunque tiene como desventaja la complejidad de su administración.

4.2. Impacto de las tecnologías de red en la función de ruteo

Tradicionalmente, las redes de comunicación han sido divididas en conmutación de circuitos o conmutación de paquetes [Steenstrup 95], lo cual no solo distingue la tecnología de conmutación sino el propósito para el cual fue desarrollada la red. Sin embargo, el crecimiento en el número de computadoras personales poderosas y portátiles y de las redes de comunicación accesibles globalmente han resultado en la demanda de servicios de comunicación más sofisticados por lo cual han surgido nuevas tecnologías tales como redes de alta velocidad y redes móviles. Cada una de estas tecnologías usan esquemas de ruteo distintos. Analicemos algunas características de cada una de estas tecnologías para determinar las razones de estas diferencias en cuanto a la función de ruteo.

4.2.1. Conmutación de circuitos

Este tipo es el más antiguo de tecnología y también la que más ha durado. Originalmente fue diseñada para la comunicación de voz y lo que hace es establecer circuitos físicos de las fuentes a los destinos en respuesta a las demandas de conexión hechas por los usuarios. Tales circuitos (y los recursos que estos utilicen) son dedicados a quien fueron asignados independientemente de sí son usados o no hasta que explícitamente se termina la conexión entre los usuarios. Las redes de este tipo más familiares son las de los sistemas telefónicos, las cuales, de inicialmente llevar tráfico de voz de manera análoga han evolucionado hasta manejar voz, datos y vídeo de manera digital.

El ruteo en este tipo de redes se ha manejado principalmente por los objetivos económicos de las compañías telefónicas y por las tecnologías de conmutación disponibles

[Steenstrup 95]. Su objetivo principal es asegurar que las redes cumplan con las demandas pico de tráfico y la recuperación de fallas y de otros eventos poco probables, por lo cual mucho de sus recursos se canalizan al control de tráfico y diseño de la red (de la cual forma parte integral el ruteo).

Históricamente, las redes telefónicas se han apoyado en rutas estáticas las cuales son preconfiguradas fuera de línea y que dependen de los pronósticos de la demanda de tráfico y no del estado actual de la red. Cada conmutador tiene múltiples rutas a cada destino con el propósito de maximizar la probabilidad de proporcionar el servicio; tales rutas pueden ser:

- Rutas alternativas utilizadas cuando las llamadas son bloqueadas en la ruta principal
- Rutas dependientes del tiempo para uso en diferentes horas del día.

Las ventajas de este tipo de ruteo son el control total sobre las rutas seleccionadas y que los requerimientos de procesamiento de los conmutadores son mínimos. Sin embargo, tiene como desventaja la lenta adaptación a los eventos no predecibles, la incapacidad de optimizar las rutas dentro de la red y la gran cantidad de memoria que deben tener los conmutadores para almacenar todas las rutas.

Ahora bien, el uso de este tipo de ruteo en el pasado se debió a dos factores importantes: la poca capacidad de procesamiento de los conmutadores de ese tiempo y el hecho de que las empresas rehuían dar parte del control de la red a la red misma por miedo a las consecuencias económicas que pudieran traer decisiones incorrectas de ruteo. Sin embargo, esto ha cambiado con la aparición de conmutadores capaces de almacenar programas y de procesar a altas velocidades, haciendo que el ruteo dinámico sea una alternativa factible para el ruteo de redes telefónicas al permitir la reducción de bloqueo de

llamadas, número de circuitos necesarios y costos de operación.

Una estrategia de ruteo dinámico usa información del estado de la red al seleccionar una ruta para una llamada y la velocidad a la que se adapta a los cambios depende de sí la selección de la ruta se basa en información global o local. Estas estrategias obtienen una mayor confiabilidad y rendimiento y permiten simplificar el diseño y administración de la red así como la posibilidad de agregar nuevas características tales como los servicios integrados.

Una clase importante de estrategias de ruteo dinámico son los algoritmos de *ruteo dependiente del estado*, los cuales intentan rutear cada llamada de tal forma que minimicen la probabilidad de llamadas futuras por lo cual se apoyan en el conocimiento de las demandas de tráfico pronosticadas y requiere que los conmutadores midan y recolecten información de estado de la red tal como la oferta de carga y la tasa de bloqueo de llamadas [Pack 90]. Otras estrategias de ruteo dinámico se describen brevemente a continuación:

- DCR (Dynamically Controlled Routing), posteriormente renombrado como HPR (High Performance Routing). Aquí se alimenta a un procesador central con la información del estado de la red en intervalos de 10 segundos, tal procesador actualiza las tablas de ruteo de los conmutadores con la misma periodicidad [Girard 83].
- DNHR (Dynamic Non Hierarchical Routing). En este esquema las tablas de ruteo son optimizadas y actualizadas cada hora por un procesador central, la variación de las rutas es por la hora del día [Ash 81]. Fue utilizado en la red de larga distancia de AT&T a finales de los 80's [Schwartz 80]. La esencia de este algoritmo es la examinación secuencial de un subconjunto de las rutas alternas posibles cuando la ruta directa está llena, tales rutas alternas varían como ya se mencionó de acuerdo a la hora del día y se calculan fuera de línea basándose en

pronósticos de tráfico [Gupta 95].

- RTNR (Real Time Network Routing). Aquí por cada llamada en sobreflujo se hace una petición del estado del enlace al conmutador destino. El ruteo varía de acuerdo a la llamada y depende del estado del enlace, el ruteo es distribuido entre los conmutadores [Ash 92]. Ha estado en operación en la red doméstica de larga distancia de AT&T desde principios de los 90´s.
- DAR (Dynamic Alternative Routing). Aquí se mantiene una cierta ruta alterna de entre las que se encuentran en una lista mientras ésta tenga éxito. Si la ruta es bloqueada entonces se intenta con otra ruta de la lista tomada de manera aleatoria y se mantiene hasta que falla [Gibbens 95, Key 88]. Está implementada en la red doméstica British Telecom.
- IN/DR (Intelligent Network based Dynamic Routing). Este esquema se construye sobre las funcionalidad de una red inteligente lo cual reduce su costo, usando satélites para distribuir la información de ruteo en tiempo real y utilizando el patrón de los eventos bloqueantes para determinar la carga del enlace [Bella 98].

Ya se mencionó anteriormente que las redes telefónicas actuales han evolucionado hasta proporcionar servicios integrados de vídeo, voz y datos. Esto se ha logrado a través del desarrollo de *redes digitales de servicios integrados de banda ancha (B-ISDN* por sus siglas en inglés) para la cual se han desarrollado técnicas de conmutación tales como ATM (Modo de Transferencia Asíncrono), el cual es lo suficientemente versátil para integrar una amplia gama de fuentes de tráfico. Esta tecnología es también muy adecuada para las redes corporativas o ambientes LAN de alta velocidad. Debido a las nuevas características de esta tecnología el ruteo que en esta se utiliza es más complejo y se analiza en [Gupta 95].

4.2.2. Conmutación de paquetes

La conmutación de paquetes es una tecnología más reciente que la conmutación de circuitos, la cual fue concebida para proporcionar comunicaciones de datos eficientes entre computadoras "grandes" y usuarios de computadoras [Steenstrup 95]. Su evolución se ha debido a la necesidad de los usuarios de acceder a múltiples computadoras, las cuales han incrementado su disponibilidad y distribución gracias al incremento en las velocidades de procesamiento y capacidad de memoria de las mismas y la reducción en su costo. Otro factor importante que ha impactado en el rápido desarrollo de la conmutación de paquetes es el desarrollo de redes y protocolos para la misma por parte de compañías de tiempo compartido (TYMNET de Tymshare), de fabricantes de computadoras (SNA de IBM) y de la Agencia de Proyectos de Investigación Avanzada (ARPA), la cual desarrolló ARPANET.

Las redes de conmutación de paquetes varían en términos de técnicas de paso de paquetes, algoritmos de generación y selección de rutas así como en la descentralización y dinamismo del control de ruteo [Steenstrup 95].

Las técnicas de paso de paquetes incluyen:

- **Circuito virtual.** Se establece un "circuito virtual" que dura lo que la sesión de transmisión a través de todos los conmutadores en la ruta. Cada uno de estos dirige el paquete de datos a su destino de acuerdo al circuito virtual asociado.
- **Especificado por fuente.** Cada paquete lleva directivas de ruteo que se siguen por los conmutadores intermedios a través de la ruta.
- **Datagrama.** Permite a cada conmutador que cada paquete sea pasado de acuerdo a la preferencia del conmutador. Este es más flexible que los anteriores pero requiere que se tenga una consistencia entre los conmutadores o de otra

forma el paquete puede no llegar a su destino.

La mayoría de las redes de conmutación de paquetes operan en modo *store-and-forward*, independientemente de que técnica para pasar paquetes se use. En este modo cada paquete es recibido y almacenado en su totalidad antes de pasarlo al destino (o siguiente conmutador). Un modo alterno es el denominado *cut-through virtual* [Kermani 79], en el cual cada conmutador empieza a pasar el paquete al destino en cuanto se ha recibido lo suficiente del mismo para determinar a donde pasarlo.

Los algoritmos de generación y selección de rutas para este tipo de redes se pueden dividir en:

- **Algoritmos de ruteo óptimo**. Su objetivo es determinar rutas de tal forma que se minimice el costo de las mismas, a menudo resultan en el uso de múltiples rutas para el tráfico individual entre una fuente y un destino. Debido a que requiere de muchos recursos de cómputo ha sido usado principalmente para porciones estáticas de ruteo en el diseño de red.
- **Algoritmos de Ruta Mínima**. Su objetivo es determinar una ruta de mínimo costo (el cual se calcula de acuerdo a la métrica usada). La mayoría de las redes de conmutación de paquetes usan algoritmos de este tipo para generar dinámicamente y seleccionar rutas de acuerdo al usuario y al estado de la red. Estos algoritmos se dividen en dos clases:
  1. *Vector de distancia.* Estos algoritmos son usualmente implantados de manera asíncrona y distribuida y hacen la selección de la ruta basados en estimados locales. Hacen uso del algoritmo de Bellman-Ford [Bertsekas 92, Stallings 94].
  2. *Estado de enlace.* Estos algoritmos son implantados usualmente de manera replicada (cada conmutador realiza un cómputo de ruta independiente) y

construye rutas basándose en estimados globales de los costos de los enlaces. Hacen uso principalmente del algoritmo de Dijkstra para el cómputo de las rutas mínimas [Moy 95, Moy 98, Bertsekas 92].

Cada uno de estos algoritmos genera rutas de costo mínimo en tiempo polinomial. Sin embargo, el encontrar rutas que cumplan múltiples restricciones impuestas por el usuario (límites específicos para el retardo y tasa de error de los paquetes) es un problema NP [Steenstrup 95] aunque se pueden obtener soluciones factibles usando algoritmos de ruta mínima.

También se puede requerir en ciertas circunstancias de tener múltiples rutas para una sesión, por ejemplo para obtener una mayor confiabilidad o un mayor rendimiento. Para esto se han propuesto varios algoritmos los cuales se encuentran en [Suurballe 74, Topkis 88, Ogier 89, Torrieri 92].

Otro tipo de comunicación requerida en las aplicaciones distribuidas de hoy en día es la comunicación multipunto para lo cual se requiere de un tipo de ruteo de multitransmisión y algunas de las soluciones propuestas son RPF (Reverse Path Forwarding) [Dalal 78], el cual fue desarrollado originalmente para difundir eficientemente un paquete a todos los destinos, aunque hay versiones mejoradas que restringen la distribución de los paquetes [Deering 88]. Cabe mencionar que el problema de la construcción de árboles de distribución de multitransmisión, es decir, de generar un árbol que se expanda sobre un subconjunto de destinos (problema de Steiner) es NP completo.

El éxito de este tipo de redes es evidente por la existencia de muchas redes privadas y por la expansión de Internet como red global de comunicación de datos, aunque tal expansión ha hecho que se investiguen nuevas técnicas de ruteo que permitan la reducción de la cantidad de información de ruteo que se tiene que intercambiar de tal forma que no se sacrifique la calidad de las rutas seleccionadas.

4.2.3. Redes de alta velocidad

La demanda de los usuarios de tener integrada la comunicación de vídeo, datos y voz (multimedia) ha obligado a que se desarrollen nuevas tecnologías de red. Las redes de alta velocidad son una alternativa eficiente y altamente flexible pues ofrecen estos servicios a un gran número de usuarios en una red simple. El desarrollo de estas redes fue posible gracias a avances tecnológicos tales como la fibra óptica, servicios digitales y arquitecturas de conmutación basadas en redes de interconexión de conmutadores simples con mínimas capacidades de buffer [Hui 90].

Debido a que los proveedores de servicio ofrecen o redes de conmutación de circuitos o redes de conmutación de paquetes, las redes de alta velocidad han creado una sinergia entre ambas. Las técnicas de conmutación más promisorias para las redes de alta velocidad son híbridos de ambos tipos de conmutaciones; estos híbridos son capaces de eliminar la varianza de retardos y garantizar el ancho de banda y al mismo tiempo son capaces de usar los recursos eficientemente y acomodar tráfico de tasa variable. Ejemplos de estas técnicas son:

- **Conmutación de ráfaga.** Obtiene recursos de red para sesiones de tráfico sólo cuando los usuarios lo requieren [Amstutz 83].
- **Conmutación rápida de paquetes.** Reduce el retardo del manejo del paquete al usar paquetes cortos de longitud fija o variable en los cuales se lleva información de ruta diseñada para minimizar su tiempo de procesamiento [Turner 86].

En las redes de alta velocidad, los principales problemas de ruteo involucran la selección de rutas factibles así como el paso de tráfico que reduzcan la pérdida, para lo cual se requieren de modelos de tráfico de usuario y de carga de enlaces así como de algoritmos

de generación de rutas con múltiples objetivos y restricciones. Hay que tomar en cuenta también que los conmutadores para este tipo de redes deben enfrentar la contención interna por rutas lo cual puede resultar en un retardo o en pérdida de datos.

Un ejemplo de red de alta velocidad es el dado por las redes ópticas. Estas redes llevan los mensajes codificados como señales de luz a velocidades muy altas y con una confiabilidad superior [Bannister 95]. Abarcan desde sistemas que son simples actualizaciones de redes de líneas de transmisión eléctrica a fibra óptica a redes completamente ópticas en donde toda la transmisión y procesamiento son hechos en su totalidad en el dominio óptico.

Las redes de fibra óptica usan el multiplexado de división de longitud de onda (WDM) y ofrecen el potencial de construir grandes redes de área amplia capaces de soportar miles de nodos y proporcionar capacidades en el orden de los gigabits por segundo a cada uno de estos nodos [Green 92]. En WDM el vasto ancho de banda disponible es utilizado particionándolo en varios *canales*, cada uno en una *longitud de onda* óptica diferente. Cada uno de estos con la capacidad ya mencionada[Aggarwal 96]. Se requiere por lo tanto de conmutadores especiales denominados ruteadores de longitud de onda para manejar estos enlaces.

En general, las redes de este tipo se pueden clasificar de acuerdo a como se hace el ruteo en las siguientes categorías [Bannister 95]:

- **Redes de Enlace Optico.** En estas solo se reemplazan los enlaces por fibra óptica y el ruteo se logra de manera análoga a las redes electrónicas convencionales tales como Internet. Un ejemplo de estas redes es B-ISDN.
- **Redes Opticas de un Solo Salto.** Aquí se rutean los mensajes de la fuente al destino en un solo salto, es decir, no se hace ningún procesamiento en puntos intermedios (conmutadores electrónicos) aunque se puede hacer ruteo de

longitud de onda. El procedimiento de ruteo debe por lo tanto seleccionar una ruta física que conecte la fuente y el destino sin intervención de conmutadores electrónicos. Un ejemplo es la red LAMDANET.

- **Redes ópticas multisalto.** Permiten que el mensaje sea procesado en puntos intermedios entregándolo después de varias conversiones electro-ópticas. El ruteo en este tipo de redes se enfoca en el problema de determinar la mejor secuencia de conmutadores electrónicos por los cuales pasar el mensaje.
- **Redes Opticas Híbridas.** Combinan las técnicas de un solo salto y multisalto y consisten de subredes de ambos tipos que deben incorporar los procedimientos de ruteo de su respectivo tipo de red.
- **Redes Fotónicas.** Son redes completamente ópticas y pueden procesar la información fotónicamente.

En general, las redes ópticas generan nuevos problemas de ruteo debido a:

- Las altas tasas de velocidad de transferencia
- La introducción del WDM
- La intención de reducir los dispositivos electrónicos para evitar cuellos de botella.

4.2.4. Redes móviles

Una red móvil puede definirse como una red de comunicaciones en la cual al menos una de las entidades constituyentes (puntos extremos, conmutadores) cambian de localidad en relación con las otras [Steenstrup 95]. Durante tal movimiento, la comunicación puede continuar o puede ser suspendida, dependiendo de las capacidades de la red móvil y de la naturaleza de la comunicación.

Las redes inalámbricas móviles han tenido un dramático incremento en su popularidad durante los últimos años, aunque la tecnología ha existido por más de 20 años y ha estado disponible comercialmente por más de 10 [Ramanathan 96]. Esta reciente popularidad se puede atribuir a 2 factores:

1. Los avances en el diseño de hardware que han hecho que los dispositivos de cómputo y comunicaciones inalámbricas sean portables, de bajo consumo de energía y al alcance de todos.
2. El rápido crecimiento en la infraestructura de comunicación lo cual resulta en un fácil acceso a las redes telefónicas y de datos.

Los usuarios acostumbrados a los servicios disponibles en las redes estacionarias demandan lo mismo de las redes móviles inalámbricas lo cual es un gran reto ya que algunos servicios pueden no estar disponibles entre todos los pares de localidades resultando en la imposibilidad de proporcionar un tráfico consistente conforme los puntos extremos se mueven.

Idealmente, el sistema de ruteo en este tipo de redes debe ser capaz de manejar la movilidad de los nodos tal que dos puntos extremos en la comunicación no estén al tanto de tal movilidad. Los cambios en la red o estado de la sesión pueden necesitar cambios en las rutas existentes para poder mantener su factibilidad y es más probable que tales cambios ocurran con frecuencia en las redes móviles que en las redes estacionarias. El sistema de ruteo debe ser capaz de detectar rápidamente y responder a tales cambios de estado para poder minimizar la degradación de los servicios proporcionados a las sesiones de tráfico existente, y debe hacerlo usando un mínimo de recursos de la red para poder maximizar el rendimiento de ésta [Ramanathan 96].

En el proceso de rastrear y adaptarse a los cambios de localidad de un nodo el sistema de ruteo consume recursos de la red que están en proporción directa a la frecuencia

y velocidad del movimiento del nodo por lo cual para dar soporte a nodos altamente móviles requiere ser capaz de predecir las localidades futuras del nodo aparte de reaccionar a los movimientos actuales, para lo cual se requiere tener disponible información a priori de la trayectoria del nodo, de otro forma no sería posible reaccionar a cambios demasiado rápidos (por ejemplo sí el nodo está en un avión).

Ahora bien, una red consiste de nodos y enlaces; los nodos pueden ser puntos extremos (hosts o terminales) y conmutadores (o ruteadores). Dependiendo de sí estos son estacionarios o móviles se obtienen los siguientes 4 tipos de red:

- **Redes celulares.** En esta, los conmutadores son estacionarios (y se denominan *estaciones base*) mientras que los puntos extremos pueden ser móviles e inalámbricos. Los puntos extremos son particionados en *celdas*, cada una de las cuales tiene una estación base asociada y que es usada para conectarse a la porción fija de la red. Una organización celular permite el reuso de frecuencias entre celdas geográficamente distantes, incrementado la capacidad del sistema [Ketchum 95]. El uso de las redes celulares da lugar a 2 problemas:
    - La partición del área cubierta en celdas y la determinación del tamaño de la misma así como la ubicación de la estación base.
    - La asociación de las estaciones base con los centros de conmutación, denominados Centros de Conmutación Móvil.
- **Redes de radio paquetes.** Esta consiste de un número de nodos (fijos o móviles) que se comunican unos con otros, vía ondas de radio. En muchas de estas redes los radios de paquetes no tienen enlaces directos a los otros radios por lo cual se usa la técnica de *store-and-forward*. Este tipo de redes acelera la instalación de la red ya que no se requiere de enlaces físicos y los enlaces

radiales soportan de manera directa los nodos móviles aunque tiene como desventaja el uso compartido del medio y el alto dinamismo de la topología de red [Lauer 95]. Los algoritmos de ruteo usados en estas redes van de un solo salto a multisalto, de circuito virtual a datagrama y de casi estáticos a altamente adaptivos. Cabe mencionar que típicamente todos los nodos son conmutadores. Ejemplos de este tipo de red son DARPA PRNET y SURAN.

- **Redes satelitales.** Aquí, las funciones de conmutación son hechas por satélites. Los enlaces pueden ser tierra - satélite o intersatélite. Los puntos extremos son estacionarios. Dependiendo de la red, los usuarios no cubiertos por el mismo satélite pueden comunicarse a través de múltiples saltos tierra - satélite o pueden usar varios enlaces intersatélite. Aunque tienen topologías dinámicas no requieren de mecanismos adaptivos ya que los satélites se mueven en patrones determinísticos y por lo tanto completamente predecibles. El problema en estas redes es el diseño de la mejor distribución topológica, incluyendo el número de satélites, sus posiciones, órbitas, ángulos y áreas cubiertas así como la conectividad intersatélite.

4.3. Ruteo en las redes de conmutación de paquetes

Debido a que nos interesa determinar la confiabilidad de un sistema distribuido enfocándonos al subsistema de comunicaciones sobre el cual esta construido, que existen muchos sistemas distribuidos en Internet [Coulouris 95] y que tal interred esta en su mayor parte funcionando sobre redes de conmutación de paquetes [Steenstrup 95] analizaremos en esta sección los algoritmos de ruteo usados en este tipo de redes y que ya fueron mencionados de manera breve con anterioridad. Estos algoritmos de ruteo nos van a permitir desarrollar un modelo que arroje resultados más aproximados a la realidad.

Las redes individuales en Internet pueden ser unidas por una computadora que actúa como un conmutador entre las redes, tal computadora recibe el nombre de ruteador (aunque anteriormente también se le denominaba gateway), las operaciones de tal conmutador se programan para rutear el tráfico a la red apropiada al examinar la dirección destino en la Unidad de Datos del Protocolo y compararla con las entradas en su tabla de ruteo [Black 92]. Tales entradas indican la mejor ruta a la red destino. Internet esta organizada en regiones denominadas *Sistemas Autónomos* (dominio en la terminología del modelo OSI). Cada sistema autónomo consiste de una colección de ruteadores bajo el control de una entidad administrativa [Moy 98].

La colección de sistemas autónomos está organizada de manera jerárquica. Entre más cerca se encuentre al tope de la jerarquía, más rutas aparecen en el sistema autónomo. Los que están en el tope tienen la tabla completa de rutas, actualmente 45,000.

Ahora bien, debido a esta división en sistemas autónomos se usan dos formas de ruteo (protocolos):

- Ruteo interior (intradominio en OSI). Usado para el ruteo dentro de un sistema autónomo.
- Ruteo exterior (interdominio en OSI). Usado para el ruteo entre distintos sistemas autónomos.

Los algoritmos de ruteo interior más populares son los de vector de distancia y de estado de enlace [Lynch 93, Black 92, Moy 98, Scott 95, Huitema 95], los cuales son dinámicos en el sentido que cambian las tablas de ruteo con base en los cambios que se den en la topología de la red. Estos dos tipos de algoritmos podrían usarse también para el ruteo interdominio, pero debido a que los dominios (sistemas autónomos) pudieran ser de distintos países (hablando en un contexto geográfico) o distintas empresas (hablando en un contexto comercial) habrá información que no desee que se pase de un dominio a otro

debido entre otras cosas a restricciones legales; podemos citar como ejemplo el algoritmo de encriptación usado por el programa Netscape, el cual puede ser de 128 o de 40 bits, más sin embargo, el uso de la versión de 128 bits está restringido a Estados Unidos y Canadá.

4.3.1. Algoritmos de vector de distancia

Operan haciendo que cada ruteador mantenga una tabla que da la mejor métrica (las cuales ya fueron mencionadas) conocida a cada destino y la vía a usar para llegar ahí. Estas tablas se actualizan intercambiando información con los vecinos [Tanenbaum 97, Hedrick 88]. Este intercambio de información se realiza cada T segundos (cada ruteador le envía su tabla a sus vecinos y recibe la tabla de cada uno de ellos). Supongamos que el ruteador X sabe que la distancia al nodo i es m, y recibe de su vecino Y el valor Y_i (el cual denota la distancia de Y al nodo i), entonces el ruteador X sabe que si pasa por Y, la distancia a i sería $Y_i + 1$, y determinaría cual de los dos valores es el mejor y es el que guardaría en su tabla. Ejemplifiquemos esto usando la colección de ruteadores que se muestra la figura 4.1 suponiendo que la métrica es el retardo cuyo valor aparece en cada arista:

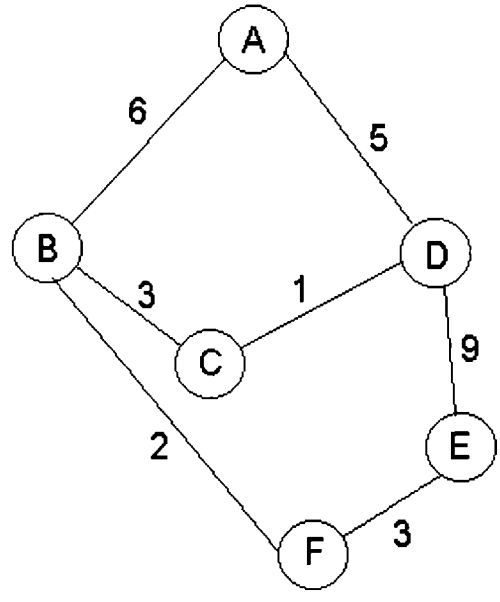

Figura 4.1 Colección de ruteadores de una red

Supongamos que las tablas de ruteo de A, B y D, en las cuales se indica el retardo que conoce cada uno de ellos a los demás así como la vía que hay que tomar son las siguientes en un instante de tiempo T_1:

Tabla de A

Ruteador	Retardo	Vía
A	0	-
B	6	B
C	9	B
D	5	D
E	14	D
F	17	D

Tabla de B

Ruteador	Retardo	Vía
A	6	A
B	0	-
C	3	C
D	4	C
E	5	F
F	2	F

Tabla de D

Ruteador	Retardo	Vía
A	5	A
B	4	C
C	1	C
D	0	-
E	9	E
F	12	E

Entonces, de acuerdo al algoritmo, después de T segundos, cada ruteador intercambiará información con sus vecinos, así pues, A pasará su tabla a B y a D, y recibirá las tablas de B y D (sucederá de manera similar con los restantes ruteadores). Después de este intercambio A comparará el valor que tiene para cada una de las entradas con el valor

de cada una de las entradas de las tablas que recibió y determinará cual es el mejor valor, así pues, por ejemplo, determinará que para alcanzar a F lo mejor es pasar a través de B, el cual le ofrece un retardo de 2 segundos para llegar a B, más el retardo que le toma a A llegar a B, que es 6, le da un total de 8, a diferencia del valor 17 que es el que tenía almacenado en el tiempo T_1 y el que se obtendría al pasar por D, que es también 17, por lo tanto la entrada en la tabla para F sería modificada con el nuevo valor de 8. La nueva tabla quedaría de la siguiente manera en el tiempo $T_1 + T$:

Tabla de A

Ruteador	**Retardo**	**Vía**
A	0	-
B	6	B
C	6	D
D	5	D
E	11	B
F	8	B

Hay que hacer notar que si después del intervalo de T algún ruteador no recibe mensaje de algún vecino entonces supondría que ha fallado (el enlace o ruteador) y en su tabla pondría el valor infinito para indicar que no es posible llegar (el valor es en realidad un valor grande seleccionado de acuerdo a la implantación) así como todos los destinos para los cuales tenía que atravesarlo, posteriormente se compararía con las tablas de los demás vecinos para ver si alguno tiene algún valor menor para los destinos. Por ejemplo, si después de que se estableció la anterior tabla en A, B o el enlace de B a A fallara entonces al momento en que se haga el siguiente intercambio de información a A solamente le llegaría la tabla de D, por lo cual, los destinos que pasan por B se consideran con un valor infinito, pero como los valores para esos mismos destinos son menores en la tabla de D, A quedaría con la siguiente tabla:

Ruteador	Retardo	Vía
A	0	-
B	9	D
C	6	D
D	5	D
E	14	D
F	17	D

El algoritmo de vector de distancia también recibe el nombre de algoritmo de Bellman-Ford distribuido, por los investigadores que lo desarrollaron. Una explicación detallada de tal algoritmo se puede encontrar en [Bertsekas 92, Stallings 94, Cormen 90, Scott 95]. Ejemplos de protocolos que usan este algoritmo son RIP, IGRP o RTMP.

Este tipo de algoritmo aunque fácil de implementar, tiene un problema denominado de conteo a infinito el cual hace que su convergencia sea lenta, es decir, tarda en actualizar sus tablas para reflejar que un enlace o nodo ha dejado de funcionar [Black 92, Tanenbaum 97, Comer 95]. Ejemplifiquemos esto con el siguiente grafo:

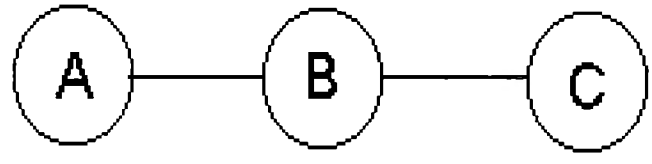

Figura 4.2 Ejemplo de conteo a infinito

Supongamos aquí que la métrica usada es la distancia, por lo tanto la distancia de A a C es 2, de B a C es 1, de B a A es 1, etc. Supongamos que C falla, entonces B no recibiría ya información de C, por lo cual le daría un valor de infinito (digamos 16), pero recibiría de A una tabla donde se especifica una distancia de 2 a C, por lo cual B decidiría llegar a C a través de A, guardando en su tabla una distancia de 3 hacia C (la distancia que ofrece A más la distancia de B a A). Después de T segundos cuando debe haber otro intercambio, A recibe de B el nuevo valor que tiene hacia C, que es 3, y debido a que cuando se envía un

mensaje a C debe pasar por B considera con autoridad tal valor y almacena en su tabla 4 como la distancia de A hasta C. Esto se da repetitivamente hasta que se logre llegar al valor 16 y se den cuenta tanto A como B que C no se puede alcanzar. Esta operación se dio repetitivamente pues se formó un ciclo de ruteo.

Para poder mejorar la convergencia de este tipo de algoritmo, se le han hecho varias modificaciones, las cuales son las siguientes:

- Para reducir el tiempo de convergencia, muchos ruteadores envían actualizaciones cuando sus tablas de ruteo cambian en vez de esperar el tiempo preestablecido (30 segundos en el caso de RIP). Estas actualizaciones se denominan *actualizaciones disparadas*.
- Para evitar que se formen ciclos durante la convergencia se usa un procedimiento denominado *división de horizonte* [Scott 95, Moy 98, Huitema 95]. En este procedimiento, cuando un ruteador envía una actualización por su interfaz X, omite todas las rutas en la actualización cuya interfaz de salida sea X. En el ejemplo anterior, el ruteador A no enviaría la ruta que él tiene a C al ruteador B pues la vía que debe tomar es B evitándose el problema del ciclo.
- Una modificación al procedimiento anterior, denominada *división de horizonte infinita*, hace que cuando un ruteador envía una actualización a través de su interfaz X anuncie las rutas que tiene por vía de X como inalcanzables (esto es, con un valor infinito). Este procedimiento es más efectivo pues causa que se rompa el ciclo entre 2 ruteadores inmediatamente mientras que en el anterior las rutas ya inalcanzables se eliminan hasta que el ruteador las considera inválidas por el tiempo transcurrido sin una actualización.

Estas modificaciones pueden mejorar el tiempo de convergencia pero no pueden eliminar por completo la formación de ciclos de ruteo durante la convergencia. Para ello existen otros algoritmos que requieren de más intercambios de mensajes de control, más información en los mensajes de actualización o de cómputos adicionales [Scott 95]. Uno de esos algoritmos es el LPA (Loop-free path finding algorithm), el cual es descrito en [García 97] aunque no se ha implantado en la práctica.

4.3.2. Algoritmo de estado de enlaces

En el contexto de estado de enlaces, una red se convierte en un grafo de nodos y enlaces interconectados. Cada nodo forma una vista local de la red en términos de las propiedades de los enlaces que conectan a dos nodos vecinos, de aquí el término "estado de enlace". En la forma más común de ruteo de estado de enlace cada nodo actúa autónomamente, distribuyendo su información de estado de enlace a todos los otros nodos y calculando las rutas más cortas de sí mismo a todos los destinos basado en la información de estado de enlace obtenido de otros nodos [Moy 95].

Este tipo de algoritmo ha reemplazado al anterior en redes prácticas tal como es Internet debido a las siguientes ventajas [Huitema 95]:

- **Convergencia más rápida y libre de ciclos**. Esto se debe a que a diferencia de los protocolos de vector de distancia en los cuales el número de pasos a realizar es proporcional al número de nodos en la red que en el peor de los casos es el número de nodos menos 1, los algoritmos de estado de enlace solo consisten de 2 pasos:
    - Una rápida transmisión de nueva información a través de un algoritmo de inundación.
    - Un cómputo local

El algoritmo de inundación se usa para distribuir la información de ruteo a través del dominio, cuando un ruteador recibe un mensaje de actualización por una de sus interfaces la envía a través de las restantes interfaces.

El cómputo local se refiere al cálculo de las mejoras rutas para el tráfico de red usando la información que recibió en el mensaje de actualización. El método más popular para hacer este cálculo es el algoritmo de Dijkstra [Bertsekas 92, Stallings 94, Aho 74, Cormen 90, Grimaldi 89, Huitema 95, Moy 95, Moy 98], también llamado de primer camino más corto, por lo cual los algoritmos de estado de enlace son a menudo llamados algoritmos de primer camino más corto.

Inmediatamente después de esta inundación y el cómputo local, todas las rutas en la red están sanas - sin ciclos intermedios ni conteo al infinito.

- **Soporte de múltiples métricas.** Debido a que el cómputo de las rutas más cortas es ejecutado con un completo conocimiento de la topología se pueden usar métricas arbitrarias y precisas sin ralentizar la convergencia. De hecho, la precisión del cómputo hace posible el soporte de varias métricas en paralelo, cosa que para los algoritmos de vector de distancia es bastante complicado. Esto es muy eficiente debido a la diversidad de tipos de enlace que existen (DS0 a 64 Kbps, T1 a 1.544 Mbps, E1 a 2.048 Mbps entre otras), y se puede ver claramente que es mejor dar 5 saltos en líneas E1 que dos saltos en líneas DS0 (que es lo que haría el algoritmo de vector de distancia usando como métrica el número de saltos).
- **Soporte de múltiples rutas a un destino.** El algoritmo de Dijkstra requiere sólo de una pequeña extensión para poder calcular rutas alternas la cual se encuentra en [Huitema 95]. Tal extensión permite determinar todas las rutas

secundarias a un destino dado.

- **Rutas externas.** La red se conecta generalmente a través de uno o varios "ruteadores externos" a una o varias "redes de tránsito" que alcanzan a millones de hosts en el mundo. Cuando solo existe uno de esos ruteadores el asunto es simple pues basta con anunciar una "ruta por default" a ese ruteador, ambos tipos de algoritmos soportan esto. Pero cuando se tienen varios, el problema se agudiza para los algoritmos de vector de distancia pues debe agregar una entrada por cada uno lo cual lo hace más lento aparte de estar limitado en la distancia (hay que recordar que se maneja un numero como valor infinito y por lo tanto si sobrepasa de este valor se consideraría como inalcanzable). En cambio, en los algoritmos de estado de enlace no se da el problema del valor infinito y las rutas de este tipo se almacenan de manera separada pues son marcados de manera especial lo que hace que su cálculo sea más eficiente.

Ejemplos de protocolos que usan este algoritmo son Phase V de DECnet y OSPF [Moy 97, Moy 98, Huitema 95].

Los componentes de un algoritmo de estado de enlace son: la descripción de la red, un algoritmo de inundación confiable y el cálculo de las rutas.

4.3.2.1. Descripción de la red (AEE)

Los algoritmos de estado de enlace se basan en el concepto de "mapa distribuido": todos los nodos que participan en el algoritmo (denominados conmutadores de manera general) tienen una copia del mapa de la red, el cual es actualizado de manera regular [Huitema 95]. Este mapa o descripción de la red se denomina base de datos de estado de enlace y las partes que lo constituyen se denominan anuncios de estado de enlace (AEE). Cada conmutador es responsable de originar un AEE que describa el conjunto de interfaces

que tenga. Este AEE es entonces distribuido a todos los demás conmutadores en la red utilizando un algoritmo de *inundación* (flooding). Cuando algo en la red cambia (como por ejemplo el fallo de una línea) se debe crear un nuevo AEE e inundarse a todos los demás conmutadores.

El AEE de un conmutador contiene los siguientes datos [Moy 95]:

- La identidad del conmutador
- Una lista de los enlaces o interfaces activas
- El costo que cada enlace tiene para pasar información
- La identidad del conmutador al que cada enlace conecta

Así, para la red representada en la figura 4.1 se muestran en la figura 4.3 los AEE que conforman la base de datos de estado de enlace:

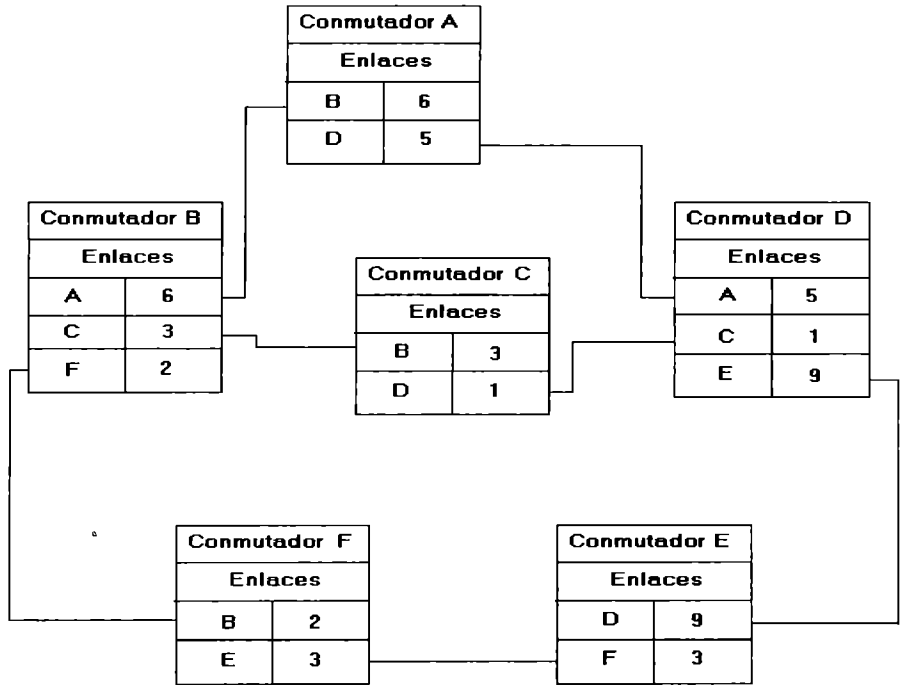

Figura 4.3 Anuncios de estado de enlace

Otros datos que puede tener un AEE son:

- La identificación de las aplicaciones a las que el conmutador da servicio

- Número de secuencia. Para determinar el AEE más actual
- Edad del AEE. Para determinar cuando se debe descartar
- Suma de chequeo del contenido del AEE. Para prevenir corrupciones de datos tanto durante la inundación como mientras se mantiene en la base de datos del conmutador.

4.3.2.2. Algoritmo de inundación

Un buen algoritmo de inundación es crucial para un algoritmo de estado de enlace [Moy 95]. El algoritmo de estado de enlace requiere que los conmutadores tengan bases de datos idénticas para asegurar que el ruteo es correcto y libre de ciclos. Durante la inundación, no todos los conmutadores tendrán bases de datos idénticas pero un algoritmo de inundación confiable asegura que las bases de datos se sincronizarán después de un período de convergencia, el cual es pequeño si el algoritmo es rápido. La confiabilidad de estos algoritmos se obtiene a través de reconocimientos explícitos de todos los AEE enviados de un conmutador a los conmutadores vecinos. La velocidad se obtiene al darle precedencia a la inundación sobre el tráfico de datos normal. De hecho, el algoritmo de inundación es la parte más complicada de un algoritmo de estado de enlace.

Antes de que el algoritmo de inundación inicie, un conmutador requiere de una copia inicial de la base de estados de enlace. Este intercambio sucede cuando se enciende el conmutador o cuando se activa un nuevo enlace. En ese momento el conmutador descubre sus nuevos vecinos y estos bajan una copia de la base de datos al primero. Después de esta sincronización inicial, la inundación se da por 2 razones:

- Un cambio en la red.
- Una actualización periódica para hacer robusto el algoritmo (capaz de detectar errores inesperados tales como fallas en el conmutador).

En resumen, el algoritmo trabaja como sigue [Huitema 95]:

1. El conmutador recibe un AEE por una de sus interfaces o enlaces.
2. El conmutador examina el número de secuencia del AEE y lo compara con el número de secuencia de la copia actual (en caso de que exista) del AEE en la base local, la cual determina si el AEE recibido es más nuevo.
3. Si el AEE es nuevo, se instala en la base de datos local reemplazando la copia actual (o se agrega en caso de no existir) y se envía a todos los enlaces restantes que tenga el conmutador enviando un reconocimiento por el enlace que llegó.
4. Si el AEE local es más nuevo entonces se transmite este AEE local a través del enlace por el cual llegó la inundación.
5. Si los AEE local y recibido tienen un número de secuencia igual entonces solo se envía un reconocimiento del mismo por el enlace que llegó.

Este algoritmo se hace confiable al continuar retransmitiendo el AEE hasta que el otro conmutador envía un reconocimiento. Para optimizar el caso en que ambos extremos del enlace transmitan el AEE simultáneamente, el recibir un AEE se contempla como un reconocimiento implícito.

Como ejemplo de inundación, supongamos que el conmutador A de la figura 4.3 necesita actualizar su AEE y asumiendo que la mayor parte del tiempo transcurrido por la inundación se debe a los tiempos de transmisión por los enlaces, la inundación puede suceder en las siguientes 4 fases:

1. El conmutador A inunda su AEE a los conmutadores B y D.
2. Los conmutadores B y D aceptan el AEE como nuevo, lo instalan en su base de datos local y envían su reconocimiento a A. B entonces inunda el AEE a los conmutadores C y F, mientras que D inunda el AEE a los conmutadores C y E.

3. Suponiendo que el primer AEE lo recibe C de B, entonces este envía el reconocimiento a B y lo inunda a D, quién al recibirlo lo toma como reconocimiento implícito del que él envió y C al recibir el AEE de D lo toma de la misma manera. F recibe el AEE de B y le envía el reconocimiento inundándolo después a E, E recibe el AEE de D, le envía el reconocimiento respectivo y lo inunda a F.
4. Los AEE enviados por F y E se toman como reconocimiento implícito de la contraparte.

Se pueden observar dos hechos importantes de este ejemplo:

- Primero, durante la inundación un nuevo AEE transita cada enlace por lo menos una vez y algunas veces dos (en los reconocimientos implícitos).
- Segundo, la inundación no depende del ruteo, por lo cual evita dependencias circulares.

Algunas versiones del algoritmo de estado de enlace usan la inundación también para remover el AEE de la base de datos de enlaces.

4.3.2.3. Cálculo de rutas

Teniendo como base la base de datos de estado de enlace, los conmutadores calculan las rutas a través de la red para el envío de información. Existen muchos algoritmos para lograr esto pero el más popular como ya se mencionó es el de Dijkstra [Bertsekas 92] debido a su simpleza y eficiencia. Este algoritmo requiere que cada enlace tenga asignado un costo no negativo y produce un conjunto de rutas más cortas del conmutador que hace el cálculo a los restantes en la red. El algoritmo separa los conmutadores en 2 conjuntos: uno consiste de los conmutadores cuya ruta más corta se ha encontrado (S) y el otro consiste de los conmutadores para los cuales se han encontrado

rutas tentativas (T). El algoritmo empieza poniendo el conmutador que está haciendo el cálculo en el conjunto de rutas más cortas e iniciando la lista de rutas tentativas vacía. El algoritmo entra entonces en un ciclo en el cual en cada iteración el conmutador (X) que se acaba de agregar al conjunto S se examina y se realizan los siguientes pasos:

1. Los vecinos del conmutador X que no pertenecen al conjunto S son agregados al conjunto T (si es que no están) o se actualizan si la ruta a través de X es más corta que la anterior.
2. Si T está vacío, el algoritmo termina.
3. El conmutador en el conjunto T que esté más cercano al conmutador que hace el cálculo se mueve al conjunto S y el algoritmo itera otra vez.

Para ejemplificar este proceso veamos el proceso que el conmutador A del grafo representado en la figura 4.3 realiza para obtener su tabla de ruteo (los elementos que aparecerán en el conjunto T serán triplas que definen el conmutador destino, el conmutador por el cual se coloca o se actualiza y el costo):

Estado inicial.

S={(A,-,0)}, T={}

Iteración 1.

T={(D,D,5),(B,B,6)} y se mueve D a S puesto que tiene un costo menor quedando T={(B,B,6)}.

Iteración 2.

T={(B,B,6),(C,D,6),(E,D,14)}, se mueve B a S (también pudo haberse movido C) quedando T={(C,D,6),(E,D,14)}

Iteración 3.

T={(C,D,6),(F,B,8),(E,D,14)}, se mueve C a S quedando T={(F,B,8),(E,D,14)}.

Iteración 4.

T={(F,B,8),(E,D,14)}, se mueve F a S quedando T={(E,14)}

Iteración 5.

T={(E,B,11)}, se mueve E a S quedando T={} por lo que el algoritmo termina.

Al final en S queda por lo tanto la tabla de ruteo de la siguiente manera:

Destino	Siguiente Salto	Costo
A	*	*
B	B	6
C	D	6
D	D	5
E	B	11
F	B	8

El rendimiento de este algoritmo es del orden O((n+e)log(n)) donde e es el número de enlaces y n el número de conmutadores (nodos). El término log(n) caracteriza la complejidad de encontrar la ruta más corta en el conjunto de rutas tentativas cuando se usa una estructura de heap para representarla. A menudo se asume que conforme el número de conmutadores crece, el número de interfaces por conmutador permanece constante y de ser así, el comportamiento asintótico del algoritmo de Dijkstra puede ser dado por O(e log(n)). Visto de otra manera, una implantación del algoritmo de Dijkstra que corre en un procesador a 10 MIPS toma 15 milisegundos para calcular las rutas para una red de 200 conmutadores [Moy 91].

Debido a las ventajas que dan los algoritmos de estado de enlace sobre los de vector de distancia usaremos este tipo de algoritmos en nuestro modelo.

CAPITULO 5

MODELO DE CONFIABILIDAD DE RED

5.1. Introducción

En este capítulo analizaremos la implantación de un modelo para el análisis de la confiabilidad de red tomando como base el modelo propuesto en [Valdivia 89] y extendiéndolo haciendo uso del algoritmo de ruteo de estado de enlace analizado en el capítulo anterior. Ahora bien, de acuerdo a lo que discutimos en la sección 3.3, existen varios problemas relacionados con el análisis de la confiabilidad de redes (dos terminales, k terminales, todas las terminales); en este modelo nos limitaremos al cálculo de la confiabilidad para 2 terminales, es decir, la obtención de la probabilidad de que exista una ruta operativa entre 2 nodos dados asumiendo además que los enlaces son bidireccionales (las redes por lo tanto representadas con grafos no dirigidos).

El análisis se hará tanto de manera determinista (estructural) así como probabilísticamente usando como parámetro principal la conectividad al momento de calcular la confiabilidad.

5.2. Modelado determinista

En este modelo la confiabilidad depende de la distancia, grado y el número de caminos disjuntos entre los nodos que representan al sistema [Valdivia 89], por lo cual, la implantación consiste del cálculo de los siguientes parámetros: densidad, grado, distancia y conectividad así como el análisis de la variación de estos parámetros al degradar el grafo que representa la red con ayuda de simulación de fallas en uno o más nodos y/o enlaces.

5.2.1. Densidad

La densidad se obtiene de manera simple al dividir el número de enlaces por el número de nodos en el grafo.

$$densidad = \frac{e}{n} \qquad \ldots (5.1)$$

5.2.2. Grado

El grado de un nodo para un grafo no dirigido es el número de vecinos que tiene y se calcula con el siguiente algoritmo:

```
Algoritmo ObtenGrado;
  For ∀ Vi ∈ V(G) do
    Grado[Vi] := 0;
    For ∀ Vj ∈ Adj[Vi] do
      grado[Vi] := grado[Vi] + 1;
    Endfor;
  Endfor;
  Se obtiene el grado máximo, mínimo y promedio
End;
```

Algoritmo 5.1 Obtención del grado de cada vértice

5.2.3. Distancia

Para obtener la distancia haremos uso de los resultados obtenidos por el algoritmo de ruteo, el cual determina cual es la distancia más corta entre dos nodos. Asumiremos pues la existencia de un algoritmo que dada como entrada una fuente y un destino devuelva con base en la tabla de ruteo T que tiene el nodo fuente la distancia que tiene al nodo destino. Recordemos que cada entrada de la tabla de ruteo es una tripla (vértice destino, vértice por el cual sale y distancia), para lo cual se usa la siguiente definición:

```
Entrada=record
        destino:integer;
        salida:integer;
        distancia:real;
end;
```

```
Algoritmo ObtenDistancia(fuente, destino);
  If ∃ t | t∈ T[fuente] ∧ t.destino=destino
    Then distancia := t.distancia
    Else distancia := ∞
  Endif
End;
```

Algoritmo 5.2 Obtención de la distancia de la fuente al destino

El algoritmo usado en el ruteo de estado de enlace (Dijkstra), el cual es ejecutado por cada nodo y basado en la distancia como métrica se muestra a continuación:

```
Algoritmo RuteoEE;
 (1)  S := { V0 }      // V0 es el nodo que está haciendo el cálculo (fuente)
 (2)  d[V0] := 0
 (3)  vía[V0] := V0
 (4)  RT := {}   // Conjunto de rutas tentativas
 (5)  For ∀ Vi ∈ Adj[V0] do
 (6)    RT := RT ∪ {Vi}
 (7)    d[Vi] := distancia(V0,Vi)
 (8)    vía[Vi] := Vi
 (9)  Endfor
(10)  While |RT| >0 do
(11)    Extraer Vj de T tal que d[Vj] sea la mínima
(12)    S := S ∪ { Vj }
(13)    For ∀ Vi ∈ Adj[Vj] | Vi ∉ S do
(14)      If Vi ∉ RT
(15)        Then RT = RT ∪ {Vi}
(16)             d[Vi] := d[Vj]+distancia(Vj, Vi)
(17)             vía[Vi] := vía[Vj]
(18)        Else If d[Vi]>d[Vj]+distancia(Vj, Vi)
(19)             Then d[Vi] := d[Vj]+distancia(Vj, Vi)
(20)                  vía[Vi] := vía[Vj]
(15)    Endfor
(16)  EndWhile
End;
```

Algoritmo 5.3 Obtención de las rutas mínimas de un vértice V_0 a los demás

En este algoritmo el conjunto S contiene todos los vértices o nodos para los cuales ya se encontró la ruta más corta desde V_0, el conjunto T contiene los vértices para los cuales existen rutas tentativas pues ya fueron visitados pero aún no se determina si es la de

mínima distancia, Adj[V_i] indica los vecinos de V_i, d[V_i] contiene la distancia desde V_0 hasta V_i que se encuentra almacenada en la tabla de ruteo T del vértice V_0, vía[V_i] contiene el vértice hacia el cual sale desde V_0 para llegar a V_i y distancia(V_j,V_i) es la distancia que se tiene del vértice V_j al vértice V_i.

5.2.4. Conectividad de Enlaces

La conectividad de enlaces (K_e), es el mínimo número de enlaces cuya eliminación hace que el grafo quede desconectado [Valdivia 89, Gabow 91]. Para poder obtener este parámetro nos basamos en 2 teoremas; el Teorema de Menger, el cual dice que el número máximo de rutas con enlaces disjuntos en un grafo G que unen a un vértice *s* y un vértice *t* es igual al número mínimo de enlaces de E(G) cuya eliminación separa a *s* y *t* [Diestel 97]; y el Teorema del Max-Flow-Min-Cut (Flujo Máximo, Corte Mínimo), el cual nos dice que el peso máximo de todos los flujos de un grafo dirigido D de un vértice *s* a un vértice *t* es igual a la capacidad mínima de un corte (conjunto de enlaces de E(D) cuya eliminación destruye todas las rutas dirigidas de *s* a *t*) [Sedgewick 88, Cormen 90]. Aprovechando esto, se puede utilizar un algoritmo para la obtención del flujo máximo, dándole una capacidad de 1 a cada enlace, por lo cual, el flujo máximo nos daría la cantidad de enlaces que conforman al corte, es decir, el valor de K_e. Uno de los métodos más conocidos para obtener el flujo máximo es el de Ford-Fulkerson [Sedgewick 88, Cormen 90, Allen 93], el cual es iterativo y consta de los siguientes pasos:

- Se da un flujo inicial de 0 a todos los enlaces que pertenecen a E(G)
- En cada iteración se incrementa el flujo al encontrar una "ruta de aumento", la cual se puede ver como una ruta simple por la cual se puede meter mas flujo del vértice fuente *s* al vértice destino *t*.
- Este proceso se repite hasta que no se pueda encontrar una ruta de aumento.

Para poder lograr esto, se requiere de una red residual representada por grafo G^f, el cual contiene todos los enlaces que admiten un incremento en el flujo, este flujo adicional se denomina capacidad residual del enlace y esta dado por:

$\Delta(u,v)$ = capacidad(u,v) - flujo(u,v)

Entonces, dado un grafo G=(V,E) y un flujo *f*, la red residual de G inducida por *f* es $G^f = (V, E^f)$ donde $E^f = \{ (u,v) \in E \mid \Delta(u,v) > 0\}$.

Ahora bien, el método de Ford-Fulkerson selecciona la ruta de aumento de manera arbitraria, una mejora es seleccionar esta ruta de aumento como una ruta mínima entre *s* y *t*, lo cual se hace en el algoritmo de Edmonds-Karp [Valdivia 89, Cormen 90] mediante el algoritmo de búsqueda en anchura, la cual supone una distancia de 1 entre cada par de vértices. Nosotros podemos hacer uso de las tablas de ruteo generadas por el algoritmo de ruteo de estado de enlace las cuales tienen exactamente eso, la ruta más corta de una fuente a un destino.

Se puede ver entonces que requerimos de 3 algoritmos para obtener la conectividad de enlaces:

1. Un algoritmo de flujo máximo
2. Un algoritmo para obtener la red residual (G^f)
3. Un algoritmo para obtener la ruta de aumento

Estos algoritmos se muestran a continuación:

```
Algoritmo FlujoMaximo(G,fuente,destino);
For ∀ (Vi,Vj) ∈ E(G) do
    flujo(Vi,Vj) := 0
    flujo(Vj,Vi) := 0
Endfor;
ConstruyeGF(G,GF);
RutaAumento(fuente,destino,GF,Ruta,P);
While Ruta do
    For ∀ (Vi,Vj) ∈ P do
      flujo(Vi,Vj) := flujo(Vi,Vj) + 1
```

```
        flujo(Vj,Vi) := - flujo(Vi,Vj)
        E(GF) := E(GF) - (Vi,Vj)
    EndFor;
    RutaAumento(fuente,destino,GF,Ruta,P)
EndWhile;
FlujoMaximo:=Σ flujo(fuente, Vj), para ∀ Vj ∈ Adj[fuente]
End;
```

Algoritmo 5.4 Obtención del flujo máximo

```
Algoritmo ConstruyeGF(G,GF);
V(GF) := V(G);
E(GF) := {};
For ∀ (Vi,Vj) ∈ E(G) do
    Δ(Vi,Vj) := capacidad(Vj,Vi) - flujo(Vj,Vi)
Endfor;
If Δ(Vi,Vj) > 0 then
    E(GF) := E(GF) ∪ (Vi,Vj)
End;
```

Algoritmo 5.5 Construcción de la red residual inicial

```
Algoritmo RutaAumento(fuente,destino,G,Ruta,P);
P := {}
Se ejecuta el algoritmo de ruteo en los vértices del grafo G
If ObtenDistancia(fuente,destino) = ∞ then
   Ruta := false
Else
   Ruta:=true;
   Vi := fuente;
   While Vi ≠ destino do
      Se obtiene vía[destino] de la tabla T de Vi y se asigna a Vj
      P := P ∪ {(Vi, Vj)}
      Vi := Vj
   EndWhile;
End;
```

Algoritmo 5.6 Obtención de la ruta de aumento

```
Algoritmo ConectividadEnlaces(fuente,destino);
For ∀ (Vi,Vj) ∈ E do
    capacidad(Vi,Vj) := 1
    capacidad(Vj,Vi) := 1
Endfor;
Ke := FlujoMaximo(G,fuente,destino);
End;
```

Algoritmo 5.7 Obtención de la conectividad de enlaces

5.2.5. Conectividad de Nodos

La conectividad de nodos (K_n) es el número mínimo de nodos (vértices) cuya eliminación deja al grafo desconectado [Valdivia 89]. De acuerdo al Teorema de Menger, dados dos vértices no adyacentes en un grafo G, *s* y *t*, el número máximo de rutas disjuntas en cuanto a los vértices que se tienen entre *s* y *t* es igual al número mínimo de vértices de V(G) - {s,t} cuya eliminación separa a *s* de *t*. Debido a esto, para obtener K_n se puede utilizar un procedimiento similar al que se uso para obtener K_e pero modificando al grafo de tal forma que se tome en cuenta ahora que un vértice o nodo no pueda estar en más de una ruta de *s* a *t*. El grafo modificado G' se construye siguiendo los pasos descritos en [Valdivia 89] modificado para aprovechar el trabajo realizado por el algoritmo de ruteo:

Para cada vértice $v \in V(G)$, G' contiene 2 enlace v' y v'' y un enlace (v',v'') denominado *enlace interno*. Además por cada $(V_i, V_j) \in E(G)$, G' contiene 2 enlaces (V_i'',V_j') y (V_j'',V_i') denominados *enlaces externos*. La capacidad de cada enlace interno y externo es de 1. Una vez construido G' se obtiene el flujo máximo del vértice fuente *s*'' al vértice terminal *t*'. En la figura 5.1 se muestra los grafos G y G' modificado.

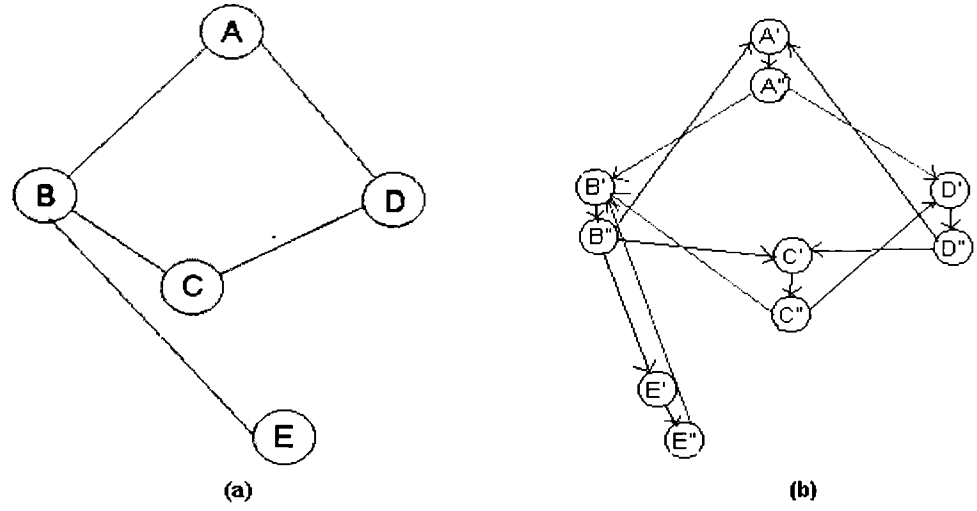

Figura 5.1 (a)Grafo G, (b)Grafo G' modificado para obtener Kn

A continuación se muestran los algoritmos necesarios:

```
Algoritmo ConstruyeG'(G,G',fuente,destino);
V(G') := {};
E(G') := {};
For ∀ Vi ∈ V(G) do
    V(G') := V(G') ∪ {Vi', Vi"};
    E(G') := E(G') ∪ {(Vi', Vi")};
    capacidad(Vi', Vi") := 1
Endfor;
For ∀ (Vi,Vj) ∈ E(G) do
    E(G') := E(G') ∪ {(Vi",Vj'), (Vj",Vi')};
    capacidad(Vi",Vj') := 1;
    capacidad(Vj",Vi') := 1
Endfor;
End;
```

Algoritmo 5.8 Construcción del grafo auxiliar G'

```
Algoritmo ConectividadNodos(fuente,destino);
ConstruyeG'(G,G',fuente,destino);
Kn := FlujoMaximo(G',fuente,destino);
End;
```

Algoritmo 5.9 Obtención de la conectividad de nodos

5.2.6. Simulación de Fallas

Siguiendo el modelo determinista propuesto en [Valdivia 89] se simula en el modelo la falla de vértices o enlaces con el propósito de observar el comportamiento del grafo degradado en cuanto a los parámetros determinísticos mencionados (densidad, grado, distancia, conectividad de enlaces y nodos). La simulación se hace de tal manera que la conectividad de enlace o de nodo se vaya decrementando en uno. Para simular la falla en un enlace se selecciona un enlace (V_i,V_j) incidente a un vértice V_i de grado mínimo y para simular la falla de un vértice se selecciona cualquier vértice V_i vecino de un vértice V_j con grado mínimo (que no sea ni la fuente ni el destino) y se elimina junto con todos sus enlaces incidentes; esto asegura que la conectividad correspondiente se reduce en uno. Este proceso de simulación se repite mientras no se desconecte el grafo restante. El algoritmo se muestra a continuación:

```
Algoritmo SimulaFallas(clase,fuente,destino);
Repeat
   Case clase Of
      FallaEnlace:
         Vi := Vértice con grado mínimo;
         (Vi,Vj) := enlace incidente;
         E(G) := E(G) - (Vi,Vj)
      FallaNodo:
         Vi := Vértice con grado mínimo | Vi ∉ {fuente,destino}
         Vj := Vértice vecino;
         For ∀ Vk ∈ Adj[Vj]do
            E(G) := E(G) - (Vj,Vk)
         Endfor;
         V(G) := V(G) - Vj
   Endcase;
   Calcula parámetros determinísticos de G
Until ObtenDistancia(fuente,destino)=∞
End;
```

Algoritmo 5.10 Simulación de fallas en enlaces y nodos

5.3. Modelado Probabilístico

El modelo probabilístico toma en cuenta la probabilidad de funcionamiento de cada uno de los componentes de la red (vértices y enlaces) para el cálculo de la confiabilidad; asumiendo que tales probabilidades se conocen de antemano y se dan manera independiente estadísticamente.

En la sección 3.3, se mencionaron los enfoques existentes utilizados para la obtención de la confiabilidad así como sus restricciones. En el modelo propuesto lo que se busca es una aproximación al valor exacto de la confiabilidad haciendo uso de expresiones algebraicas. El método a seguir consiste de 3 pasos [Valdivia 89]:

1. Se obtienen todas las rutas disponibles entre la fuente y el destino representándolas por una expresión denominada *cubo*.
2. Se realizan ciertas operaciones a estos cubos para obtener una expresión algebraica booleana.

3. Esta expresión algebraica se interpreta como una expresión que simboliza las medidas de interés tales como la probabilidad estacionaria de éxito o el MTTF, lo cual se logra al representar la expresión como una suma de cubos disjuntos.

Los pasos 1 y 2 se ejecutarán de manera recursiva para obtener la expresión booleana de manera gradual de tal manera que no se requiera de mucho espacio de memoria ni tome mucho tiempo su cómputo tal como se sugiere en [Valdivia 89].

5.3.1. Cubos y la Operación "Sharp"

5.3.1.1. Identificador de ruta

Para un grafo que consiste de n vértices y e enlaces, un identificador de ruta se define como sigue:

El identificador de ruta IR_a para ruta R_a es una cadena de k variables binarias

$IR_a = x_1, x_2, x_3, \ldots, x_i, \ldots, x_k$

donde

$$x_i = \begin{cases} 1 \textit{ si el } i-\textit{ésimo elemento del} \text{ grafo } \textit{es incluido en el árbol} \\ x \textit{ de otro} \text{ modo} \end{cases}$$

y k es el número de elementos sujetos a falla que puede ser igual a n, e o $n+e$ dependiendo de que se considera puede fallar, vértices, enlaces o ambos respectivamente, en nuestro modelo se consideraran fallas en vértices y enlaces por lo cual k=n+e. En la siguiente figura se muestra una ruta del vértice E al D y su respectivo IR, en el cual primero se representan los enlaces y posteriormente los vértices:

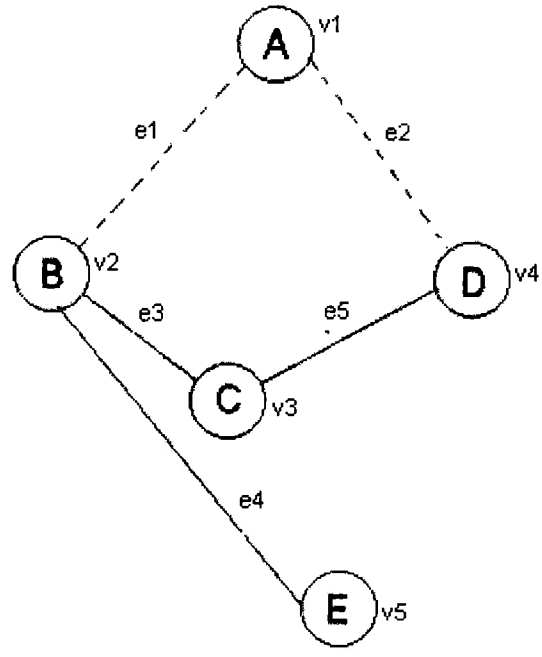

Figura 5.2 Ruta simple de E a D. IR=xx111x1111.

5.3.1.2. Cubo

Un cubo en el álgebra booleana es una representación geométrica de una función booleana al mapear una función de n variables a una unidad n-dimensional (cubo-n) [Miller 65]. Debido a que un identificador de ruta tiene la forma de un cubo se usará un cubo para representar la ruta en el álgebra booleana.

Un cubo es una cadena de variables de la forma:

$C = s_1, s_2, s_3, \ldots, s_i, \ldots, s_k$

donde k es el número de elementos en el grafo del sistema y s_i es el estado del elemento x_i, y tiene los siguientes valores:

$$s_i = \begin{cases} 0 \; si \; x_i \; tiene una \; falla \\ 1 \; si \; x_i \; esta \; funcionando \\ x \quad estado \; arbitrario \end{cases}$$

5.3.1.3. Operación sharp

Una expresión booleana se genera al aplicar la operación "sharp" (operación #) entre 2 cubos, denotada por A#B, obteniéndose así el conjunto de subcubos de A no incluidos en B, lo cual es una suma de cubos disjuntos. Para la descripción algebraica de la operación # usaremos las siguientes 2 tablas que representan la operación coordinada # (no conmutativa) y la operación de intersección coordinada:

	b_i			
a_i	**#**	**0**	**1**	**x**
	0	Z	y	z
	1	Y	z	z
	x	1	0	z

Tabla 5.1 Operación coordinada #, $a_i \# b_i$

	b_i			
a_i	$\cap$	**0**	**1**	**x**
	0	0	$\varnothing$	0
	1	$\varnothing$	1	1
	x	0	0	x

Tabla 5.2 Operación de intersección coordinada, $a_i b_i$

La operación # entre 2 cubos $A=a_1,a_2,a_3,...,a_n$ y $B=b_1,b_2,b_3,...,b_n$ se define como sigue:

$$A\#B = \begin{cases} A \text{ si } a_i \# b_i = \text{y para cualquier i} \\ \varnothing \text{ si } a_i \# b_i = \text{z para toda i} \\ \bigcup_{i=1}^{n} C_i \text{ de otro modo} \end{cases}$$

donde $C_i = ((a_1 b_1),...,(a_{i-1} b_{i-1}),\alpha_i,a_{i+1},...,a_n)$

$\alpha_i = a_i \# bi = 0$ o 1

y $a_i b_i$ es la intersección coordinada tal como se definió en la tabla 5.2.

La intersección entre 2 cubos de define como:

$$A \cap B = \begin{cases} \varnothing \ si \ a_i b_i = \varnothing \\ (a_1 b_1, a_2 b_2, ..., a_n b_n) \ de \ otro \text{ modo} \end{cases}$$

La operación # tiene las siguientes propiedades:

a) $A \# B = A$ si $A \cap B = \varnothing$

b) $A \# B = \varnothing$ si $A \cap B = A$

c) $A \# B = \sum_{i=1}^{n} C_i$ si $A\#B = \cup C_i$ ya que todos los cubos C_i son disjuntos par a par

5.3.2. Algoritmo para la Obtención de la Expresión Booleana

A continuación se muestra el algoritmo recursivo utilizado para la obtención del conjunto de los cubos disjuntos de un grafo G (la expresión booleana), el cual fue tomado de [Valdivia 89]. La variable ExprBool representa la expresión booleana simbólica, la cual se usará posteriormente para determinar las medidas numéricas de la confiabilidad.

Las condiciones iniciales son:

- Y es el valor universal para el espacio muestral, $Y=(x_1,x_2,...,x_k)$
- ExprBool esta vacía antes de llamar por primera vez al algoritmo.
- Se supone que se van a tomar en cuenta las fallas en vértices y enlaces, por lo cual $k = e + n$.

```
Algoritmo ObtenExprBooleana(Y,G,fuente,destino);
ObtenRuta(fuente,destino,G,T);  // Se obtiene una ruta de la fuente al destino del grafo G
                                // y se almacena en T
ObtenIdent(A',T); // Se representa como identificador de ruta en A'
A=Y∩A';   // Se obtiene la representación de cubo
ExprBool:=ExprBool+A;
ObtenCubosDisjuntos(Y,A,B,r) // Se obtienen los r cubos Bi disjuntos de Y#A
// Se aplica este algoritmo recursivamente a cada subgrafo Gi conectado representado por
// Bi hasta que todos los subgrafos resultantes estén desconectados
For i:=1 to r do
    ObtenSubgrafo(Gi,Bi)
    If Gi esta conectado then
      ObtenExprBooleana(Bi,Gi)
Endfor
End;
```

Algoritmo 5.11 Obtención de la Expresión Booleana

Lo primero que se hace en el algoritmo es obtener la ruta de la fuente al destino mediante el algoritmo ObtenRuta, el cual accede a las tablas generadas por el algoritmo de ruteo para obtener la ruta más corta de la fuente al destino. Una vez hecho esto, se obtiene el identificador de la ruta y se almacena en A' mediante el algoritmo ObtenIdent, a este identificador se le hace una intersección con Y para obtener la representación de cubo de la ruta, la cual se agrega a la expresión booleana. Posteriormente mediante el algoritmo ObtenCubosDisjuntos, se obtienen los r cubos disjuntos al realizar la operación # entre Y y A, cada uno de estos cubos disjuntos almacenados en B, representan un subgrafo de G, de acuerdo a la siguiente correspondencia:

$x_j \notin G_i$ si $b_j = 0$

$x_j \in G_i$ de otro modo, $b_j = 1$ o x

donde x_j es cada uno de los elementos de cada cubo B_i, el cual representa al subgrafo G_i.

Los algoritmos auxiliares se muestran a continuación:

```
Algoritmo ObtenRuta (fuente,destino,G,P);
P := {}
Se ejecuta el algoritmo de ruteo en los vértices del grafo G
If ObtenDistancia(fuente,destino) <> ∞ then
   Vi := fuente;
   While Vi ≠ destino do
      Se obtiene vía[destino] de la tabla T de Vi y se asigna a Vj
      P := P ∪ {(Vi, Vj)}
      Vi := Vj
    EndWhile;
End;
```

Algoritmo 5.12 Obtención de una ruta de la fuente al destino

```
Algoritmo ObtenIdent (IR,P);
IR := x1x2,...,xk
While P ≠ ∅ do
      Se extrae el primer enlace de P, (Vi, Vj)
      P := P - (Vi, Vj);
      IR[i] := 1;
      IR[Indice((Vi, Vj))] := 1;  // Indice devuelve la posición que tiene el enlace en IR
```

```
EndWhile;
IR[j] := 1;
ObtenIdent := IR
End;
```

Algoritmo 5.13 Obtención del identificador IR de la ruta P

```
Algoritmo ObtenCubosDisjuntos(Y,A,B,r);
Res:=Y#A;
r :=0;
If Res ≠ A AND Res ≠ ∅ then
   Pos := 1;
   While Pos <= k do
      While Pos<=k AND Res[Pos] ∉ {0,1} do
         Pos := Pos+1;
      EndWhile
      If Pos<=k then
         r := r+1;
         For j:=1 to pos-1 do
            B_r[j]:=y_j a_j;
         B_r[pos] := Res_pos;
         For j:=1 to k do
            B_r[k] := a_j;
   EndWhile;
End;
```

Algoritmo 5.14 Obtención de los cubos disjuntos de Y#A

```
Algoritmo ObtenSubgrafo(G_i,B_i);
E(G_i) := ∅
V(G_i) := ∅
For j:=1 to n do
   If B_i[j] = 1 Or B_i[j] = x then
      V(G_i) := V(G_i) ∪ {V_j}
Endfor;
For j:=1+n to k do
   If B_i[j] = 1 Or B_i[j] = x then
      E(G_i):=E(G_i) ∪ {enlace(j)}  //enlace devuelve el enlace correspondiente a la posición j
Endfor;
End;
```

Algoritmo 5.15 Obtención del subgrafo G_i correspondiente a B_i

En la siguiente figura se muestra el comportamiento de este algoritmo conforme encuentra la expresión booleana para el grafo de la figura 5.2, solo se muestran los subgrafos conectados (representados por círculos) y algunos de los que no quedan conectados (representados por cuadrados):

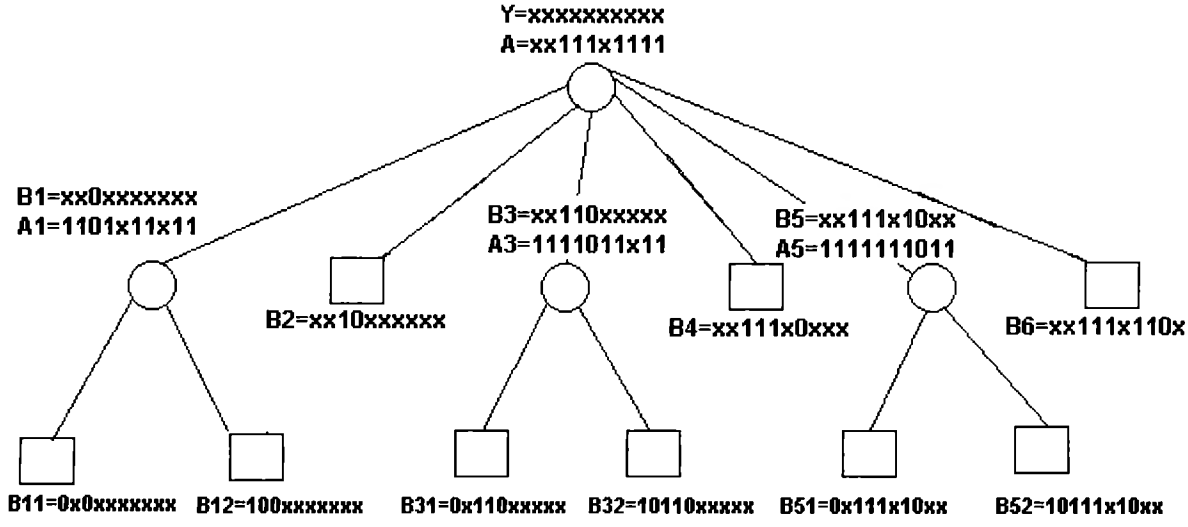

Figura 5.3 Arbol de cómputo del grafo de la figura 5.2

Conforme se profundiza en la recursión, el número de ceros en los cubos obtenidos se hace demasiado grande como lo veremos en el siguiente ejemplo, por lo cual no aporta mucho a las medidas de confiabilidad que se obtendrán posteriormente. Sería recomendable dar un valor N que represente hasta que nivel de recursión se va a ejecutar el algoritmo ahorrando tiempo de ejecución y espacio en memoria, aunque claro, con el costo de obtener valores de confiabilidad menos exactos.

5.3.3. Medidas de Confiabilidad

Una vez ejecutado el algoritmo 5.11 tenemos la expresión booleana consistente de la suma de los cubos disjuntos, la cual puede ser transformada a una expresión numérica de la confiabilidad al sustituir los valores del cubo por las diferentes medidas de confiabilidad, las cuales pueden ser estacionarias y dinámicas.

5.3.3.1. Confiabilidad estacionaria o estática

Esta es la medida de confiabilidad que se obtiene al suponer que la probabilidad de que los elementos estén funcionando es constante durante el tiempo que se analiza el

sistema. El cálculo de esta medida para cada cubo C_j se hace de la siguiente manera:

Probabilidad de que C_j esté funcionando, $Pr\{C_j\} = P * Q$ ... (5.2)

donde

$P = \prod p_i$ para todo i que satisfaga $s_i = 1$

$Q = \prod (1-p_i)$ para todo i que satisfaga $s_i = 0$

p_i = probabilidad de que el elemento i este funcionando

La expresión simbólica de la confiabilidad es entonces:

$$R = \sum_{j=1}^{r} Pr\{C_j\} \quad \text{... (5.3)}$$

donde r es el número total de cubos de la expresión booleana.

La no confiabilidad U, es por lo tanto:

$$U = 1 - R \quad \text{... (5.4)}$$

5.3.3.2. Confiabilidad dinámica

Es cuando se considera que las probabilidades de operación de los elementos varían en el tiempo de acuerdo a una distribución de probabilidad, es decir, ahora la confiabilidad es dependiente del tiempo dado(¿funcionará después de t tiempo?). En este tipo de evaluación de confiabilidad existen lo que se denomina sistemas cerrados, que es cuando los elementos que fallan no pueden ser reparados y los sistemas reparables, en donde sucede lo contrario. Algunas de las medidas que interesan para los elementos individuales en estos sistemas ya fueron mencionadas en la sección 3.1, como son el MTTF, MTTR, Disponibilidad, etc. Veamos ahora como calcular estas medidas para el sistema completo.

La confiabilidad dependiente del tiempo se calcula para cada componente con la siguiente ecuación:

$$R(x_i,t) = e^{-\lambda_i t} \qquad \ldots (5.5)$$

donde λ_i es la tasa de falla del elemento i.

Y para el sistema en totalidad se calcularía obteniendo nuevamente la de cada cubo de manera análoga a como se realizó en la confiabilidad estacionaria, sólo que ahora P y Q tendrían el siguiente valor:

$P = \prod R(x_i,t)$ para todo i que satisfaga $s_i = 1$

$Q = \prod (1 - R(x_i,t))$ para todo i que satisfaga $s_i = 0$

Y la confiabilidad global del sistema se obtendría aplicando la siguiente fórmula sobre varios valores de t a lo largo de un intervalo dado:

$$R(t) = \sum_{j=1}^{r} \Pr\{C_j\} \qquad \ldots (5.6)$$

El MTTF se calcula de acuerdo a lo que ya habíamos mencionado con la siguiente fórmula:

$$MTTF = \int_0^\infty R(t)dt \qquad \ldots (5.7)$$

Pero debido a la complejidad de su cálculo se opta por utilizar integración numérica, siendo uno de los métodos para ello el método de 1/3 de Simpson [Chapra 87, Nakamura 92], el cual obtiene valores muy cercanos al real. Esta regla se define como sigue:

$$\int_a^b f(x)dx = (b-a)\frac{f(x_0) + 4\sum_{i=1,3,5,\ldots}^{n-1} f(x_i) + 2\sum_{i=2,4,6,\ldots}^{n-2} f(x_i) + f(x_n)}{3n} \qquad \ldots (5.8)$$

Para que el método obtenga los mejores resultados al obtener el MTTF se recomienda usar como valor de *b* un valor bastante grande, como *a*, un valor de cero y utilizar varios intervalos (valores de t), la cantidad *n* de estos debe ser par de acuerdo al método, quedando entonces la fórmula como sigue:

$$MTTF = t_n \frac{R(t_0) + 4\sum_{i=1,3,5,\ldots}^{n-1} R(t_i) + 2\sum_{i=2,4,6,\ldots}^{n-2} R(t_i) + R(t_n)}{3n} \quad \ldots (5.9)$$

La disponibilidad del elemento i esta dada por:

$$A(x_i,t) = \frac{\mu_i}{\lambda_i + \mu_i} + \frac{\lambda_i}{\lambda_i + \mu_i} e^{\lambda_i t} \quad \ldots (5.10)$$

Y por tanto, la disponibilidad de todo el sistema A(t) se calcularía de manera análoga a la de R(t), pero dando los siguientes valores a P y Q para los cubos:

$P = \prod A(x_i,t)$ para todo i que satisfaga $s_i = 1$

$Q = \prod (1 - A(x_i,t))$ para todo i que satisfaga $s_i = 0$

La disponibilidad en estado estable, es decir, la disponibilidad que se obtiene en el tiempo ∞, se calcula para cada elemento como sigue:

$$SA(x_i) = \frac{\mu_i}{\lambda_i + \mu_i} \quad \ldots (5.11)$$

Y la del sistema SA, se calcula de manera análoga a la forma en que se obtiene R(t) y A(t).

Por último, el MTBF del sistema se calcula de la siguiente forma:

$$MTBF = \frac{MTTF}{SA} \quad \ldots (5.12)$$

5.3.4. Simulación de fallas

Como se hizo en el modelo determinista, en este también se simulan las fallas en los enlaces y vértices o nodos. El algoritmo para lograr esto se muestra a continuación:

```
Algoritmo SimulaFallas(clase,fuente,destino);
Repeat
   Case clase Of
      FallaEnlace:
         Vi := Vértice con grado mínimo;
         (Vi,Vj) := enlace incidente;
         E(G) := E(G) - (Vi,Vj)
      FallaNodo:
         Vi := Vértice con grado mínimo | Vi ∉ {fuente,destino}
         Vj := Vértice vecino;
         For ∀ Vk ∈ Adj[Vj]do
            E(G) := E(G) - (Vj,Vk)
         Endfor;
         V(G) := V(G) - Vj
  Endcase;
   Calcula parámetros probabilísticos de G
Until ObtenDistancia(fuente,destino)=∞
End;
```

Algoritmo 5.16 Simulación de Fallas en Enlaces y Nodos (Probabilístico)

Tomando como ejemplo el cálculo de la confiabilidad estacionaria, siguiendo el algoritmo de simulación de fallas de nodos y teniendo como base el grafo de la figura 5.2, tenemos que inicialmente la expresión booleana calculada (usando como métrica para la determinación de las rutas la confiabilidad de los enlaces) es xx111x11111101x11x111111011x111111111011, tal como se observó en la figura 5.3; y si le damos un valor de confiabilidad a cada nodo de 1.0, a los enlaces e1 y e2 de 0.8 y a los enlaces restantes de 0.9 obtenemos de acuerdo a la ecuaciones 5.2 y 5.3 una confiabilidad de 0.838439; al eliminarse un nodo (el cual sería el A), la expresión booleana queda en 1111111, dando como resultado una confiabilidad estacionaria de 0.7289999; el segundo nodo eliminado (B) hace que no exista ya una ruta de E a D y por tanto, la confiabilidad es 0.

CAPITULO 6

IMPLANTANCION DEL MODELO

6.1. Introducción

Una vez que se han desarrollado los algoritmos del modelo de dependencia se procederá ahora a ver los detalles de su implantación en computadora; esta implantación busca tener una interfaz gráfica amigable, poder ser ejecutada en diversas plataformas y permitir el reuso del código ya desarrollado (esto con el fin de agregarle funcionalidad sin tener que modificar lo ya hecho sino simplemente anexarle nuevos módulos); por lo cual se optó por el uso del lenguaje de programación Java ya que proporciona una muy buena portabilidad, por ser orientado a objetos da la posibilidad del reuso de código y gracias a su librería de gráficos, AWT (*Abstract Windowing Toolkit*), se puede generar una interfaz de usuario de manera sencilla, sin tener que conocer el funcionamiento interno del ambiente gráfico subyacente [Hernández 97, Horvilleur 97, Vizcaíno 97].

La interfaz de la implantación permite dibujar un grafo que represente una red con el uso del mouse y pide la probabilidad de funcionamiento de cada uno de los componentes que se agregan (vértices y enlaces). Una vez dibujado el grafo se pueden obtener las rutas de acuerdo al algoritmo de estado de enlace o los AEE (Anuncios de Estado de Enlace) de cada uno de sus vértices, podemos obtener los parámetros deterministas o probabilísticos del grafo así como hacer una simulación de las fallas en vértices y enlaces. También permite guardar las representaciones de los grafos o leerlas desde disco. A continuación se muestra cada una de las partes en que se divide la interfaz.

6.2. Interfaz de usuario

La interfaz de usuario se muestra en la figura 6.1.

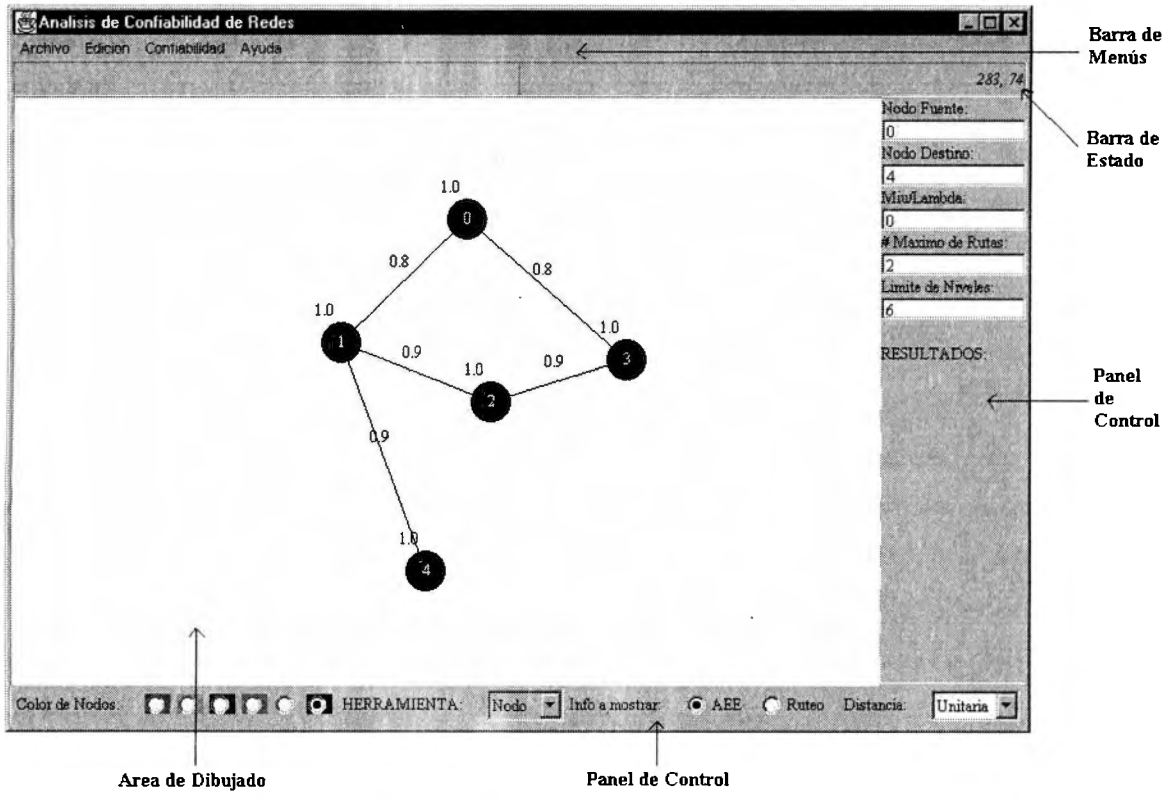

Figura 6.1 Interfaz de usuario de la implantación del modelo

Se puede observar que la interfaz se divide en varias partes, por un lado en la parte superior se encuentra la barra de menús, abajo de la barra de menús se encuentra una barra de estado, en la parte inferior y en la parte central derecha tenemos un panel de control y en la parte central izquierda tenemos el área de dibujado del grafo. Se procede ahora al análisis de cada una de estas partes.

6.2.1. Barra de Menús

La barra contiene 4 menús tal como ve en la figura 6.1, y son:

- **Archivo**. Este da los comandos para el manejo de la representación de los grafos:
  - Nuevo. Para limpiar el área de dibujado.
  - Abrir. Para leer la representación desde un archivo.
    - Guardar como. Para guardar la representación en un archivo.
  - Salir. Para terminar la ejecución.

- **Edición.** Este proporciona solo un comando para la eliminación del último nodo o vértice dibujado.
- **Confiabilidad.** Aquí se encuentran los comandos para obtener los parámetros de confiabilidad del modelo determinista y probabilístico así como hacer una simulación de las fallas en nodos y enlaces, estos comandos se encuentran organizados como se muestra en la figura 6.2.

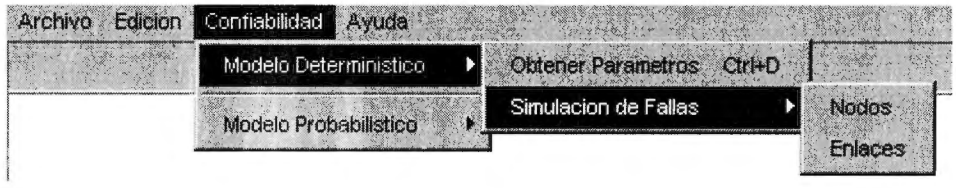

Figura 6.2 Organización del menú de confiabilidad.

- **Ayuda.** En este menú solo encontramos un cuadro de diálogo que muestra los datos del programa, tal como se muestra en la figura 6.3.

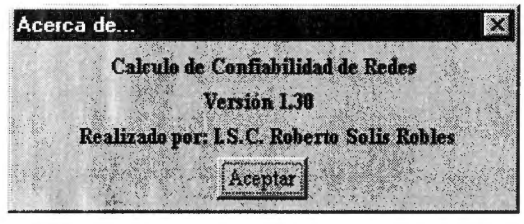

Figura 6.3 Cuadro de diálogo con los datos del programa.

6.2.2. Barra de Estado

Esta barra se encuentra dividida en 2 partes, la parte izquierda muestra un mensaje que nos indica si hay algún error o la acción que se está llevando a cabo, mientras que la parte derecha indica la posición del mouse en el área de dibujado.

Por ejemplo, la figura 6.4 muestra la barra de estado una vez que se dibuja un nuevo nodo.

Figura 6.4 Barra de Estado después de agregar un nuevo nodo.

6.2.3. Panel de controles de la parte inferior

En este panel se selecciona el color con el que se van a dibujar los nodos así como el elemento que vamos a dibujar, el cual se decide con base en el valor que tiene la herramienta, la cual puede ser Nodos o Enlaces. También se selecciona que información se va a mostrar cuando sé de un click con el botón derecho a un nodo, la cual puede ser el AEE o bien, su tabla de ruteo. Finalmente se decide sobre la base de qué se calculan las tablas de ruteo, si tomando a los enlaces como una distancia unitaria o bien tomando como base de decisión el valor de las probabilidades de cada enlace en cuyo caso se seleccionan las rutas con un mayor valor probabilístico. El panel se muestra en la figura 6.5.

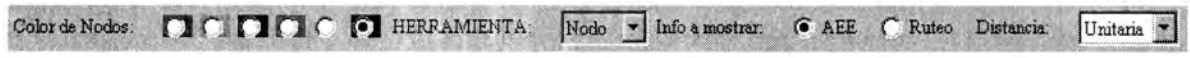

Figura 6.5 Panel de controles de la interfaz

6.2.4. Panel de controles de la parte central derecha

En este panel se seleccionan los nodos fuente y destino para la obtención de los parámetros de ambos modelos así como la relación μ/λ, el número máximo de rutas y el número máximo de niveles de profundidad a realizarse para la obtención de la expresión booleana mencionada en la sección 5.3.2 y la cual se utiliza en el modelo probabilístico. También en este panel se muestran los resultados obtenidos en el modelo seleccionado. En la figura 6.6 se muestra el resultado obtenido al ejecutar el modelo determinista sobre el grafo de la figura 5.2 haciendo uso de rutas con base en distancias unitarias.

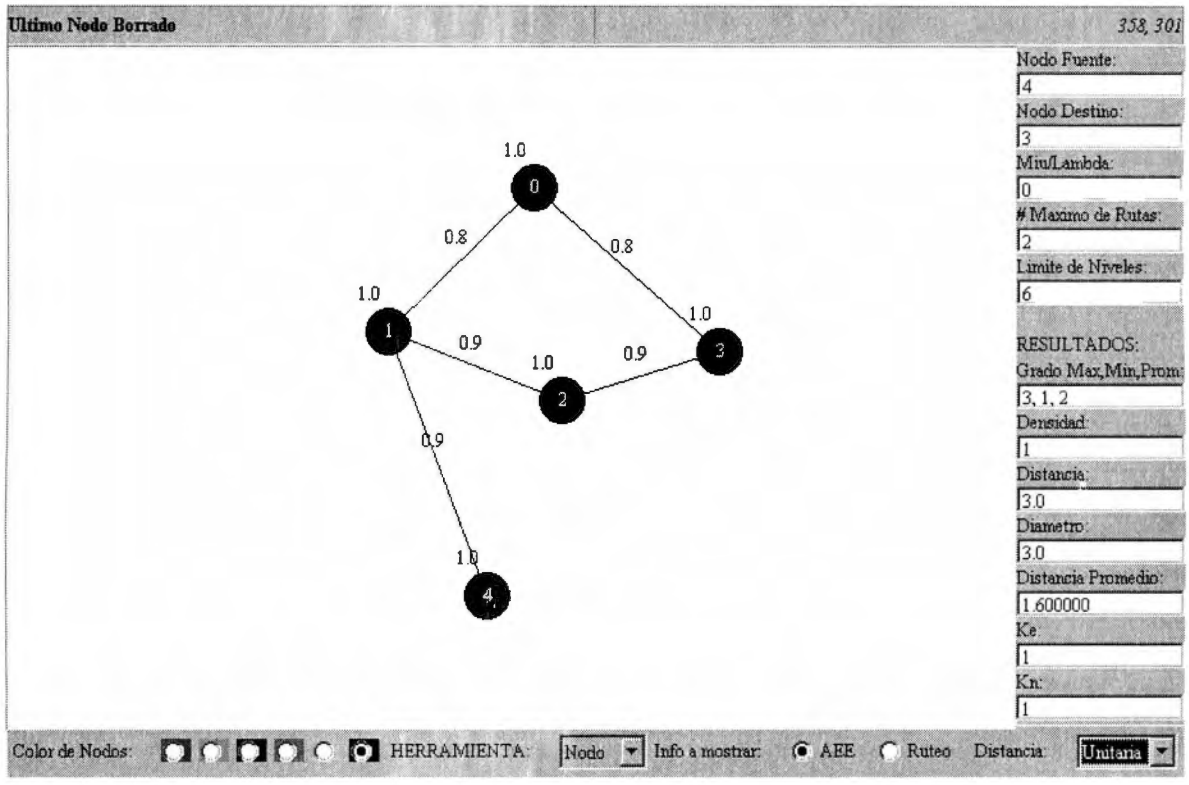

Figura 6.6 Resultados del modelo determinista sobre el grafo de la figura 5.2

6.2.5. Area de dibujado

En esta parte de la interfaz de usuario se pueden dibujar los grafos, para lo cual primeramente se selecciona la herramienta a utilizar del panel de control inferior. Si se va a dibujar un nodo se da click al mouse con el botón izquierdo, una vez hecho esto, aparecerá un cuadro de diálogo pidiendo la probabilidad que tendrá el elemento, se teclea tal probabilidad y se dibuja junto con su probabilidad. Si es un enlace se da click con el botón izquierdo en uno de los nodos extremos y se arrastra hasta el otro, una vez seleccionados los 2 extremos se pide la probabilidad del enlace y se dibuja junto con su probabilidad. Si un nodo es de grado cero se puede mover de posición seleccionando con el botón izquierdo del mouse y arrastrándolo a su nueva posición. En la figura 6.7 se muestra el cuadro de diálogo que pide la probabilidad del elemento.

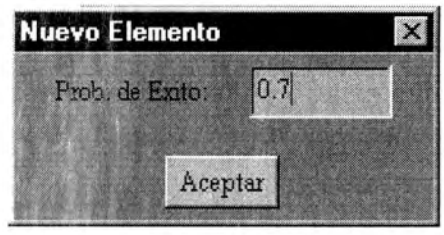

Figura 6.7 Cuadro de diálogo de petición de probabilidad del elemento

Una vez que se tienen enlaces en un nodo se puede presionar el botón derecho del mouse sobre él y se mostrará su AEE o su tabla de ruteo dependiendo de que se seleccionó en el panel de control. En la figura 6.8 se muestra la tabla de ruteo obtenida al seleccionar el nodo 4.

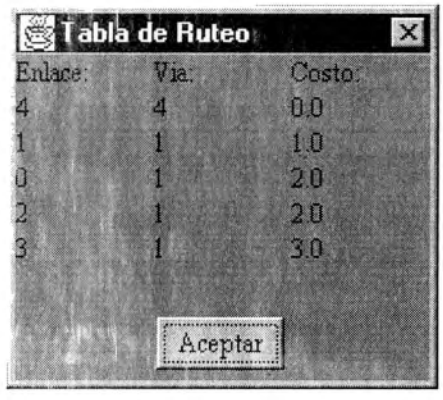

Figura 6.8 Tabla de ruteo del nodo 4.

6.3. Simulación de Fallas

Una vez que se tiene un grafo se puede simular las fallas en nodos o enlaces para cualquiera de los modelos vistos en el anterior capítulo seleccionando del menú de confiabilidad la opción deseada, cada vez que se elimina un nodo o enlace aparece un cuadro de diálogo que pide la aceptación para la continuación de la simulación. En las figuras 6.9, 6.10 y 6.11 se muestran los 3 pasos de la simulación de fallas en nodos del modelo determinista para el grafo de la figura 5.2.

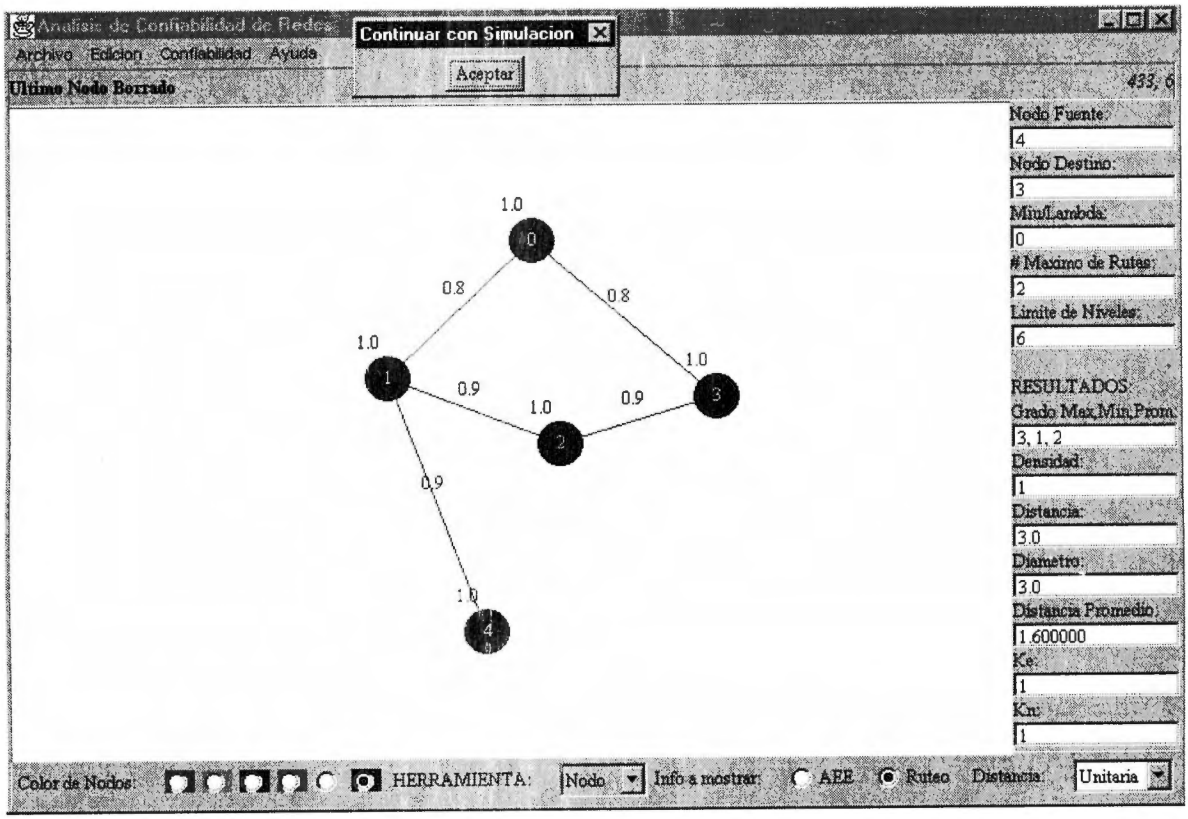

Figura 6.9 Primer paso en la simulación de fallas en nodos, el grafo todavía completo.

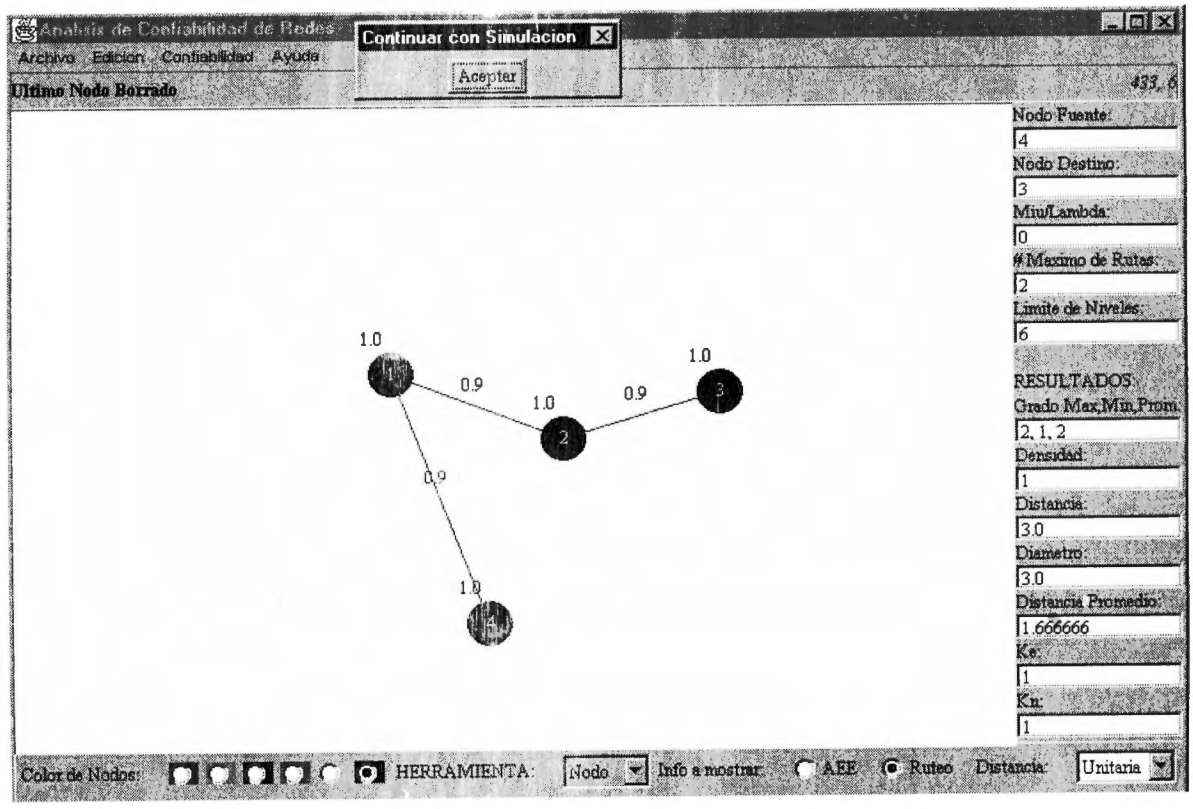

Figura 6.10 Segundo paso en la simulación, el nodo 0 eliminado.

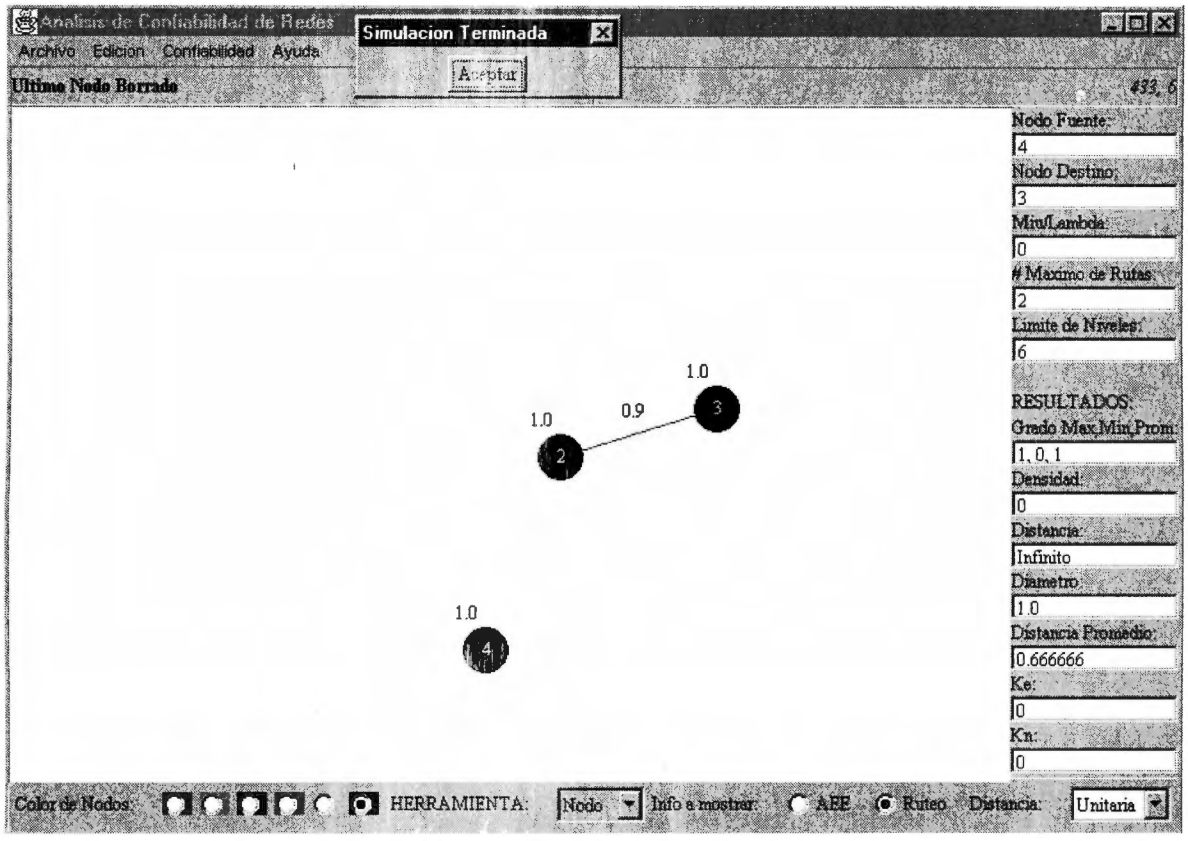

Figura 6.11 Tercer paso en la simulación, dos nodos eliminados (el nodo 0 y el nodo 1)

6.4. Detalles de la implantación

El programa que implanta el modelo de dependencia se construye haciendo uso de orientación a objetos pues como ya se mencionó es el enfoque utilizado por el lenguaje seleccionado, por lo cual el programa se divide en varias clases, cada una con diversos métodos para realizar sus funciones. Tales clases son:

- **Flujos**. Esta es la clase principal la cual pone a ejecutar el programa.
- **MyFrame.** Esta clase es la que genera la ventana del programa y le asigna su espacio a cada una de las partes ya mencionadas (área de dibujado, paneles de control, etc.)
- **Mensaje.** Esta clase solicita una confirmación para la realización de alguna tarea, como es la de borrar un nodo o de salir del programa.

- **AboutDialog**. Esta clase genera el cuadro de diálogo con la información del programa.
- **StateBar.** Esta clase genera la barra de estado donde se indica la acción realizada así como la posición del apuntador del mouse.
- **NewData.** Esta clase genera el cuadro de diálogo en el cual se pide la probabilidad que tiene el nuevo elemento agregado.
- **AEEData.** Esta clase genera el cuadro de diálogo donde se muestra el AEE de un nodo al presionar el botón derecho del mouse.
- **RutaData.** Esta clase genera el cuadro de diálogo donde se muestra la tabla de ruteo de un nodo.
- **DrawControls.** Esta clase genera el panel de control de la parte inferior de la interfaz donde se selecciona el color, herramienta, información a mostrar y la base de decisión para el ruteo.
- **DrawCon2.** Esta clase genera el panel de control de la parte central derecha donde se muestran los resultados de la ejecución del modelo de dependencia.
- **DrawPanel.** Esta clase se encarga del área de dibujado del grafo.
- **BD.** Esta clase se encarga del manejo de la representación de los grafos en disco, ya sea para guardarla o para leerla.
- **Grafo.** Esta clase es prácticamente el kernel del programa pues es la que se encarga de representar mediante listas de adyacencia a los grafos obtenidos y de calcular los diferentes parámetros de los modelos así como de realizar las simulaciones de fallas, para ello utiliza otras clases tales como:
    - **ListaAdy.** Para representar la lista de adyacencia de un nodo.
    - **InfoNodo.** Para representar a un nodo.
    - **ElemLista.** Para representar cada enlace en la lista de adyacencia.

6.5. Comentarios sobre la implantación

El lenguaje Java fue seleccionado por su gran portabilidad, la cual es a nivel de código objeto. Esto se debe a que el programa fuente es compilado una sola vez obteniéndose un código objeto intermedio (bytecode), el cual puede ejecutarse en varias plataformas mediante la ayuda de un intérprete. El programa fue desarrollado usando la versión 1.2 del JDK (Java Development Kit) de Sun Microsystems y fue probado en Windows 95/98, Windows NT, Solaris 2.6 y Linux 2.x con éxito y sin ningún cambio en el código.

Sin embargo, esta portabilidad tiene un precio; debido a que el programa es interpretado se hace lento con relación a un programa hecho específicamente para la plataforma, aunque con las velocidades de procesamiento actuales quizá no sea muy notable en cuanto a lo que a cálculos matemáticos se refiere pero en cuanto al despliegue en pantalla de los gráficos si es bastante notable.

También hay que recalcar que al programa le hace falta optimizar el código relacionado al dibujado del grafo pues como debe redibujar los elementos conforme se van agregando nuevos, entre mas sean, más lento se hace y se nota un parpadeo que puede ser molesto al usuario.

CAPITULO 7

RESULTADOS Y CONCLUSIONES

7.1. Modelo Determinista

Se mostrarán ahora algunos resultados obtenidos con el modelo determinista desarrollado aplicado a algunas configuraciones de grafo comunes. Las configuraciones se muestran en la figura 7.1 y de las figuras 7.2 a la 7.5 se muestran algunos parámetros de tales configuraciones conforme se van eliminando nodos.

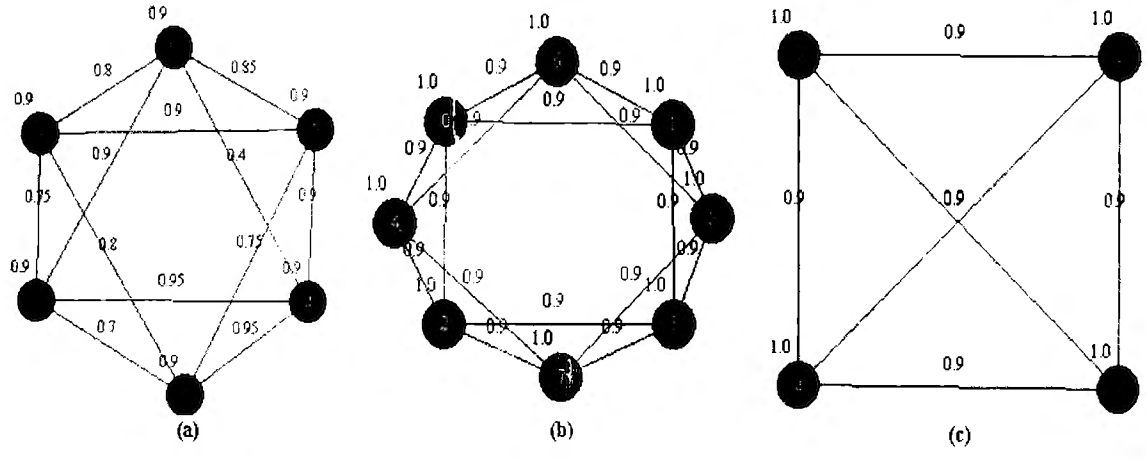

Figura 7.1 (a) Anillo mallado 3x2. (b) Anillo Mallado 4x2. (c) K4

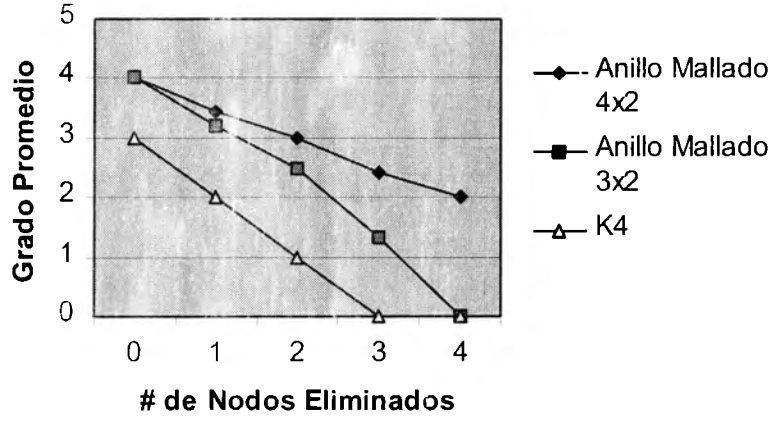

Figura 7.2 Grado promedio conforme se eliminan nodos

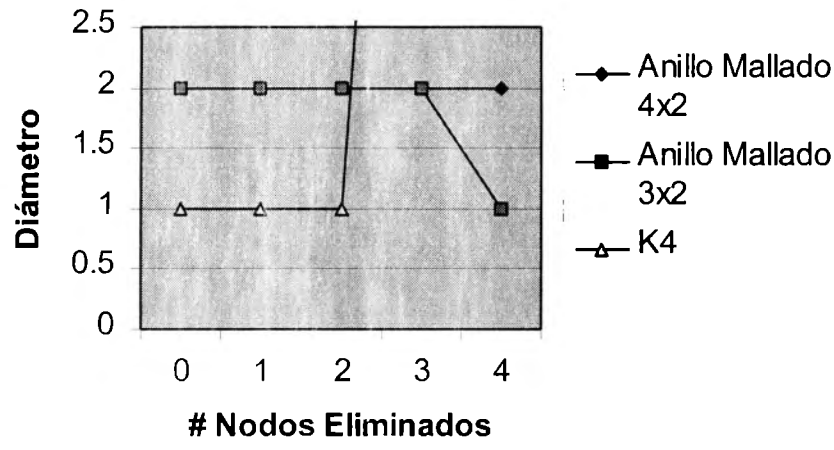

Figura 7.3 Diámetro conforme se eliminan nodos

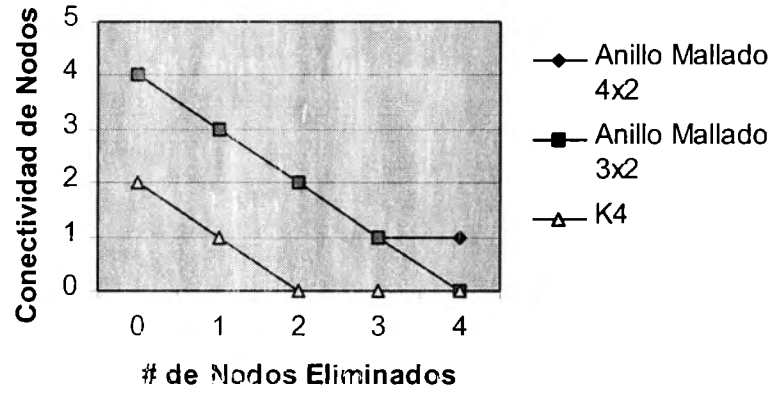

Figura 7.4 Conectividad de nodos conforme se eliminan nodos

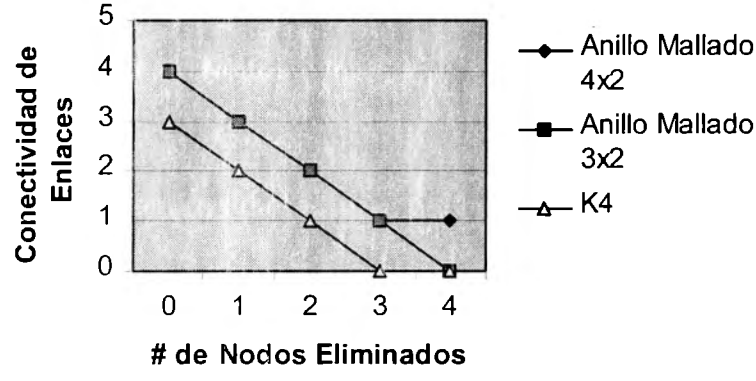

Figura 7.5 Conectividad de enlaces conforme se eliminan nodos

Se puede observar de acuerdo a los resultados obtenidos que la confiabilidad en el modelo determinista depende en gran medida de la conectividad de nodos y enlaces que se tenga en el grafo de la red, entre mayor valor tengan estos parámetros mayor será la confiabilidad. Por otro lado se puede notar que en la simulación de fallas se selecciona siempre un nodo que reduce el grado del grafo.

La figura 7.6 muestra una simplificación de la red Arpanet y la figura 7.7 muestra los parámetros deterministas obtenidos suponiendo que la fuente es el nodo 2 y el destino es el nodo 10.

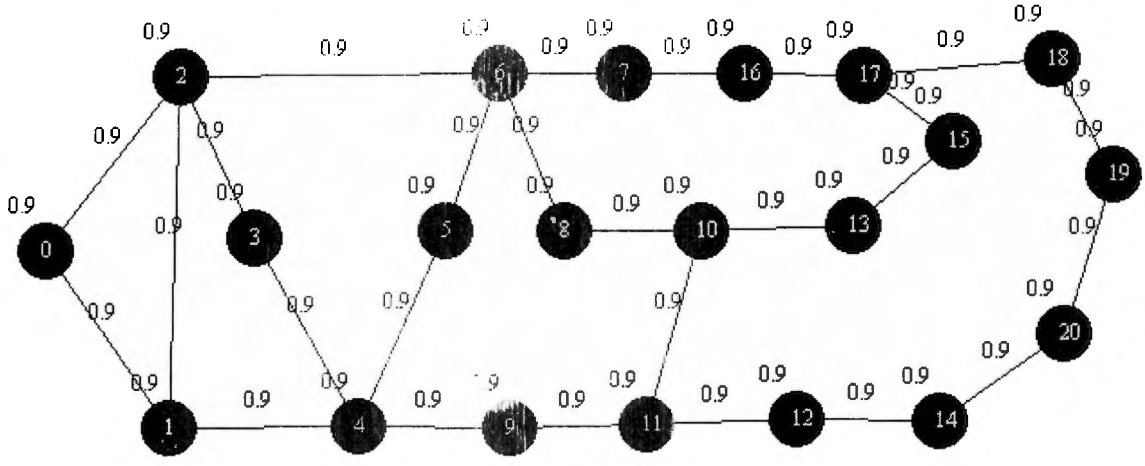

Figura 7.6 Arpanet simplificado

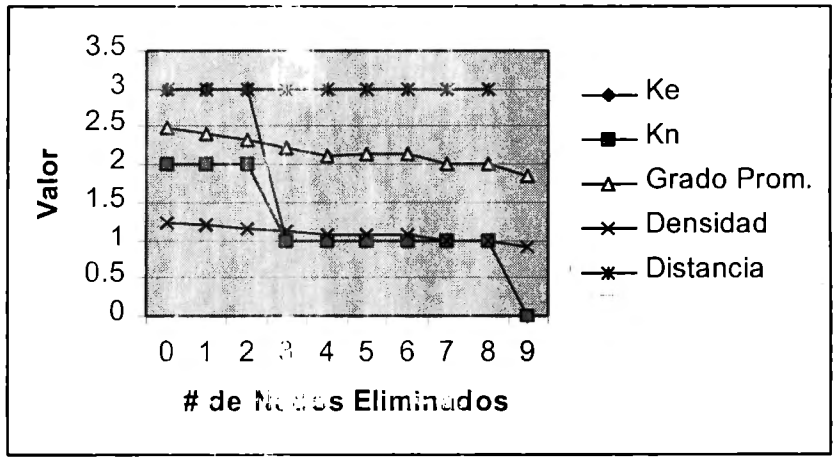

Figura 7.7 Parámetros deterministas de la red Arpanet simplificada

7.2. Modelo Probabilístico

Se obtienen ahora los parámetros de importancia para este modelo para redes que se han utilizado o utilizan actualmente con frecuencia. También se comparan los resultados obtenidos en ciertas configuraciones con los reales para determinar que tan cerca se está de ellos y también se muestra el tiempo requerido para los cálculos.

Se comienza con un subgrafo de la red RTN (Red Tecnológica Nacional), en el cual se ha dado una confiabilidad de 0.9 a los enlaces E1, una confiabilidad de 0.6 a los enlaces E0 o DS0 y una confiabilidad de 0.25 para los enlaces satelitales. El subgrafo de tal red se muestra en la figura 7.8 y en la figura 7.9 se muestra su confiabilidad estacionaria suponiendo que el nodo fuente es el 12 (el cual representa al D.F) y el nodo destino es el 2 (el cual representa a Torreón, Coahuila) en varios niveles conforme se eliminan enlaces.

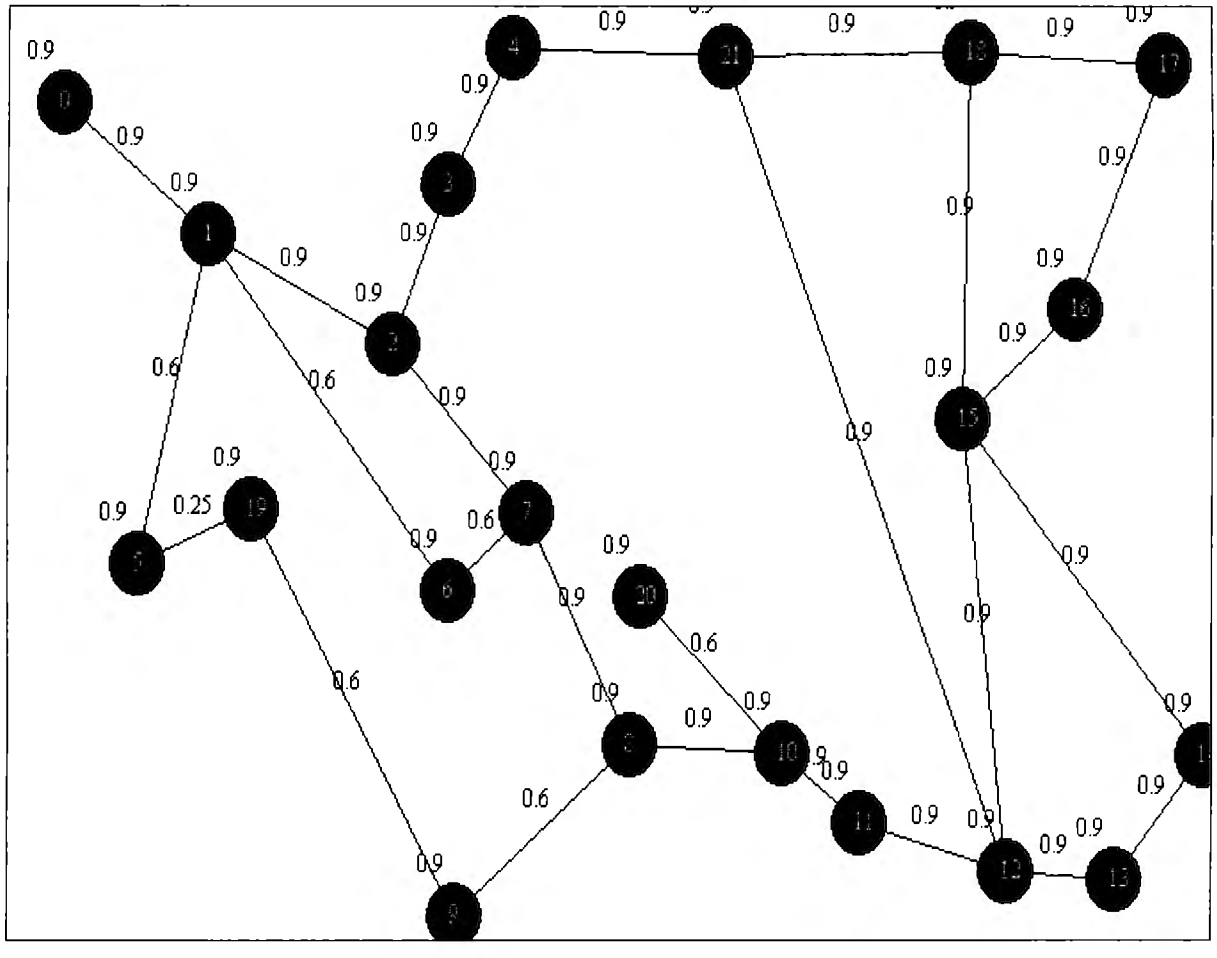

Figura 7.8 Representación de una subred de RTN

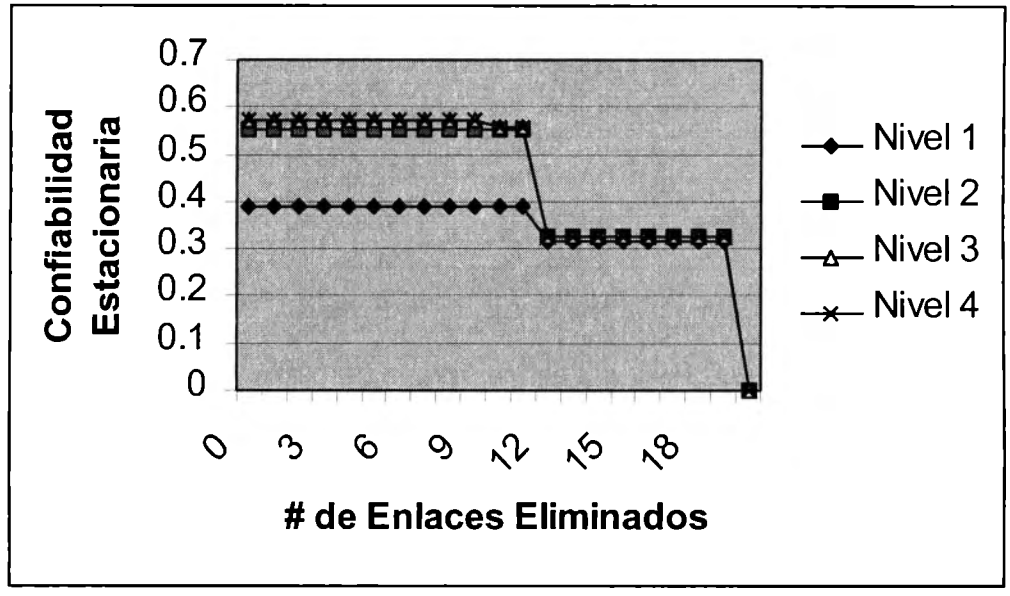

Figura 7.9 Confiabilidad estacionaria en la subred de RTN

De la misma manera se obtiene a continuación la confiabilidad estacionaria de la figura 7.6 tomando como nodo fuente el 16 y como nodo destino el 9.

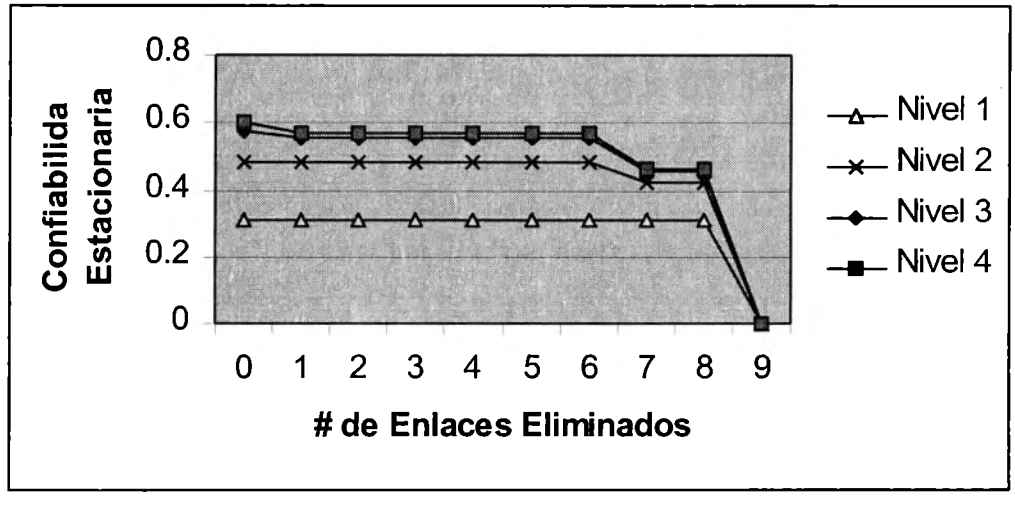

Figura 7.10 Confiabilidad Estacionaria para la red Arpanet Simplificada

Una vez que se han obtenido los resultados en cuanto a confiabilidad se refiere se muestra ahora el tiempo que toma generar la expresión booleana necesaria para el cálculo de la confiabilidad incluyendo el tiempo que lleva tal cálculo en diferentes niveles de profundidad, lo cual se refiere a lo mencionado en la sección 5.3.2; qué tanto va a avanzar

en la recursión el algoritmo de obtención de la expresión booleana para la determinación de los parámetros probabilísticos; y al mismo tiempo observaremos que tan cercanas están estas aproximaciones a las reales. Para ello se usan algunas configuraciones cuyo valor de confiabilidad real se puede consultar en [Bulteau 97].

La primera configuración se muestra en la figura 7.1(a), su confiabilidad calculada en distintos niveles junto con el valor real se muestra en la figura 7.11, y el tiempo para obtenerlo se muestra en la figura 7.12.

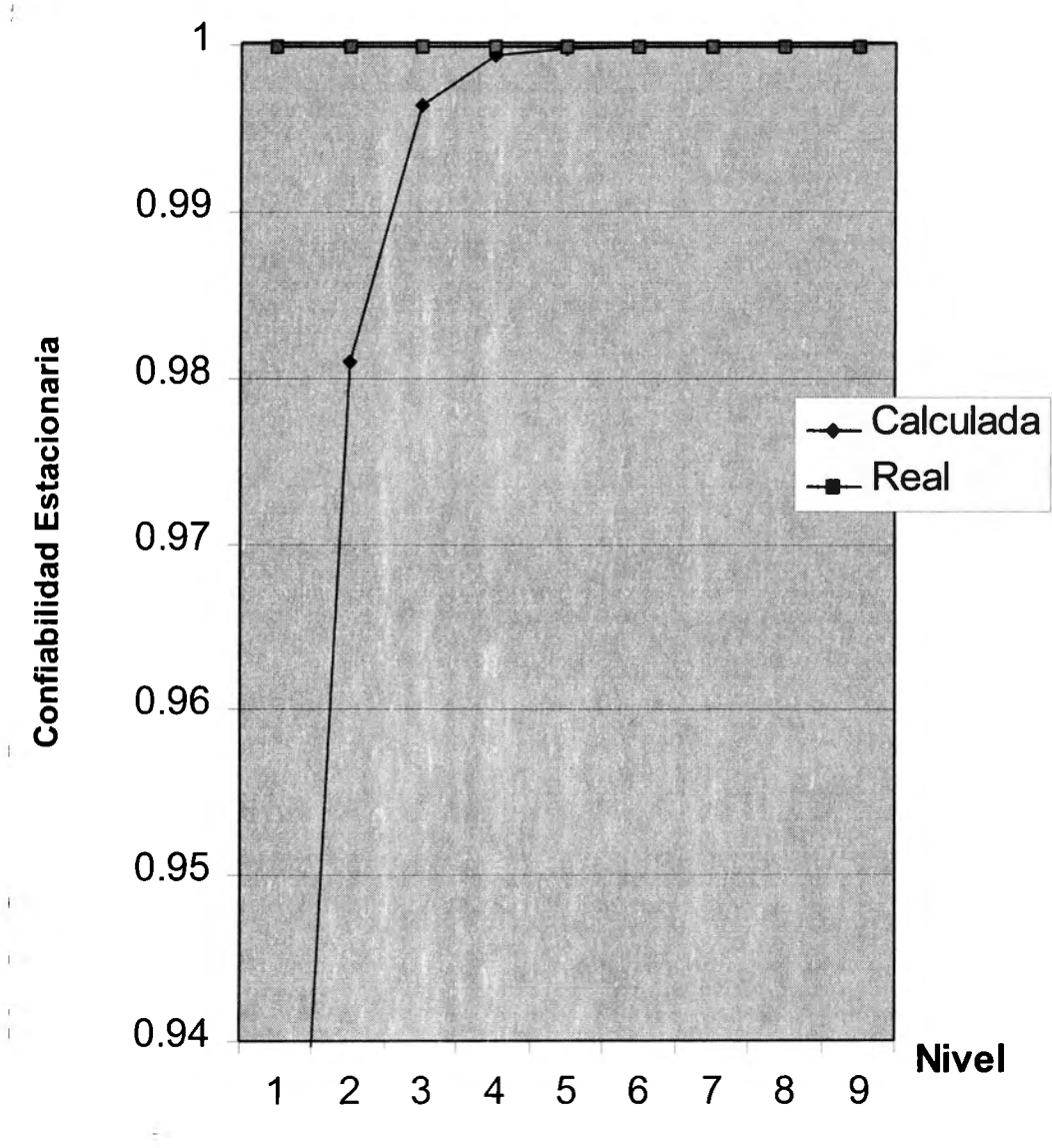

Figura 7.11 Confiabilidad real y calculada de la figura 7.1(a)

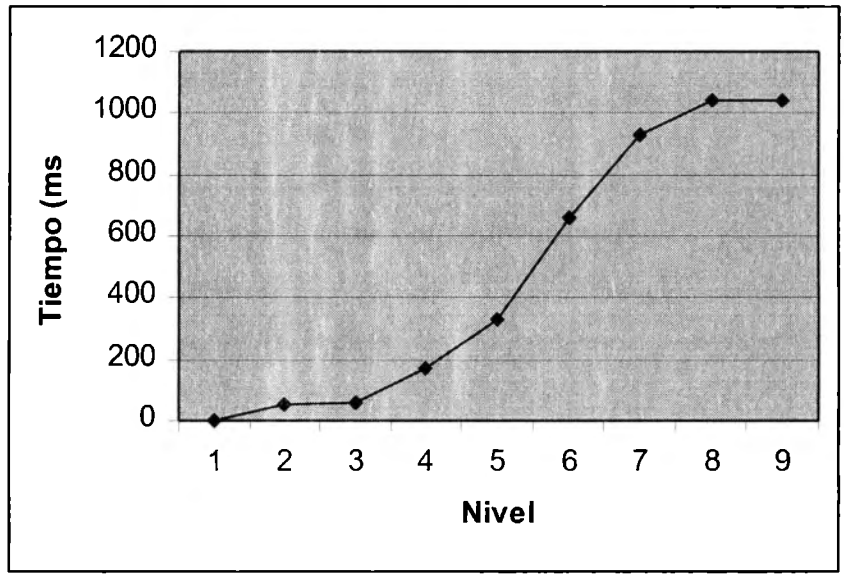

Figura 7.12 Tiempo necesario para el cálculo en cada nivel de la figura 7.1(a)

La segunda configuración se muestra en la figura 7.13, sus valores de confiabilidad en la figura 7.15 y los tiempos de cálculo en la 7.16.

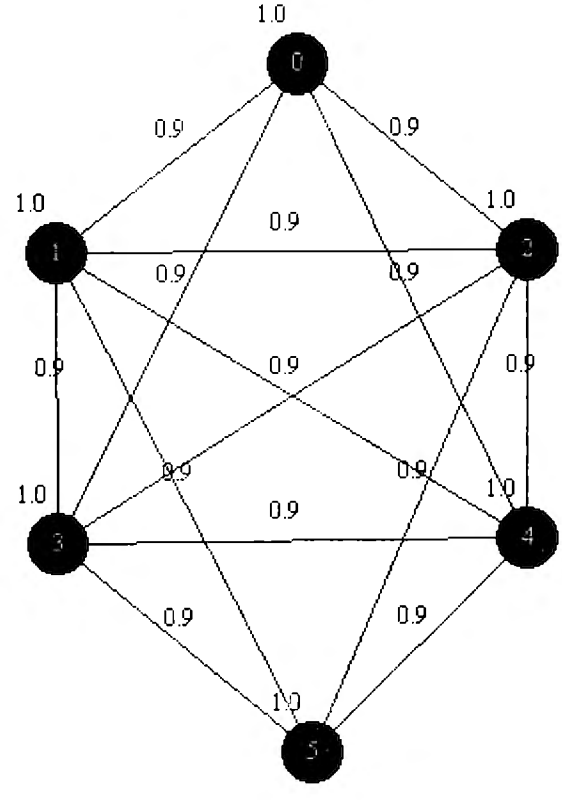

Figura 7.13 Grafo de Ejemplo

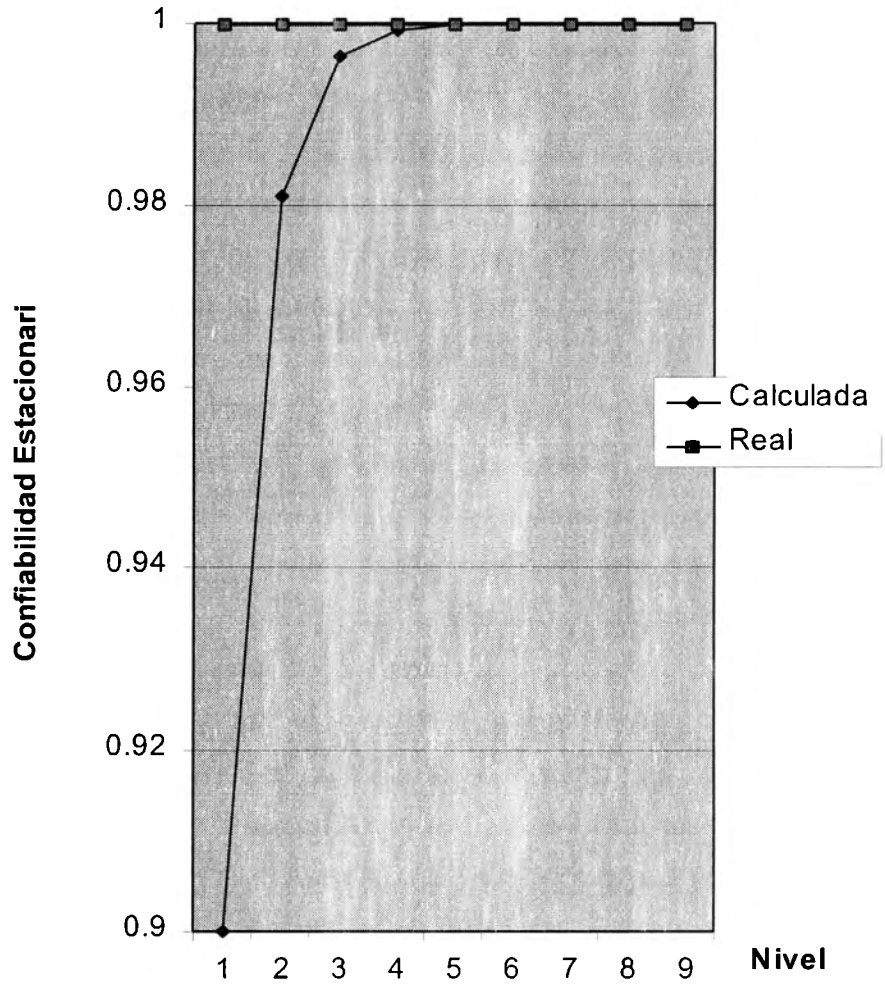

Figura 7.14 Confiabilidad real y calculada de la figura 7.13

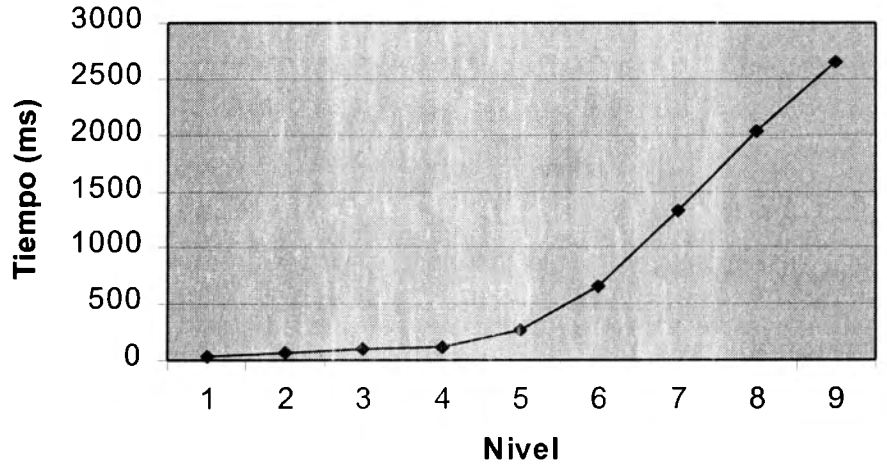

Figura 7.15 Tiempo necesario para el cálculo en cada nivel de la figura 7.13

La última configuración se muestra en la figura 7.16, sus valores de confiabilidad en la 7.17 y los tiempos de cálculo en la 7.18.

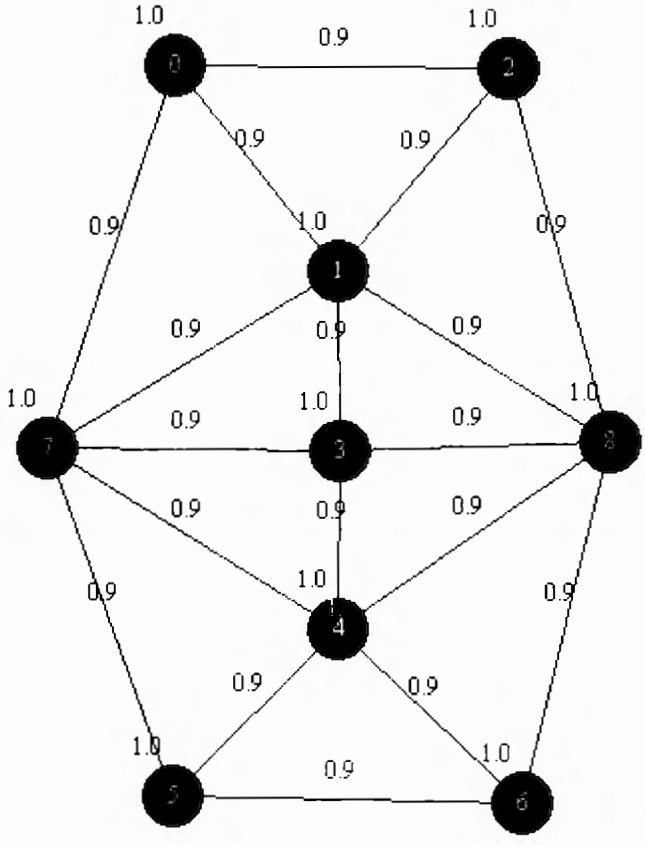

Figura 7.16 Otro grafo de ejemplo

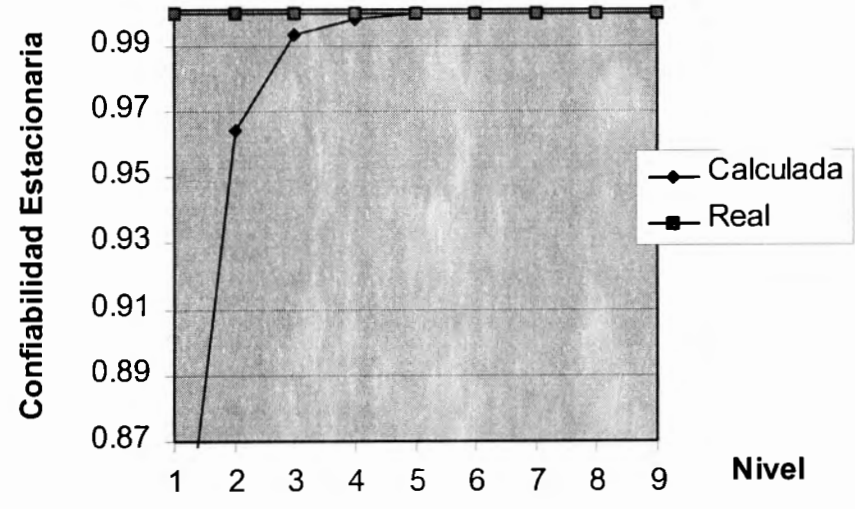

Figura 7.17 Confiabilidad real y calculada de la figura 7.16

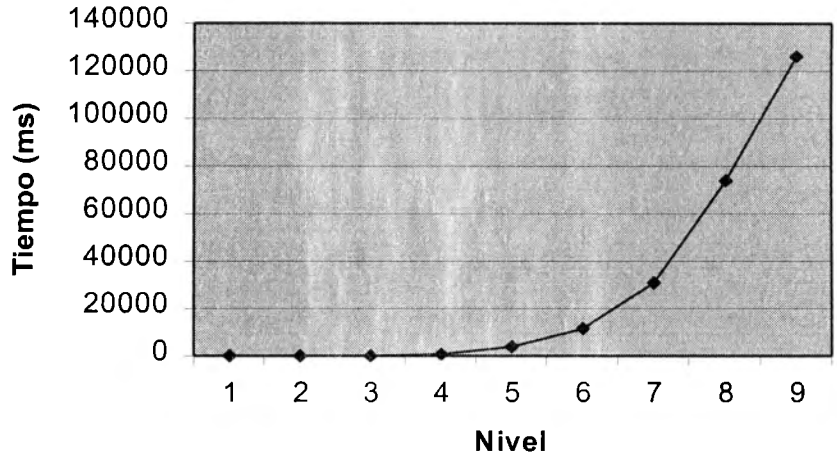

Figura 7.18 Tiempo necesario para el cálculo en cada nivel de la figura 7.16

Por otro lado se pudo observar al estar probando con configuraciones con confiabilidad de enlace diferente un comportamiento que se muestra en las siguientes figuras. Las figuras 7.19 y 7.20 muestran dos configuraciones con diferentes valores de confiabilidad de enlace, las figuras 7.21 y 7.22 muestran la confiabilidad estacionaria calculada en diversos niveles si las rutas se calculan con base en una distancia unitaria o con base en la probabilidad del enlace.

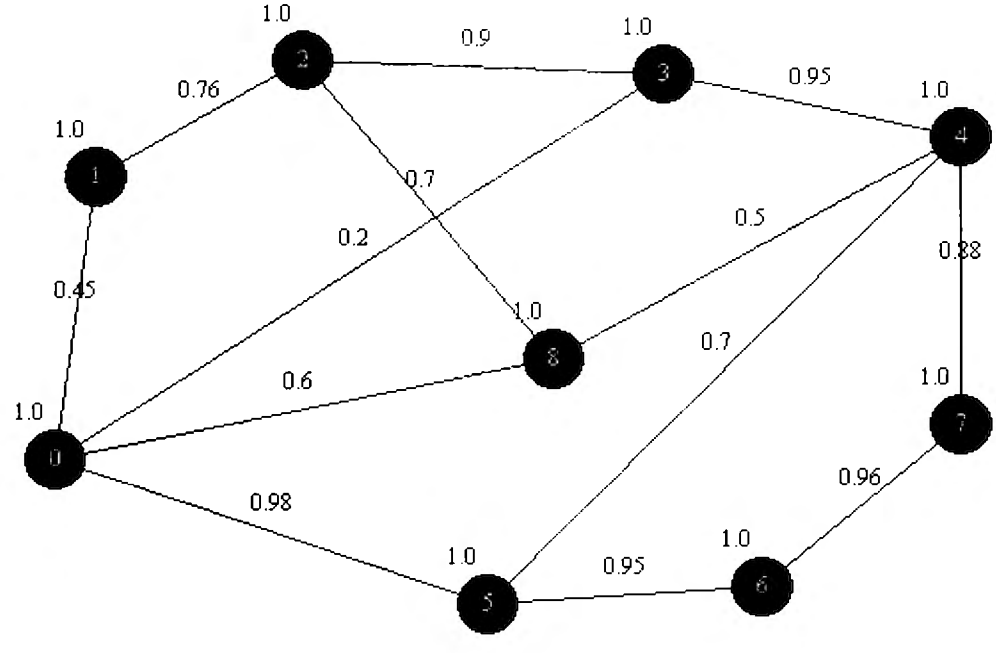

Figura 7.19 Configuración con distintas confiabilidades de enlace

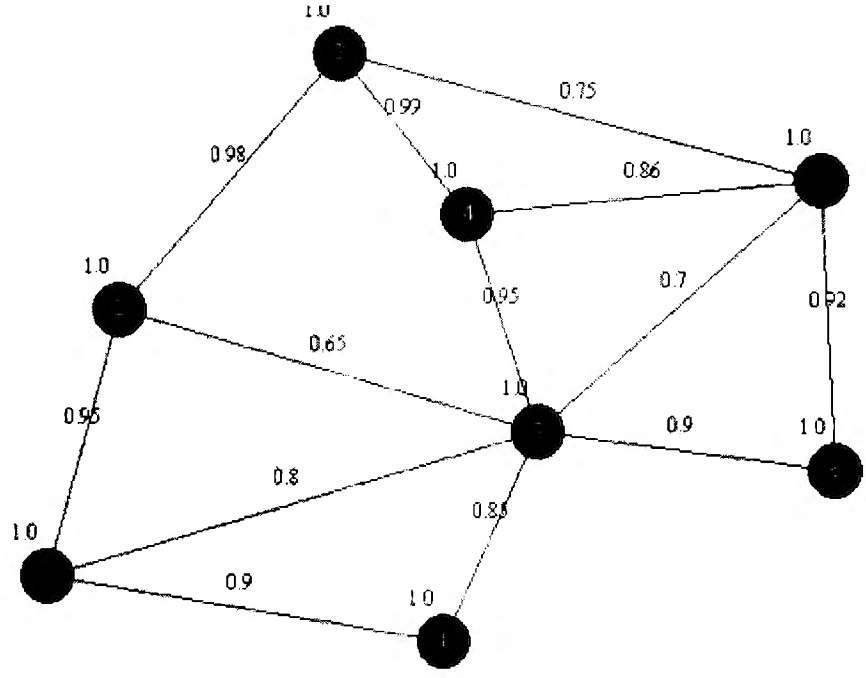

Figura 7.20 Segunda configuración con distintas confiabilidades de enlace

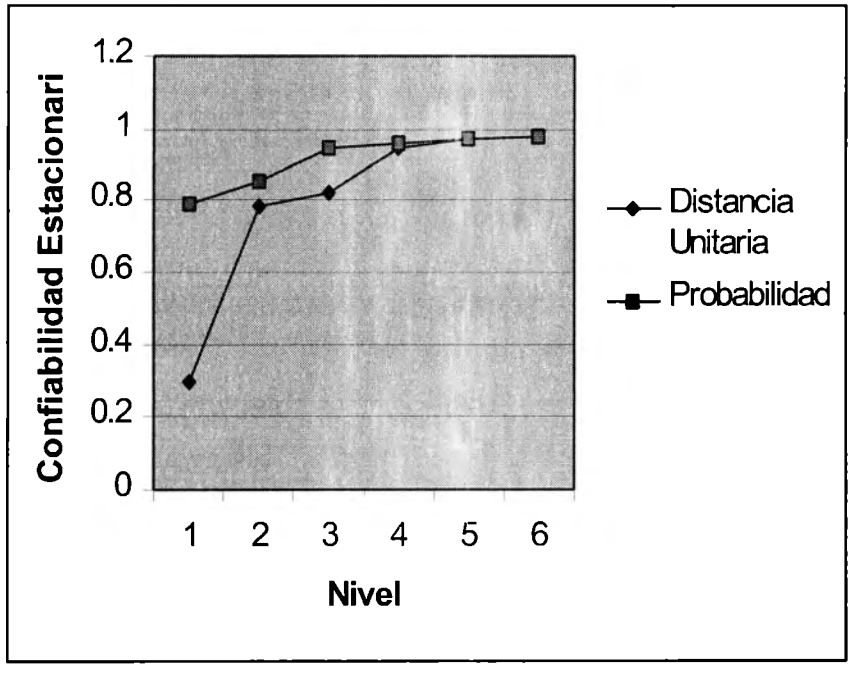

Figura 7.21 Confiabilidades obtenidas con distintas bases de ruteo de la figura 7.19

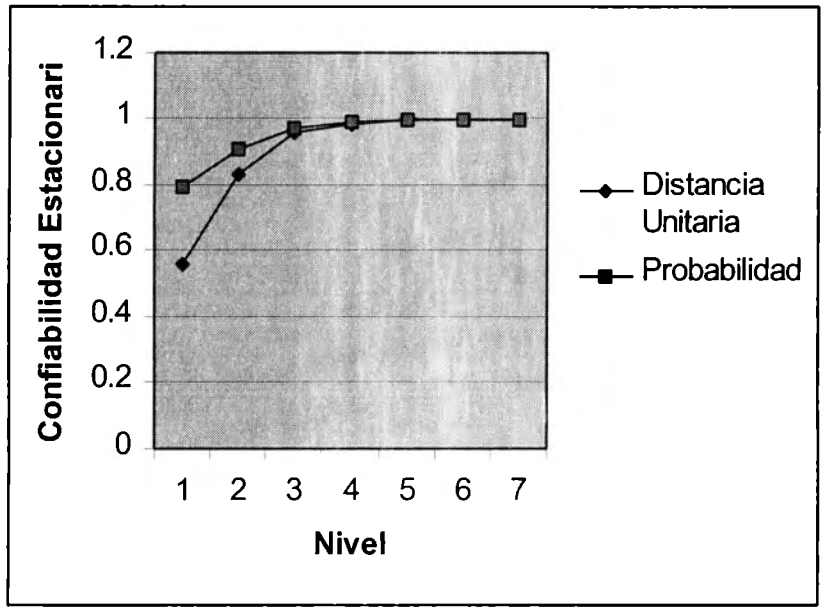

Figura 7.22 Confiabilidades obtenidas con distintas bases de ruteo de la figura 7.20

Este comportamiento distinto también impacta en el número de rutas utilizadas para el cálculo, para la figura 7.20 el número de rutas usadas de acuerdo al ruteo seleccionado se muestra en la figura 7.23.

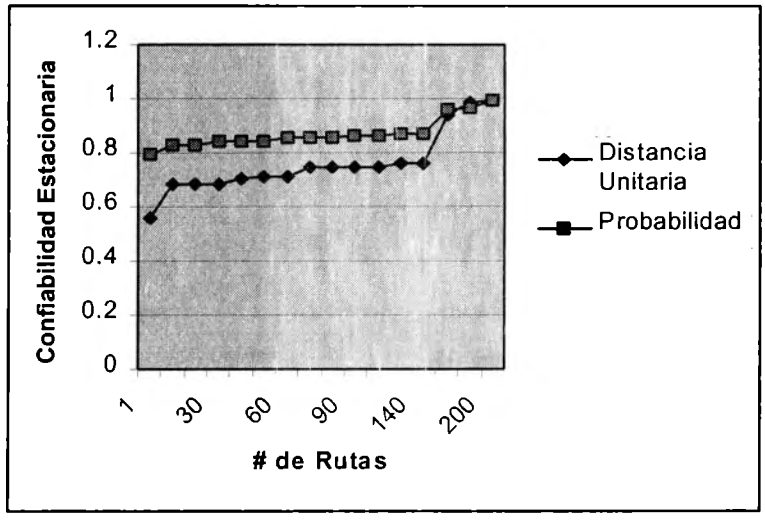

Figura 7.23 Número de rutas usadas de acuerdo al ruteo seleccionado para la figura 7.20

7.3. Análisis de los resultados

Con base en los resultados obtenidos se pueden hacer las siguientes observaciones:

a) El método obtiene resultados muy cercanos a los reales sin necesidad de muchos niveles de profundidad en la obtención de la expresión booleana, con un nivel de 3 o 4 la aproximación es bastante buena.

b) El tiempo necesario para el cálculo, el cual fue tomado en una computadora personal con procesador Pentium MMX a 166 MHz y con 80 MB de RAM, es razonable para configuraciones de tamaño pequeño o medio, aunque conforme el número de elementos y el nivel de profundidad para la obtención de la expresión booleana aumenta el tiempo de cálculo tiende a crecer de manera exponencial.

c) El número de niveles necesarios para obtener una buena aproximación varía dependiendo de la métrica utilizada para la generación de las tablas de ruteo, las cuales son utilizadas para la obtención de la expresión booleana, las métricas comparadas son la distancia unitaria en los enlaces, la cual fue la utilizada en el trabajo en que se basó esta tesis y la otra es la probabilidad que se le asignó a cada enlace.

d) El número de rutas usadas para obtener una aproximación al valor de la confiabilidad también varía de acuerdo a la métrica usada para el ruteo.

e) Estas dos últimas observaciones son ciertas siempre y cuando las probabilidades asignadas a cada enlace sean distintas pues de otra forma se obtienen las mismas rutas independientemente de la métrica usada.

7.4. Conclusiones

Se puede concluir con base en los resultados obtenidos lo siguiente:

a) El método aplicado obtiene resultados bastante aproximados a la realidad (97 a 98%) en un tiempo razonable para configuraciones pequeñas o medias aunque para configuraciones grandes se requiere de algún método de descomposición que reduzca el tamaño del grafo y permita obtener los valores de confiabilidad en menor tiempo; o bien, la modificación de estos algoritmos para que funcionen de manera paralela lo cual mejoraría el rendimiento de los mismos, de hecho, el cálculo de las rutas o de la expresión booleana se presta para su descomposición y distribución entre varias tareas.

b) Aunque la complejidad de utilizar el ruteo de estado de enlace eleva la complejidad temporal de la selección de las rutas de O(e+n), que es la complejidad de la búsqueda en anchura primero utilizada en el algoritmo de la tesis en que se basa este trabajo a O(e log (n)) que es la complejidad del algoritmo de Dijkstra, la generación de ruteo con base en las probabilidades de cada enlace mejora el rendimiento de manera significativa pues permite ejecutar el algoritmo de la expresión booleana con un nivel de profundidad inferior al que se debe tomar con las distancias unitarias de manera que se obtenga una aproximación similar, lo cual disminuye el tiempo de ejecución hasta en un 50%. Esto aparte de que el algoritmo de ruteo no seleccionará todas las rutas posibles en un grafo dado sino solo los que pertenezcan al dominio o sistema autónomo, lo cual de hecho podría utilizarse para generar una especie de descomposición para lograr lo mencionado en el inciso (a).

c) Este modelo y la correspondiente implantación pueden ser muy útiles para diagnosticar un sistema distribuido en cuanto a su red de interconexión, la cual es una de las partes más importantes para su funcionamiento.

ANEXO

CONCEPTOS BASICOS SOBRE GRAFOS

A.1 Terminología

Un **grafo** es un par de conjuntos G = (V,E). Los elementos de V se denominan **vértices** (o **nodos**) del grafo G, los elementos de E son sus **enlaces** (o **arcos** o **aristas**). Un enlace $e \in E$ es un par (u,v) donde $u,v \in V$, este par indica que los vértices u y v están conectados. El conjunto de vértices de un grafo G se representa por V(G) y su conjunto de enlace se representa por E(G). El número de vértices de un grafo es su orden y se denota por |G|, |V(G)| o simplemente |V|. El número de enlaces se denota por ||G||, |E(G)| o |E|. Un grafo puede ser no dirigido o dirigido, la diferencia es que cada par $(u,v) \in E$ es un par ordenado cuando E es de un grafo dirigido, es decir, el par indica de donde a donde es el enlace mientras que en un grafo no dirigido cada enlace es bidireccional. Cada uno de estos tipos de grafo se ilustra a continuación:

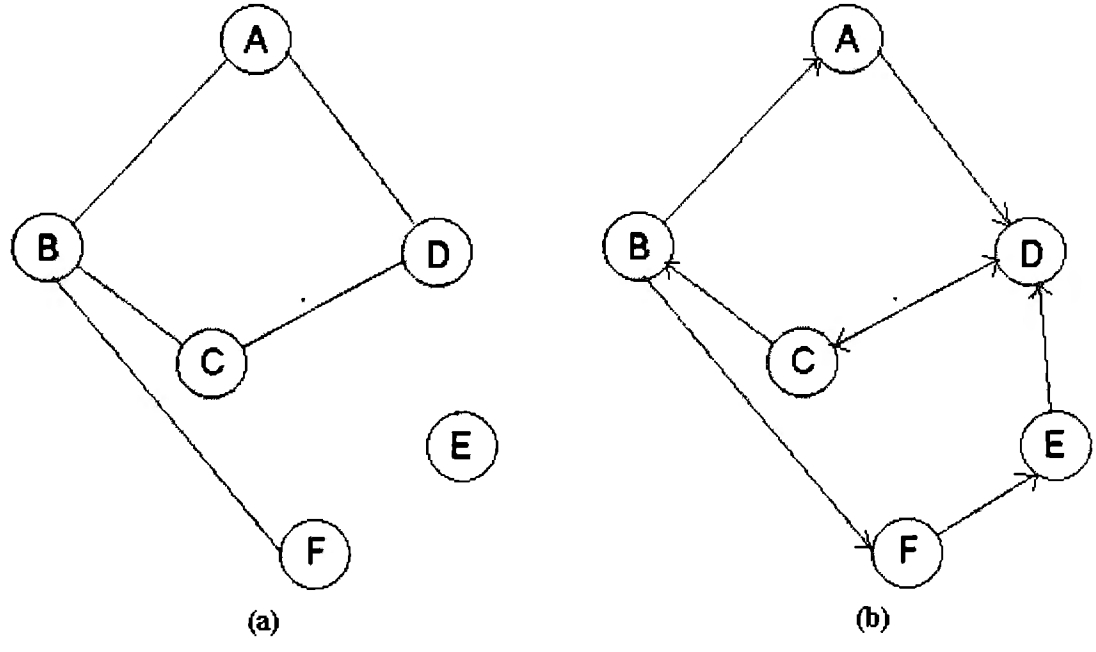

Figura A.1 (a) Grafo no dirigido, (b) Grafo dirigido

Muchas definiciones son comunes a los grafos dirigidos y no dirigidos, aunque ciertos términos tienen diferentes significados para cada uno. Si (u,v) es un enlace en un grafo no dirigido, (u,v) es **incidente sobre** los vértices u y v. Sin embargo, si el grafo es dirigido, entonces el enlace(u,v) es **incidente desde** el vértice u y es **incidente hacia** el vértice v, también se dice en este caso que u es **predecesor** de v y que v es **sucesor** de u. Por ejemplo, en la figura A.1(a) el enlace (B,C) es incidente sobre los vértices B y C, mientras que en la figura A.1(b) el enlace (B,C) es incidente desde B y es incidente hacia C. Si (u,v) es un enlace en un grafo no dirigido, los vértices u y v se dice que son **adyacentes** o **vecinos**, mientras que si el grafo es dirigido se dice que el vértice v es **adyacente** al vértice u. En un grafo no dirigido el **grado** o **valencia** de un vértice v es el número de enlaces incidentes sobre v. En un grafo dirigido existen 2 grados para cada vértice, el **grado de entrada** de un vértice v es el número de enlaces que son incidentes hacia v, mientras que el **grado de salida** del vértice v es el número de enlaces que son incidentes desde v. Un vértice de grado 0 en un grafo no dirigido (o un vértice con grado de entrada y de salida igual a cero en un grafo dirigido) es un vértice **aislado**. Si todos los vértices tienen el mismo grado entonces se dice que es un grafo **regular**.

Dos enlaces son **adyacentes** si tienen un vértice en común. Si todos los vértices de G son adyacentes par a par, entonces G es un grafo **completo**. Un grafo completo con n vértices es un K^n; por ejemplo, K^3 es un triángulo.

Una **ruta** de un vértice v a un vértice u es una secuencia $(v_0,v_1,v_2,...,v_k)$ de vértices donde $v_0=v$, $v_k=u$ y $(v_i,v_{i+1}) \in E$ para i=0,1,2,...,k-1. La **longitud** de una ruta se define como el número de enlaces en la ruta, la **distancia** entre 2 vértices es la longitud de la ruta más corta entre esos 2 vértices y el **diámetro** es la longitud de la ruta más larga entre los 2 vértices. Si existe una ruta de v a u se dice que u se puede alcanzar desde v. Una ruta es **simple** si todos los vértices son distintos. Una ruta forma un **ciclo** si empieza y termina con

el mismo vértice. Un grafo sin ciclos se denomina **acíclico**. Un ciclo es **simple** si todos los vértices intermedios son distintos. Dos rutas son **independientes** si ninguna de ellas contiene los vértices interiores de la otra. Dos rutas son **de enlaces disjuntos** si no tienen enlaces en común.

Dos vértices en un grafo se dice que están **conectados** si existe una ruta del primero al segundo. Un grafo esta conectado si existe una ruta entre cada par de vértices que este contenga.

Se dice que un grafo G'=(V',E') es un **subgrafo** de G=(V,E) si $V' \subseteq V$ y $E' \subseteq E$. Un **subgrafo propio** de G es un grafo que se obtiene al eliminar un número de enlaces y/o vértices de G. La eliminación de un vértice implica la eliminación de todos los enlaces incidentes sobre el mismo. Dado un conjunto $V' \subseteq V$, el subgrafo de G **inducido** por V' es el grafo G'=(V',E') donde $E'=\{ (u,v) \in E \mid u,v \in V' \}$.

Un **árbol** es un grafo conectado acíclico. Un **bosque** es un grupo de árboles desconectados. Un **árbol de expansión** de un grafo es un subgrafo que contiene todos los vértices pero sólo los enlaces necesarios para formar un árbol. Un árbol es un **árbol de Steiner** si se expande sobre un subconjunto de los vértices del grafo.

Un grafo que tiene asociado un peso con cada enlace se denomina **grafo con peso** y se denota por G=(V,E,w) donde $w : E \rightarrow \Re$ es una función de valor real definida sobre E. El peso del grafo se define como la suma de los pesos de sus enlaces. El peso de una ruta es la suma de los pesos de sus enlaces.

A.2 Formas de representación

Existen dos métodos estándar de representación de un grafo en una computadora. El primer método se denomina **matriz de adyacencias** y consiste en lo siguiente: Considere

un grafo G=(V,E) con n vértices numerados 1,2,...,n. La matriz de adyacencias de este grafo es un arreglo de n x n $A= (a_{i,j})$, que se define como sigue:

$$a_{i,j} = \begin{cases} 1 \; si \; (v_i, v_j) \in E \\ 0 \;\; \text{de otro modo} \end{cases}$$

Por ejemplo, la matriz de adyacencias para el grafo de la figura A.1(a) sería:

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Se puede que cuando se representa a un grafo no dirigido la matriz es simétrica. La representación de matriz de adyacencias puede ser modificada para manejar grafos con peso. En este caso, $A= (a_{i,j})$ se define como sigue:

$$a_{i,j} = \begin{cases} w(v_i, v_j) \; si (v_i, v_j) \in E \\ 0 \;\; \text{si i = j} \\ \infty \;\; \text{de otro modo} \end{cases}$$

Esta representación recibe el nombre de **matriz de adyacencias con peso**. El espacio para almacenar la matriz de adyacencias de un grafo con n vértices es $\Theta(n^2)$.

El otro método de representación de un grafo es la **lista de adyacencias**. Esta representación consiste de un arreglo Adj[1..|V|] de listas. Para cada $v \in V$, $Adj[v]$ es una lista enlazada de todos los vértices u tales que G contiene un enlace $(v,u) \in E$. En otras palabras, Adj[v] es una lista de todos los vértices adyacentes a v. La figura siguiente muestra la lista enlazada para la figura A.1(a).

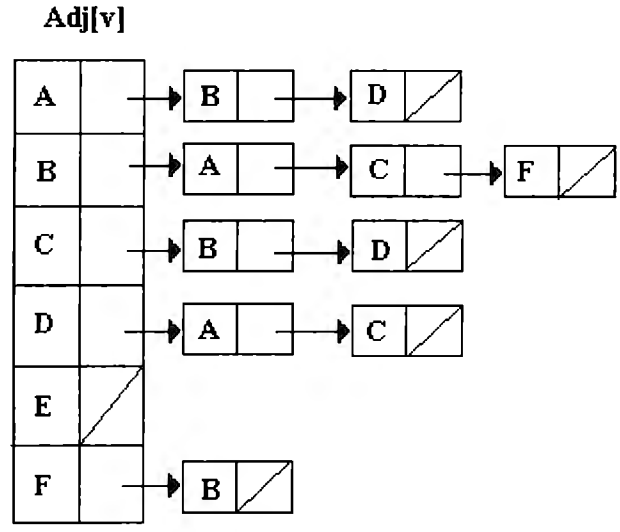

Figura A.2 Lista de adyacencias del grafo de la figura A.1(b)

La representación de lista de adyacencias puede ser modificada para acomodar grafos con peso al almacenar el peso de cada enlace $(v,u) \in E$ en la lista de adyacencias del vértice v. El espacio requerido para almacenar la lista de adyacencia es $\Theta(|E|)$.

La naturaleza del grafo determina que representación debe ser usada. Un grafo G=(V,E) es **no denso** si |E| es mucho más pequeño que $O(|V|^2)$; de otra manera es **denso**. La representación de matriz es útil para grafos densos y la de listas de adyacencia es buena para grafos **no densos**.

BIBLIOGRAFIA

[Aggarwal 96] Aggarwal, Alok; et al. (1996, Noviembre). Efficient Routing in Optical Networks. Journal of the ACM, Vol. 46, 6, 973-1001.

[Aho 74] Aho, Alfred; et al. (1974) The Design and Analysis of Computer Algorithms. Estados Unidos: Addison Wesley.

[Allen 93] Allen Weiss, Mark. (1993). Data Structures and Algorithm Analysis. Estados Unidos: Benjamin Cummings Publishing Co.

[Amstutz 83] Amstutz, S.R. Burst-Switching - An Introduction. (1983, Febrero). IEEE Communications, Vol. 31,2.

[Anderson 90] Anderson, T.; Lee, P.A. (2^{nd} Edition).(1990). Fault Tolerance: Principles and Practice. Estados Unidos: Springer-Verlag.

[Ash 81] Ash, G.R.; et al. (1981, Octubre). Design and optimization of networks with dynamic routing. Bell System Technical Journal, 60, 1787-1820

[Ash 92] Ash, G.R.; et al. (1992). Real-time network routing in the AT&T network, improved service quality at lower cost. En Proceedings of Globecom '92.

[Bannister 95] Bannister, Joseph; Gerla, Mario; Kovacevic, Milan. Routing in Optical Networks. En: Steenstrup, Martha E. (1995) Routing in Communication Networks. Estados Unidos: Prentice Hall.

[BCS 95] British Computer Society. (1995). Standards for Software Component Testing. Working Draft 3.0 BCS Specialist Group in Software Testing. Inglaterra.

[Bella 98] Bella, Luigu; et al. (1998) Performance Evaluation of dynamic routing based on the use of satellites and intelligent networks. Wireless Networks, 4, 167-180

[Bertsekas 92] Bertsekas, Dimitri; Gallager, Robert. (2^a ed.). (1992). Data Networks. Estados Unidos: Prentice Hall.

[Black 92] Black, Uyless. (1992). TCP/IP and Related Protocols. Estados Unidos: McGraw Hill.

[Bowen 92] Bowen, J.P.; Stavridou, V. (1992). Safety-critical systems, formal methods and standards. En: Proceedings Safecomp '92: Safety of Computer Control Systems. Zurich: Pergamon Press.

[Bowen 93] Bowen, J.P.; Stavridou, V. (1992, Octubre). Formal methods and software safety. Software Engineering Journal, Vol. 8, 4, 189-209.

[Bulteau 97] Bulteau, Stéphane; Rubino, Gerardo. (1997) Evaluating network vulnerability with the mincuts frequency vector. Reporte de Investigación 3125. Francia: INRIA.

[Cancela 95] Cancela, Hector; El Khadiri, Mohamed. (1995). Recursive Path Conditioning Monte Carlo Simulation of Communication Network Reliability. Francia: INRIA.

[CISCO 97] Internetworking Technology Overview. (1997) Cisco Systems, Inc. URL: http://www.cisco.com/univercd/cc/td/ doc/csintwk/ito_doc.

[Colbourn 91] Colbourn, C.J.; Litvak, Eugene I. Bounding Network Parameters by Approximating Graphs. En: Roberts, Fred; et al. (1991). Reliability of Computer and Communications Network. Estados Unidos: American Mathematical Society. ACM.

[Colbourn 92] Colbourn, C.J.; et al. (1992) Network Reliability. Estados Unidos: Reporte Técnico TR 92-74. Institute for Systems Research.

[Colbourn 95] Colbourn, C.J.; et al. (1995). Network Reliability. Experiments with a Symbolic Algebra Environment. Estados Unidos: CRC Press.

[Comer 95] Comer, Douglas E. (3ª ed.).(1995). Internetworking with TCP/IP Vol I: Principles, Protocols and Architecture. Estados Unidos: Prentice Hall.

[Cormen 90] Cormen, Thomas H.; et al. (1990). Introduction to Algorithms. Estados Unidos: MIT Press.

[Coulouris 95] Coulouris, George; Dollimore, Jean; Kindberg, Tim (2ª ed.). (1995). Distributed Systems. Concepts and Design. Estados Unidos: Addison Wesley.

[Cristian 89] Cristian, Flaviu. Exception Handling. En: Anderson, T. (1989). Dependability of Resilient Computers. Blackwell Scientific Publications.

[Cristian 91] Cristian, Flaviu. (1991, Febrero) Understanding fault-tolerant distributed systems. Communications of the ACM, Vol. 34, 2, 56-78

[Cullyer 94] Cullyer, W.J.; Storey, Neil. (1994). Tools and Techniques for the testing of safety-critical software. Computer Control Engineering Journal, Vol 5, 5, 239-244.

[Chapra 87] Chapra, Steven C.; Canale, Raymond P. (1987). Métodos Numéricos para Ingenieros. Estados Unidos: McGraw Hill

[Chien 95] Chien, Andrew A.; Kim, Jae H. (1995, Enero). Planar-Adaptive Routing: Low-cost Adaptive Networks for Multiprocessors. Journal of the ACM, Vol. 42, 1, 91-123.

[Dalal 78] Dalal, Y.K.; Metcalfe, R.M. (1978, Diciembre). Reverse Path Forwarding of Broadcast Packets. Communications of the ACM, Vol. 21, 1040-1048.

[Deering 88] Deering, S.E. (1988, Agosto). Multicast Routing in Internetworks and Extended LANs. Computer Communications Review, Vol. 18, 4, 55-64.

[Diestel 97] Diestel, Reinhard. (1997). Graph Teory. Estados Unidos: Springer-Verlag.

[Fencott 92] Fencott, C.; Fleming, C. (1992). Practical formal methods for process control engineering. En: Proceedings Safecomp '92: Safety of Computer Control Systems. Zurich: Pergamon Press.

[Gabow 91] Gabow, Harold N. (1991). A Matroid Approach to Finding Edge Connectivity and Packing Arborescences. ACM Proceedings on Theory of Computation 1991.112-122.

[Garcia 97] García Luna Aceves, J.J. (1997, Febrero). A Path Finding Algorithm for Loop-Free Routing. IEEE/ACM Transactions on Networking, Vol. 5, 1, 148-160.

[Gavoille 96] Gavoille, Cyrill; Pérennès, Stéphane (1996). Memory Requirements for Routing in Distributed Networks. ACM Proceedings of Distributed Computing 1996. 125-133.

[Gibbens 95] Gibbens, Richard J.; et al. Dynamic Alternative Routing. En: Steenstrup, Martha E. (1995) Routing in Communication Networks. Estados Unidos: Prentice Hall.

[Girard 83] Girard, A.; Côté, Y. (1983). Sequential routing optimization for circuit switched networks. En: Proceedings of the 10th ITE.

[Green 92] Green, P.E. (1992). Fiber-Optic Networks.Estados Unidos: Prentice Hall.

[Grimaldi 89] Grimaldi, Ralph P. (2nd Edition) (1989). Discrete an Combinatorial Mathematics.

[Gupta 95] Gupta, Sanjay; et al. On Routing in ATM Networks. En: Steenstrup, Martha E. (1995) Routing in Communication Networks. Estados Unidos: Prentice Hall.

[Guy 91] Guy, C.G. Errors, reliability and redundancy. En: Holdsworth, B., Martin, G.R. (1991). Digital Systems Reference Book. Oxford: Butterworth-Heinemann.

[Hedrick 88] Hedrick, C. (1988, Junio). RFC 1058: Routing Information Protocol. Bajado el 20 de marzo de 1998 del World Wide Web: http://www.cis.ohio-state.edu/htbin/rfc/rfc1058.html

[Hernández 97] Hernández, D., León; C., Rukoz, M. (1997). Integración del Modelo PAC y el Lenguaje Java para el Desarrollo de Sistemas Interactivos. En Memoria del Simposio Internacional de Computación "Nuevas Aplicaciones e Innovaciones Tecnológicas en Computación". 332-338.

[Hiltunen 96] Hiltunen, Matti Arno. (1996). Configurable Fault-Tolerant Distributed Services. Ph. D. Dissertation. Estados Unidos: Universidad de Arizona.

[Horning 74] Horning, J.J.; et al. (1974). A program structure for error detection and recovery. En: Proceedings of the International Sysmposium on Operating Systems, Francia, 171-187.

[Horvilleur 97] Horvilleur, Gerardo; Hardy, Martin. (1997, Marzo 15). ¿Por qué Java es tan Importante?. *Soluciones Avanzadas*, 43, 43-48.

[Hui 90] Hui, J.Y. Switching and Traffic Theory for Integrated Broadband Networks, Estados Unidos: Kluwer Publishers, 1990.

[Huitema 95] Huitema, Christian (1995) Routing in the Internet. Estados Unidos: Prentice Hall.

[Johnson 89] Johnson, B.W. (1989). Design and Analysis of Fault Tolerant Digital Systems. Estados Unidos: Addison Wesley.

[Karger 95] Karger, David R. (1995). A Randomized Fully Polinomial Time Approximation Scheme for the All Terminal Network Reliability Problem. Estados Unidos: ACM STOC '95.

[Kermani 79] Kermani, P.; Kleinrock, L. (1979). Virtual Cut-Through: A New Computer Communication Switching Technique. Computer Networks, Vol. 3, 267-286.

[Ketchum 95] Ketchum, John W. Routing in Cellular Mobile Radio Communications Networks. En: Steenstrup, Martha E. (1995) Routing in Communication Networks. Estados Unidos: Prentice Hall.

[Key 88] Key, P.B.; Whitehead, M.J. (1988, Junio). Cost-effective use of networks employing dynamic alternative routing. En: Proceedings of the 12th Teletraffic Conference, Torino, Italia.

[Kopetz 93] Kopetz, Hermann; Veríssimo, Paulo. Real Time and Dependability Concepts. En Mullender, Sape (2ª ed.).(1993). Distributed Systems. Estados Unidos: Addison Wesley.

[Laprie 92] Laprie, J.C. (1992). Dependability: Basic Concepts and Terminology. Vienna: Springer-Verlag.

[Lauer 95] Lauer, Gregory S. Packet Radio Routing. En: Steenstrup, Martha E. (1995) Routing in Communication Networks. Estados Unidos: Prentice Hall.

[Lewis 96] Lewis, E.E. (2nd Edition) (1996). Introduction to Reliability Engineering. New York: John Wiley and Sons.

[Lynch 93] Lynch, Daniel C.; Rose, Marshal T. (1993). Internet System Handbook. Estados Unidos: Addison Wesley.

[Miller 65] Miller, R.E. (1965). Switching Theory, vol. I: Combinatorial Circuits. Estados Unidos: John Wiley and Sons.

[Morison 82] Morison, J.D.; et al. (1982). ELLA: a hardware description language. En: Proceedings IEEE International Conference on Circuits and Computers. 604-607.

[Moy 91] Moy, John T. (1991, Julio). RFC 1245: OSPF Protocol Analysis. Bajado el 24 de agosto de 1998 del World Wide Web: http://www.cis.ohio-state.edu/htbin/rfc/rfc1245.html

[Moy 95] Moy, John T. Link-State Routing. En: Steenstrup, Martha E. (1995) Routing in Communication Networks. Estados Unidos: Prentice Hall.

[Moy 97] Moy, J. (1997, Julio). RFC 2178: OSPF Version 2. Bajado el 20 de marzo de 1998 del World Wide Web: : http://www.cis.ohio-state.edu/htbin/rfc/rfc2178.html

[Moy 98] Moy, John T. (1998) OSPF. Anatomy of an Internet Routing Protocol. Estados Unidos: Addison Wesley.

[Nakamura 92] Nakamura, Shoichiro. (1992). Métodos Numéricos Aplicados con Software. Estados Unidos: Prentice Hall.

[NASA 95] NASA Lewis Research Center. (1995, Mayo). Reliability Block Diagrams and Reliability Modeling. Bajado el 22 de marzo de 1998 del World Wide Web: http://www-osma.lerc.nasa.gov/rbd.

[Ogier 89] Ogier, R., Shacham, N. (1989, Abril). A Distributed Algorithm for Finding Shortest Pairs of Disjoint Paths. Proceedings of the IEEE INFOCOM '89, 173-182.

[Pack 90] Pack, C.D.; Olson, D.W. Advanced routing techniques using advanced intelligent network functional components and data base controls. En: Kershenbaum, A., et al. (1990) Network Management and Control, Estados Unidos: Plenum Press.

[Ramanathan 96] Ramanatha, S.; Steenstrup, Martha. (1996). A survey of routing techniques for mobile communications networks. Mobile Networks and Applications,1, 89-104.

[Randell 75] Randell, B. (1975, Junio). System structure for software fault tolerance. IEEE Transactions on Software Engineering. Vol I, 2, 220-232.

[Rushby 93] Rushby, J. (1993). Formal methods and the certification of critical systems. Estados Unidos. Reporte Técnico CSL-93-7. SRI International.

[Rushby 94] Rushby, J. (1994, Febrero). Critical Systems Properties: Survey and Taxonomy. En: Reliability Engineering and System Safety, Vol. 43, 2, 189-219.

[Scheideler 96] Scheideler, Christian; Vöcking, Berthold (1996). Universal Continuos Routing Strategies. Proceedings of Parallel Algorithms and Architectures. 142-151.

[Schwartz 80] Schwartz, M.; Stern, T.E. (1980, Abril) Routing Techniques Used in Computer Communications Networks, IEEE Transactions on Communications, Vol. 28, 4, 539-552

[Scott 95] Scott Malkin, Gary; Steenstrup, Martha E. Distance-Vector Routing. En: Steenstrup, Martha E. (1995) Routing in Communication Networks. Estados Unidos: Prentice Hall.

[Sedgewick 88] Sedgewick, Robert (1988). Algorithms. Estados Unidos: Addison Wesley.

[Shahdad 86] Shahdad, M. (1986). An overvier of VHDL language and technology. En: Proceedings 23rd ACM/IEEE Design Automation Conference. 320-326.

[Shier 91] Shier, Douglas R. Algebraic Methods for Bounding Network Reliability. En: Roberts, Fred et al. (1991). Reliability of Computer and Communications Network. Estados Unidos: American Mathematical Society. ACM.

[Stallings 94] Stallings, William. (4ª ed.).(1994). Data and Computer Communications. Estados Unidos: Prentice Hall.

[Steenstrup 95] Steenstrup, Martha E.(1995) Routing in Communications Networks. Estados Unidos: Prentice Hall.

[Storey 96] Storey, Neil. (1996). Safety-Critical Computer Systems. Estados Unidos: Addison Wesley.

[Suurballe 74] Suurballe, J.W. (1974). Disjoint Paths in a Network. En: Networks, Vol. 4, 125-145.

[Tanenbaum 97] Tanenbaum, Andrew S. (3ª ed.).(1997). Redes de Computadoras. Estados Unidos: Prentice Hall.

[Topkis 88] Topkis, D.M. (1988, Julio). A Shortest Path Algorithm for Adaptive Routing in Communications Networks", IEEE Transactions on Communications, Vol. 36, 7, 855-859.

[Torrieri 92] Torrieri, D. (1992, Noviembre). Algorithms for Finding an Optimal Set of Short Disjoint Paths in a Communications Network. IEEE Transactions on Communications, Vol. 40, 11, 1698-1702.

[Turner 86] Turner, J.S. (1986, Octubre). New Directions in Communications (or Which Way to the Information Age?). IEEE Communications, Vol. 24, 8-15.

[Valdivia 89] Valdivia B., Roberto. (1989). Reliability and Fault Tolerance Modelling of Multiprocessor Systems. Ph. D. Dissertation. Universidad de Brunel, Inglaterra.

[Vizcaíno 97] Vizcaíno Sahagún, Carlos. (1997, Mayo 15). Java es Productividad. Soluciones Avanzadas, 45, 6-7.

[Wald 83] Wald, J.A.; Colbourn, C.J. (1983). Steiner trees in probabilistic networks, Microelectronics and Reliability 23, 837-840.

[Wood 95] Wood, R.K.; Satyanarayana, A. (1985). A linear time algorithm for computing k-terminal reliability in series-parallel networks. SIAMS Journal, 14, 818-832.