Universitat Politècnica de València
Departamento de Informática de Sistemas y Computadores

Una Nueva Familia de Topologías Indirectas, Eficientes y Tolerantes a Fallos

Tesis Doctoral

Presentada por

Diego Fernando Bermúdez Garzón

Dirigida por

Dra. María Engracia Gómez Requena
Dr. Pedro Juan López Rodríguez

Valencia, Diciembre de 2015

Esta tesis ha sido escrita usando LaTeX. Las Figuras han sido realizadas con InkScape y GNUPLOT.

Agradecimientos

La culminación de este trabajo ha sido un gran reto, el cual no hubiera podido lograr sin la ayuda de todas aquellas personas que me han apoyado a lo largo de este camino. Por ello, quiero aprovechar este espacio para agradecer a cada uno de ellos su valiosa colaboración.

En primer lugar, doy gracias a Dios por darme la fuerza necesaria para continuar y finalizar mis estudios.

A mis padres y mis hermanos por estar siempre presentes brindándome su apoyo y nunca dejar de creer en mí.

A Karen Sulay por su apoyo incondicional a lo largo de estos años y por estar siempre a mi lado.

A mis tutores, Pedro López y María Engracia Gómez, por aceptarme en su grupo de investigación y permitirme llevar a cabo este trabajo. Por sus consejos, enseñanzas y colaboración durante la elaboración de esta tesis.

A Ricardo Marín, por su gran labor en el laboratorio del GAP, por mantener todos los equipos 'a punto' para que cada uno de los integrantes pueda realizar sus investigaciones sin ningún contratiempo.

A todos mis compañeros del GAP, por hacer que la estancia en el laboratorio fuese más amena.

Índice general

Índice de figuras

Índice de tablas

Abstract

Large parallel computers are currently adopting the cluster architecture as the basis for their construction. These machines are being built with thousands of processing nodes that are interconnected through high-speed interconnection networks.

Performance, fault tolerance and network cost are key factors in the overall design of these systems. The levels of computing power required can only be reached by increasing the number of network nodes. As systems grow, however, so does the amount of network components and with it, the probability of network faults. Since availability is important with these computer systems, fault-tolerance mechanisms are often implemented that are based on increasing network size and duplicating components, which directly affects cost.

In the field of interconnection networks, indirect topologies are often the design of choice for HPC systems. The most commonly used indirect topology is the fat-tree, which is a multi-stage bidirectional-link topology providing good network performance and high fault-tolerance levels, but at a high cost. To reduce cost, RUFT has been proposed, a multi-stage unidirectional topology providing network performance similar to that of the fat-tree but using fewer hardware resources (approximately half). RUFTs weak point, however, is that it has zero fault tolerance.

This work focuses on designing a simple indirect topology that offers high performance and fault tolerance while keeping hardware cost as low as possible. In particular, we propose a set of new topologies with different properties in terms of cost, performance and fault tolerance. All of them are capable of achieving performances similar to or better than that of the fat-tree, while also providing good fault tolerance levels and tolerating faults in the links

connecting to end nodes, which most available topologies cannot do.

Our first contribution is RUFT-PL, a topology that duplicates the number of injection, network and ejection links, while using the RUFT connection pattern to interconnect all network elements. This topology provides high network performance and a slight level of fault tolerance, using the same hardware resources as a fat-tree.

Our second contribution is the FT-RUFT-212 topology, which provides better network performance than the fat-tree, as well as good fault tolerance for a low design cost, thanks to the proposed injection/ejection system implemented by the processing nodes.

The third contribution, FT-RUFT-222, is a topology combining the best properties of the previous two proposals. In particular, this topology implements the injection/ejection used by FT-RUFT-212 and the double network links used by RUFT-PL to interconnect the switches. It provides high performance and fault-tolerance levels while using the same hardware resources required by a fat-tree.

Our fourth and last contribution is FT-RUFT-XL, a topology in which both the injection/ejection and the connection between the switches have been redesigned. It offers a significant improvement on the other proposals' fault-tolerance levels, and also provides high network performance. Furthermore, unlike many unidirectional topologies, it allows packets to take different routes at every network stage, always bringing them closer to their destination with each hop.

Resumen

Actualmente, los grandes sistemas de cómputo paralelo están adoptando la arquitectura de *cluster* como base de su construcción (lista Top500). Estos *clusters* están siendo construidos con miles de nodos de procesamiento, los cuales se conectan a través de una red de interconexión de altas prestaciones.

En estos sistemas, el rendimiento, la tolerancia a fallos y el coste de la red juegan un factor clave en el diseño de todo el sistema. Los niveles de cómputo requeridos solo pueden ser alcanzados incrementando el número de nodos que lo componen. Sin embargo, a medida que el sistema crece también lo hace la cantidad de componentes de la red, y con ello la probabilidad de un fallo en la misma. Dado que la disponibilidad de estos sistemas es una preocupación, los mecanismos de tolerancia a fallos son implementados regularmente basados en el aumento y replicación de componentes, afectando de forma directa a su coste.

En este campo, las topologías indirectas a menudo son elegidas en el diseño de *clusters* de alto rendimiento. Entre ellas, la más utilizada es el fat-tree, la cual es una topología bidireccional multietapa que provee un buen rendimiento de red y un buen nivel de tolerancia a fallos, pero a un alto coste. Para reducir su coste, se propuso RUFT, una topología unidireccional multietapa que obtiene un rendimiento de red similar al fat-tree, utilizando menos recursos de hardware (aproximadamente la mitad). Sin embargo, el punto débil de RUFT es que no ofrece ningún tipo de tolerancia a fallos.

En este trabajo, nos enfocamos en diseñar una topología indirecta que ofrezca un alto rendimiento de red y sea tolerante a fallos, a la vez que mantiene un bajo coste del hardware.

En particular, proponemos una nueva familia de topologías indirectas con

diferentes propiedades en términos de coste, rendimiento y tolerancia a fallos. Estas nuevas topologías son capaces de alcanzar un rendimiento similar o mejor al ofrecido por el fat-tree, además de ofrecer un buen nivel de tolerancia a fallos y, a diferencia de la mayoría de topologías disponibles, también son capaces de tolerar fallos en los enlaces que conectan con los nodos de procesamiento.

Nuestra primera contribución es RUFT-PL, una topología que duplica los enlaces de inyección, red y eyección, siguiendo el mismo patrón de conexión utilizado por RUFT para interconectar todos los elementos de la red. Esta topología obtiene un alto rendimiento de red y un ligero grado de tolerancia a fallos, usando los mismos recursos de hardware que el fat-tree.

Como segunda contribución, proponemos la topología FT-RUFT-212. Esta topología incrementa el rendimiento de red con respecto al fat-tree, ofreciendo además un buen nivel de tolerancia a fallos a un bajo coste de diseño, gracias al sistema de inyección/eyección propuesto que implementan los nodos de procesamiento.

La tercera contribución, FT-RUFT-222, es una topología que aprovecha las mejores propiedades de las dos propuestas anteriores. En particular, esta topología implementa la inyección/eyección utilizada por FT-RUFT-212 y los dobles enlaces de red de RUFT-PL para conectar los conmutadores. Esta propuesta ofrece un alto rendimiento de red y de tolerancia a fallos, utilizando los mismos recursos de hardware requeridos por el fat-tree.

Nuestra última contribución es FT-RUFT-XL, una topología que rediseña tanto la inyección/eyección como la conexión entre los conmutadores. Esta topología incrementa notablemente el nivel de tolerancia a fallos ofrecido por las demás propuestas, ofreciendo también un alto rendimiento de red. Además, a diferencia de muchas topologías unidireccionales, ésta permite que los paquetes tomen diferentes rutas en cada etapa de la red, acercándolos siempre a su destino en cada salto.

Resum

Actualment, els grans sistemes de còmput paral·lel estan adoptant l'arquitectura *cluster* com a base per la seua construcció (Llista Top500). Aquests *clusters* estan sent construïts amb milers de nodes de processament, els quals es connecten mitjançant una xarxa d'interconnexió d'altes prestacions.

En aquests sistemes, el rendiment, la tolerància a fallades i el cost de la xarxa són un factor clau en el disseny de tot el sistema. Per altra banda, els nivells de còmput requerits només poden ser aconseguits incrementant el nombre de nodes que componen el *cluster*. Per tant, a mesura que el sistema creix també ho fa la quantitat de components de la xarxa, i amb això la probabilitat d'una fallada en la mateixa. Atès que la disponibilitat d'aquests sistemes és una gran preocupació, és habitual que les xarxes d'interconnexió implementen mecanismes de tolerància a fallades, que solen consistir en l'augment i replicació de components, incrementant el cost total de la xarxa.

En aquest camp, les topologies indirectes sovint són triades en el disseny de *clusters* d'alt rendiment. Entre elles, la més utilitzada és el fat-tree, una topologia bidireccional multietapa que presenta un bon rendiment de xarxa i un bon nivell de tolerància a fallades, però a un alt cost. Per reduir aquest cost, es va proposar RUFT, una topologia unidireccional multietapa que obté un rendiment de xarxa similar al fat-tree utilitzant menys recursos hardware (aproximadament la meitat). No obstant això, el punt feble de RUFT és que no ofereix cap tipus de tolerància a fallades.

En aquest treball, ens centrem en dissenyar una topologia indirecta que, per una banda, aconseguisca un alt rendiment de xarxa i siga tolerant a fallades i, per altra banda, tinga un baix cost.

Concretament, proposem una nova família de topologies indirectes amb di-

ferents propietats pel que fa a cost, rendiment i tolerància a fallades. Aquestes noves topologies obtenen un rendiment similar o millor al que ofereix el fat-tree, a més d'oferir un bon nivell de tolerància a fallades. A més, a diferència de la majoria de topologies disponibles, toleren fallades en els enllaços que connecten amb els nodes de processament.

La nostra primera contribució és RUFT-PL, una topologia que duplica els enllaços d'injecció, xarxa i ejecció, seguint el mateix patró de connexió utilitzat per RUFT per interconnectar tots els elements de la xarxa. Aquesta topologia obté un alt rendiment de xarxa i un lleuger grau de tolerància a fallades, emprant els mateixos recursos de hardware que el fat-tree.

Com a segona contribució, proposem la topologia FT-RUFT-212. Aquesta topologia incrementa el rendiment de xarxa respecte al fat-tree, oferint a més a més un bon nivell de tolerància a fallades amb un baix cost de disseny, gràcies al sistema d'injecció/ejecció proposat que implementen els nodes de processament.

La tercera contribució, FT-RUFT-222, és una topologia que aprofita les millors propietats de les dues propostes anteriors. En particular, aquesta topologia implementa la injecció/ejecció utilitzada per FT-RUFT-212 i els dobles enllaços de xarxa de RUFT-PL per a connectar els commutadors. Aquesta proposta ofereix un alt rendiment de xarxa i de tolerància a fallades, utilitzant els mateixos recursos hardware requerits pel fat-tree.

La nostra última contribució és FT-RUFT-XL, una topologia que redissenya tant la injecció / ejecció com la connexió entre els commutadors. Aquesta topologia incrementa notablement el nivell de tolerància a fallades oferit per les altres propostes, presentant alhora un elevat rendiment de xarxa. A més a més, a diferència de moltes topologies unidireccionals, aquesta permet que els paquets prenguin rutes diferents en cada etapa de la xarxa, acostant-se sempre al seu destí en cada salt.

Capítulo 1

Introducción

"Da tu primer paso ahora... No importa que no veas el camino completo. Sólo da tu primer paso y el resto del camino irá apareciendo a medida que camines."

Martín Luther King

En la Sección 1.1 del presente capítulo se describen las razones que han motivado esta tesis doctoral. Luego, en la Sección 1.2, definimos los objetivos de la misma. Finalmente, en la Sección 1.3, concluimos este capítulo presentando la estructura y desarrollo que se llevará a lo largo de esta memoria.

1.1. Motivación

La computación de altas prestaciones, mejor conocida como *High Performance Computing* o por su sigla en inglés HPC, se define como el uso del procesamiento paralelo para ejecutar aplicaciones de manera eficiente, fiable y rápida.

Hoy en día, se requiere el uso de grandes sistemas de cómputo con procesamiento paralelo y gran cantidad de nodos para resolver problemas de gran envergadura. Anteriormente, el uso de estos sistemas solo estaba asociado a grandes proyectos militares o gubernamentales. Un ejemplo de ello es el supercomputador Pleiades [7] de la NASA usado para el análisis de datos captados

por sus satélites. Sin embargo, dado el avance tecnológico que estamos viviendo, los tenemos más presentes que nunca en la sociedad actual.

Hoy contamos con investigadores participando en la secuenciación del genoma humano, el diseño de medicamentos, análisis de productos o la gestión de riesgos; equipos de investigación trabajando en ciencias de la vida que requieren entornos técnicos de alto rendimiento con capacidad de procesar grandes cantidades de datos y soportar simulaciones y análisis cada vez más sofisticados. Organizaciones que ayudan a encontrar las causas y curas para enfermedades necesitan la velocidad, la agilidad y el control a lo largo del ciclo de vida del desarrollo clínico para aumentar la productividad, fomentar la innovación y competir de manera más efectiva.

Debido a la gran demanda de estos sistemas por parte de la sociedad, se han desarrollado grandes supercomputadores con miles de nodos de procesamiento e interconectados entre sí usando redes de alto rendimiento. El uso de estos supercomputadores, normalmente, está destinado al sector industrial privado, a la investigación y otros a nivel gubernamental.

La información de los supercomputadores más potentes del mundo la podemos encontrar en la lista del Top500 (*Top500 List*) [10]. Estos sistemas suelen clasificarse según su arquitectura. En la Figura 1.1 podemos ver la distribución de las arquitecturas usadas en estos supercomputadores a lo largo de los últimos años. Como se puede apreciar, más del 85 % de los supercomputadores listados en el *Top500* de junio de 2015 trabajan con arquitecturas basadas en *clusters*.

Un *cluster* está compuesto por una colección de computadores interconectados, que combinan sus capacidades y prestaciones para ofrecer una imagen de un único sistema de gran potencia de cómputo. La arquitectura *cluster* también se utiliza en los grandes servidores de internet, utilizados diariamente por miles de personas.

Hoy en día, cualquier persona puede hacer uso de estos sistemas basados en *cluster* a través de empresas [1, 9] dedicadas a prestar servicios basados en la computación en la nube (*cloud computing*). Por estos motivos, esta tesis se centrará en la computación de altas prestaciones basada en *clusters*.

Como hemos visto, la computación de altas prestaciones basada en *clusters* está ampliamente extendida. Este crecimiento ha traído consigo una serie de

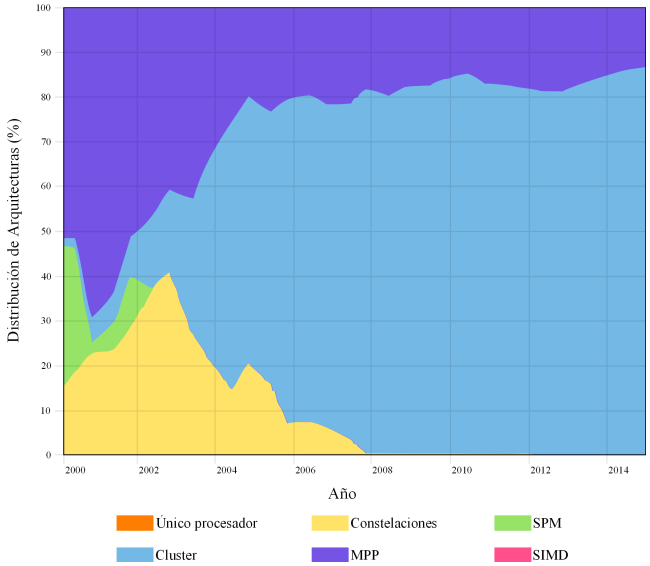


Figura 1.1: Arquitecturas usadas por supercomputadores en el *Top500 List*.

retos al campo de las redes de interconexión. Estos retos se pueden dividir en dos puntos: *1*) reducir el coste total del sistema, y *2*) proporcionar una red de interconexión de altas prestaciones que ofrezca tolerancia a fallos y que afecte lo mínimo posible el rendimiento del sistema en presencia de fallos.

El primer desafío a la hora de implementar un *cluster* de altas prestaciones está relacionado con el coste total del sistema. La cantidad de elementos que lo componen es enorme, entre ellos tenemos: nodos de procesamiento, racks, servidores de discos, servidores de respaldo, KVMs, PDUs, elementos de red tales como conmutadores, enlaces, interfaces de red, etc. Por este motivo, a la hora de diseñar un *cluster* de tales características se debe tener en cuenta la forma de reducir el coste del sistema sin afectar las prestaciones previstas, o al menos evitar que el coste se incremente sin obtener un rendimiento extra a cambio. El coste de la red de interconexión suele llevarse gran parte del presupuesto de diseño, ya que a medida que se incrementa el número de nodos de procesamiento, también lo hace el número de elementos de red, teniendo en cuenta además que muchos de estos elementos suelen replicarse para ofrecer tolerancia a fallos. Este último factor es de vital importancia, ya que un *cluster* de cientos de millones de dólares no se puede permitir estar parado a causa de

Tecnología de interconexión	**Gigabit Ethernet**	**Infiniband**
Cluster	Cluster Platform SL230s	Darwin
CPU	Xeon E5-2670 8C	Xeon E5-2670 8C
Frecuencia del reloj	2.6 GHz	2.6 GHz
Número de *cores*	17280	9728
Prestaciones (Rmax)	182.7 TFlop/s	183.4 TFlop/s
Posición en el *Top500 List*	434	433

Tabla 1.1: Comparación entre *clusters* con tecnología de interconexión GigaEthernet e Infiniband. *Top500 list, junio de 2015.*

un simple fallo.

El segundo reto con el que se encuentran los diseñadores de *clusters* de altas prestaciones es la red de interconexión que deben usar para unir todos los elementos de cómputo. Este factor juega un papel muy importante en cualquier diseño. Aunque el *cluster* disponga de un gran de número de nodos de procesamiento, las prestaciones máximas alcanzadas por el sistema se verán afectadas de forma directa por la red de interconexión. Un ejemplo de ello lo podemos ver en la Tabla 1.1, donde tenemos dos *clusters* listados en el *Top500* de junio de 2015, uno en la posición 434 y otro en la posición 433.

Como podemos apreciar, ambos *clusters* usan procesadores Xeon E5-2670 8C a una frecuencia de 2.6 GHz. Aunque el primer *cluster* dispone de 7552 *cores* más que el segundo, éste último logra obtener unas prestaciones (Rmax) superiores. El *cluster* que usa tecnología Infiniband, usando un 43.7 % menos en *cores*, sobrepasa las prestaciones que ofrece el *cluster* que usa Gigabit Ethernet. Este ejemplo ilustra como la red de interconexión juega un papel decisivo en cuanto al rendimiento del sistema.

Es por este motivo que más del 44 % de *clusters* listados en el *Top500* de junio de 2015 usan redes de interconexión de altas prestaciones basadas en tecnología Infiniband para poder explotar al máximo la potencia de cómputo. Sin embargo, para poder obtener una prestaciones óptimas, se debe implementar una topología de red y un algoritmo de encaminamiento adecuados para hacer frente al tráfico que circula por la red.

La topología define la forma en la que se conectan los nodos de procesamiento y los conmutadores de interconexión. Muchos de los supercomputadores que usan la tecnología Infiniband como sistema de interconexión implementan

Figura 1.2: Supercomputador Tianhe-2 (MilkyWay-2). Número 1 en el *Top500 List*, junio de 2015.

la topología fat-tree. Esta topología ha ganado una creciente popularidad al proporcionar tolerancia a fallos y ofrecer elevadas prestaciones.

Supercomputadores como el Tianhe-2, Mira-BlueGene/Q, TSUBAME 2.5, Nebulae, Tera-100, Roadrunner y JUROPA, entre otros, utilizan la topología fat-tree en su sistema de interconexión. Por ejemplo, el Tianhe-2 (ver Figura 1.2), desarrollado por la Universidad Nacional de Tecnología y Defensa [8] (NUDT) en Guangzhou (China), que encabeza la lista del *Top500* en sus últimas cuatro ediciones, desde junio de 2013 hasta junio de 2015, tiene su propio sistema de interconexión llamado TH Express, el cual usa una topología fat-tree con 13 conmutadores en el nivel superior, cada uno de 576 puertos, para conectar los 16000 nodos que componen el *cluster* [31].

Dado el creciente uso de esta topología en sistemas de altas prestaciones, esta tesis se centrará en redes de interconexión basados en la topología fat-tree [42] y en especial su variante k-ary n-tree [52].

Por otra parte, el algoritmo de encaminamiento determina el camino que toma cada paquete entre dos nodos de procesamiento a través de la red de interconexión, y suele tener un gran impacto en el rendimiento de la red, la complejidad de los conmutadores y sobre todo en la tolerancia a fallos.

El encaminamiento lo podemos clasificar como adaptativo o determinista. En el primer caso, los paquetes inyectados por los nodos de procesamiento a

la red pueden tomar cualquiera de las posibles rutas disponibles, usando información acerca del estado del tráfico y de los puertos para elegir el mejor canal de salida, realizando a la vez un mejor equilibrado del tráfico al distribuir los paquetes a través de las posibles diferentes rutas. Sin embargo, este esquema no garantiza la entrega en orden de los paquetes, lo cual es obligatorio para algunas aplicaciones, como algunos protocolos de coherencia de caché [50].

Por otra parte, el encaminamiento determinista siempre utiliza la misma ruta para cada par origen-destino, típicamente la ruta más corta, de modo que en cada punto de la red siempre se toma la misma decisión para encaminar un paquete destinado a un nodo dado. Este esquema es fácil de implementar, rápido, y ofrece buenos resultados bajo cierto tipo de patrones de tráfico. Sin embargo, debido a que carece de diversidad de caminos para cada par origen-destino, el equilibrado del tráfico en la red es muy pobre bajo ciertos patrones de tráfico.

En esta tesis nos centraremos en la propuesta de alternativas a la topología fat-tree para conseguir mejores prestaciones de red y ofrecer tolerancia a fallos, usando hardware de menor complejidad para rebajar el coste del sistema o usando hardware de características similares a las que usa la topología fat-tree, pero haciendo mejor uso de los recursos para ofrecer unas prestaciones y tolerancia a fallos superiores.

1.2. Objetivos

Como se ha mencionado antes, el objetivo principal de esta tesis es proponer alternativas a la topología fat-tree, con el fin de obtener mejores prestaciones en la red de interconexión, tratando de mantener el coste de ésta lo más bajo posible.

Este objetivo principal lo podemos dividir en una serie de objetivos parciales, los cuales se describen a continuación:

- Analizar la topología fat-tree y demás propuestas existentes en la literatura para conocer los actuales planteamientos, avances y mejoras que se han realizado en este campo.

- Proponer, desarrollar e implementar nuevas topologías de redes de in-

terconexión indirectas que nos permitan explotar el hardware de manera más eficiente para obtener el máximo beneficio de la red.

- Optimizar el algoritmo de encaminamiento para reducir el *Head of Line Blocking* y alcanzar máximas prestaciones en la red, equilibrando el tráfico y reduciendo la congestión del mismo.

- Proveer tolerancia a fallos en la topología, incrementando el número de caminos para cada par origen-destino, haciendo uso de los recursos disponibles. Además, este modelo debe soportar fallos en cualquier parte de la red, ya sea en los enlaces de inyección, de red o de eyección.

- Proponer, diseñar e implementar un mecanismo que nos permita conocer el estado de toda la red para mejorar el encaminamiento y la tolerancia a fallos.

- Evaluar mediante simulación la idoneidad de las propuestas, analizando los diferentes factores que puedan afectar el rendimiento de la red de interconexión.

1.3. Estructura de la tesis

En el presente capítulo se ha planteado la motivación y objetivos a cumplir de esta tesis. El resto de esta memoria se desarrolla de la siguiente manera:

- En el Capítulo 2 describimos los fundamentos de redes de interconexión necesarios para comprender el presente trabajo. También realizamos un análisis del estado del arte que contribuye a la materia de esta tesis.

- En el Capítulo 3 presentamos una nueva familia de redes de interconexión compuesta por cuatro topologías de red, las cuales ofrecen un alto rendimiento de red y un alto nivel de tolerancia a fallos, usando hardware con un coste inferior o similar al fat-tree.

- En el Capítulo 4 realizamos una amplia evaluación de la nueva familia de topologías propuesta, comparándolas frente a la topología fat-tree con

encaminamiento adaptativo. Las topologías propuestas no se evalúan de manera individual puesto que éstas se encuentran estrechamente relacionadas entre sí. De esta manera, al realizar un análisis general de todas las topologías podemos tener una visión global de las mejoras que ofrece cada una de ellas.

- Por último, en el Capítulo 5, finalizamos con las conclusiones y contribuciones relacionadas al campo de estudio.

Capítulo 2

Fundamentos y estado del arte

"Nunca consideres el estudio como un deber, sino como una oportunidad para penetrar en el maravilloso mundo del saber."

Albert Einstein

El objetivo de este capítulo es describir la terminología necesaria para entender los aspectos básicos de las redes de interconexión. Por brevedad y simplicidad en el documento, nos centraremos solo en los conceptos principales. No forma parte de los objetivos hacer hincapié en estos conceptos, ya que el campo de estudio en las redes de interconexión es bastante amplio, y existen muchos aspectos que están fuera del alcance de esta tesis. Por ello, invitamos al lector a consultar algunos libros [27, 30, 32] especializados en la materia que cubren de manera detallada todos estos aspectos.

Este capítulo se encuentra dividido en tres secciones. La Sección 2.1 se centra en los fundamentos de las redes de interconexión. Por otra parte, la Sección 2.2 presenta el estado del arte en cuanto a tolerancia a fallos en redes de interconexión multietapa. Por último, en la Sección 2.3 describimos la topología fat-tree y algunas propuestas de optimización realizadas sobre la misma.

2.1. Redes de interconexión

La red de interconexión es uno de los pilares centrales que permite a un sistema de cómputo de alto rendimiento alcanzar sus máximas prestaciones. Hoy en día estos sistemas están compuestos por miles de nodos de procesamiento que cooperan entre sí, con el mismo propósito. Sin embargo, para poder cumplir con este objetivo, los nodos, en cierta forma, deben estar conectados entre sí a través de una infraestructura de comunicación que permita compartir información de manera fiable entre un par de nodos cualquiera, de la manera más rápida posible. Por ejemplo, el centro de datos de Facebook dispone de más de cien mil servidores, y utiliza una base de datos distribuida, la cual es accedida por millones de usuarios a la vez. Un sistema de estas dimensiones, que mueve más de 350GB de información por minuto, requiere una red de altas prestaciones y baja latencia para explotar el potencial del sistema de procesamiento y almacenamiento. Si un sistema de altas prestaciones no implementa una red adecuada, se verán negativamente afectadas sus prestaciones.

Por otra parte, la red de interconexión no solo desempeña un papel principal en el rendimiento del sistema, sino también, es un punto crítico en el soporte de tolerancia a fallos. Como veremos en la Sección 2.2, han sido muchos los trabajos de investigación que se han centrado en la tolerancia a fallos, dado que un fallo en la red de interconexión puede aislar un gran número de nodos de procesamiento de forma indirecta, comprometiendo las prestaciones y la estabilidad del sistema.

Con el fin de comprender el estado del arte citado en este trabajo, además de las propuestas realizadas por nuestra parte, proporcionaremos los aspectos básicos acerca de las redes de interconexión. Por ello, en esta sección, haremos una breve introducción a las redes de interconexión, presentando sus componentes y diversos conceptos que deben ser considerados en cualquier diseño.

2.1.1. Conceptos básicos de redes de interconexión

La red de interconexión es un sistema programable compuesto por un gran número de elementos, tales como: conmutadores, buffers, enlaces y lógica de control que trabajan con el mismo objetivo para entregar la información que comparten los terminales. Los datos a ser transmitidos son enviados desde un

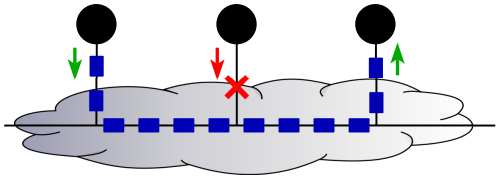


Figura 2.1: Red de medio compartido.

dispositivo origen hacia un dispositivo destino. Dicha información, antes de ser enviada a través de la red, es particionada en unidades de datos adecuadas para la transmisión, a través de un protocolo, donde es encapsulada con información de control adicional. Esta operación suele realizarse en el nodo origen. Cuando los datos están listos para ser enviados, la tarjeta de red o NIC (*Network Interface Card*) es la encargada de inyectar dicha información a través de un enlace que da acceso al dispositivo a la red de interconexión. A lo largo de este trabajo, denominaremos estos enlaces como enlaces de inyección o eyección, dependiendo de la dirección de los paquetes. Normalmente, los enlaces utilizados son bidireccionales.

En las primeras propuestas sobre redes, antes de inyectar un paquete, la tarjeta de red debía monitorizar constantemente el enlace con el cual estaba conectado, para determinar si podía utilizar el medio de transmisión. Esta topología se conoce como red de medio compartido (Figura 2.1), donde la red de interconexión comparte un único medio entre cada nodo.

Sin embargo, debido a la naturaleza de estas redes, el arbitraje utilizado suele ser distribuido. Esto significa que, cada nodo perteneciente a la red tiene que escuchar en su enlace para determinar si hay tráfico o no, antes de transmitir cualquier información, a fin de evitar/reducir las colisiones. El protocolo utilizado para realizar este arbitraje es el de Acceso Múltiple por Detección de Portadora o CSMA, que dispone de dos versiones: CSMA/CD[1] y CSMA/CA[2]. En las redes de acceso compartido, la conmutación y el encaminamiento son procesos triviales. Tras el arbitraje, el nodo origen vuelca el paquete en el

[1]CSMA/CD es el protocolo utilizado en redes IP para manejar el acceso al medio, definido en la especificación IEEE 802.3.

[2]CSMA/CA es un protocolo de acceso al medio que se relaciona con CSMA/CD. Desafortunadamente, este protocolo crea una carga significante y agrega tráfico innecesario a la red, ralentizando el sistema.

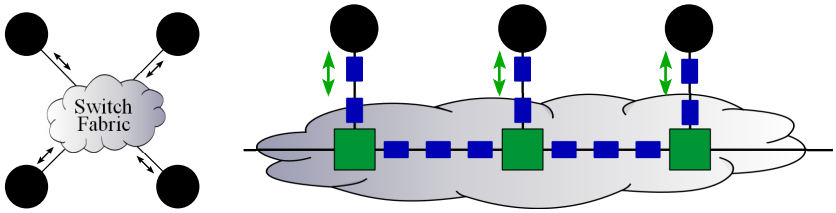


Figura 2.2: Red segmentada o conmutada.

medio compartido. Como todos los nodos están conectados a dicho medio, el destinatario lo recibirá. Sin embargo, a medida que aumenta el número de nodos, también aumenta el tiempo de espera para poder transmitir. Por otra parte, solo se permite realizar una transferencia en un momento dado. Por lo tanto el rendimiento de la red se ve afectado.

Las limitaciones ofrecidas por este modelo, han hecho que se abandone el paradigma de red de medio compartido, dando paso a una solución más óptima, donde la red es segmentada y cada nodo se conecta directamente a un conmutador. Este modelo se conoce como una *red segmentada* o *red conmutada*. Ver Figura 2.2.

Cuando un paquete llega a un conmutador, éste debe determinar si el destino del paquete se encuentra conectado al mismo conmutador, para entregar la información. Por el contrario, si el mensaje debe ser enviado a otro conmutador, si hay un canal de salida disponible, éste es encaminado a través de dicho canal. En caso contrario, se almacenará en un buffer para ser enviado posteriormente cuando dicho canal esté disponible.

Con este nuevo modelo, los nodos no tienen que estar escuchando el tráfico de la red, sino que inyectan directamente los paquetes, y es el conmutador quien debe encargarse de gestionar la información recibida. A diferencia de la red de medio compartido, la red conmutada ofrece un mayor ancho de banda y más flexibilidad, siendo utilizada cuando se necesita conectar un alto número de nodos.

Visto de otra manera, un conmutador actúa como un semáforo, controlando el tráfico que circula por la red en los puntos de intersección. Estos conmutadores están compuestos por una gran cantidad de elementos. En la Figura 2.3, tenemos una vista simplificada con los principales componentes.

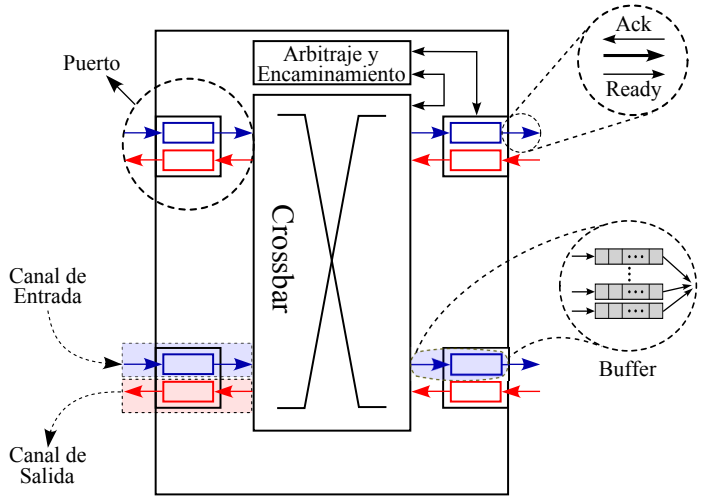


Figura 2.3: Vista simplificada de un conmutador de 2×2 con puertos bidireccionales.

Los enlaces están conectados a un puerto del conmutador; estos puertos internamente se encuentran divididos en dos partes: un canal de entrada y un canal de salida, si son bidireccionales. Cada canal en un único sentido está compuesto lógicamente por 3 interfaces: una ruta de datos *'data path'* que transmite los mensajes a nivel de *flit*[3], una línea de *'ready'* que marca la presencia de un *flit* en la ruta de datos y especifica el canal virtual (ver Sección 2.1.5) donde el *flit* debe almacenarse, y por último, una línea *'ack'* en la dirección inversa que envía un reconocimiento cada vez que es liberado un buffer en las líneas de entrada [52].

El canal de entrada es el responsable de recibir el paquete que proviene del enlace. En este punto, el algoritmo de encaminamiento debe decidir con qué canal de salida ha de conectarse el canal de entrada, para pasar la información a través del crossbar. Una vez tomada esta decisión, el arbitraje resuelve los conflictos entre los múltiples aspirantes al recurso. Si no está disponible el canal de salida, los paquetes son almacenados[4] en el buffer del canal hasta que

[3]Un *flit* (*flow control digit*) es la unidad más pequeña de la asignación de recursos en un conmutador, utilizado por la mayoría de los métodos de control de flujo.

[4]Dependiendo de la técnica de encaminamiento, la información puede ser almacenada a nivel de paquete o a nivel de *flit*.

se libere algún recurso que permita su envío.

La conmutación es el proceso que permite transmitir los datos desde el canal de entrada hasta el canal de salida. Esta conmutación normalmente es implementada con una red crossbar, la cual permite la comunicación entre cualquier par entrada-salida libres.

Una vez el paquete es transmitido al canal de salida, el buffer del canal de entrada es liberado para dar paso a otros paquetes. Cuando el paquete está en el canal de salida, si éste no puede ser enviado al siguiente conmutador, se almacenará en el buffer de dicho canal, hasta su envío. Los enlaces que interconectan los conmutadores, serán referidos como enlaces de red.

Finalmente, cuando el paquete alcanza el último conmutador de su recorrido, y se encuentra en el canal de salida, la tarjeta de red del nodo destino es la responsable de eyectar el paquete de la red.

2.1.2. Parámetros de diseño

Como se ha mencionado, las redes de interconexión son un tema de gran interés en la comunidad científica, dado que juegan un papel muy importante en el diseño de *clusters* de altas prestaciones. A la hora de realizar un diseño para una red de altas prestaciones debemos tener en cuenta muchos factores. A continuación explicaremos, brevemente, los parámetros más importantes en el diseño de una red de interconexión.

- *Prestaciones*: Las prestaciones de una red de interconexión se describen principalmente por la latencia de red y la productividad. La latencia es un factor de diseño crítico en muchos sistemas, tales como los sistemas de tiempo real. La productividad es la máxima cantidad de tráfico que puede gestionar la red, antes de llegar al punto de saturación.

- *Escalabilidad*: La escalabilidad en redes de interconexión implica que el ancho de banda de la red debe crecer proporcionalmente al tamaño de la red. Además, el incremento del tamaño de la red no debería impactar abruptamente sobre la latencia, de modo que, ésta debería mantenerse en unos límites razonables a medida que incrementa el tamaño de la red. De otra manera, la red de interconexión se convertiría en un cuello de botella, limitando la eficiencia de todo el sistema.

- *Fiabilidad y tolerancia a fallos*: Es una característica de la red que indica que el sistema es capaz de entregar la información de manera fiable. En redes de interconexión el número de elementos que compone el sistema suele ser bastante alto. Tecnológicamente, el número de fallos que se pueden producir en el sistema crece de forma lineal con el número de elementos que lo componen. Por ello, es muy importante que una red tolere fallos, esto es, que sea capaz de continuar funcionando, aún en presencia de fallos.

- *Simplicidad*: Una red de interconexión con un diseño sencillo, normalmente conduce a una arquitectura de conmutadores más simples, a la vez que permite la implementación de algoritmos de encaminamiento menos complejos. Los diseños simples son habitualmente más rápidos y más económicos.

- *Coste*: Las redes de interconexión para sistemas de alto rendimiento, suelen representar un alto porcentaje del coste total del sistema. Afortunadamente, el coste no es el único parámetro que influye en las prestaciones. Por ello, los diseñadores tienen que buscar la mejor relación entre coste y prestaciones, obviamente, explotando al máximo el hardware para obtener los mejores beneficios.

Los parámetros mencionados anteriormente son muy importantes en cualquier diseño. Las contribuciones realizadas en esta tesis han tenido en cuenta, directa o indirectamente, estos parámetros.

Con el objeto de optimizar el resultado final, se han tenido en cuenta diferentes aspectos o factores de diseño de la red de interconexión. Los factores de diseño más importantes son [32] la topología, la técnica de conmutación y el encaminamiento.

2.1.3. Topología

La topología es la estructura física de interconexión del grafo que representa la red, es decir, es la forma en que están interconectados los nodos y conmutadores de la red, a través de enlaces que permiten el intercambio de información entre dichos elementos. Estas topologías, que normalmente vienen

dadas por un patrón, definen el número de nodos que pueden conectar y el número de enlaces y conmutadores requeridos para su conexión, afectando de forma directa la latencia, el ancho de bisección, el máximo ancho de banda de los enlaces y otros factores. La topología, en cualquier red de interconexión, es de vital importancia, ya que de ella dependen las prestaciones de la red, su simplicidad, y por ende, el algoritmo de encaminamiento implementado.

Actualmente existen muchas topologías para redes de interconexión. Sin embargo, éstas comparten muchas propiedades primordiales que han de tenerse en cuenta en cualquier diseño:

- *Grado de un conmutador*: Esta propiedad se refiere al número de puertos que conectan directamente a un conmutador con sus vecinos, a través de un enlace directo entre cada par de componentes.

- *Ancho de bisección*: Es el mínimo número de enlaces cortados para dividir la red en dos partes iguales.

- *Ancho de banda de la bisección*: Es una medida de la capacidad de comunicación de la red, dada por el producto del ancho de bisección por el ancho de banda de cada enlace. Normalmente, una topología con un alto ancho de banda de la bisección es preferida sobre una que ofrezca un bajo ancho de banda de la bisección.

- *Diámetro*: Es el máximo camino de ruta mínima entre dos nodos cualquiera, medido por el número de enlaces recorridos. Un diámetro menor indica mayor habilidad de comunicación en la red. Al diseñar una red, debemos procurar que el diámetro de la red sea lo menor posible.

- *Distancia media*: Es el promedio de la distancia de encaminamiento (número de enlaces atravesados) entre todos los pares de nodos. Ésta es la distancia esperada entre un par aleatorio de nodos.

- *Simetría*: Una red es simétrica si su topología tiene el mismo aspecto, vista desde cualquier nodo. Esto implica que la red tenga que ser regular.

Tipos de topologías

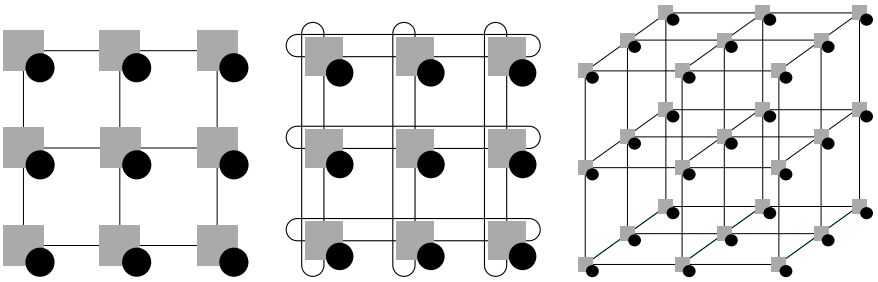
Existen principalmente dos categorías para clasificar las redes de interconexión: redes directas y redes indirectas.

Las redes directas o punto a punto, son aquellas en las que los conmutadores están directamente conectados al nodo de procesamiento, o en ocasiones, formando parte del nodo. Un ejemplo de este caso podrían ser las tarjetas EXTOLL [3], las cuales estan conectadas a la placa base del nodo a través del bus PCI.

El modelo de conexión utilizado por este tipo de red es muy simple. Cada nodo de procesamiento está directamente conectado con sus vecinos, a través de un enlace punto a punto, siguiendo un patrón de conexión que normalmente establece una simetría en toda la red. La simplicidad de este modelo hace que la implementación y cableado de la red sea bastante sencilla.

Dada la naturaleza de conexión en estas topologías, cada vez que se envía un paquete desde un origen hasta un destino, que no tienen conexión directa, éste tiene que pasar por nodos intermedios. Es entonces cuando toma protagonismo el algoritmo de encaminamiento, que determina el camino adecuado para que el paquete alcance su destino. En cada nodo intermedio, el algoritmo de encaminamiento indica el siguiente canal a ser usado. Si todos los canales candidatos están ocupados, el paquete es bloqueado y no puede avanzar. El encaminamiento es un parámetro determinante en las redes directas, dado que, si no se implementa adecuadamente, las prestaciones de la red decaen considerablemente.

En la Figura 2.4, mostramos algunos ejemplos de redes directas. Los círculos negros representan los nodos de procesamiento y los cuadros grises los conmutadores. La simplicidad que presenta este tipo de redes de interconexión hace que éstas sean fáciles de replicar en otra dimensión. Por ejemplo, si tomamos la topología en malla de la Figura 2.4a, y la replicamos dos veces más en el eje z y las interconectamos entre sí, obtenemos una 3-ary 3D-mesh, representada en la Figura 2.4c, interconectada en todas sus dimensiones. De hecho, una de las características interesantes de las topologías derivadas de la malla, es que tienen en común que, todos sus enlaces están dispuestos en



(a) Malla (b) Toro (c) 3-ary 3D-mesh

Figura 2.4: Ejemplo de topologías directas.

varias dimensiones en forma ortogonal[5] de una manera regular.

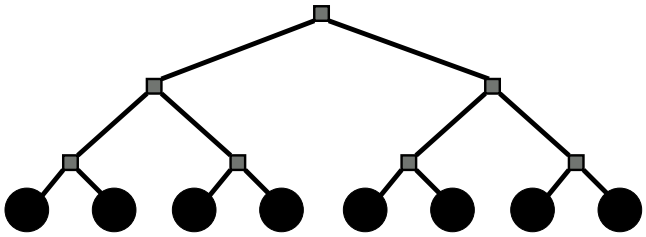
En realidad, estas topologías son una instancia particular de una clase de redes directas más amplia, denominadas k-ary n-cube, donde k es el número de nodos interconectados en cada dimensión y n es el número de dimensiones de la red. Las redes directas más comunes son la malla n-dimensional, el hipercubo y el k-ary n-cube o toro.

Otra ventaja inherente a las redes directas es que, a medida que se incrementa el número de nodos, también se incrementan los recursos de la red y en teoría la capacidad de comunicación del sistema. Esta ventaja, junto a otras propiedades, ha llevado a que se implementen estas topologías sobre sistemas reales, tal como el BlueGene/L [11] y el Cray/SGI T3D [57], que usan un toro-3D como sistema de interconexión.

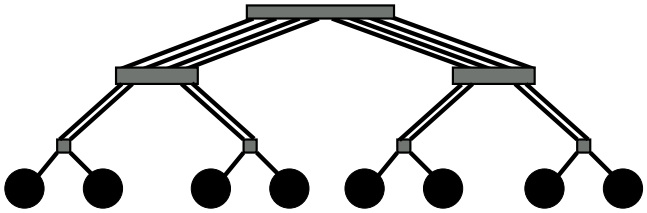
Sin embargo, estas redes presentan un inconveniente, y es que, a medida que aumenta el número de nodos en el sistema se debe incrementar el número de dimensiones de la red, para evitar que se genere un cuello de botella y reduzcan las prestaciones de la red. Esto supone incrementar la complejidad de los conmutadores y el cableado requerido. Otra posibilidad es usar una topología con un bajo número de dimensiones, pero con un gran número de nodos por dimensión, pero en este caso, el ancho de banda por nodo se reduce drásticamente.

Como solución a los inconvenientes que presentaban las redes directas, se

[5]Una topología de red es ortogonal, sí y solo sí, los nodos pueden ser dispuestos en un espacio n-dimensional ortogonal, y cada enlace puede ser dispuestos de tal manera que produzcan un desplazamiento en una única dimensión [32].



(a) Topología en árbol.



(b) Topología fat-tree.

Figura 2.5: Ejemplo de topologías indirectas.

introdujo otra clase de redes de interconexión, denominadas redes indirectas. A diferencia de las redes directas que establecen una comunicación directa entre nodos, en las redes indirectas, la comunicación entre un par de nodos debe llevarse a cabo a través de uno o varios conmutadores. La Figura 2.5 muestra dos ejemplos de redes indirectas. A diferencia de las redes directas, en las redes indirectas la comunicación entre un par de nodos debe llevarse a través de un conmutador, como se puede ver en la Figura 2.5, donde los conmutadores son representados por cuadros grises y los nodos por círculos negros. En este tipo de redes, los nodos de procesamiento disponen de una interfaz de red que se conecta con algunos de los conmutadores de la red. Los puertos de algunos conmutadores de la red pueden no ser utilizados [32], como sería el caso de los conmutadores de la última etapa.

En las redes indirectas, gracias a que los conmutadores no tienen ningún nodo conectado entre ellos, el ancho de banda por nodo escala con el número de nodos en el sistema. Por otra parte, si deseamos duplicar el número de nodos en la red en cualquiera de las topologías de la Figura 2.5, debemos replicar toda la red y agregar otra capa de conmutadores para conectar ambas partes de la red. De esta forma, el ancho de banda por nodo permanece constante.

Es habitual que la red indirecta se organice en varias etapas de conmu-

tadores, interconectados con un patrón de interconexión determinado. Estas redes son conocidas como redes multietapa o MINs (*Multistage Interconnection Networks*) y son el objeto de esta tesis.

Las MINs son una familia paramétrica de las redes de interconexión definidas por dos parámetros: k y n. k es el número de puertos de subida o bajada de cada conmutador. Eso significa que el grado de un conmutador es $2 \times k$, dado que cada conmutador tiene el mismo número de puertos de entrada que de salida. Por otra parte, el parámetro n define el número de etapas de la red. De manera que, una MIN puede conectar hasta N nodos, siendo $N = k^n$, utilizando nk^{n-1} conmutadores y nk^n enlaces para interconectar los conmutadores.

Las MINs tienen la particularidad de que todos los conmutadores de la red están distribuidos por etapas. Cada etapa dispone del mismo número de conmutadores, y cada conmutador solo se conecta con los conmutadores de la etapa anterior o la etapa siguiente. Por otra parte, los nodos de procesamiento solo tienen conexión con los conmutadores de la primera y/o última etapa, dependiendo de la topología.

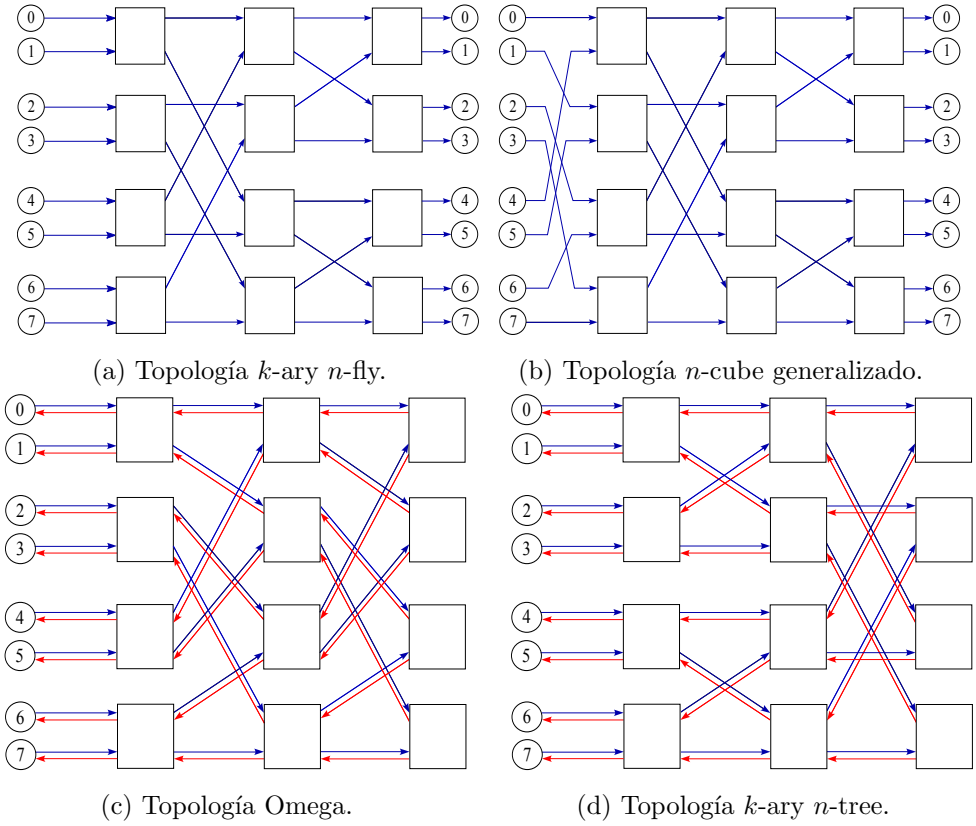
Gracias a que todos los conmutadores no tienen conexión directa con los nodos, como en el caso de las redes directas, el ancho de banda por nodo escala con el tamaño de la red. No obstante, cabe notar que, aunque las MINs ofrecen buenas prestaciones, la relación nodo-conmutador es mayor a 1:1, lo que hace que el coste de la red se incremente notablemente.

Dependiendo del esquema de interconexión entre los conmutadores, y el número de etapas de la red, se han propuesto diversas topologías. En la Figura 2.6, tenemos varios ejemplos de redes de interconexión multietapa.

Teniendo en cuenta el tipo de enlaces y conmutadores utilizados, estas redes las podemos clasificar en: redes de interconexión multietapa unidireccionales (UMINs) y bidireccionales (BMINs).

- Las UMINs (Figuras 2.6a y 2.6b), solo utilizan conmutadores y enlaces unidireccionales. De esta manera, los paquetes que viajan por la red, siempre tienen que alcanzar la última etapa para llegar a su destino.

- Las BMINs (Figuras 2.6c y 2.6d), utilizan conmutadores y enlaces bidireccionales. Esto indica que la información puede ser transmitida si-



(a) Topología k-ary n-fly.

(b) Topología n-cube generalizado.

(c) Topología Omega.

(d) Topología k-ary n-tree.

Figura 2.6: Ejemplo de redes de interconexión multietapa. Unidireccionales (a y b). Bidireccionales (c y d).

multáneamente en ambas direcciones. Además, dependiendo del encaminamiento utilizado, los paquetes, no necesariamente deben llegar a la última etapa de la red.

Aunque topológicamente las UMINs y las BMINs son muy similares, las diferencias entre éstas incluyen factores tales como: costes, complejidad de los elementos de red, encaminamiento y tolerancia a fallos. Estos factores se explicarán a lo largo de este capítulo.

Fat-trees

En el campo de las redes de interconexión la topología en árbol fue una de las primeras propuestas gracias a la facilidad de su implementación. En esta

topología, como podemos ver en la Figura 2.5a, los conmutadores (representados por el recuadro gris) se encuentran distribuidos por etapas, y a la vez, cada conmutador se conecta con un conmutador padre de la etapa superior, quien agrupa varios conmutadores hijos, formando un árbol invertido. Por otra parte, los nodos de procesamiento (representados por los círculos negros) solo tienen conexión con los conmutadores de la primera etapa, o lo que podríamos describir como las hojas del árbol.

Como hemos visto, el patrón de interconexión en la topología es bastante simple. Sin embargo, la conexión entre cada par de elementos de red, conmutador-conmutador o nodo-conmutador, se realiza a través de un único enlace, dando como resultado un ancho de banda fijo en cada punto de la red. Como podemos apreciar en la Figura 2.5a, si más de un conmutador de una mitad de la red necesita comunicarse con alguno de los nodos de la otra mitad, claramente formará un cuello de botella en la raíz del árbol.

Para solventar este inconveniente, en 1985, Charles E. Leiserson propuso la topología fat-tree [47]. Como podemos ver en la Figura 2.5b, la principal diferencia con la topología en árbol es que, a medida que nos acercamos más a la raíz del árbol, se incrementa el número de enlaces en cada etapa, haciendo que el árbol sea más grueso a medida que nos movemos hacia las etapas superiores. De este modo, el ancho de banda que conecta un conmutador con la siguiente etapa será igual al ancho de banda acumulado de todos los enlaces que conectan dicho conmutador. No obstante, la implementación de la topología fat-tree se hace prácticamente imposible, ya que, a medida que se incrementa el tamaño de la red también lo hace el tamaño del conmutador, duplicando además la aridad de los conmutadores en cada etapa, a medida que nos acercamos a la raíz. Considerando que el número de puertos de un conmutador está limitado tecnológicamente, la implementación de la topología fat-tree mostrada en la Figura 2.5b, es imposible de llevar a cabo en *clusters* con cientos o miles de nodos de procesamiento, como demandan los sistemas actuales.

Para generar una solución óptima a este problema, en 1995, Petrini y Vanneschi propusieron una nueva topología, equivalente al fat-tree, denominada k-ary n-tree [52]. Al igual que la k-ary n-cube y la k-ary n-butterfly [28], el k-ary n-tree es una familia paramétrica de topologías regulares y una subfamilia

de las redes de interconexión multietapa (MINs) que pueden ser construidas variando los parámetros k y n. Recuerde que k representa la aridad de los conmutadores y n representa el número de etapas de la red. En la Figura 2.6d podemos ver una red 2-ary 3-tree con ocho nodos de procesamiento y doce conmutadores. Cada conmutador de la red utiliza $2k$ puertos bidireccionales.

A diferencia de fat-tree, el k-ary n-tree usa conmutadores de igual aridad en toda la red, lo que permite realizar implementaciones más realistas. Por ello, en esta tesis nos centraremos en esta topología, teniendo en cuenta también que su popularidad ha aumentado en los últimos años, y se ha convertido en la topología preferida por fabricantes de conmutadores para redes de altas prestaciones, como Mellanox [5].

Para comprender mejor el funcionamiento y evaluación de la topología k-ary n-tree es necesario conocer la nomenclatura usada en su implementación.

En la Figura 2.7, podemos ver la enumeración utilizada para hacer referencia a los nodos y conmutadores de la red. En la parte izquierda de la figura, representado por los círculos, tenemos los nodos de procesamiento, y a la derecha de éstos tenemos los conmutadores, distribuidos por etapas. Cada nodo lleva asociado un identificador (Id). La numeración de los nodos se realiza de forma consecutiva, de arriba hacia abajo, e inicia en 0 hasta k^{n-1}. Además, el identificador del nodo suele dividirse en componentes, en base k (en base 2 en la Figura 2.7). Al igual que los nodos de procesamiento, los conmutadores también llevan asociado un identificador consecutivo, que podemos ver en la parte superior de cada conmutador, que se asigna comenzando por los conmutadores de la etapa inferior, de arriba hacia abajo, y continuando por la etapa siguiente en el mismo orden. El identificador de los conmutadores consta de dos componentes: etapa y número de conmutador dentro de la etapa. Dentro de la segunda componente, existen tantas componentes como etapas menos 1 $(n-1)$.

De manera formal, tenemos que, cada nodo de procesamiento está representado como una n-tupla $\{0,1,...,k-1\}^n$, es decir, los nodos están etiquetados con n componentes cuyos valores están entre 0 y $k-1$. Esta etiqueta la podemos encontrar dentro del círculo que representa el nodo. Los conmutadores están definidos como un par ordenado $\langle s, o\rangle$, siendo s la etapa donde está localizado el conmutador, con $s \in \{0,1,...,n\text{-}1\}$, y o una $(n-1)$-tupla $\{0,1,...,k-1\}^{n-1}$

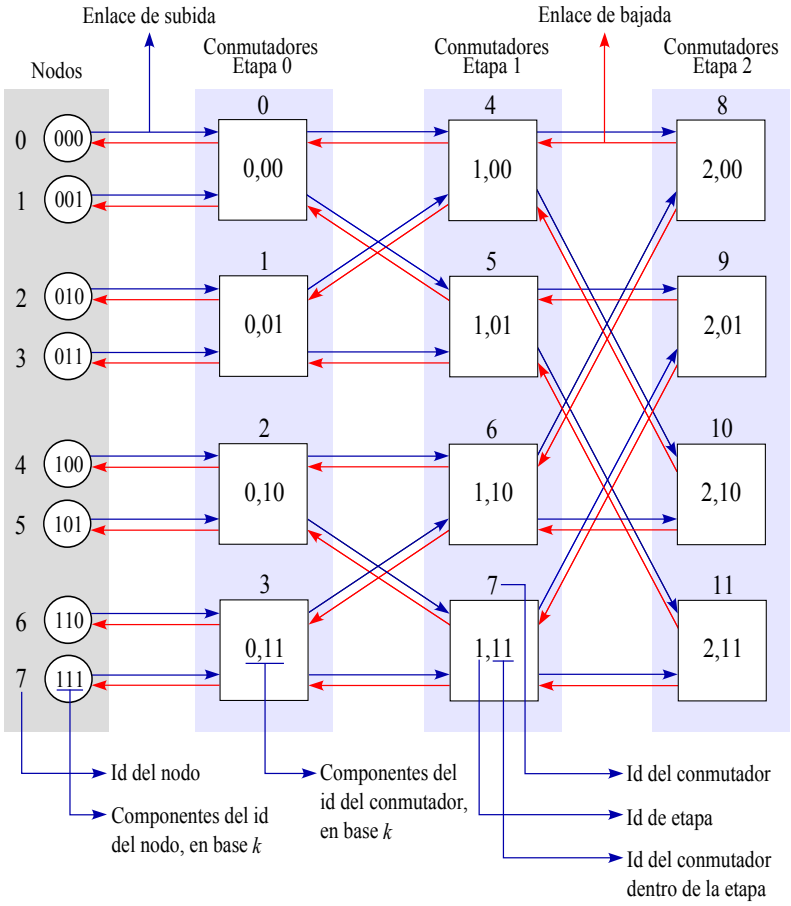


Figura 2.7: Nomenclatura usada en una red k-ary n-tree.

que identifica el conmutador dentro de la etapa que se encuentra. La componente del identificador del conmutador dentro de la etapa es la misma que la de los nodos de la fila correspondiente, pero sin la componente de menor peso.

En una red k-ary n-tree, los nodos y conmutadores de la red se encuentran unidos a través de enlaces bidireccionales. Sin embargo, puesto que en esta tesis también se diseñan y analizan UMINs, éstos serán representados como dos enlaces unidireccionales, lo que da lugar a un enlace de subida y un enlace de bajada, donde antes solo teníamos un único enlace bidireccional. Como podemos apreciar en la Figura 2.7, los enlaces de subida se representan en color azul, y los enlaces de bajada en color rojo. Como hemos explicado ante-

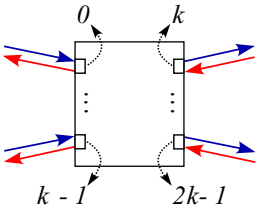


Figura 2.8: Enumeración de puertos de un conmutador en una red k-ary n-tree.

riormente, cada conmutador dispone de $2k$ puertos, los cuales también llevan un sistema de enumeración asociado que podemos ver en la Figura 2.8. Esta enumeración se realiza etiquetando los puertos desde 0 hasta $k-1$, para los puertos que comunican al conmutador con los nodos o con los conmutadores de la etapa anterior, y desde k hasta $2k-1$ para los puertos que conectan al conmutador con los elementos de la etapa siguiente. A lo largo de esta tesis, nos referimos a estos puertos como de entrada o salida, dependiendo de la dirección que lleven los paquetes.

Usando este sistema de etiquetado, podemos saber si dos conmutadores están conectados por un enlace realizando una simple comparación. Formalmente tenemos que, dos conmutadores $c - \langle s, o_{n-2}, ..., o_1, o_0 \rangle$ y $c' - \langle s', o'_{n-2}, ..., o'_1, o'_0 \rangle$ están conectados por un enlace, si $s' = s + 1$ y $o_i = o'_i$ para todo $i \neq s$. En otras palabras, dos conmutadores están directamente conectados por un enlace si pertenecen a dos etapas consecutivas de la red, y las componentes del segundo elemento de sus etiquetas son iguales, a diferencia del s^o elemento, siendo ésta la componente correspondiente a la etapa donde el conmutador c está localizado. Respecto a los nodos de procesamiento y los conmutadores de la primera etapa, se puede establecer una relación similar. Existe un enlace entre el conmutador $\langle 0, o_{n-2}, ..., o_1, o_0 \rangle$ y el nodo de procesamiento $p_{n-1}, ..., p_1, p_0$ si $o_i = p_i + 1$ para todo $i \in \{n-2, ..., 1, 0\}$. Lo anterior nos indica que, un nodo de procesamiento y un conmutador están conectados, si el conmutador pertenece a la primera etapa de la red, y las componentes del segundo elemento de su etiqueta son iguales a las componentes del nodo de procesamiento, descartando la componente menos significativa del nodo de procesamiento [52].

2.1.4. Técnicas de conmutación

Como hemos mencionado anteriormente, las técnicas de conmutación son las responsables de reservar los recursos de la red para poder transmitir la información desde un nodo origen hasta un nodo destino. Estos recursos pueden incluir las conexiones del crossbar, los buffers de los puertos del conmutador y el control de flujo.

La función básica consiste en configurar las conexiones entre los buffers del puerto de entrada y del puerto de salida del conmutador que encaminará los datos, dando paso al control de flujo que regulará la transmisión de datos.

La técnica de conmutación a utilizar en la red de interconexión es un factor de diseño muy importante, ya que ésta afecta de forma directa al tamaño de los buffers utilizados en los puertos de los conmutadores, el arbitraje, el encaminamiento, la latencia de la red y sobre todo el coste de diseño del sistema.

A continuación describiremos algunas de las técnicas usadas en redes de interconexión.

Circuit Switching

Circuit Switching o conmutación de circuitos es una técnica que opera reservando previamente los recursos de la red para formar un circuito desde el origen hasta el destino. Cuando se ha establecido la comunicación, y la información haya sido transferida, el circuito reservado previamente es liberado. Este proceso se lleva a cabo en 4 fases [30]:

1. El nodo origen inyecta en la red un *flit* cabecera (también conocido como *header flit*), el cual contiene el identificador del nodo destino. Este *flit* se desplaza desde el origen hasta el destino, reservando los canales por donde pasa.

2. Cuando el *flit* de cabecera llega al nodo destino, se envía una señal de reconocimiento, un *ack*, hasta el nodo origen.

3. Cuando el nodo origen recibe la señal de *ack*, el circuito ya se encuentra establecido, y se puede dar inicio al envío de un número arbitrario de paquetes, no necesariamente del mismo tamaño.

4. Por último, cuando se ha realizado toda la transmisión, se envía un *flit* final (también conocido como *tail flit*) para liberar todos los canales utilizados a lo largo de la transmisión, para que puedan ser usados por otros circuitos.

Esta técnica de conmutación tiene la ventaja que, una vez establecido el circuito, la información no sufre ningún tipo de contención, retraso o bloqueo a lo largo del camino, dando como resultado el poder utilizar todo el ancho de banda disponible en el medio de comunicación, siendo muy útil cuando los paquetes a transferir son frecuentes y de gran tamaño. Sin embargo, esta técnica presenta varios inconvenientes.

- Cuando se está reservando el circuito, si el recurso solicitado en un conmutador está ocupado, el tiempo que tarde éste en ser liberado se sumará al tiempo de establecimiento de dicho circuito.
- El circuito entre el origen y destino debe ser recorrido tres veces, uno para el establecimiento, otro para el *ack* y otro para el envío de datos, haciendo que la latencia de red se incremente de forma significativa, sobre todo en aquellos casos donde se envía un único paquete corto.
- Si el tiempo de reserva del circuito es más largo que el tiempo transmisión de los datos, el rendimiento de la red se verá penalizado, dado que los enlaces serán usados pobremente.
- Además, cuando los canales son reservados para un flujo de tráfico dado, estos no pueden ser reservados para otros flujos.

Las desventajas que presenta esta técnica, han hecho que su uso no sea muy común en el campo de las redes de interconexión, debido al bajo rendimiento que ofrece la red.

Store & Forward

Para solucionar el problema de *Circuit Switching*, en *Store & Forward* se propone dividir el mensaje en paquetes de una longitud fija, donde los primeros bytes de cada paquete contienen información de control para su encaminamiento. Esta información se conoce como cabecera del paquete.

Mediante este método, cada paquete es transmitido de manera individual, sin necesidad de tener un circuito previamente establecido entre origen y destino, permitiendo que haya varios paquetes en la red de manera simultánea, aun si el primer paquete no ha llegado a su destino. Por eso esta técnica es también conocida como *Packet Switching* o conmutación de paquetes.

Cuando se envía un paquete de datos, cada conmutador, a lo largo del camino, debe esperar a que dicho paquete haya sido recibido por completo, almacenándolo en su propio buffer (*Store*); luego su cabecera es examinada para determinar el puerto de salida y enviarlo al conmutador del siguiente salto (*Forward*). Esta es la razón por la que esta técnica se conoce como *Store & Forward*.

La ventaja de este método, es que pueden existir distintos flujos de datos a la vez dentro del mismo conmutador, compartiendo un mismo recurso, lo que permite que los enlaces de la red puedan ser utilizados al 100 %. Además, cada vez que se recibe un paquete, se verifica la integridad de éste.

Sin embargo, este método requiere que los buffers utilizados sean lo suficientemente grandes para albergar por completo el paquete recibido, lo cual influye sobre el coste de la red. Además, la latencia de los paquetes se multiplica en cada salto, por la operación de *Store* que se debe efectuar antes de reenviar la información.

Virtual Cut-Through

La conmutación a nivel de paquete está basada en la premisa de que un paquete debe ser recibido por completo antes de ser encaminado hacia el siguiente salto. Sin embargo, como hemos mencionado antes, en *Store & Forward*, cada paquete tiene una cabecera con información que se utiliza para el encaminamiento. Por lo tanto, en este caso, en vez de esperar a recibir el paquete por completo, tan pronto llega la cabecera (normalmente compuesta por un *flit*), ésta se examina y se inicia de inmediato el encaminamiento de dicha cabecera, y los bytes de datos siguientes, conforme van llegando, son encaminados tan pronto se tome la decisión de encaminamiento, y el canal de salida esté libre. Esta técnica es conocida como *Virtual Cut-Through*, y será la técnica de conmutación utilizada en nuestro simulador para evaluar todas nuestras propuestas.

Con esta técnica de conmutación la latencia de los paquetes se reduce, ya que éstos no tienen que esperar a ser recibidos por completo antes de su encaminamiento.

Por otra parte, aunque el tiempo que le lleva a un paquete atravesar un conmutador sea menor, la latencia base que ofrece este método todavía se ve afectada por el número de saltos que deban cruzar los paquetes. Además de esto, los buffers requeridos por *Virtual Cut-Through* son iguales a los que requiere *Store & Forward*, ya que, cuando un canal de salida en un conmutador se encuentra ocupado, el paquete debe ser almacenado por completo.

Wormhole

La necesidad de almacenar por completo un paquete en los buffers, afecta de forma directa al diseño de los conmutadores, sobre todo cuando se busca que éstos sean más sencillos y compactos. Por ello, a diferencia de *Virtual Cut-Through*, *Wormhole* opera con buffers a nivel de *flit* [32], de este modo, los paquetes son divididos en *flits*, sobre los que se ejerce el control de flujo, y los buffers de entrada y salida solo van a requerir capacidad para almacenar unos pocos *flits*.

En este caso, cuando un mensaje es enviado a través de la red, y se detiene a lo largo del camino, dado que los buffers no tienen la capacidad para almacenar todo un paquete, éste quedará almacenado a lo largo de diferentes conmutadores de la red, en *flits*.

La mayor ventaja que ofrece *Wormhole* frente a *Virtual Cut-Through*, es la reducción del tamaño de los buffers en los conmutadores. Por otra parte, la latencia de los mensajes depende principalmente del tamaño de los mismos y muy poco de la distancia recorrida. Sin embargo, el mayor inconveniente que presenta, es que se puede generar un alto nivel de contención en la red, porque un mensaje puede bloquear muchos recursos mientras la atraviesa, causando una baja utilización de los enlaces y los buffers. Además es más complicado garantizar la ausencia de interbloqueos en el algoritmo de encaminamiento.

2.1.5. Canales virtuales

Wormhole permite reducir la latencia de los mensajes al tiempo que reduce el tamaño de los buffers. Sin embargo, si un mensaje reserva un canal, pero debido a la saturación de la red éste permanece bloqueado en el conmutador actual, ningún otro mensaje detrás de este mensaje puede usar el canal físico, aun si el puerto de salida solicitado por los otros está libre. Este problema es conocido como bloqueo de cabecera o *Head-of-Line Blocking* (*HoL Blocking*)[6].

Para reducir el *HoL Blocking* generado por *Wormhole*, se propuso el uso de canales virtuales (*Virtual Channels*) [29], asociando distintos canales virtuales dentro de un único canal físico. En la Figura 2.9, podemos ver el diagrama simplificado de un conmutador que implementa canales virtuales.

Cuando se usan canales virtuales, los buffers son divididos en diferentes colas y cada canal físico es compartido por todas las colas. Cada canal virtual de entrada incluye una copia completa del estado de todo el canal incluyendo un registro de estado (***A**ctive, **I**dle o **W**aiting*) [30], además del identificador del canal virtual de salida asignado al paquete actual, y los *flits* del buffer.

Cuando un paquete llega al puerto de entrada de un conmutador y el canal de salida por donde éste debe ser enviado está ocupado, en vez de bloquear la comunicación de todo el puerto de entrada, el paquete solo bloquea el canal virtual donde está almacenado, dando la posibilidad de dejar paso a los mensajes en otras colas.

De este modo, los canales virtuales reducen la latencia e incrementan las prestaciones de la red, permitiendo que diferentes mensajes compartan un canal físico y avancen a través de la red en vez de permanecer bloqueados.

En vista de la optimización agregada por los canales virtuales, podríamos pensar que a mayor cantidad de canales virtuales obtendríamos mejores prestaciones. Sin embargo, un alto incremento en la cantidad de canales virtuales reduciría la tasa de transferencia de los mensajes a nivel individual, los controladores de enlaces serían más complejos e incrementaría la lógica del conmutador, afectando negativamente el rendimiento y la latencia de la red.

Si bien la técnica de los canales virtuales se asoció directamente a *Wormhole*, puede emplearse también con otras técnicas de conmutación, como *Virtual*

[6]De ahora en adelante usaremos este término, para mantener la coherencia a través de la diferente literatura.

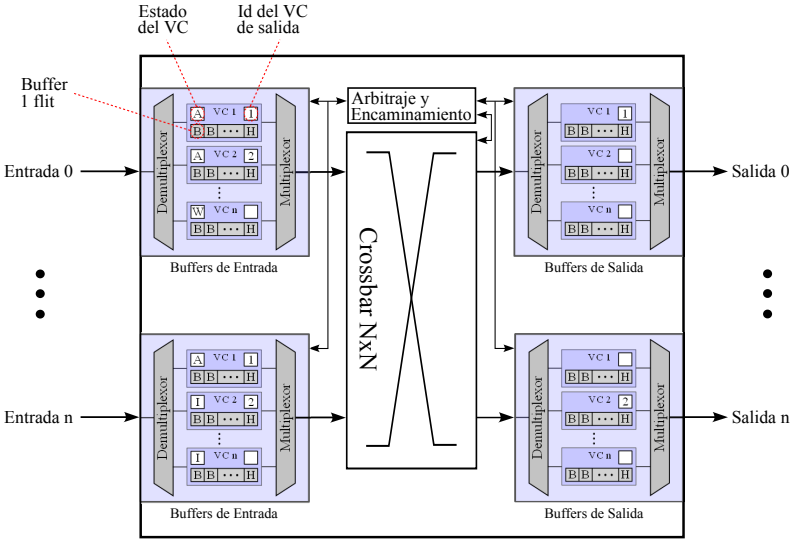


Figura 2.9: Diagrama simplificado de un conmutador que implementa canales virtuales.

Cut-Through. Por supuesto, en este caso, los buffers asociados a los canales virtuales deben poder albergar paquetes completos y no *flits*.

2.1.6. Control de flujo

El control de flujo es un mecanismo que está estrechamente relacionado con la técnica de conmutación. Este mecanismo establece un diálogo entre el emisor y el receptor, controlando el envío de información. Cuando un paquete está bloqueado, éste debe ser almacenado en un buffer. Cuando los buffers están llenos y no hay más espacio disponible para otros paquetes, el envío de información debe detenerse para evitar que los paquetes sean descartados o desviados a través de otro canal. De igual forma, cuando se liberan los buffers, el control de flujo debe reanudar el envío de información.

En el campo de las redes de interconexión se han propuesto diferentes mecanismos para el control de flujo, los cuales influyen en diferente medida sobre el rendimiento de la red. A continuación explicaremos brevemente algunos de ellos:

- *ack/nack* es un mecanismo que está basado en el reconocimiento de da-

tos. Cuando un *flit* llega a un buffer, si el buffer tiene espacio disponible, entonces el *flit* es aceptado y se envía una señal, *ack*, de reconocimiento. Por el contrario, si no hay disponibilidad para almacenar el *flit*, éste se descarta y se envía una señal *nack* para notificarlo. El *flit* tiene que conservarse en el origen hasta que reciba una señal *ack* que permita su avance. Este mecanismo es raramente usado, debido al ineficiente uso del ancho de banda y sus buffers [30].

- *Stop & Go* es un mecanismo alternativo diseñado para reducir la señalización, entre el emisor y el receptor, introducida por *ack/nack*. El principio de funcionamiento se basa en que cada buffer dispone de dos umbrales, correspondientes a cierto tamaño, calculados a partir del tiempo de ida y vuelta (*Round-trip time*)[7]. Cuando el espacio ocupado en el buffer alcanza el umbral de *Stop*, se envía una señal al emisor para indicarle que detenga la transmisión, teniendo en cuenta que haya espacio suficiente en el buffer para almacenar los *flits* que aún estén siendo transmitidos. Cuando la capacidad del buffer haya disminuido por debajo, o igual, al segundo umbral (*Go*), entonces se envía otra señal al emisor para reactivar la transmisión.

- En el control de flujo basado en créditos (*Credit-Based*), cada emisor, en el extremo de su enlace, mantiene un contador de créditos, el cual es igual al número de *flits* o paquetes que puede almacenar el receptor en su buffer. Cuando un *flit* o paquete es enviado al buffer receptor, y este ocupa un slot, entonces el contador es decrementado. Si el contador es igual a cero, quiere decir que no hay espacio suficiente en el receptor para almacenar más *flits* o paquetes. Por otra parte, cuando un *flit* o paquete es reenviado y se libera el espacio que ocupaba, se devuelve un crédito, y se incrementa el contador.

El mecanismo de control de flujo basado en créditos tienen una amplia difusión a nivel comercial. Éste lo podemos encontrar en dispositivos como Infiniband y PCIe, entre otros. Por ello, el simulador utilizado en esta tesis,

[7]Round-trip time puede ser definido como el tiempo transcurrido entre el tiempo que toma una unidad de información en ser enviada y el tiempo que toma en volver la señal de reconocimiento de la transmisión.

con el cual se evaluarán todas nuestras propuestas, utilizará este mecanismo para el control de flujo.

2.1.7. Encaminamiento

Como hemos visto anteriormente, la topología proporciona el esquema o grafo de interconexión entre todos los elementos de la red, describiendo la organización física de los componentes. De hecho, la topología define todos los caminos que hay entre cualquier origen-destino. Sin embargo, es el algoritmo de encaminamiento quien define qué ruta, o rutas, son las que deben seguir los paquetes.

Muchas de las propiedades de las redes de interconexión se derivan del algoritmo de encaminamiento, tales como la conectividad, la adaptatividad, la tolerancia a fallos y la ausencia de interbloqueos (*Deadlock*). La elección de este protocolo se convierte en uno de los factores determinantes en el rendimiento de la red.

Es de principal interés conocer la taxonomía de los protocolos de encaminamiento [30, 35], ya que cubre una serie de factores de gran importancia que deben tenerse en cuenta a la hora de desarrollar/implementar este mecanismo.

A continuación explicaremos brevemente los conceptos que conforman el esquema de clasificación.

Decisiones de encaminamiento

Las decisiones que deben tomarse en el encaminamiento son un criterio muy importante, ya que involucran el dónde y cuándo se ejecuta dicho proceso.

Los caminos que siguen los paquetes pueden ser calculados en los conmutadores o nodos a medida que van cruzando la red (encaminamiento distribuido), donde la cabecera del paquete solo contiene la dirección de destino, usada por los conmutadores para seleccionar el canal, o canales de salida. Este encaminamiento es especialmente favorable en topologías simétricas o regulares, donde todos los conmutadores usan el mismo mecanismo.

Otros métodos proponen usar un encaminamiento fuente, donde los nodos predeterminan el camino completo de los paquetes antes de ser inyectados en la red, de modo que cada vez que un paquete llega a un conmutador, la cabecera

es leída y mecánicamente se solicita el canal de salida indicado. Sin embargo, este método tiene la desventaja que, a medida que aumenta el número de puertos del conmutador y la distancia a recorrer, la cabecera de los paquetes también lo hace, requiriendo más bits para codificar el número de canales de salida. Este esquema ha sido usado en el IBM SP-2 [62].

También existe un modelo centralizado, donde la función de encaminamiento es determinada por un controlador central, como hacen típicamente las máquinas SIMD; o los modelos híbridos multifase, como el implementado en el sistema nCube-2, donde el nodo origen calcula de manera previa solo la dirección de algunos nodos intermedios y la ruta entre éstos se calcula de manera distribuida entre los routers.

Tipos de implementación

El encaminamiento puede ser implementado de diferentes maneras, pero siempre con el objetivo de ofrecer rapidez en la toma de decisiones.

Existen dos enfoques básicos, dirigidos a la implementación. El primero de ellos está basado en una implementación por hardware, donde un autómata de estados finitos ha sido trasladado a un circuito dentro del conmutador, de modo que cuando llega una cabecera, ésta es analizada en los puertos de entrada, y la salida es computada basándose en el hardware que representa la función de encaminamiento. Aunque esta implementación ofrece bajas latencias, no es la más conveniente, ya que no ofrece flexibilidad para realizar actualizaciones o cambios en el dispositivo, cuando surgen cambios en la topología.

Alternativamente, existe un encaminamiento basado en tablas donde el nodo origen y/o conmutadores disponen de una tabla, donde se especifica qué canal de salida que ha de ser utilizado para un paquete dado, basado en su identificador de destino. El número de entradas en la tabla es del orden $O(N)$, donde N es el número de nodos en la red.

El encaminamiento fuente basado en tablas contiene la especificación de todas las rutas, mientras que el encaminamiento distribuido basado en tablas solo indica que canales de salida corresponden a cada destino. La ventaja que ofrecen las tablas de encaminamiento, es que no es necesario realizar cálculos cada vez que se quiere encaminar un paquete, sino que solo basta con buscar la entrada en la tabla y la asignación del canal de salida. Sin embargo, esta

implementación no es escalable, ya que a medida que crece la red, y, por lo tanto, el tamaño de la tabla, se incrementa la latencia del paquete. Para reducir el tamaño lineal de estas estructuras, se pueden utilizar intervalos de encaminamiento [13, 34, 41, 46] y/o intervalos de exclusión [40].

El encaminamiento basado en tablas puede ser estático o dinámico. Un ejemplo de un sistema con tablas de encaminamiento actualizadas dinámicamente es Myrinet [6], el cual permite recomputar automáticamente las tablas cada vez que se produce un cambio en la red.

Adaptatividad

Básicamente, la adaptatividad se refiere a la habilidad del algoritmo de encaminamiento para ofrecer diferentes rutas entre cada par origen-destino. Estos pueden ser clasificados como deterministas o adaptativos.

El encaminamiento determinista solo ofrece una única ruta entre cada par de nodos, típicamente la ruta más corta. Cuando un paquete es encaminado, durante su trayecto, éste no puede desviarse por ningún camino alterno, sin importar si la ruta usada está congestionada o los otros canales de salida están libres. Este tipo de encaminamiento es usado en la mayoría de máquinas paralelas comercialmente disponibles, dado que es simple de implementar, rápido y con un buen rendimiento bajo la asunción de tráfico uniforme. Sin embargo, bajo otro tipo de patrones de tráfico se genera un desequilibrio en el uso de los enlaces de la red. La ventaja de este encaminamiento es que por defecto, ofrece entrega en orden de paquetes, como es requerido en muchas aplicaciones [50].

Existe un algoritmo de encaminamiento similar al determinista, denominado algoritmo de encaminamiento *oblivious* (*oblivious routing algorithm*). Este algoritmo no tiene en consideración ninguna otra información excepto la dirección del nodo destino. En este caso las decisiones de encaminamiento son ajenas al estado del tráfico de la red. Cualquier algoritmo determinista es de tipo *oblivious*, pero los de tipo *oblivious* no necesariamente son deterministas. Por ejemplo, pueden existir muchas rutas mínimas entre el origen y el destino, y el algoritmo de encaminamiento puede seleccionar la salida de forma aleatoria o de forma cíclica. El encaminamiento *oblivious* no determinista puede distribuir uniformemente la carga de tráfico en situaciones donde las soluciones adaptativas son muy costosas o lentas.

El encaminamiento adaptativo ofrece diferentes rutas entre el origen y el destino. La decisión acerca de la ruta que seguirá cada paquete es tomada en cada conmutador, por la denominada función de selección. El algoritmo de encaminamiento devuelve varios puertos y la función de selección elige cuál de ellos es el más conveniente para enviar el paquete. Esta decisión es tomada a nivel local, usando información del propio conmutador, tablas de historial, sondeando los nodos vecinos o usando algún método heurístico para comprobar el estado de la red. Uno de los métodos más comunes es usar el estado de los buffers de los puertos de salida. Cuando el algoritmo de encaminamiento devuelve un conjunto de posibles puertos de salida, la función de selección analiza cuál de ellos tiene más espacio libre, y selecciona aquel con más capacidad, dando a entender que dicho puerto está siendo menos utilizado. Existen diversos métodos a utilizar en la función de selección, tal como se expone en [37].

Progresivo y de ruta mínima

La finalidad del encaminamiento es hacer avanzar los paquetes por la red hacia el destino establecido, reservando un nuevo canal de salida para transmitir la información. El desplazamiento de los paquetes por la red, normalmente va incrementando el número de saltos (encaminamiento adaptativo progresivo), aunque no necesariamente acercándolos al destino.

Sin embargo, existen algoritmos que permiten que la cabecera del paquete regrese por el mismo camino que ha llegado (encaminamiento adaptativo con retorno), liberando previamente los canales reservados. Este algoritmo es usado principalmente en encaminamientos que proveen tolerancia a fallos, cuando no hay un camino alterno progresivo; sin embargo, requiere de un historial para asegurar que no está tratando de utilizar un camino previamente usado. El encaminamiento con retorno es costoso y difícil de implementar en *Wormhole*, en comparación con *Circuit Switching*.

Los algoritmos de encaminamiento adaptativo también suelen clasificarse de acuerdo al tipo de rutas ofrecidas entre el origen y destino. Aquellos que en cada decisión acercan más al paquete hacia el destino son conocidos como algoritmos de ruta mínima, directos o de ruta más corta. Por otra parte, los que proporcionan canales de salida que envían los paquetes lejos de su destino

son conocidos como algoritmos de encaminamiento de ruta no mínima. Sin embargo, este tipo de encaminamiento requiere de recursos adicionales y es propenso a que los paquetes se queden circulando indefinidamente por la red sin alcanzar su destino (*Livelock*).

Los algoritmos descritos anteriormente pueden ser subdivididos en dos grupos dependiendo de si permiten usar todas las posibles rutas o no. Si el encaminamiento solo permite usar un subconjunto de todas las rutas posibles, entonces el algoritmo es parcialmente adaptativo. Por el contrario, si el algoritmo permite usar todas las rutas provistas por la topología entre el origen y el destino, el algoritmo es completamente adaptativo.

El encaminamiento adaptativo, bajo ciertos patrones de tráfico, provee un mejor rendimiento que el encaminamiento determinista debido a su capacidad de adaptar la ruta de los paquetes a las condiciones de la red. No obstante, el encaminamiento adaptativo requiere una función de selección en los conmutadores, incrementando su complejidad y aumentando su retardo, mientras que el encaminamiento determinista no requiere ninguna función de selección, ya que solo existe una ruta para cada par origen-destino.

Por otra parte, el encaminamiento adaptativo, debido a la diversidad de rutas que ofrece puede llevar a que la red sea bloqueante, es decir, que se pueden producir interbloqueos (*Deadlock*) en la transmisión de datos, impidiendo que los paquetes avancen hacia su destino. Sin embargo, con un diseño adecuado del algoritmo de encaminamiento puede evitarse.

Deadlock y Livelock

Los interbloqueos, también conocidos como *Deadlock*, ocurren cuando un mensaje no puede avanzar hacia su destino porque los recursos solicitados para la transmisión del paquete están llenos, siendo bloqueados a su vez por otro mensaje que también está esperando la liberación de otro recurso, todos ellos esperando de forma cíclica.

Cuando se produce esta situación, los paquetes quedarán bloqueados permanentemente porque cada mensaje depende de otro para poder hacer uso de dicho recurso. Por ejemplo, consideremos una red en malla de 4×4, donde la función de encaminamiento usa rutas mínimas. En este instante, existen 4 paquetes localizados en diferentes buffers, como podemos ver en la Figura

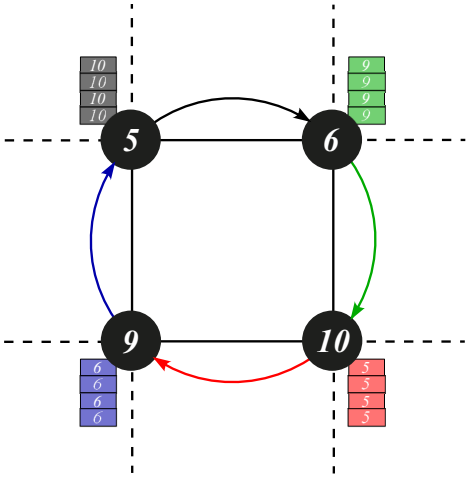


Figura 2.10: Interbloqueo o Deadlock.

2.10, donde cada uno de ellos está esperando a que el buffer en el siguiente salto sea liberado. El ciclo formado a la espera de la liberación hará que los paquetes estén bloqueados permanentemente.

Una situación diferente, denominada *Livelock*, ocurre cuando los paquetes no pueden alcanzar su destino, aun sin haber bloqueos permanentes en la red. En este caso los paquetes pueden estar transitando cerca de su destino, pero nunca llegando a éste, porque los canales requeridos para la entrega están ocupados por otros paquetes. Esta situación suele darse cuando el algoritmo de encaminamiento adaptativo permite que los paquetes sigan rutas no mínimas.

Arbitraje

De manera general, cuando una red de interconexión debe comunicar diferentes pares de dispositivos, usualmente, deben compartir las rutas o parte de ellas para poder enviar la información de un punto a otro. Aquellos recursos que se deben compartir, normalmente, se encuentran en los conmutadores que realizan la conexión entre los elementos de la red.

Como se ha visto anteriormente, un conmutador está compuesto de múltiples puertos, cada uno con sus buffers y canales de entrada y salida. Cuando se está llevando a cabo la transmisión de datos, múltiples canales de entrada en un conmutador pueden solicitar al mismo tiempo un mismo canal de sa-

lida, de acuerdo al encaminamiento. Es en este punto donde necesitamos un mecanismo para resolver estos conflictos, denominado algoritmo de arbitraje.

El arbitraje determina cuando las rutas solicitadas están disponibles para los paquetes. Idealmente, el arbitraje debe maximizar la asignación de los recursos disponibles de la red y los paquetes que demandan dichos recursos. Visto de otra manera, en cada conmutador de la red, el árbitro maximiza el emparejamiento de los canales de salida disponibles y los paquetes que están en los canales de entrada del conmutador demandando una salida. Cuando todas las solicitudes no pueden ser concedidas simultáneamente, el árbitro resuelve los conflictos mediante la concesión de los canales de salida a los paquetes de una manera justa, previniendo que se agoten los recursos solicitados.

Cada vez que se ejecuta el arbitraje, hay un ganador y posiblemente muchos perdedores. Los perdedores no tienen acceso a los recursos solicitados y típicamente son almacenados en el buffer. Como indicamos en la Sección 2.1.6, se puede implementar un control de flujo que evite que los buffers se desborden.

En resumen, el principal objetivo del mecanismo de arbitraje es ofrecer equidad entre todos los canales a la vez que maximiza la asignación de recursos solicitados, es decir, que asigne aproximadamente el mismo número de veces los recursos disponibles a los solicitantes. El significado de igualdad puede variar de una aplicación a otra.

Existen diferentes implementaciones en el arbitraje que varían según la manera de asignar las prioridades a los paquetes solicitantes. El arbitraje utilizado en la red aparte de ser equitativo en la asignación de recursos debe ser suficientemente eficiente para evitar incrementar más la latencia introducida por el propio mecanismo.

Para ver los diferentes métodos de arbitraje utilizados en redes de interconexión, por favor refiérase al Capítulo 18 de [30].

2.2. Tolerancia a fallos en redes de interconexión

Es esta sección revisaremos una serie de trabajos relacionados con la tolerancia a fallos en redes de interconexión. Veremos cómo se ha afrontado el reto de proveer tolerancia a fallos en las redes de interconexión a lo largo de la lite-

ratura, y cómo afecta de forma directa la implementación en las prestaciones del sistema.

Esta tesis está centrada en el campo de tolerancia a fallos y prestaciones de la red, siendo éstas las áreas donde realizamos nuestras contribuciones.

2.2.1. Tolerancia a fallos en MINs

A lo largo de la historia de las redes de interconexión, gran cantidad de literatura ha sido dedicada al desarrollo de mecanismos para proveer tolerancia a fallos [14, 19, 20, 22–26, 33, 36, 40, 43–45, 49, 51, 53, 54, 59, 60, 65–67], tanto en UMINs como en BMINs.

Las redes de interconexión multietapa unidireccionales son de especial interés, ya que su diseño suele requerir hardware menos complejo, reduciendo el coste de la red. Sin embargo, su nivel de tolerancia a fallos suele ser bajo o nulo. Normalmente, la tolerancia a fallos en redes de interconexión está basada principalmente en el uso de hardware adicional. Para dotar al sistema de esta característica, los investigadores proponen tres opciones: *1*) replicar toda la red *2*) agregar etapas extra, y/o *3*) incrementar la aridad de los conmutadores y añadir enlaces extra para comunicar distintas etapas. Todos estos diseños incrementan el hardware, la complejidad de los conmutadores utilizados y de la topología.

En las UMINs se han propuesto muchas topologías de red. La red Gamma (GIN - *Gamma Interconnection Network*) [51], por ejemplo, es una topología de red unidireccional propuesta para mejorar los inconvenientes de las redes Omega, n-cube binario, shuffle-exchange, Delta, Banyan y la IADM, en las que solo existe un camino entre cada par origen-destino. Sin embargo, la red GIN requiere de un alto número de conmutadores, y aun más, de un elevado número de enlaces para interconectar toda la red, con un bajo número de nodos de procesamiento, sin ofrecer tolerancia a fallos. Además, en esta topología, los conmutadores que interconectan los nodos, solo disponen de una única entrada/salida, lo que puede provocar congestión en la última etapa, al eyectar los paquetes.

También se han realizado otra serie de propuestas basadas en la GIN. Por ejemplo, la MGIN (*Monogamma Interconnection Networks*) [26], elimina la última etapa de conmutadores, para reducir los enlaces, y replica la primera

etapa de la red. De forma similar la CGIN (*Cyclic Gamma Interconnection Network*) [26] mueve la primera etapa de la MGIN a la última etapa, para crear al menos dos caminos disjuntos entre cada par origen-destino. Aun así, estas dos nuevas propuestas siguen utilizando un elevado número de enlaces y de conmutadores.

En [24], los autores hacen dos nuevas propuestas sobre la red Gamma. La primera es denominada PCGIN (*Partially Chained Gamma Interconnection Network*), en la cual todos los conmutadores de la primera etapa de la red Gamma son unidos a través de un enlace, formando un anillo, que les permite alcanzar caminos alternos. Sin embargo, para poder llevar a cabo la comunicación entre estos conmutadores, se debe incrementar su complejidad. La segunda propuesta, denominada FCGIN (*Fully Chained Gamma Interconnection Network*), interconecta todos los conmutadores de cada etapa de la red GIN entre sí, a excepción de la última, para alcanzar un mayor número de caminos. El inconveniente de estas topologías es que al interconectar los conmutadores de la misma etapa, los paquetes pueden tardar más tiempo en avanzar hacia la siguiente etapa, incrementando la latencia de la red y la complejidad del algoritmo de encaminamiento. Con esta modificación, consiguen soportar un fallo de enlace o de conmutador en cualquier etapa intermedia.

En CSMIN (*Combining Switches Multistage Interconnection Network*) [22] introducen un nuevo modelo de interconexión para mejorar la red, reduciendo el número de conmutadores de la primera etapa, pero incrementando el número de puertos de entrada y salida de estos conmutadores. Con este nuevo patrón de interconexión obtienen dos caminos disjuntos, soportando nuevamente un fallo en la red.

Otros trabajos como la EGN (*Extra Group Network*) [66] proponen dividir la red en partes más pequeñas, con igual cantidad de nodos en cada parte, y agregar un grupo extra de conmutadores en medio de cada división, además de usar multiplexores y demultiplexores entre los nodos y los conmutadores. A pesar del alto número de elementos utilizados, solo consiguen proveer dos caminos disjuntos.

En [33] los autores han propuesto un modelo óptimo en el cual describen cuantas etapas, mínimo, deben replicarse para soportar un número dado de fallos en los conmutadores, y ofrecer tolerancia a fallos en una MIN.

Otras estrategias como 3DON (*3-Disjoint Paths Fault-Tolerant Omega Multi-stage Interconnection Network*) [53] y 3DGIN (*3-Disjoint Paths Fault-Tolerant Multi-stage Interconnection Networks*) proponen un esquema de etiquetado de conmutadores para distinguir y alcanzar diferentes rutas. A través de esta propuesta, ofrecen tres rutas para cada par origen-destino. Sin embargo, para aumentar la tolerancia a fallos, el origen envía una copia del mismo paquete a través de las diferentes rutas para aumentar la tasa de entrega, lo cual incrementa la congestión de paquetes y por lo tanto, la degradación del rendimiento de la red.

Otras estrategias como las propuestas en [14, 44, 45, 60] también hacen uso de hardware adicional para incrementar el número de caminos alternativos, agregando más elementos de conmutación, etapas y enlaces extra. En [65] los autores usan diversas redes MIN en paralelo para crear redundancia, sin ningún enlace de interconexión entre ellos, incrementando el coste de la red.

En [36] también se analizan las propiedades de tolerancia a fallos en las MINs, sin usar redundancia. En [43], se introduce una metodología de encaminamiento tolerante a fallos que sacrifica un cierto número de nodos funcionales con el fin de no usar más de dos canales virtuales, y reducir el tiempo de encaminamiento.

Los autores de [25] analizan diferentes métodos para proveer múltiples caminos en redes MINs, y describen métodos para la identificación y reconfiguración de la red. Sin embargo, llegan a la conclusión que, para mantener un alto rendimiento de la red se deben eliminar nodos con una pobre conectividad.

Trabajos más recientes como el descrito en [19], proponen una mejora a la topología IASEN [12] para incrementar el nivel de tolerancia a fallos, obteniendo hasta doce rutas dentro de la red. Sin embargo, el problema con esta propuesta es que la etapa intermedia de la red podría convertirse en un cuello de botella si uno o más de sus conmutadores o enlaces fallan, dado que en dicha etapa hay una gran concentración de tráfico y un bajo número de conmutadores.

En [20], los autores también proponen ofrecer tolerancia a fallos, sin embargo su propuesta requiere incrementar el número total de conmutadores en la red, además de requerir un alto número de multiplexores/demultiplexores para conectar todos los nodos de procesamiento a la red. Por otra parte, dicho

trabajo solo se centra en la utilización de conmutadores de 2×2, haciendo que éste no sea adecuado para implementar redes que utilicen conmutadores de mayor aridad.

De igual manera, en las redes bidireccionales se suelen plantear soluciones similares. Por ejemplo, en [59], para ofrecer tolerancia a fallos en la red fat-tree (k-ary n-tree), los autores proponen duplicar toda la red, agregando enlaces cruzados entre todos los conmutadores de ambas redes. Esta propuesta incrementa más del doble el coste del sistema, y además, el rendimiento obtenido no compensa el excesivo hardware utilizado. De hecho han sido pocas las propuestas sobre las BMINs que logran proveer tolerancia a fallos sin apenas incrementar su coste. Este es el caso de [40], donde los autores desarrollaron un novedoso método para soportar fallos dentro de la red, utilizando el sistema de intervalos de encaminamiento propuesto en [41], además de incluir en cada puerto de salida un intervalo de exclusión. De esta manera incrementan el nivel de tolerancia a fallos y evitan replicar el hardware de la red.

En [49], se propone un novedoso y simple rediseño de los enlaces de red de la topología fat-tree para reducir el número de rutas y conmutadores afectados cuando se detecta un fallo dentro de la red. Sin embargo, esta propuesta no considera los fallos de los enlaces que conectan con los nodos de procesamiento.

Como hemos visto, a lo largo de diversos trabajos, gran parte de las propuestas realizadas en el campo de tolerancia a fallos en redes de interconexión multietapa se basan en la replicación de hardware. Este incremento de componentes solo aporta unas pocas rutas alternas para cada par origen-destino, y el rendimiento de la red no se ve beneficiado del hardware extra. También hay que tener en cuenta que al replicar el hardware se modifica la topología y el coste de la red se eleva considerablemente.

2.3. Topología fat-tree

Fat-tree es una topología de red que ha sido ampliamente usada en grandes sistemas de cómputo y máquinas comerciales, ofreciendo un buen rendimiento de red pero a un alto coste en el hardware utilizado.

En vista de las buenas propiedades que ofrece fat-tree, diferentes autores se han centrado en optimizar esta topología con el fin de obtener mejores

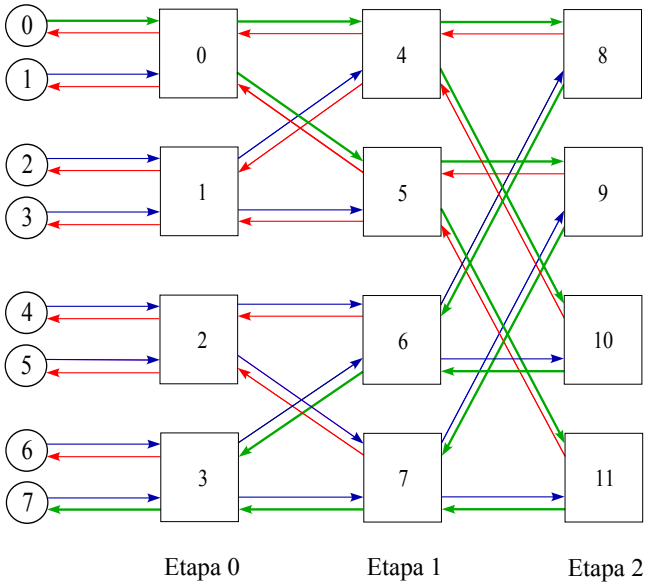


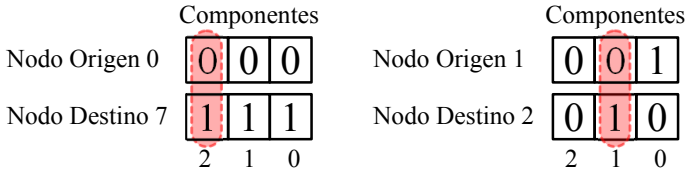
Figura 2.11: Rutas disponibles para un paquete con origen 0 y destino 7 en una red 2-ary 3-tree con encaminamiento adaptativo.

beneficios y reducir costes. En esta tesis se realiza una propuesta en esta línea, dando paso a una nueva familia de redes de interconexión, la cual será presentada en el Capítulo 3. Por este motivo, para comprender mejor el trabajo realizado en esta tesis, es importante conocer el funcionamiento de la topología fat-tree y su evolución.

2.3.1. Fat-tree con encaminamiento adaptativo

Fat-tree es una topología que utiliza encaminamiento adaptativo, que ofrece gran diversidad de rutas para cada par de nodos que deban comunicarse. En la Figura 2.11 podemos ver las rutas disponibles para un paquete con origen 0 y destino 7, en una red 2-ary 3-tree. Como podemos ver, existen cuatro rutas disponibles equivalentes (marcadas en color verde) que nos permiten alcanzar el destino, cada una de ellas con un recorrido mínimo.

Para calcular las rutas mínimas entre cualquier par de nodos, en cada conmutador por donde avanza el paquete, se debe identificar la etapa de red que los paquetes deben alcanzar. Este proceso de identificación se realiza compa-



(a) Comparación de componentes entre el nodo 0 y 7.

(b) Comparación de componentes entre el nodo 1 y 2.

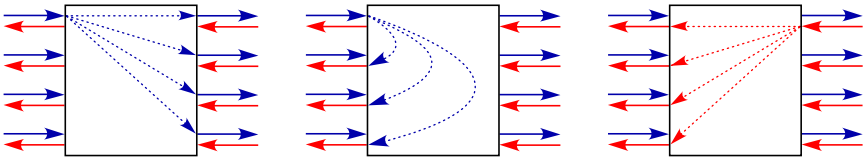
Figura 2.12: Identificación de la última etapa a la que debe llegar un paquete por comparación de componentes en una red 2-ary 3-tree.

rando las componentes de los identificadores de los nodos origen y destino, comenzando por el bit más significativo. Por ejemplo, si hay un paquete con origen 0 $\langle 000 \rangle$ y destino 7 $\langle 111 \rangle$ solo basta con comparar una a una ambas componentes, ver Figura 2.12a, y el primer par de ellos que difiera indica la etapa hasta la cual debe llegar el paquete, en este caso hasta la etapa 2. Los conmutadores alcanzables en esta etapa son los conmutadores 8, 9, 10 y 11, siendo éstos los conmutadores comunes al destino. Otro ejemplo de ello sería un paquete con origen 1 $\langle 001 \rangle$ y destino 2 $\langle 010 \rangle$. En este caso la componente de mayor peso es igual en ambos identificadores, por eso debemos seguir comparando la siguientes componentes. Como apreciamos en la Figura 2.12b, la componente que difiere entre ambos identificadores es la número 1, lo que indica que el paquete debe llegar hasta la etapa 1 para poder alcanzar su destino. Desde el conmutador que conecta con el nodo 1 (conmutador 0), los conmutadores de la primera etapa que podemos alcanzar son el 4 y el 5.

El encaminamiento de paquetes es un proceso que se lleva a cabo a través de dos fases: una fase ascendente y una descendente.

En la fase ascendente el encaminamiento es completamente adaptativo, es decir, los paquetes pueden usar cualquier puerto de salida que los conduzca hasta la etapa requerida. De manera formal podemos decir que, para enviar un paquete desde el nodo $p_{n-1}, ..., p_1, p_0$ hasta el nodo $p'_{n-1}, ..., p'_1, p'_0$ el paquete tiene que enviarse hasta la etapa i, si $p_j = p'_j$ para $j \in \{n-1...i+1\}$ y $p_i \neq p'_j$.

Una vez alcanzada la última etapa a la que debe llegar un paquete, se inicia la fase descendente, la cual es determinista; esto quiere decir que solo



(a) Fase de subida. (b) Giro, para iniciar fase de bajada. (c) Fase de bajada.

Figura 2.13: Puertos demandados en un conmutador de 4×4 en una red k-ary n-tree usando enlaces bidireccionales.

existe un único camino hasta el nodo destino, tal como podemos apreciar en la Figura 2.11. En cada conmutador por el que pasa un paquete, el puerto seleccionado es indicado por la componente del nodo destino, correspondiente a esa etapa. De manera formal, tenemos que, en la etapa i, el puerto seleccionado viene dado por la componente i del nodo destino (p'_i). Si nos referimos de nuevo a la Figura 2.11, vemos que en la fase de bajada, en cada conmutador, los puertos seleccionados en cada etapa se corresponden con las componentes $\langle 111 \rangle$, asociados al nodo destino.

2.3.2. Fat-tree con encaminamiento determinista: DESTRO

Como hemos visto anteriormente, fat-tree utiliza un mecanismo de encaminamiento que consta de dos fases: una fase de subida que es adaptativa, y una fase de bajada que es determinista. Este mecanismo permite que un nodo origen alcance un nodo destino a través de diferentes rutas.

En la fase de subida, un puerto de entrada puede demandar cualquier puerto de salida (Figura 2.13a), para llegar a la etapa requerida por el paquete. Una vez el paquete llega a la etapa requerida, el paquete debe dar un giro para iniciar la fase de descenso (Figura 2.13b), solicitando cualquiera de los puertos que dan inicio a la fase de bajada. Una vez el paquete se encuentra en la fase de descenso, el puerto de entrada donde se encuentra el paquete puede solicitar cualquiera de los puertos de salida (Figura 2.13c). A través de estas dos fases, los conmutadores llevan la información desde el origen hasta el destino, siguiendo las pautas marcadas por el encaminamiento.

La forma más común de construir conmutadores es usar tantos multiple-

xores como número de puertos de salida requeridos. Cada multiplexor tiene un número de entradas igual al número de puertos de entrada que pueden solicitar los correspondientes puertos de salida. De esta manera, los puertos de subida requieren k multiplexores con k entradas cada uno, lo que conduce a una complejidad de $k \times k = k^2$. En cuanto a los puertos de bajada, estos requieren k multiplexores con $2k$ entradas cada uno, lo que conduce a una complejidad de $k \times 2k = 2k^2$. Ahora, la complejidad de un conmutador requerido por fat-tree puede ser calculada sumando la complejidad que requiere la fase de subida y la fase de bajada, es decir, $k^2 + 2k^2 = 3k^2$ elementos de conmutación. Como podemos notar, el mecanismo de encaminamiento influye de forma directa en el diseño y la complejidad del conmutador.

Por otra parte, el número de rutas que ofrece la topología fat-tree en la fase de subida puede hacer que el tráfico de la red no se equilibre de forma adecuada, creando congestión en la fase de descenso, por ser determinista y disponer de una única ruta de bajada.

Con el fin de reducir la complejidad de los conmutadores y el efecto de *HoL Blocking* existente en la fase de bajada y equilibrar el tráfico en la red, se propuso un nuevo mecanismo denominado DESTRO [38].

Este nuevo mecanismo, al igual que el encaminamiento adaptativo, también usa dos fases para encaminar los paquetes, una de subida y una de bajada, pero con la diferencia de que ambas fases son deterministas. Para conseguir una fase ascendente determinista, los autores propusieron reducir el número de rutas ofrecidas por fat-tree a una sola ruta entre cada par origen-destino. Cada vez que llega un paquete a un conmutador, se debe comprobar si en la etapa actual es donde se debe dar inicio a la fase de descenso del paquete, de la misma manera que hace el encaminamiento adaptativo, comparando las componentes de los nodos origen y destino. Sin embargo, ahora el encaminamiento de cada paquete, en la fase de subida, se realiza basado en las componentes del nodo destino (empezando por la componente de menor peso) de este modo los paquetes son clasificados en cada etapa, reduciendo el número de destinos por canal y enlace, a medida que ascienden por la red.

En la Figura 2.14 podemos ver la distribución de paquetes en una red 2-ary 3-tree, usando DESTRO. Los destinos marcados en azul, son los paquetes que se encuentran en fase de subida, y los marcados en rojo son los paquetes que

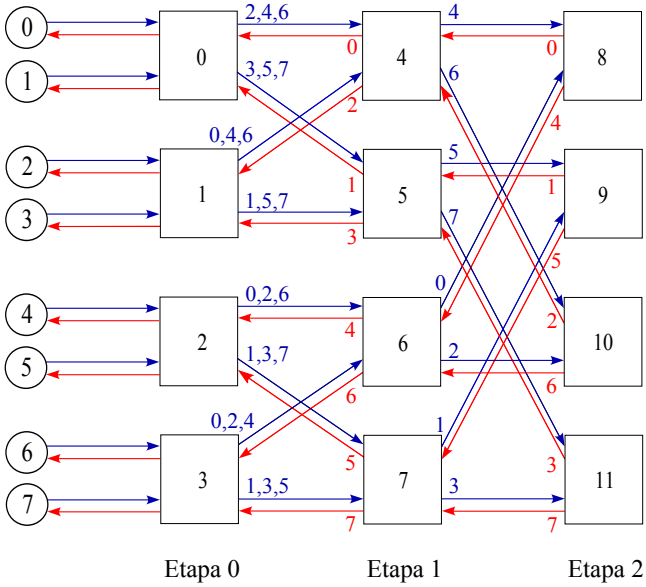


Figura 2.14: Distribución de paquetes en una red 2-ary 3-tree usando DESTRO.

están en fase de bajada.

En la primera etapa, todos aquellos destinos que tengan igual componente usarán el mismo puerto de salida; por ejemplo, el 0 $\langle 00\mathbf{0}\rangle$, 2 $\langle 01\mathbf{0}\rangle$, 4 $\langle 10\mathbf{0}\rangle$ y 6 $\langle 11\mathbf{0}\rangle$ tienen en común la primera componente (0), de esta manera estarán agrupados en el canal de salida asociado al puerto 2 $(k+0)$. Por otra parte, el 1 $\langle 00\mathbf{1}\rangle$, 3 $\langle 01\mathbf{1}\rangle$, 5 $\langle 10\mathbf{1}\rangle$ y 7 $\langle 11\mathbf{1}\rangle$ tienen en común la primera componente (1), usando el canal de salida asociado al puerto 3 $(k+1)$. De este modo, los conmutadores de la siguiente etapa solo contendrán un subgrupo de destinos previamente clasificados. En la siguiente etapa se realizará el mismo procedimiento, pero usando la siguiente componente. Por ejemplo, el conmutador 4 que recibe paquetes con destinos 0, 2, 4, 6, dará inicio a la fase de bajada a los destinos 0 y 2 (por haber alcanzado la etapa a la que deben llegar), mientras que el 4 y 6 usarán puertos de salida diferentes de acuerdo a lo que indican sus componentes; el 4 $\langle 1\mathbf{0}0\rangle$ usará el puerto de salida 2 $(k+0)$, mientras que el 6 $\langle 1\mathbf{1}0\rangle$ usará el puerto de salida 3 $(k+1)$.

De manera general, la selección del puerto de salida está basada en el identificador de destino y la etapa del conmutador que está encaminando el

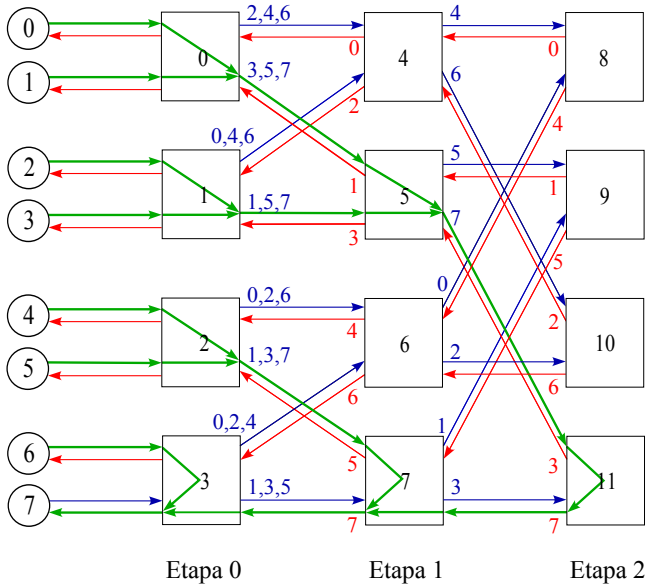


Figura 2.15: Encaminamiento determinista en una red 2-ary 3-tree usando DESTRO.

paquete, es decir, en el conmutador $\langle s, o_{n-2}, ..., o_1, o_0 \rangle$, el puerto de salida para un paquete que tiene que ser enviado a la siguiente etapa, cuyas componentes están dadas por $p_{n-1}, ..., p_1, p_0$, será $k + p_s$. El valor de k debe ser agregado a la componente del nodo destino ya que los puertos que comunican con la siguiente etapa empiezan por k, tal como hemos visto en la Figura 2.8.

La fase de bajada en DESTRO se conserva igual que en el fat-tree, siguiendo las componentes del nodo destino, empezando por el bit de mayor peso.

Con este nuevo mecanismo, solo existirá una ruta única entre cada par origen-destino, equilibrando el tráfico de la red, contrario a [48], donde usan diferentes rutas disjuntas en la fase ascendente y descendente, creando congestión en la red. Por ejemplo, en la Figura 2.15 podemos ver que las rutas desde cualquier nodo hasta el destino 7 solo disponen de un camino. La simplificación introducida por DESTRO consigue reducir el *HoL Blocking* que existía en la fase de bajada de fat-tree, ya que cada enlace de bajada es utilizado exclusivamente por un destino.

Por otra parte, usando DESTRO, la complejidad del conmutador es re-

ducida en comparación con fat-tree. La fase de subida mantiene la misma actividad que fat-tree con encaminamiento adaptativo, es decir, cada puerto de entrada puede demandar cualquiera de los puertos de salida, o cualquiera de los puertos de bajada, al iniciar la fase descendente. Sin embargo, en la fase de bajada, cada enlace, puerto de entrada y de salida es usado exclusivamente por paquetes enviados a un único destino. Como consecuencia, un puerto de entrada dado siempre solicitará el mismo puerto de salida, dado que todos los paquetes que llegan a un puerto de entrada en particular, en su fase de bajada, están destinados al mismo nodo y siempre son enviados al mismo puerto de salida. Esto permite una notable reducción en la complejidad del conmutador.

Usando multiplexores para implementar el conmutador, los puertos en la fase ascendente requieren k multiplexores, con k entradas, o visto de otra manera, una complejidad de $k \times k = k^2$. Los puertos en la fase de bajada requieren k multiplexores con $k+1$ entradas (los k puertos de subida más el único puerto de la fase de bajada) o $k \times (k+1) = k^2 + k$ elementos de conmutación. Por lo tanto la complejidad del conmutador requerida por DESTRO es $2k^2 + k$.

2.3.3. RUFT

RUFT [39] es una topología de red derivada de la topología fat-tree, haciendo uso de la buenas propiedades que ofrece DESTRO. Ver Figura 2.16.

En la Sección 2.3.2, hemos visto que DESTRO ha reducido la complejidad del conmutador, convirtiendo la fase ascendente adaptativa de fat-tree en determinista. Esta reducción no solo aporta economía en el diseño del conmutador, sino también una mejora de rendimiento en la red, dado que el tráfico permanece clasificado y el *HoL Blocking* de la fase de bajada es totalmente eliminado.

Por otra parte, la lógica que implementa DESTRO requiere que cada paquete sea comprobado al llegar a un conmutador, para determinar si éste debe iniciar la fase de bajada o debe proseguir. Con el fin de eliminar esta comprobación y reducir aún más la complejidad de los conmutadores, los autores han eliminado por completo la fase de bajada, convirtiendo el fat-tree en una red unidireccional, implementando una nueva permutación en la última etapa, para conectar los puertos de salida con los nodos de procesamiento. La topología resultante se asemeja a una *butterfly* unidireccional, pero con una

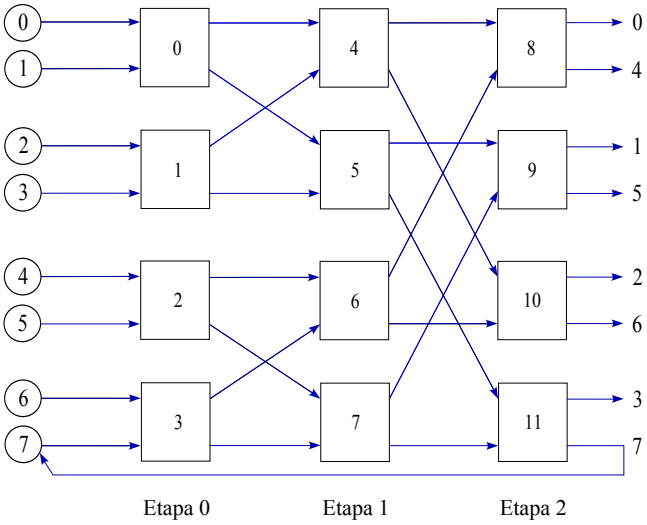


Figura 2.16: Topología RUFT.

permutación diferente en la última etapa que conecta con los nodos, tal como vemos en la Figura 2.16.

Ahora que la fase de bajada ha sido eliminada, los nodos de procesamiento han sido conectados a los conmutadores de la última etapa, a través de un enlace unidireccional. Aunque este enlace, por diseño, parezca más largo, en un buen diseño de red los conmutadores podrían estar "plegados" o distribuidos de tal manera que el enlace de la última etapa sea de igual o menor tamaño que otro cualquier otro enlace.

Eliminando la fase de bajada, el mecanismo de encaminamiento es simplificado, ya que solo se cuenta con una fase ascendente para el envío de paquetes. El encaminamiento en RUFT se realiza de la misma manera que hace DESTRO, con la diferencia que los paquetes deben alcanzar siempre la última etapa para llegar al destino. Al usar las componentes del nodo destino para encaminar los paquetes, éstos son clasificados a través de los diferentes puertos de salida, distribuyéndosen por los diferentes canales a medida que atraviesan las diferentes etapas.

En la Figura 2.17 podemos ver la topología RUFT con la distribución de paquetes a medida que avanzan por la red, y la ruta que sigue un paquete con origen 0 y destino 7. Nótese que en la ruta seguida por el paquete (el camino

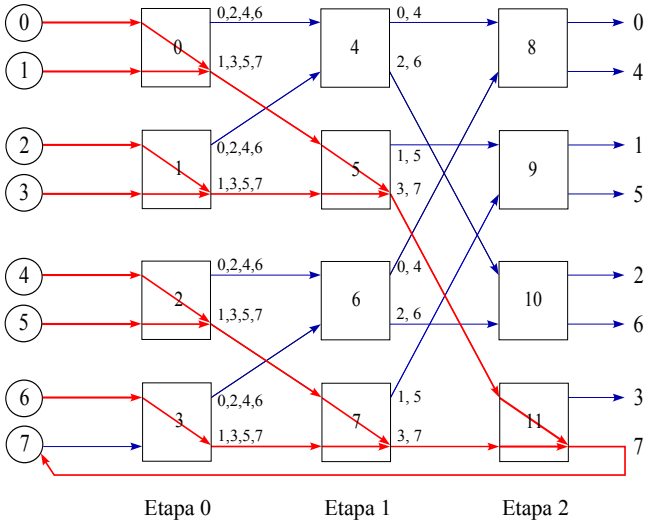


Figura 2.17: RUFT 2-ary 3-tree, derivada de la topología DESTRO.

marcado en rojo) no hay ninguna posibilidad de iniciar la fase de bajada antes de llegar a la última etapa, por lo tanto, los caminos son más largos dado que todos los paquetes tiene que alcanzar la última etapa, contrario al fat-tree donde dependiendo del par origen-destino, se debe atravesar un número diferente de etapas.

Usando RUFT, la complejidad del conmutador corresponde a la complejidad de un conmutador unidireccional de k puertos de entrada y k puertos de salida, donde cualquiera de los k puertos de entrada pueden demandar cualquiera de los k puertos de salida, de modo que la complejidad del conmutador es de k^2.

Como podemos ver, la complejidad del conmutador ha sido reducida más del doble en comparación con DESTRO y tres veces en comparación con el fat-tree usando encaminamiento adaptativo.

Capítulo 3

Una nueva familia de topologías indirectas, eficientes y tolerantes a fallos

"El científico no busca un resultado inmediato. No espera que sus ideas avanzadas sean fácilmente aceptadas. Su deber es sentar las bases para los que vendrán, señalar el camino."

Nikola Tesla

Las prestaciones y la tolerancia a fallos en redes de interconexión son factores clave de diseño para los sistemas de cómputo de altas prestaciones (HPC). Otro factor importante es el coste de la red. Las topologías indirectas a menudo son elegidas en el diseño de estos sistemas. Entre ellas, la topología más usada es el fat-tree. Por ello, en este capítulo nos centramos en obtener los máximos beneficios de los recursos de la red, diseñando una topología indirecta que ofrezca altas prestaciones y sea tolerante a fallos, mientras mantenemos el coste del hardware lo más bajo posible. Para alcanzar estos objetivos, proponemos algunas modificaciones a la topología fat-tree para sacar el máximo provecho de los recursos hardware consumidos por la topología. En particular, proponemos una nueva familia de topologías con diferentes propiedades en términos de coste, prestaciones y tolerancia a fallos.

Primero, proponemos una topología sencilla, a la que denominamos RUFT-PL. Esta duplica los enlaces de RUFT, alcanzando un coste de hardware similar a la topología fat-tree. Los enlaces extra siguen el mismo patrón de conexión usado por RUFT, siendo paralelos a los enlaces de la topología base. Esta topología, como veremos en el Capítulo 4, es capaz de doblar las prestaciones de RUFT y de fat-tree, además de proporcionar tolerancia a fallos.

La siguiente propuesta de este capítulo, referida como FT-RUFT, incrementa el grado de tolerancia a fallos de RUFT-PL que solo soporta un fallo en los enlaces de inyección, red y eyección. Para aumentar este nivel de tolerancia a fallos, se ha rediseñado la inyección y la eyección para incrementar el número de rutas disjuntas que se proporcionan en la red a cada par origen-destino. Esta propuesta consta, a su vez, de tres variantes: FT-RUFT-212, FT-RUFT-222 y FT-RUFT-XL. A diferencia de RUFT-PL, estas topologías modifican el patrón de conexión de los nodos de procesamiento con la red para incrementar el nivel de tolerancia a fallos. Además, FT-RUFT-XL introduce un nuevo patrón de interconexión en los enlaces de red que permite incrementar el número de rutas ofrecidas por las anteriores topologías. Un aspecto muy importante de estas topologías es que el hardware requerido para su implementación tiene un coste inferior o similar al requerido por el fat-tree.

El resto del capítulo está organizado de la siguiente manera: primero, en la Sección 3.2 introducimos la topología RUFT-PL, describiendo cómo usando los mismos recursos hardware requeridos por el fat-tree podemos duplicar las prestaciones de la red. Luego en la Sección 3.3 describimos FT-RUFT-212, una topología que usa un hardware similar a RUFT, con la diferencia de que se introduce una modificación en los conmutadores que conectan con los nodos de procesamiento. En la Sección 3.4 describimos la topología FT-RUFT-222, que se deriva de las dos topologías previas y que incrementa las prestaciones de red. En la Sección 3.5 presentamos la topología FT-RUFT-XL, la cual modifica el patrón de conexión de los enlaces de red para incrementar significativamente el número de rutas para cada par origen-destino. En la Sección 3.6 describimos un nuevo mecanismo de tolerancia a fallos para las topologías propuestas, basado en un modelo estático, el cual nos permite conocer el estado de toda la red, haciendo uso de pocos recursos hardware. Este mecanismo trabaja tanto con fallos de enlaces como de conmutadores. Por último, algunas conclusiones

son presentadas en la Sección 3.7.

3.1. Introducción

Los grandes sistemas de cómputo paralelo han sido o están siendo construidos con miles de nodos de procesamiento [10]. En estos sistemas, las prestaciones, la tolerancia a fallos y el coste de la red de interconexión juegan un papel primordial en el diseño de todo el sistema. Los niveles requeridos en la potencia de cálculo solo pueden ser alcanzados incrementando el número de nodos que lo componen. A medida que el sistema crece también lo hace la cantidad de recursos de red y por lo tanto la probabilidad de un fallo en la red. Dado que la disponibilidad de estos sistemas es una preocupación, los mecanismos de tolerancia a fallos a menudo son implementados basándose en el incremento de los recursos de la red y por lo tanto incrementando su coste.

El fat-tree, una topología multietapa bidireccional, es una de las topologías más utilizadas en máquinas de gran tamaño [10] dado que ofrece unas buenas prestaciones y tolerancia a fallos; sin embargo el hardware utilizado por la topología no es aprovechado de la mejor manera. La topología RUFT (Reduced Unidirectional Fat Tree) [39] es una MIN unidireccional que fue propuesta como una topología alternativa más simple y de menor coste que el fat-tree, que reduce a más de la mitad la complejidad del hardware de los conmutadores de interconexión, usando un encaminamiento determinista que permite eliminar la fase de bajada de los paquetes en la topología fat-tree. Las prestaciones que ofrece RUFT son similares al fat-tree a pesar de que utiliza menor cantidad de hardware (aproximadamente la mitad). Sin embargo, el punto débil de RUFT es que no ofrece ningún soporte en cuanto a tolerancia a fallos.

Por ello, en esta tesis nos centramos en la elaboración de nuevas topologías MIN que mejoren las prestaciones, la tolerancia a fallos y coste con respecto a otras topologías MIN comunes, organizando los recursos de hardware de una manera diferente y manteniendo el coste del hardware tan bajo como sea posible.

Como hemos visto en el Capítulo 2, el fat-tree con encaminamiento adaptativo, fat-tree con DESTRO y RUFT son topologías que implementan diferentes algoritmos de encaminamiento, y que requieren conmutadores de dife-

rente complejidad, pero con unas prestaciones de red muy similares. Teniendo en cuenta la diferencia en la complejidad del conmutador que requiere cada una de ellas, hemos propuesto una nueva familia de topologías que mejoran las prestaciones de RUFT, usando conmutadores más complejos. La idea es obtener nuevas topologías con una complejidad en el conmutador similar al fat-tree.

Al igual que RUFT, estas nuevas topologías son MINs unidireccionales, por lo tanto, todos los paquetes tendrán que dar el mismo número de saltos (el número de etapas de la red más uno) sin tener en cuenta el par origen-destino que comunican. Este comportamiento podría ser una desventaja ante las aplicaciones que explotan el parámetro de localidad en sistemas paralelos. Por otra parte, la latencia de los paquetes será más uniforme y menos sensible a los cambios, reduciendo el jitter experimentado por las aplicaciones.

3.2. RUFT-PL

RUFT-PL (RUFT with Parallel Links) es una topología de red que usa el mismo número de conmutadores que el fat-tree y RUFT, y con el mismo esquema de conexión entre los elementos de la red, con la diferencia de que la complejidad del conmutador es similar a la del fat-tree. El propósito es implementar la topología RUFT pero sin reducir la complejidad del conmutador. Dado que el fat-tree usa conmutadores con puertos bidireccionales, los conmutadores de RUFT-PL pueden doblar el número de puertos unidireccionales que usa RUFT.

En lo que sigue, en nuestras propuestas, nos referiremos como puerto agregado a la unión de dos canales unidireccionales que comunican con otro conmutador o nodo de procesamiento. En la Figura 3.1 puede verse el diagrama simplificado de un conmutador utilizado por RUFT-PL.

Como se puede apreciar, el conmutador tiene doble número de canales de entrada que RUFT, lo que conlleva un incremento en su complejidad. Nuestra propuesta es usar los k canales adicionales para tener dos enlaces en paralelo conectando cada par de conmutadores y nodos de la topología RUFT original. Estos dos enlaces paralelos proporcionan tolerancia a fallos y flexibilidad en el encaminamiento que puede ser explotado de diferentes formas. En la Figura

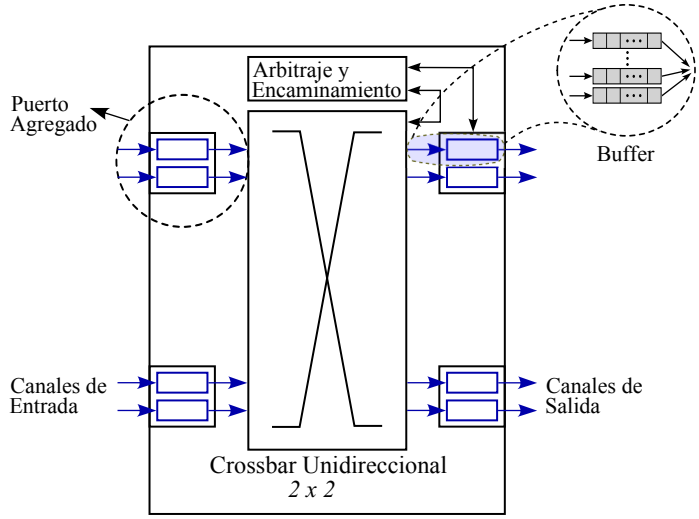


Figura 3.1: Vista simplificada de un conmutador de 2×2 con puertos unidireccionales.

3.2 se muestra un ejemplo de la topología RUFT-PL.

3.2.1. Selección de los enlaces paralelos

Como puede verse en la Figura 3.2, cada nodo de procesamiento está conectado a la red a través de dos enlaces. Para proporcionar tolerancia a fallos en este punto, la inyección de paquetes se realiza de forma dinámica, es decir, empleando cualquiera de los dos enlaces. En particular, usaremos el enlace que conecta con el canal del conmutador con más espacio en el buffer.

Desde aquí, en un conmutador dado, el par de canales de salida (es decir, el puerto agregado de salida) a ser usados están dados por la componente de destino correspondiente a la etapa del conmutador (como en DESTRO y RUFT). Sin embargo, para seleccionar cuál de los dos enlaces paralelos será usado hemos considerado dos criterios diferentes: *1*) seleccionar el enlace correspondiente al bit menos significativo de la componente del siguiente salto con el objetivo de clasificar los destinos en las dos colas disponibles, o *2*) seleccionar el enlace con más espacio en el buffer con el objetivo de dar flexibilidad al encaminamiento.

El primer enfoque aísla diferentes destinos entre ambos enlaces paralelos,

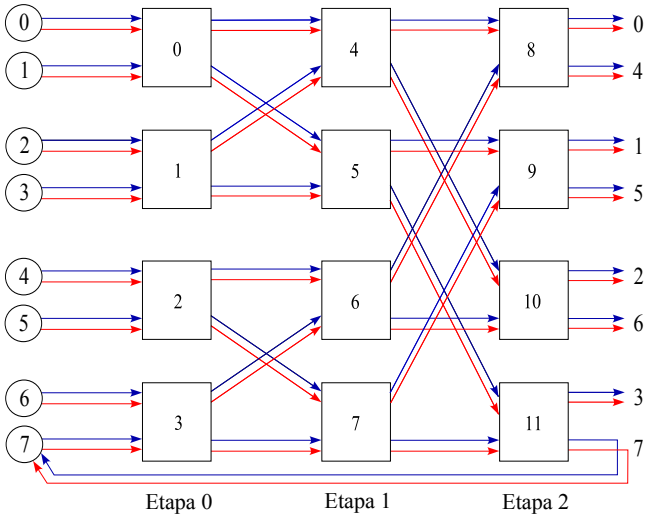


Figura 3.2: Topología RUFT-PL 2-ary 3-tree.

de esta manera el *HoL Blocking* que pueda existir en la fase ascendente de RUFT es reducido. La Figura 3.3 muestra como los nodos destino son clasificados entre los enlaces paralelos del conmutador, en una red 2-ary 3-tree utilizando este mecanismo de selección de los dos enlaces paralelos.

El inconveniente de este esquema es que implica una selección estática de los canales, lo cual no permite implementar tolerancia a fallos. Además, cuando la red está experimentando algún patrón de tráfico no uniforme, algunos enlaces no se utilizan.

Por otra parte, aplicando una función de selección dinámica para realizar la selección de los enlaces paralelos, podemos incrementar hasta dos el número de caminos para cada par origen-destino, tolerando un fallo en la red.

Usando el segundo criterio, podemos usar siempre ambos canales en paralelo, por puerto agregado, sin importar qué tipo de tráfico esté usando la red. Sin embargo, en este caso, la complejidad del conmutador se incrementa ya que cada canal de entrada puede solicitar cualquiera de los canales de salida, así que, la complejidad del conmutador será de $2k \times 2k = 4k^2$ elementos de conmutación. Por otra parte, como hemos mencionado, la tolerancia a fallos en la red se incrementa al poder enviar los paquetes por un enlace u otro, indiferentemente. Cuando ambos canales de salida tienen el mismo espacio li-

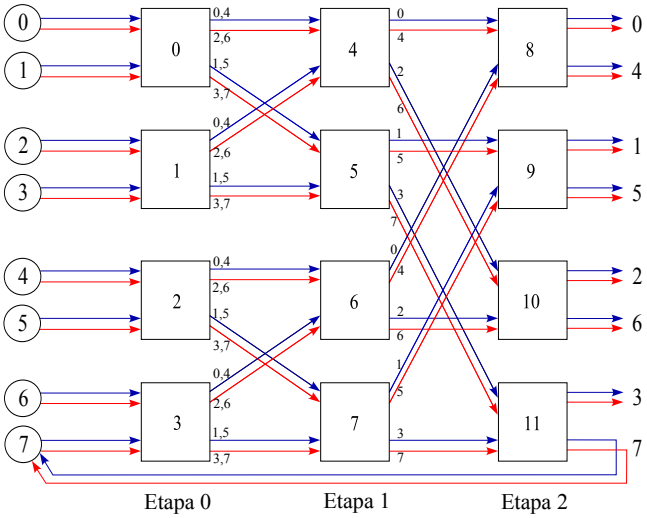


Figura 3.3: RUFT-PL 2-ary 3-tree usando encaminamiento con clasificación de destinos.

bre en el buffer, se realiza una selección aleatoria entre ambos para no cargar siempre la misma salida. En la Figura 3.4 podemos ver como los destinos pueden usar los dos enlaces paralelos. En cada una de las etapas el paquete puede usar cualquiera de los canales de salida disponibles durante todo su recorrido, incrementando las prestaciones de la red, como veremos en el Capítulo 4.

3.3. FT-RUFT-212

Esta propuesta es otra alternativa para proporcionar tolerancia a fallos e incrementar las prestaciones de RUFT haciendo uso de unos pocos enlaces adicionales con respecto a RUFT. El punto clave de esta nueva propuesta es la conexión de los nodos de procesamiento con la red.

En particular, cada nodo de procesamiento está conectado a dos conmutadores diferentes de la primera etapa. Por lo tanto, los nodos de procesamiento pueden usar dos rutas alternativas a través de diferentes conmutadores de la primera etapa. Los dos conmutadores son elegidos de tal manera que cada conmutador pertenezca a un subárbol diferente de la red y proporcione rutas completamente disjuntas para cada par origen-destino y por lo tanto tolerancia

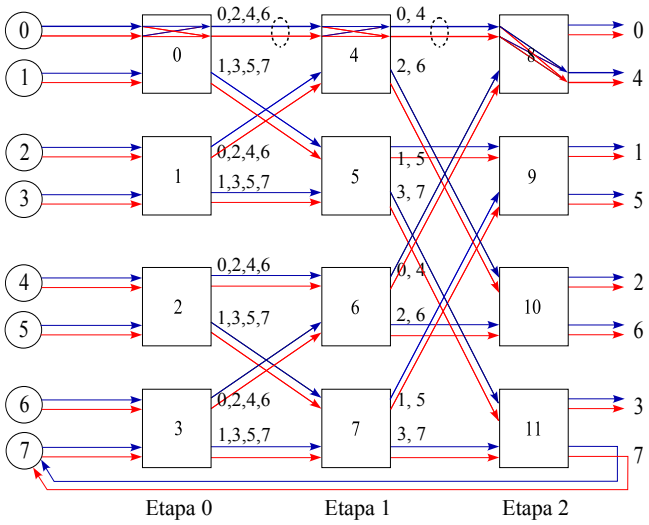


Figura 3.4: RUFT-PL 2-ary 3-tree usando encaminamiento con selección dinámica.

a fallos en la red, aparte de tolerar un fallo en la inyección.

Por otra parte, los nodos destino también tienen doble conexión en la última etapa, lo cual incrementa hasta cuatro el número de rutas alternativas para cada par origen-destino dentro de la red. Los nodos en la última etapa están conectados de tal manera que las rutas alternas sean lo más disjuntas posibles.

Esta propuesta es referida como FT-RUFT-212 (Fault-Tolerant RUFT 212) para indicar que es una variante de RUFT que tolera fallos en la cual existen 2 enlaces de inyección, 1 enlace de red y 2 enlaces de eyección. La Figura 3.5 muestra la conexión de los nodos en una red FT-RUFT-212 2-ary 3-tree. Nótese que los conmutadores de la primera y última etapa son asimétricos, con $2k \times k$ y $k \times 2k$ canales, respectivamente.

Cada nodo está conectado a la red a través de dos enlaces de inyección. El primer enlace se conecta al mismo conmutador que se conecta en RUFT y se le llamará enlace primario (mostrado en color azul en la figura). El identificador del conmutador con el que se debe conectar dicho enlace es obtenido dividiendo el identificador del nodo por la aridad de la red (id_nodo/k), por 2 en este ejemplo. El enlace alternativo que conecta cada nodo con la red será referido

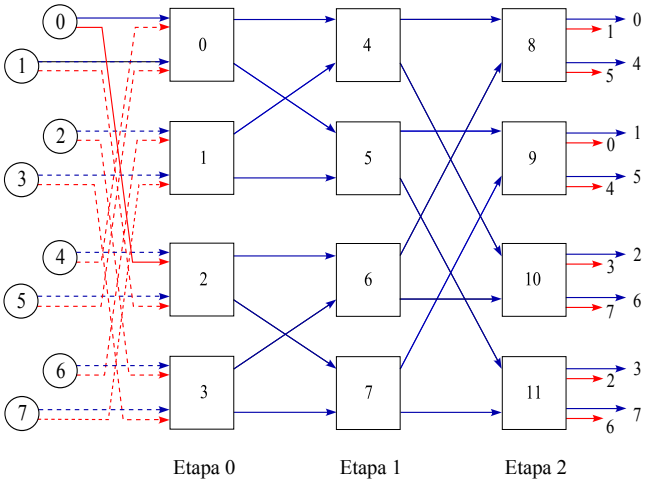


Figura 3.5: FT-RUFT-212 2-ary 3-tree.

como enlace secundario o alternativo (mostrado en rojo en la figura). Este enlace secundario se conecta a un conmutador diferente al usado por el enlace primario. En particular, éste se conecta al mismo conmutador donde se conecta el nodo con el mismo identificador pero invirtiendo su bit más significativo, por ejemplo, el enlace secundario del nodo 0 $\langle 000 \rangle$ va al conmutador donde conecta el nodo 4 $\langle 100 \rangle$, de esta misma forma, el enlace secundario del nodo 4 $\langle 100 \rangle$ va al conmutador donde conecta el nodo 0 $\langle 000 \rangle$. Siguiendo este criterio, nos aseguramos de tener doble inyección en dos puntos diferentes de la red que nos brinda dos rutas completamente disjuntas.

Como se ha indicado, cada nodo también está conectado a dos conmutadores diferentes en la última etapa con el fin de proporcionar tolerancia a fallos en la eyección y en la red, al incrementar el número de rutas alternas. En particular, conectamos cada nodo al conmutador donde se conecta en RUFT (a través del enlace primario, mostrado en azul en la figura) y, adicionalmente, al conmutador que conecta el nodo obtenido invirtiendo su componente menos significativa, al que llamaremos enlace secundario (mostrado en rojo en la figura). Por ejemplo, el nodo 0 $\langle 000 \rangle$ está conectado al conmutador 8, como en RUFT, y adicionalmente al conmutador 9, donde el nodo 1 $\langle 001 \rangle$ se conecta a través de su enlace primario. De esta manera, obtenemos cuatro rutas disjuntas en la red para cada par origen-destino.

3.3.1. Inyección y eyección de paquetes

Como hemos mencionado anteriormente, cada nodo tiene dos enlaces de inyección. A fin de proporcionar tolerancia a fallos y aumentar las prestaciones, la inyección de paquetes se realiza a través de los enlaces que conectan con el canal de entrada del conmutador con más espacio en el buffer para asignar el paquete. De esta manera, podemos distribuir la carga del tráfico entre dos segmentos de red diferentes. Cuando el espacio del buffer es igual en ambos canales, se realiza una selección aleatoria para no sobrecargar el mismo punto de entrada. Si hay un fallo en el camino hacia el destino a través de uno de los enlaces de inyección de un nodo fuente, el tráfico será enviado a través del enlace restante.

Cada conmutador en la última etapa de esta topología tiene dos enlaces de eyección. La doble eyección permite alcanzar un destino a través de la misma ruta que usaría RUFT (es decir, a través del enlace de eyección primario), pero también a través de la ruta que permite alcanzar el nodo con la componente menos significativa (Least Significant Bit o LSB) invertida en RUFT (es decir, a través del enlace de eyección secundario). Por ejemplo, para enviar un paquete al nodo 0 $\langle 000 \rangle$ a través del enlace de eyección secundario, éste tiene que ser enrutado como si su destino fuera el 1 $\langle 001 \rangle$. Cuando un paquete es enviado a través de la ruta secundaria, éste es etiquetado con un bit, que actúa como una marca, para indicar que debe usar el enlace de eyección alterno. La decisión acerca de que ruta tomará el paquete (la primaria o la secundaria) es tomada en la primera etapa, dependiendo del canal de salida seleccionado. En ausencia de fallos, cualquier función de selección podría ser usada, sin embargo, para proporcionar tolerancia a fallos, debería ser usada una función de selección basada en el estado de la red.

En esta propuesta, usamos el enlace de eyección en la primera etapa dependiendo del espacio disponible en los buffers de salida. Aquel con más espacio disponible, será el seleccionado para enviar el paquete. La forma de encaminar los paquetes se ha modificado en la primera etapa con respecto a RUFT, sin embargo el encaminamiento en el resto de etapas permanece igual, seleccionando el canal de salida de acuerdo a las componentes del paquete de destino (las cuales son las mismas para ambos casos, el paquete original y el alterno). Cuando el paquete alcanza la última etapa, éste es enviado a través del puerto

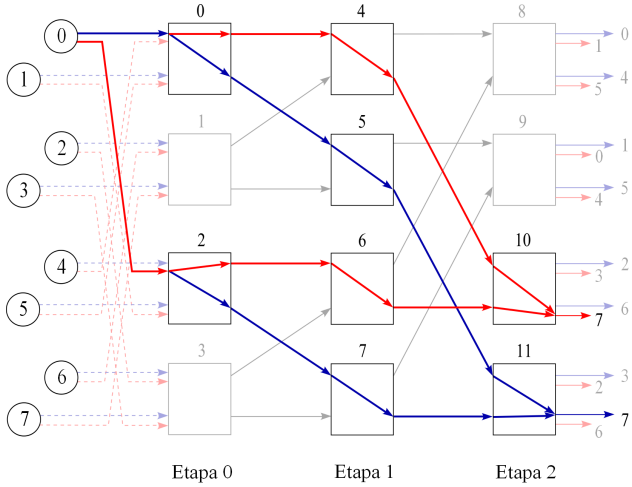


Figura 3.6: Rutas proporcionadas por FT-RUFT-212 en una red 2-ary 3-tree para un paquete con origen 0 y destino 7.

agregado que corresponda a la componente más significativa, usando el enlace primario o secundario, según indique la etiqueta del paquete.

3.3.2. Rutas disjuntas provistas por la red

Con la doble inyección/eyección descrita anteriormente, podemos proporcionar cuatro rutas diferentes dentro de la red para cada par origen-destino, las cuales son lo más disjuntas posibles, ofreciendo no solo tolerancia a fallos sino también mejoras en las prestaciones de la red, como veremos en la Capítulo 4. Como ejemplo, la Figura 3.6 muestra las distintas rutas que podría seguir un paquete desde el nodo 0 hasta el destino 7. Como se puede apreciar las rutas disponibles son lo más disjuntas posible.

En ausencia de fallos y siguiendo el mecanismo previamente descrito, la inyección se realizará a través de cualquiera de los dos enlaces de inyección (el enlace azul o el rojo). En la primera etapa, dependiendo de la carga de los buffers de los canales de salida, los paquetes usarán la ruta primaria (la ruta azul) o serán enviados a través de la ruta alterna (la ruta roja). En caso de fallos en los enlaces de inyección, red o eyección, el paquete puede ser enviado a través de cualquiera de las rutas disponibles.

Una de las grandes ventajas que ofrece FT-RUFT-212 es que, al proporcio-

nar hasta cuatro rutas para cada par origen-destino, la topología soporta hasta tres fallos de enlaces de red, y uno en la inyección y eyección. Por otra parte, a diferencia de RUFT-PL, FT-RUFT-212 también soporta fallos de conmutadores. Al tener dos enlaces de inyección conectados a segmentos o subárboles distintos en la red, en el mejor de los casos, podríamos prescindir de hasta la mitad de los conmutadores requeridos por la red, sin sacrificar ningún nodo de procesamiento.

3.4. FT-RUFT-222

En esta sección proponemos una topología de red que combina la doble inyección/eyección de FT-RUFT-212 con el uso de enlaces de red paralelos como hace RUFT-PL. Como resultado, obtenemos una topología que usa conmutadores simétricos e idénticos en todas las etapas. Esta propuesta combina las prestaciones que ofrece RUFT-PL con la tolerancia a fallos proporcionada por FT-RUFT-212. Esta topología es referida como FT-RUFT-222 para indicar que es una variante de RUFT que implementa tolerancia a fallos, que usa 2 enlaces de inyección para conectar los nodos de procesamiento a la red, 2 enlaces de red para interconectar los conmutadores y 2 enlaces de eyección que conectan con los nodos de procesamiento. FT-RUFT-222 usa dobles enlaces de inyección y de eyección como FT-RUFT-212, pero todos los conmutadores en la red son de $2k \times 2k$ (al igual que fat-tree y RUFT-PL), los cuales son acompañados por los dobles enlaces de red. La Figura 3.7 muestra un ejemplo de esta topología.

La gran ventaja ofrecida por esta topología es que, teniendo doble conexión entre todos los conmutadores, como hace RUFT-PL, y la inyección/ eyección de FT-RUFT-212, podemos incrementar el nivel de tolerancia a fallos. Como hemos visto, FT-RUFT-212 ofrece cuatro rutas diferentes, por la doble inyección/eyección. Ahora, FT-RUFT-222 gracias a los enlaces paralelos que conectan los conmutadores, puede proporcionar hasta ocho rutas alternativas, soportando un total de siete fallos en los enlaces de red.

Por otra parte, en ausencia de fallos, teniendo dobles enlaces de red podemos incrementar el número de rutas en el algoritmo de encaminamiento, y por lo tanto, las prestaciones de la red. La estrategia para inyectar tráfico en

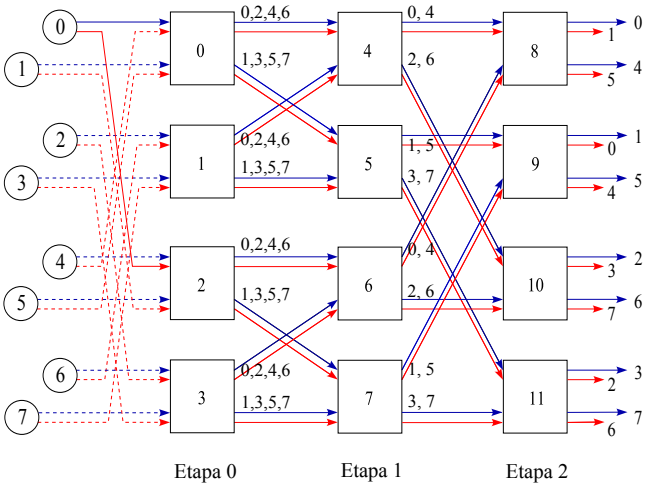


Figura 3.7: FT-RUFT-222 2-ary 3-tree.

la red es la misma utilizada por FT-RUFT-212. Para distribuir el tráfico en la primera etapa usamos los mismos criterios usados por FT-RUFT-212 (es decir, seleccionando el canal de salida con más capacidad en el buffer) con la ventaja que en esta nueva topología, desde los conmutadores de la primera etapa, tenemos cuatro posibles rutas para enviar paquetes al destino correspondiente, gracias a los dobles enlaces que permiten alternar los paquetes entre ellos. Desde la segunda etapa hasta la última etapa solo hay un par de canales de salida disponibles en cada conmutador, los cuales son seleccionados considerando la disponibilidad de sus buffers. Cuando el paquete se encuentra en la última etapa, solo hay disponible un canal para entregar el paquete. En la Figura 3.8 podemos ver todas las rutas disponibles que puede seguir un paquete con origen 0 y destino 7.

3.5. FT-RUFT-XL

Como hemos visto anteriormente, FT-RUFT-222 incrementa el nivel de tolerancia a fallos de las propuestas anteriores, ofreciendo hasta ocho rutas para cada par origen-destino, y haciendo uso del doble enlace podemos incrementar las prestaciones de la red, como veremos en el Capítulo 4.

En FT-RUFT-222, cuando un paquete va a ser inyectado en la red tiene

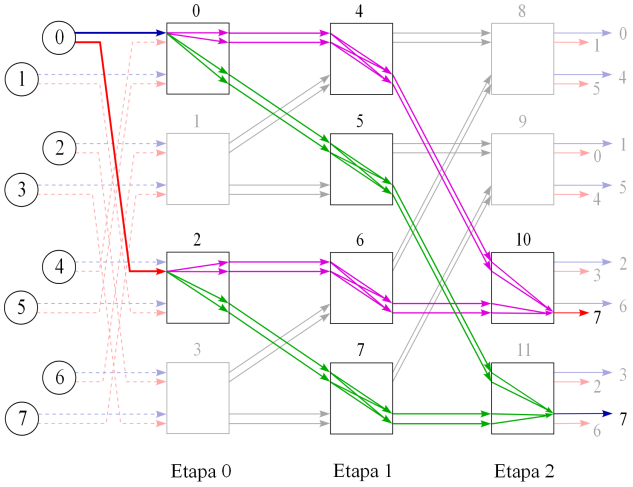


Figura 3.8: Rutas proporcionadas por FT-RUFT-222 en una red 2-ary 3-tree para un paquete con origen 0 y destino 7.

la posibilidad de usar cualquiera de los dos enlaces de inyección. Cuando el paquete ha sido inyectado y se encuentra en un conmutador de la primera etapa, éste tiene la posibilidad de ser encaminado a través de cualquiera de los cuatro enlaces disponibles. Sin embargo, cuando el paquete ha sido encaminado, desde este punto hasta el destino solo tiene dos rutas en paralelo, las cuales siempre atraviesan los mismos conmutadores. Esto quiere decir que, cualquiera de los dos canales de salida elegidos llevará al paquete al mismo conmutador en la siguiente etapa. Por consiguiente, si un conmutador en una de las etapas intermedias cae, estaría sacrificando dos rutas para cualquier par origen-destino que transite por este conmutador, además de afectar a los conmutadores de las etapas anteriores que conectan con él para enviar tráfico.

Para incrementar aún más la tolerancia a fallos y mantener unas buenas prestaciones en la red hemos propuesto una nueva topología de interconexión que, al igual que RUFT-PL y FT-RUFT-222, usa conmutadores simétricos y de igual complejidad, pero ahora los enlaces no se conectan en paralelo. Esta propuesta ha sido denominada FT-RUFT-XL (Fault-Tolerant RUFT with Cross Links) indicando que es una variante de la topología RUFT que tolera fallos y que utiliza dos topologías de red cruzadas entre sí.

3.5.1. Descripción de la topología

En FT-RUFT-XL todos los elementos de la red tienen dobles enlaces de conexión, al igual que RUFT-PL y FT-RUFT-222 (recuerde que cada conmutador dispone de $k \times k$ puertos agregados, y cada puerto dispone de dos canales físicos, los cuales denominamos como primario y secundario). Ahora, en vez de utilizar enlaces paralelos entre la conexión de los conmutadores, hemos propuesto utilizar dos topologías diferentes dentro de la misma red, una usando los canales primarios y otra distinta en lo que hemos llamado los canales secundarios. La combinación de estas dos topologías nos permite incrementar la tolerancia a fallos, a la vez que ofrece unas altas prestaciones en el sistema.

De manera formal podemos decir que, FT-RUFT-XL conecta $k^n = N$ nodos de procesamiento, usando nk^{n-1} conmutadores, cada conmutador con k puertos agregados de entrada y k puertos agregados de salida, donde cada uno de ellos consta de dos canales físicos (denominados como canal primario y canal secundario). Los puertos agregados de la red son enumerados desde 0 hasta $N - 1$ en cada etapa y en cada dirección; y cada puerto, SP, es representado por sus componentes en base k, $SP_{n-1}, SP_{n-2}, ..., SP_1, SP_0$.

Podríamos pensar que cualquier par de topologías unidireccionales existentes pueden ser implementadas en ambos canales; sin embargo, dichas topologías suelen diferir en sus patrones de conexión, y por lo tanto, serían incompatibles y no funcionales.

Por otra parte, el algoritmo de encaminamiento utilizado en nuestras propuestas siempre clasifica el tráfico, agrupándolo en los distintos conmutadores de cada etapa. Al clasificar la información, en cada etapa de la red aparecen una serie de conmutadores que comparten los mismos destinos. Estos conmutadores los hemos denominado conmutadores intercambiables.

De tal forma que, para incrementar el nivel de conectividad y tolerancia a fallos en la red, los enlaces secundarios de cada conmutador conectarán con un conmutador intercambiable, creando rutas alternas para cada par origen-destino.

Para comprender mejor el sistema de interconexión de esta propuesta, dividiremos la creación de la topología en dos esquemas: esquema de conexión primario y secundario. El esquema de conexión primario y secundario solo hará uso de los canales primarios y secundarios, respectivamente, para interconectar

los nodos de procesamiento a la red en la primera etapa, y los conmutadores entre sí. Los enlaces de retorno en la última etapa serán iguales para ambos esquemas.

3.5.2. Esquema de conexión primario

Este esquema de conexión solo utiliza los canales primarios de cada puerto agregado para conectar los nodos de procesamiento, en la inyección, y los conmutadores de las diferentes etapas, siguiendo el mismo patrón de interconexión de la topología RUFT. Es importante tener en cuenta que una red de interconexión puede ser descrita como un conjunto de funciones de interconexión, donde cada una es una permutación en el conjunto de direcciones origen-destino [61].

Por una parte, en este esquema de conexión, un nodo cuyas componentes son $p_{n-1}, p_{n-2}, ..., p_1, p_0$ estará conectado a través de su canal primario con el puerto agregado de red cuyas componentes correspondan a las del nodo en la primera etapa, es decir, el nodo i estará conectado al puerto agregado de red i, en los conmutadores de la primera etapa usando solo los canales primarios.

Por otra parte, la conexión entre los conmutadores está dada por una función de permutación que permite recrear o describir físicamente las conexiones de la topología. En RUFT, partiendo de un puerto de red (de salida), que se encuentra en la etapa s, podemos calcular el puerto de entrada en la etapa siguiente donde se conectará el enlace, permutando la componente 0 y la $s+1$, es decir, un puerto de red de la etapa s, dado por las componentes $SP_{n-1}, SP_{n-2}, ...SP_{s+1}, SP_s..., SP_1, SP_0$, estará conectado al puerto de entrada de la siguiente etapa que corresponda con las componentes $SP_{n-1}, SP_{n-2}, ...SP_0, SP_s..., SP_1, SP_{s+1}$. Por ejemplo, el puerto de red 5 $\langle 101 \rangle$ de salida de la etapa 0 estará conectado al puerto de red 6 $\langle 110 \rangle$ de entrada de la etapa 1, y el puerto de salida 6 $\langle 110 \rangle$ de la etapa 1 estará conectado al puerto de entrada 3 $\langle 011 \rangle$ de la etapa 2. Siguiendo este sencillo procedimiento, con cada puerto de red, podemos obtener el sistema de interconexión entre todos los conmutadores, al igual que RUFT, teniendo en cuenta que en nuestro caso este patrón de conexión siempre usará los canales primarios.

Por último, la conexión entre los conmutadores de la última etapa y los nodos de procesamiento se realizará de igual manera que FT-RUFT-212 y

FT-RUFT-222, utilizando dobles enlaces con rutas disjuntas; de esta manera, un puerto agregado de red cuyo identificador este dado por las componentes $SP_{n-1}, SP_{n-2}, ..., SP_1, SP_0$ a través del canal primario estará conectado al nodo de procesamiento cuyas componentes (en base k) correspondan a SP_0, SP_{n-1}, $SP_{n-2}, ..., SP_1$, es decir, rotando las componentes una vez a la derecha. Por ejemplo, en una red 2-ary 3-tree, el puerto agregado de eyección 6 $\langle 110 \rangle$ de la última etapa, a través del canal primario conectará con el nodo 3 $\langle 011 \rangle$. En cuanto a los enlaces de eyección secundarios de la última etapa, éstos conectarán con los mismos nodos que los enlaces primarios, pero invirtiendo el bit menos significativo de su identificador en binario, por ejemplo, en el puerto agregado de red 6 mencionado previamente, a través del canal secundario éste conectará con el nodo 2 $\langle 010 \rangle$, el cual se corresponde al nodo 3 $\langle 011 \rangle$ pero invirtiendo la componente de menor peso.

Como ya hemos visto, el simple hecho de tener una doble eyección disjunta nos permite elevar el nivel de tolerancia a fallos. En la Figura 3.9 podemos ver el sistema de conexión descrito previamente que, como puede observarse, la conexión de los enlaces de red y de eyección corresponden a la interconexión de FT-RUFT-212.

Gracias al encaminamiento determinista usado por RUFT, podemos ver como los paquetes son clasificados y agrupados, además de poder ser encaminados por dos rutas diferentes, gracias a esta doble eyección. Por ejemplo, en la Figura 3.9, los conmutadores 4 y 6 solo enrutan los destinos pares, y los conmutadores 5 y 7 solo enrutan los destinos impares; sin embargo cualquiera de ellos nos permite alcanzar los mismos destinos. Los paquetes que requieran usar los enlaces de eyección alternos, deberán modificar su LSB en la primera etapa, al igual que FT-RUFT-212 y FT-RUFT-222.

3.5.3. Esquema de conexión secundario

La topología descrita previamente solo utiliza los canales primarios de los puertos agregados para conectar los nodos en la inyección y los conmutadores entre sí. Ahora, con los recursos de hardware disponibles implementaremos una nueva topología que permita que los paquetes alcancen los mismos destinos, pero a través de un nuevo esquema de conexión.

En este nuevo esquema, en la inyección, los nodos de procesamiento estarán

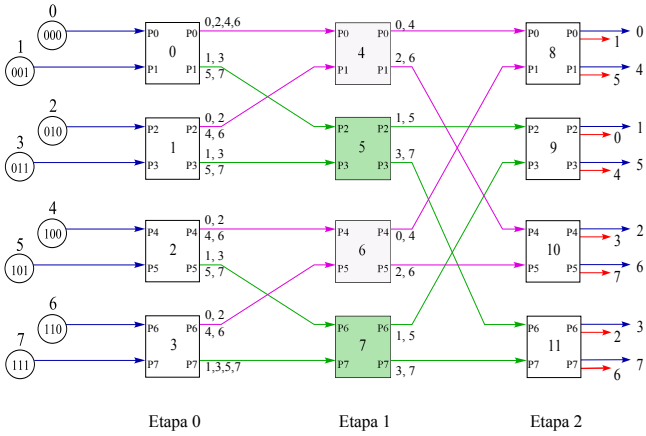


Figura 3.9: FT-RUFT-XL 2-ary 3-tree, esquema de conexión primario.

conectados con el puerto agregado de red cuyas componentes binarias correspondan con las componentes del nodo pero invirtiendo el segundo bit más significativo. De manera formal podemos decir que, un nodo de procesamiento cuyas componentes binarias esten dadas por $p_{n-1}, p_{n-2}, ..., p_1, p_0$, estará conectado al canal secundario del puerto agregado de red cuyas componentes se correspondan con $p_{n-1}, \overline{p_{n-2}}, ..., p_1, p_0$. Por ejemplo en una red 2-ary 3-tree, el nodo 0 $\langle 000 \rangle$, a través de su enlace secundario, se conecta con el puerto agregado de red 2 $\langle 010 \rangle$, de igual forma, el nodo 3 $\langle 011 \rangle$, a través de su enlace secundario, se conecta con el puerto agregado 1 $\langle 001 \rangle$.

Al invertir la segunda componente más significativa del nodo obtenemos un puerto agregado de inyección totalmente diferente al que usa el esquema de conexión primario. En este caso, no usamos el mismo patrón de conexión que FT-RUFT-212 y FT-RUFT-222 porque la conexión de los enlaces de red a través del canal secundario ya nos brindará esta misma ruta (ver más adelante).

Ahora, la conexión secundaria entre los conmutadores se realiza a través de un proceso que consta de dos partes: primero conectamos los conmutadores que se encuentran entre la etapa 0 y la etapa $n-2$, aplicando una misma función de permutación e inversión de bits; y por último, conectamos los conmutadores de la etapa $n-2$ y la etapa $n-1$, usando una inversión de bits diferente a la usada en las etapas anteriores, de modo que nos permita alcanzar los mismos

destinos e incrementar la tolerancia a fallos.

En particular, todos los conmutadores cuyo identificador de etapa s, se encuentre en el rango $0 \leq s \leq n-2$ usarán la función de permutación siguiente: un puerto agregado de red cuyas componentes están dadas por $SP_{n-1}, SP_{n-2}, ..., SP_1, SP_0$, y que se encuentra en la etapa s, estará conectado a un puerto agregado de entrada de la etapa $s+1$ donde sus componentes correspondan con las del puerto agregado de origen, permutando las componentes 0 y $s+1$ e invirtiendo además el bit $n-1$. Por ejemplo, si en una red 2-ary 3-tree tenemos el puerto agregado de salida 1 $\langle 001 \rangle$ en la etapa 0, al aplicar la permutación entre las componentes 0 y 1 obtenemos el identificador del puerto agregado de entrada 2 $\langle 010 \rangle$, en la siguiente etapa; ahora, invirtiendo el bit más significativo obtenemos el identificador 6 $\langle 110 \rangle$, que será el puerto agregado de entrada donde se conecta el enlace secundario. De la misma forma, si en una red 4-ary 3-tree tenemos el puerto agregado de salida 3 $\langle 003 \rangle$ en la etapa 0, al aplicar la permutación entre la componente 0 y 1 obtenemos el identificador 12 $\langle 030 \rangle$ (en binario $\langle 001100 \rangle$), y al invertir el bit más significativo obtenemos el 44 $\langle 101100 \rangle$, siendo este el puerto agregado de entrada para realizar la conexión en la siguiente etapa.

Por otra parte, el patrón de conexión entre los conmutadores de la etapa $n-2$ y $n-1$ (penúltima y última etapa) se calcula permutando las componentes 0 y $n-1$ del puerto agregado de partida e invirtiendo el segundo bit menos significativo del valor resultante. Por ejemplo, si en una red 2-ary 3-tree tomamos el puerto agregado de salida 4 $\langle 100 \rangle$, de la etapa 1, y aplicamos la permutación indicada obtenemos el valor 1 $\langle 001 \rangle$. Ahora, invirtiendo el segundo bit menos significativo del valor previo obtenemos el puerto agregado 3 $\langle 011 \rangle$.

Aplicando este patrón de conexión entre la penúltima y última etapa de la red, creamos una ruta para cada par origen-destino completamente distinta a la proporcionada por el esquema de conexión primario, la cual nos permite alcanzar los mismos destinos a través de diferentes conmutadores.

Por último, los enlaces de eyección en la última etapa se conectarán con los mismos nodos destino que el esquema de conexión primario, siguiendo el mismo procedimiento para calcular el nodo correspondiente a cada puerto agregado y canal de salida.

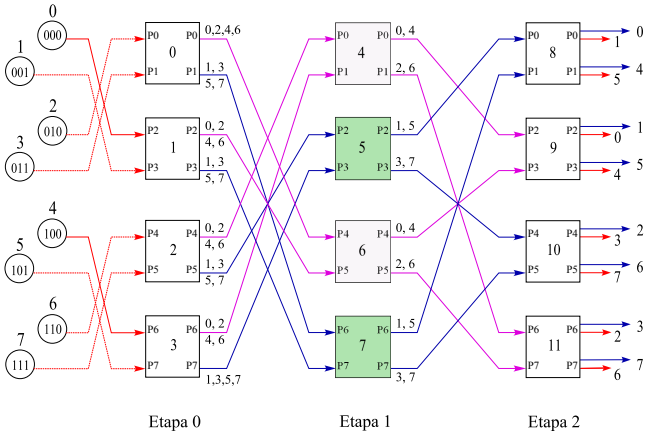


Figura 3.10: FT-RUFT-XL 2-ary 3-tree, esquema de conexión secundario.

En la Figura 3.10 se muestra la topología de red creada siguiendo los patrones de conexión previamente explicados. Como podemos apreciar, los conmutadores de la etapa intermedia siguen agrupando los mismos destinos que el esquema de conexión primario, es decir, en el ejemplo los conmutadores 4 y 6 siguen encaminando los destinos pares, y los conmutadores 5 y 7 encaminan los destinos impares. Además, a través de los enlaces de eyección de la última etapa, los destinos alcanzados concuerdan con los del esquema anterior (Figura 3.9).

Teniendo en cuenta que los esquemas de conexión explicados previamente comparten los mismos conmutadores a través de diferentes enlaces, podemos unir ambos esquemas y crear una topología que amplía el nivel de tolerancia a fallos usando un hardware con un coste similar al requerido por fat-tree, además de mantener las propiedades del encaminamiento utilizado por RUFT.

En la Figura 3.11 podemos ver la topología final, que incluye el esquema primario y secundario. También podemos notar que en la etapa intermedia hay una serie de conmutadores representados por el mismo color que, como hemos mencionado antes, hemos denominado conmutadores intercambiables.

Los conmutadores intercambiables son aquellos que nos permiten alcanzar los mismos destinos a través de conmutadores que pertenecen a diferentes subárboles de la red. En este caso, el conmutador 4 es intercambiable con el 6, y el conmutador 5 es intercambiable con el 7, ya que ambos son alcanzables

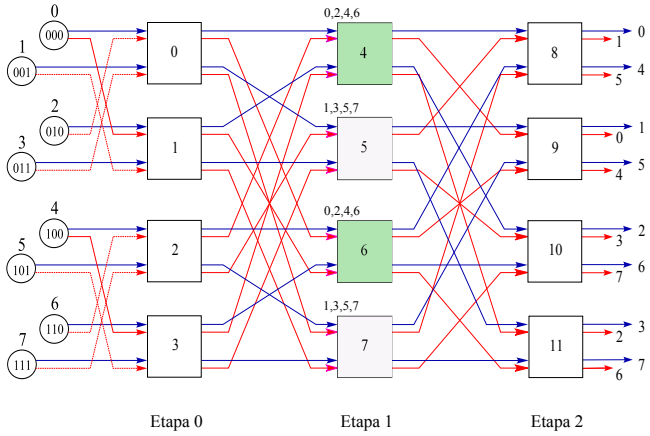


Figura 3.11: FT-RUFT-XL 2-ary 3-tree, topología final.

desde la etapa anterior, y encaminan los mismos destinos.

Dado que la topología representada en la Figura 3.11 es de un tamaño pequeño (3 etapas y 8 nodos), no podemos apreciar bien las ventajas que ofrece FT-RUFT-XL. Por ello, vamos a implementar una red 2-ary 4-tree siguiendo los procedimientos anteriores y de una forma más práctica.

Una red 2-ary 4-tree consta de 16 nodos y 4 etapas, cada una de ellas con 8 conmutadores. Cada nodo y puerto agregado de red de una etapa, en cada dirección, es enumerado de 0 a $N - 1$, es decir, de 0 a 15. Como podemos ver en la Tabla 3.1 el enlace primario de inyección de cada nodo se conecta al puerto agregado de red que comparta su mismo identificador, mientras que para calcular el puerto agregado del enlace secundario solo basta con invertir el segundo bit más significativo del identificador del nodo. Con este procedimiento, ambos enlaces de inyección del nodo estarán conectados a diferentes conmutadores en la primera etapa de la red.

Para conectar los conmutadores de la etapa 0 y 1, tomamos las componentes (en base k) del puerto agregado inicial y permutamos la componente 0 y 1. Esta permutación nos da el identificador del puerto agregado destino donde se conectará el enlace utilizado por el canal primario. Ahora, para obtener el identificador del puerto agregado donde se conectará el enlace secundario convertimos a binario el identificador calculado previamente e invertimos el bit más significativo (Tabla 3.2).

Id del nodo	Id en bits del nodo	Id puerto[1] dest. canal primario	Inversión 2º. bit más significativo	Id puerto[1] dest. canal secundario
0	0000	0	0100	4
1	0001	1	0101	5
2	0010	2	0110	6
3	0011	3	0111	7
4	0100	4	0000	0
5	0101	5	0001	1
6	0110	6	0010	2
7	0111	7	0011	3
8	1000	8	1100	12
...	...	...	...	...
15	1111	15	1011	11

[1] Identificador del puerto agregado.

Tabla 3.1: Conexión entre los nodos y conmutadores de la primera etapa, en una red 2-ary 4-tree.

Id Inicial del puerto[1] de red	Componentes del identificador	Permutación componentes 0 y 1	Id puerto[1] dest. canal primario	Inversión bit más significativo	Id puerto[1] dest. canal secundario
0	0000	0000	0	1000	8
1	0001	0010	2	1010	10
2	0010	0001	1	1001	9
3	0011	0011	3	1011	11
4	0100	0100	4	1100	12
5	0101	0110	6	1110	14
6	0110	0101	5	1101	13
7	0111	0111	7	1111	15
8	1000	1000	8	0000	0
...	...	...	...	...	...
15	1111	1111	15	0111	7

[1] Identificador del puerto agregado.

Tabla 3.2: Conexión entre los conmutadores de la etapa 0 y la etapa 1 en una red 2-ary 4-tree.

Como hemos descrito anteriormente, este proceso es aplicable a los conmutadores de todas las etapas, a excepción de la conexión de las dos últimas. De modo que, para calcular los puertos agregados de conexión entre la etapa 1 y 2, tanto para los enlaces primarios como secundarios, aplicamos el mismo proceso realizado entre la etapa 0 y 1, solo que la permutación se realizará entre las componentes 0 y 2, es decir, siempre entre la componente 0 y la que corresponda a la etapa que deseamos conectar. Ver Tabla 3.3.

La conexión entre los conmutadores de la penúltima y última etapa se realiza de forma similar a las etapas anteriores, es decir, para calcular el identificador del puerto agregado donde se conectará el enlace primario realizamos una permutación entre las componentes 0 y 3 del identificador del puerto agregado

Id Inicial del puerto[1] de red	Componentes del identificador	Permutación componentes 0 y 2	Id puerto[1] dest. canal primario	Inversión bit más significativo	Id puerto[1] dest. canal secundario
0	0000	0000	0	1000	8
1	0001	0100	4	1100	12
2	0010	0010	2	1010	10
3	0011	0110	6	1110	14
4	0100	0001	1	1001	9
5	0101	0101	5	1101	13
6	0110	0011	3	1011	11
7	0111	0111	7	1111	15
8	1000	1000	8	0000	0
...	...	...	...	...	...
15	1111	1111	15	0111	7

[1] Identificador del puerto agregado.

Tabla 3.3: Conexión entre los conmutadores de la etapa 1 y la etapa 2 en una red 2-ary 4-tree.

Id Inicial del puerto[1] de red	Componentes del identificador	Permutación componentes 0 y 3	Id puerto[1] dest. canal primario	Inversión 2º bit menos significativo	Id puerto[1] dest. canal secundario
0	0000	0000	0	0010	2
1	0001	1000	8	1010	10
2	0010	0010	2	0000	0
3	0011	1010	10	1000	8
4	0100	0100	4	0110	6
5	0101	1100	12	1110	14
6	0110	0110	6	0100	4
7	0111	1110	14	1100	12
8	1000	0001	1	0011	3
...	...	...	...	...	...
15	1111	1111	15	1101	13

[1] Identificador del puerto agregado.

Tabla 3.4: Conexión entre los conmutadores de la etapa 2 y 3 (penúltima y última etapa) en una red 2-ary 4-tree.

de red inicial, y a continuación, invirtiendo el segundo bit menos significativo de este identificador obtenemos el puerto agregado que se corresponde con el enlace secundario. Este procedimiento podemos verlo claramente en la Tabla 3.4.

Finalmente, en la última etapa solo nos queda conectar los nodos de procesamiento. Este procedimiento también se realiza de una forma muy sencilla. Tomando las componentes del identificador de red del puerto agregado de salida, en los conmutadores de la última etapa, rotamos a la derecha (una vez) las componentes, obteniendo de esta manera el identificador del nodo que se conectará con el enlace primario. Para calcular aquellos nodos que se conectan a través de los enlaces secundarios, solo basta con tomar el identificador cal-

Id del puerto[1]	Componentes del identificador	Rotación de las componentes	Id nodo dest. canal primario	Inversión bit menos significativo	Id nodo dest. canal secundario
0	0000	0000	0	0001	1
1	0001	1000	8	1001	9
2	0010	0001	1	0000	0
3	0011	1001	9	1008	8
4	0100	0010	2	0011	3
5	0101	1010	10	1011	11
6	0110	0011	3	0010	2
7	0111	1011	11	1010	10
8	1000	0100	4	0101	5
...	...	...	...	...	...
15	1111	1111	15	1110	14

[1] Identificador del puerto agregado.

Tabla 3.5: Conexión entre los conmutadores de la última etapa y los nodos de procesamiento en una red FT-RUFT-XL 2-ary 4-tree.

culado previamente e invertir el segundo bit menos significativo. En la Tabla 3.5 podemos ver reflejado este procedimiento.

Como hemos visto, la conexión entre todos los elementos de la red se realiza a través de un proceso bastante sencillo. El procedimiento anterior llevado a cabo sobre la red 2-ary 4-tree nos conduce a la topología mostrada en la Figura 3.12.

En esta topología, a medida que se incrementa el número de etapas en la red, las rutas entre cualquier par origen-destino se separan más de las rutas convencionales, aun así, dichas rutas permiten que los paquetes retomen su camino original en cualquier salto de la red, a la vez que mantienen el tráfico clasificado a lo largo de todo el encaminamiento.

En la Figura 3.12, podemos apreciar que en cada etapa intermedia aparecen una serie de conmutadores intecambiables. En esta red podemos ver que los grupos de conmutadores 8, 10, 12, 14 y 9, 11, 13, 15 son alcanzables desde cualquier nodo de procesamiento, permitiendo a su vez alcanzar los mismos destinos. Estos conmutadores intercambiables nos permiten distribuir los paquetes de la mejor manera y, aprovechando los recursos hardware disponibles en cada salto de la red.

3.5.4. Encaminamiento de paquetes

En FT-RUFT-XL, para aumentar el número de rutas y de esta forma aumentar la tolerancia a fallos y reducir la congestión de tráfico, la inyección

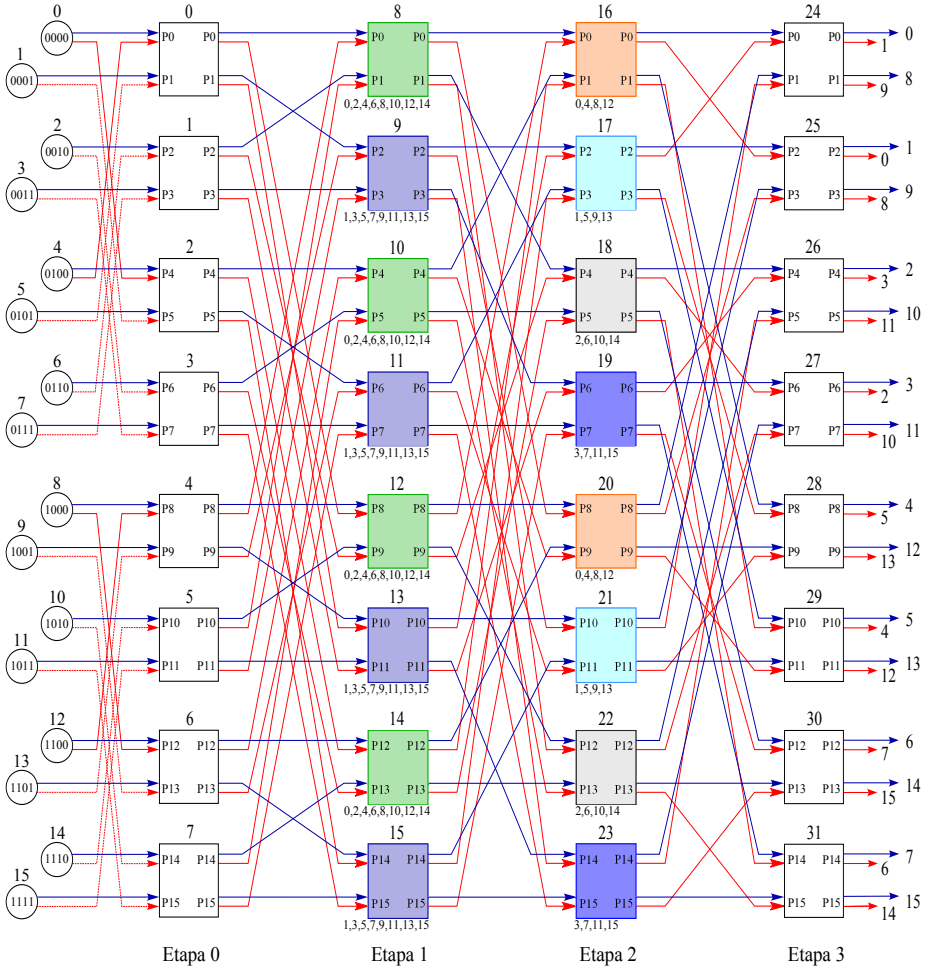


Figura 3.12: FT-RUFT-XL 2-ary 4-tree.

de paquetes se lleva a cabo de forma dinámica, es decir, seleccionando el canal de entrada al conmutador con mayor capacidad para albergar el paquete, igual que lo hace FT-RUFT-212 y FT-RUFT-222.

Por otra parte, el encaminamiento de paquetes dentro de la red se realiza de forma similar a FT-RUFT-222, es decir, seleccionando entre el par de canales de salida (primario y secundario) aquel que disponga de más espacio en el buffer para enviar el paquete, con la ventaja de que cada canal siempre conectará con un conmutador diferente en la siguiente etapa. Además, en la primera etapa podemos seguir modificando el LSB del paquete para encami-

narlo a través de la ruta alternativa (y entregarlo por el canal secundario de eyección), al igual que FT-RUFT-212 y FT-RUFT-222.

Una de las grandes ventajas de esta topología es que cada uno de los canales de salida disponible en cada conmutador proporciona una ruta diferente, distribuyendo el tráfico entre aquellos conmutadores que compartan los mismos destinos, permitiendo que los paquetes cambien de ruta en cada salto a medida que avanzan por la red, a diferencia de las propuestas anteriores, que cuando un paquete es encaminado en la primera etapa éste debe seguir la misma ruta hasta el destino (en el caso de RUFT-PL y FT-RUFT-222, la ruta proporcionada por los enlaces paralelos).

También cabe decir que, en FT-RUFT-XL, cuando un paquete selecciona la ruta alterna en la primera etapa, éste tiene la posibilidad de volver a seleccionar la ruta primaria en la penúltima etapa para elegir la ruta que se encuentre menos congestionada. En la última etapa, el paquete solo dispone de un único canal de salida para alcanzar el nodo destino.

3.5.5. Rutas disjuntas proporcionadas por la red

Con la doble inyección utilizada por los nodos de procesamiento, a través de diferentes conmutadores, garantizamos que las rutas proporcionadas por la red partan de subárboles diferentes, incrementando la tolerancia a fallos y soportando un fallo en la inyección.

En la primera etapa, cada uno de los conmutadores alcanzados proporciona cuatro canales de salida para cada par origen-destino, es decir, por medio de la doble inyección obtenemos ocho rutas de partida para cada paquete. A medida que los paquetes avanzan por la red, éstos serán enrutados a través de los conmutadores intercambiables que permiten que el tráfico se distribuya por las rutas menos congestionadas.

En la Figura 3.13 podemos ver las rutas disponibles que puede seguir un paquete con origen 0 y destino 7 en una red FT-RUFT-XL 2-ary 3-tree. Las rutas en color verde son las que alcanzan el destino a tráves del enlace de eyección primario, y las marcadas en morado son las que alcanzan el destino a través del enlace de eyección secundario. La inyección del paquete se puede realizar en el conmutador 0 a través del enlace primario (enlace azul) o en el conmutador 1 por medio del enlace secundario (enlace rojo), dependiendo del

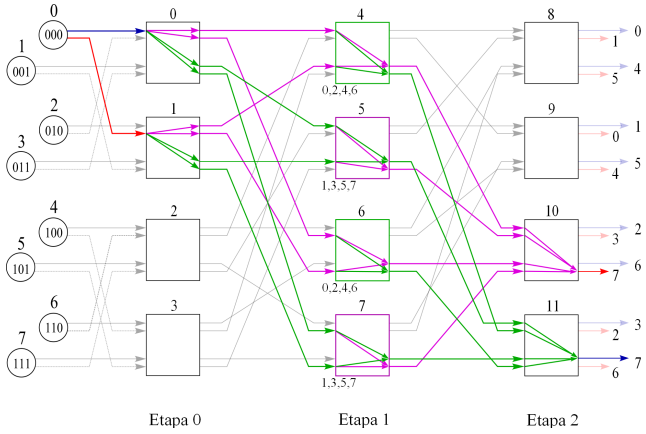


Figura 3.13: Rutas proporcionadas por FT-RUFT-XL en una red 2-ary 3-tree para un paquete con origen 0 y destino 7.

estado de los buffers. En ésta red, apreciamos que los conmutadores 4, 5, 6 y 7 de la segunda etapa son alcanzables desde los conmutadores 0 y 1, es decir, que el paquete puede ser enviado por la ruta principal (la de color verde) o por la ruta alterna (la de color morado) modificando su LSB en la primera etapa. Ahora, desde éstos conmutadores (pertenecientes a la penúltima etapa), el paquete tiene la posibilidad de elegir nuevamente la ruta que se encuentre menos congestionada en el salto previo antes de llegar al destino, basado en el estado de los buffers de salida del conmutador.

Como hemos mencionado anteriormente, FT-RUFT-XL tiene la ventaja de que cada canal de salida en un conmutador conecta con un conmutador diferente en la siguiente etapa. Esta característica hace que las rutas proporcionadas por la topología aumenten a medida que crece el número de etapas de la red. En el ejemplo anterior, para cualquier par origen-destino disponemos de dieciséis rutas compartidas por los conmutadores intercambiables, debido al tamaño de la red (solo ocho nodos de procesamiento), y que los conmutadores de la etapa intermedia se corresponden con los mismos de la penúltima etapa, lo que reduce el número de posibilidades de conexión. Para ver la diferencia con una red de mayor tamaño, veremos un ejemplo con una red que implemente más etapas.

En la Figura 3.14 podemos ver todas las rutas ofrecidas por la topología

FT-RUFT-XL en una red 2-ary 4-tree, para un paquete con origen 0 y destino 15. Como podemos notar, a medida que la red crece, la inyección de paquetes se realiza en conmutadores más distantes, separando cada vez más las rutas que proporciona la red.

A diferencia de la red mostrada en la Figura 3.13, en la red 2-ary 4-tree, el paquete inyectado puede llegar a cualquiera de los 8 conmutadores de la etapa 1 (del conmutador 8 al 15), usando la ruta principal (que conduce a los conmutadores 9, 11, 13 y 15) o modificando su LSB para usar la ruta alterna (que conduce a los conmutadores 8, 10, 12 y 14), abriendo un abanico de posibilidades a la hora de encaminar el tráfico hacia las etapas siguientes. Es decir, a medida que aumenta el número de etapas de la red, también lo hace el número de rutas disponibles para cada par origen-destino, teniendo la posibilidad de cambiar de ruta en cada salto hasta llegar a cualquiera de los cuatro conmutadores disponibles en la penúltima etapa. Si nos fijamos en los enlaces de estos conmutadores, vemos que cada uno de ellos está conectado a un conmutador diferente que nos permite alcanzar el destino por medio del enlace de eyección primario o el enlace de eyección secundario. Por lo tanto, contabilizando las rutas disponibles para cada paquete obtenemos 32 rutas dentro de la red, las cuales comparten los conmutadores intercambiables.

De manera general podemos decir que, una red FT-RUFT-XL k-ary n-tree dispone de 2^{n+1} rutas para cualquier par origen-destino, siendo estas compartidas por los diferentes conmutadores intercambiables, siempre agrupando el tráfico en cada etapa de red y reduciendo el *HoL Blocking* que pueda existir en la fase ascendente.

3.6. Tolerancia a fallos

Como se ha indicado repetidamente, la tolerancia a fallos en redes de interconexión es un tema de gran importancia en *clusters* de altas prestaciones.

Para garantizar la continuidad y funcionamiento del sistema ante la presencia de fallos se requiere de un mecanismo de tolerancia a fallos que permita mantener la comunicación entre los elementos de la red, aunque sea en un modo degradado, hasta que el componente que ha fallado sea reparado.

Cuando se consideran fallos en la red interconexión se pueden utilizar dos

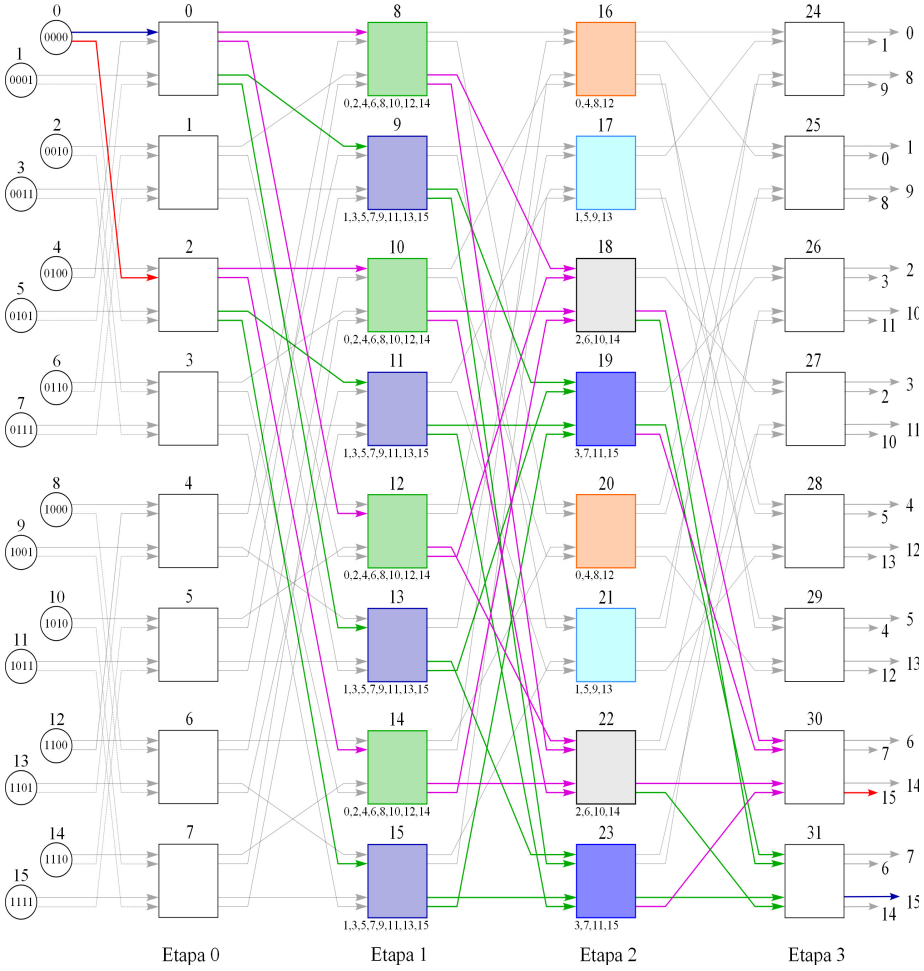


Figura 3.14: Rutas proporcionadas por FT-RUFT-XL en una red 2-ary 4-tree para un paquete con origen 0 y destino 15.

modelos de fallos, estático y dinámico.

En el modelo de fallos estático, cuando se detecta un fallo en la red, se debe realizar una reconfiguración [21] del sistema para computar el nuevo esquema de encaminamiento y actualizar el mecanismo de tolerancia a fallos utilizado.

A diferencia de los mecanismos de tolerancia a fallos estáticos, en los mecanismos dinámicos cuando se produce un fallo en la red no hace falta detener la actividad del sistema. En este caso, una vez se detecta el fallo en alguno de los conmutadores, los paquetes son redirigidos a través de otros puertos,

utilizando rutas alternativas que pueden ser menos óptimas para alcanzar su destino, pero manteniendo la actividad constante del sistema. Algunas de las técnicas de reconfiguración dinámica más novedosas que han sido propuestas en este campo las podemos encontrar en [40] y [58]. Estas dos propuestas se centran en las redes bidireccionales multietapa; específicamente en las k-ary n-tree. En esta tesis supondremos un modelo de fallos estático.

Muchos de los dispositivos comerciales, como por ejemplo, Myrinet e Infiniband, utilizan tablas para realizar su encaminamiento (*forwarding tables*). Por ejemplo, Infiniband realiza el envío de paquetes basado en tablas, que son programadas durante la inicialización y modificación de la red, las cuales son gestionadas por el software de configuración (*subnet manager*). Estas tablas pueden ser aleatorias o lineales. En las tablas lineales, el identificador local del nodo destino (DLID) del paquete entrante es usado como un índice dentro de la tabla, a fin de obtener el puerto de salida. Por otra parte, las tablas aleatorias son memorias de contenido direccionable, es decir, cada entrada en la tabla contiene dos campos: el identificador local del nodo destino y el puerto de salida a ser usado.

La ventaja que presentan las tablas lineales es que el coste del hardware es claramente más bajo que el requerido por las tablas aleatorias. Además, el acceso a las tablas lineales es directo, lo cual reduce considerablemente la latencia de red, en comparación con las tablas aleatorias, las cuales introducen cierto retardo en el envío de información, derivado del proceso de búsqueda, el cual es necesario para poder determinar la correspondencia del puerto de salida para un paquete dado.

Aun así, sin importar qué tipo de tablas se utilice para realizar el envío de información, éstas suelen tener un límite específico dado por el fabricante. Por ejemplo, en Infiniband, que es una de las tecnologías más usadas en redes de interconexión, según la especificación de su arquitectura [2], un conmutador no puede contener una tabla con más de 49152 entradas para paquetes unicast, ni más de 16383 entradas para paquetes multicast. Esta condición impide que el tamaño de la red vaya más allá de estos límites.

Como alternativa a las tablas de encaminamiento se han propuesto métodos basados en intervalos de encaminamiento [34, 41, 56] que requieren menos memoria. Estos métodos agrupan dentro de un intervalo todos los destinos

consecutivos que son físicamente alcanzables desde un puerto de salida. La ventaja que ofrece este método es que para ser implementado solo son necesarios unos pocos registros por puerto para almacenar los límites de dicho intervalo, además de reducir el tiempo de consulta usado por la tablas. Ahora, cuando un paquete va a ser enviado a través de un puerto de salida, solo basta con comprobar si el identificador del nodo destino se encuentra dentro de dicho intervalo, de esta forma se puede determinar si el nodo destino es alcanzable por dicho puerto. Para tolerar fallos, se añade a cada puerto uno o varios intervalos de exclusión [40], que indican los nodos que hay que eliminar en el intervalo (de exclusión), dado que son inalcanzables debido a los fallos producidos.

Aunque esta técnica reduce bastante la memoria y el tiempo de acceso requerido por las tablas de encaminamiento aleatorias, ésta presenta dos inconvenientes que son fundamentales para la tolerancia a fallos: el primero de ellos es que dentro de un intervalo de encaminamiento pueden existir varios nodos víctima [40], es decir, pueden aparecer nodos de procesamiento que aún son alcanzables por dicho puerto, pero al formar parte del intervalo de exclusión, para el puerto, éstos no son alcanzables. Por otra parte, la única forma de reducir el número de nodos víctima es incrementando el número de intervalos de exclusión, o visto de otra manera, la memoria requerida para almacenarlos. El otro inconveniente de los intervalos de encaminamiento es que debido a su naturaleza, éstos siempre tratan de agrupar los nodos no disponibles en conjuntos para reducir la cantidad de intervalos, y por lo tanto de registros. Sin embargo, en el momento que falla un enlace que conecta con un nodo de procesamiento (enlace de eyección) dicho fallo debe ser notificado a todos los conmutadores de la red, utilizando de esta manera, por cada puerto de salida, un registro para indicar un único nodo inalcanzable. Debido a este comportamiento, el máximo número de enlaces de eyección no consecutivos que pueden fallar y que son soportados por éste mecanismo es directamente proporcional al número de registros que implemente cada conmutador en los puertos de salida. Éste es el motivo por el cual los intervalos de encaminamiento no son una buena opción a la hora de implementar una red que requiera un alto nivel de tolerancia a fallos.

Para poder implementar un mecanismo de tolerancia a fallos en las to-

pologías propuestas en este trabajo podríamos utilizar alguna de las técnicas mencionadas previamente para poder determinar, ante la presencia de fallos, qué destinos son alcanzables desde un conmutador dado. Sin embargo, debemos tener en cuenta las siguientes consideraciones:

- Las tablas lineales nos permiten conocer el estado de toda la red. No obstante, éstas ocupan demasiada memoria, y más aún, en redes de gran tamaño.
- Las tablas aleatorias, requieren un hardware más complejo para su diseño, y además, éstas incrementan la latencia de la red debido al proceso de búsqueda requerido en ellas.
- Los intervalos de encaminamiento requieren menos cantidad de memoria pero no son una buena opción al sacrificar nodos víctima y sobre todo si consideramos fallos en los enlaces que conectan con los nodos de procesamiento.

Como podemos ver en los puntos anteriores cada uno de los mecanismos mencionados presentan su desventaja, lo cual hace que no sea óptimo para implementarlo en nuestras propuestas. Por ello, teniendo en cuenta estas consideraciones se ha desarrollado un nuevo mecanismo de tolerancia a fallos que se ajusta a nuestras topologías a través de un sencillo diseño.

Este nuevo mecanismo es de rápido acceso, fácil de implementar a nivel de hardware y además ofrece un alto nivel de detalle respecto a la conexión y a la alcanzabilidad de los elementos de la red, utilizando pocos recursos hardware en los conmutadores.

3.6.1. Vector de estados a nivel de bit

Para ofrecer tolerancia a fallos en los enlaces de red, inyección/eyección y a nivel de conmutador en las topologías propuestas, debemos conocer el estado de las rutas de todos los destinos alcanzables desde un conmutador y/o un nodo de procesamiento. De esta manera evitamos enviar paquetes por zonas donde existan fallos o existan dispositivos inalcanzables.

Conocer el estado de toda la red suele ser una tarea compleja, ya que habría que inundar toda la red con mensajes para notificar el estado de los

nodos. Además, hay que tener en cuenta que, almacenar toda esta información requiere de una cantidad considerable de memoria en los conmutadores.

A diferencia de las tablas de envío lineales, en las cuales se especifica qué puerto de salida debe tomar un paquete en base a su destino, nosotros proponemos asociar un vector a cada puerto de inyección del nodo de procesamiento y a cada puerto de salida de los conmutadores. Este vector, almacenará el estado de las rutas a los nodos destino que son alcanzables a través de dicho puerto. Teniendo en cuenta que el estado de un nodo será alcanzable o no alcanzable, solo con un bit podremos determinar el estado de alcanzabilidad de un nodo dado. De esta manera, una posición del vector que contenga un 0 indicará que el nodo no es alcanzable, y por el contrario, un 1 indicará que sí lo es. La implementación de este vector en el nodo de procesamiento es bastante simple ya que este buffer puede ser configurado/administrado desde el driver del controlador de red.

Al trabajar a nivel de bit y solo considerar la alcanzabilidad de los nodos podemos reducir la memoria requerida por cada vector asociado a los puertos. Por ejemplo, en una red 2-ary 3-tree con ocho nodos de procesamiento, en el puerto de inyección de un nodo solo será necesario un vector con ocho bits (1 *Byte*) para almacenar el estado de todos los nodos de la red. De manera formal, podemos decir que un vector asociado a un puerto de inyección de un nodo necesitará k^n bits para conocer el estado de todos los nodos.

Para saber a quién corresponde cada bit del vector utilizamos su propio índice como identificador del nodo destino. De este modo, si en un nodo de procesamiento queremos consultar si un destino con identificador i es alcanzable solo bastará con acceder a la posición i del vector y comprobar si su valor es 0 o 1 para determinar si es posible alcanzar dicho destino.

Por otra parte, en los puertos de salida de los conmutadores el tamaño requerido por un vector de estados será de k^n/k bits. Sin embargo, en cada etapa superior de conmutadores, el tamaño útil del vector se dividirá por k, dado que el número de destinos diferentes que llega a un conmutador de una etapa se divide entre la aridad del conmutador, gracias a la clasificación que realiza el algoritmo de encaminamiento. De manera formal, podemos decir que el número de bits requeridos en un conmutador que se encuentra en la etapa s es de $k^n/k^{s+1} \times k$ bits.

Red	Nodos	Vector Bits	Tabla Lineal
4-ary 3-tree	64	8	16
4-ary 4-tree	256	32	64
4-ary 5-tree	1024	128	256
8-ary 3-tree	512	64	192
8-ary 4-tree	4096	512	768
8-ary 5-tree	32768	4096	12288
16-ary 2-tree	256	32	128
16-ary 3-tree	4096	512	2048
16-ary 4-tree	65536	8192	32768

Tabla 3.6: Número de *Bytes* requeridos por conmutador en los mecanismos de tolerancia a fallos basados en vector de bits y tablas lineales.

En la Tabla 3.6 mostramos la memoria requerida (en *Bytes*) en un conmutador para el mecanismo de tolerancia a fallos descrito, basado en vector de bits, para diferentes tamaños de redes. A modo de comparación, mostramos también la memoria requerida por una tabla lineal para encaminamiento determinista. En este último caso, se requieren N entradas con $log_2 k$ bits por entrada, para representar un puerto. En la tabla, los valores mostrados para el vector de bits corresponden a un conmutador de la primera etapa. Recuerde que en cada etapa la memoria requerida por el mecanismo se reduce.

Como podemos apreciar, el vector de bits requiere una cantidad de memoria inferior a la de las tablas lineales a la vez que ofrece un punto de vista global de todo el sistema. Por ejemplo, una red 16-ary 4-tree con 65536 nodos de procesamiento que utiliza el mecanismo de tolerancia a fallos basado en vector de bits solo requiere 8KB de memoria en el conmutador para determinar si un mensaje se puede encaminar por un puerto dado.

Como hemos mencionado, cada vector solo guardará el estado de los destinos que son alcanzables a través de su correspondiente puerto. Ahora, para determinar qué bit del vector corresponde con un destino dado, solo es necesario dividir el identificador del nodo destino entre k^{s+1}, siendo s la etapa actual donde se encuentra el conmutador. De esta forma obtenemos la posición del vector que debemos acceder para determinar si un nodo destino es alcanzable por dicho puerto.

El encaminamiento utilizado en las topologías propuestas siempre clasifica el tráfico a través de los diferentes puertos de salida y nos indicará la salida que

puede tomar un paquete. Ahora, mediante el vector de bits, solo será necesario consultar que la ruta a utilizar esté disponible.

En la siguiente sección mostraremos como hacer uso de estos vectores para implementar nuestro mecanismo de tolerancia a fallos.

3.6.2. Mecanismo estático de tolerancia a fallos

La metodología propuesta en esta sección permite explotar la tolerancia a fallos ofrecida por las topologías presentadas en este capítulo. No obstante, ésta puede fácilmente ser extendida a cualquier topología que utilice encaminamiento determinista, tanto en redes unidireccionales como bidireccionales.

Concretamente, nos centramos en los fallos permanentes, porque los fallos temporales/transitorios pueden ser manejados por los protocolos de comunicación. Esta metodología trabaja con fallos de enlaces y de conmutadores en cualquier punto de la red. Sin embargo, un fallo de conmutador puede ser considerado como el fallo de todos sus enlaces. Por lo tanto, solo nos centraremos en los fallos de enlaces.

En esta sección consideramos un modelo de fallos estático, es decir, cada vez que se detecta un fallo en la red la actividad del sistema se detiene, la nueva información de encaminamiento es computada y actualizada, y finalmente se reanuda la actividad del sistema desde el último checkpoint. La detección de fallos podría ser basada en el uso de temporizadores, tal como se propone en [63]. Con respecto al checkpoint, se podría emplear cualquier técnica propuesta en la literatura, como por ejemplo la de checkpoint distribuido [64].

Como se ha indicado, nuestro mecanismo de tolerancia a fallos utiliza un vector de bits en los enlaces de inyección de los nodos y en cada puerto de salida de los conmutadores. Los vectores asociados a los puertos indicados nos permiten determinar rápidamente si el destino requerido es alcanzable.

En un principio, al inicializar el sistema, cada una de las posiciones de los vectores tendrá un valor de 1 para indicar que el destino correspondiente a este bit es alcanzable. En el momento que se detecta un fallo en un enlace, el conmutador que conecta este enlace, en el puerto de salida, será el encargado de notificar a los conmutadores de la etapa anterior los destinos que han dejado de estar disponibles a través de su ruta. Del mismo modo, si los conmutadores que reciben esta notificación no disponen de una ruta alterna para alcanzar el

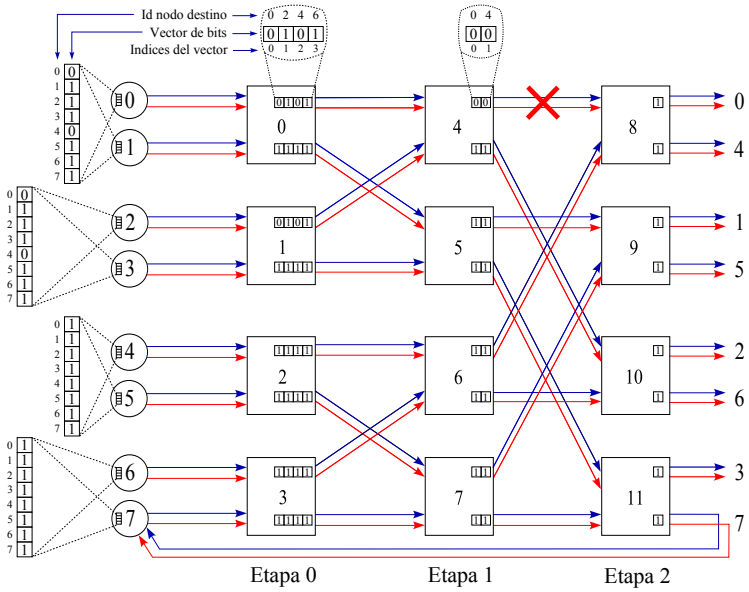


Figura 3.15: RUFT-PL 2-ary 3-tree usando vectores de estados para determinar la alcanzabilidad de los nodos destino.

destino notificado, el mensaje será notificado a los conmutadores de su etapa anterior. Luego, el conmutador de la primera etapa notifica a los nodos de procesamiento que estos destinos han dejado de estar disponibles a través del canal correspondiente. De esta manera el nodo de procesamiento tendrá una visión completa del resto de nodos pertenecientes a la red. El envío de mensajes de notificación de tolerancia a fallos se puede realizar por medio de los enlaces de control, de la misma forma que se notifican los créditos o señales utilizadas por la técnica de conmutación (Sección 2.1.4).

En la Figura 3.15 mostramos la topología RUFT-PL 2-ary 3-tree, haciendo uso del mecanismo propuesto. En este ejemplo, cada uno de los nodos de procesamiento tiene asociado un vector al puerto agregado de inyección de igual tamaño (en bits) al número de nodos de la red. También observamos que cada uno de los puertos agregados de salida de los conmutadores tiene asociado un vector, con un tamaño que varía en función de la etapa donde se encuentre el conmutador. Dado que en RUFT-PL cada par de enlaces paralelos siempre conecta con el mismo conmutador o nodo de procesamiento, solo será necesario un vector para ambos enlaces. En este caso, solo cuando fallen los dos enlaces

paralelos se notificará que algún destino ha dejado de estar alcanzable. Como podemos ver en el ejemplo, los enlaces paralelos que conectan los conmutadores 4 y 8 han fallado, dejando fuera de alcance los nodos de procesamiento 0 y 4. En este punto, el puerto agregado de salida del conmutador 4 detecta que sus enlaces han fallado, por lo tanto marca todos los destinos en su vector como inalcanzables, con un valor de 0.

Después de actualizar su vector de estados, se realiza una notificación a los conmutadores que conectan con él en la etapa anterior (conmutadores 0 y 1) para que actualicen sus correspondientes vectores. En el ejemplo podemos apreciar que los vectores asociados a los puertos agregados que conectan con el conmutador 4 tienen sus índices 0 y 2 establecidos en 0, lo cual indica que los nodos 0 y 4 no se pueden alcanzar a través de esta ruta.

Por último, los conmutadores de la primera etapa que reciben una notificación de cambio de estado comunican estos destinos a los nodos de procesamiento para que actualicen sus vectores. Como podemos ver, los nodos de procesamiento 0 a 3 tienen en su vector de estados los destinos 0 y 4 marcados como no alcanzables. Recuerden que la topología RUFT-PL soporta un fallo de enlace, y en este caso han fallado dos de ellos en paralelo. El resto de nodos y conmutadores que no se han visto afectados por este fallo mantienen su vector de estados con todos sus componentes a 1, es decir, que todos los destinos son alcanzables a través de ellos. En el Algoritmo 1 se describe el proceso de notificación de vectores de estado cuando se detecta un fallo en alguno de los enlaces de la red.

En la Figura 3.16 mostramos otro ejemplo de la utilización de los vectores de estado, pero en este caso, bajo la topología FT-RUFT-212. Como hemos visto, tanto FT-RUFT-212 como FT-RUFT-222 utilizan dobles enlaces para conectar los nodos de procesamiento con la red, pero a diferencia de RUFT-PL, la conexión de estos enlaces en la primera y última etapa se realiza a través de diferentes conmutadores para hacer que las rutas provistas por las topologías sean lo más disjuntas posibles e incrementar la tolerancia a fallos.

Dado que en la última etapa podemos alcanzar un mismo destino por medio de dos conmutadores diferentes, en cada uno de los puertos agregados de inyección del nodo de procesamiento y en cada puerto de salida de los conmutadores (a excepción de la última etapa) serán necesarios dos vectores

```
Function FalloEnlace_en_Conmutador()
    PuertoSalida.EstablecerVector(0);
    for i = 0 to Nb_PuertosEntrada − 1 do
        ObjetoPrev = PuertoEntrada[i].ObtenerVecino;
        NotificarFallos(PuertoSalida.Id, PuertoSalida.Vector, ObjetoPrev);
    end
end
Function NotificarFallos(Id_Pto, VectorBits, Objeto)
    if Objeto == Conmutador then
        Conmutador.PuertoSalida.ConfigurarVector(VectorBits, Id_Pto);
        if Conmutador.TieneRutasAlternas == Falso then
            for i = 0 to Nb_PuertosEntrada − 1 do
                ObjetoPrev = PuertoEntrada[i].ObtenerVecino;
                VectorBits = Conmutador.PuertoSalida.Vector;
                NotificarFallos(i, VectorBits, ObjetoPrev);
            end
        end
    else if Objeto == Nodo then
        PuertoInjeccion.ConfigurarVector(VectorBits, Id_Pto);
end
```

Algoritmo 1: Comunicación del vector de estados ante un fallo de enlace en un conmutador.

de estados, uno para determinar los destinos que se pueden alcanzar a través de las rutas primarias, y otro para los destinos que se pueden alcanzar a través de las rutas secundarias o alternas.

En este ejemplo (Figura 3.16), hemos supuesto que el enlace que conecta los conmutadores 4 y 8 ha fallado. Al fallar la única conexión que existe entre ambos elementos, el conmutador 4 establece en ambos vectores de estados todos sus destinos como inalcanzables. Como podemos apreciar en la figura, a través de los enlaces primarios de eyección no se puede llegar a los destinos 0 y 4, y a través de los enlaces de eyección secundarios no se puede llegar a los destinos 1 y 5. El conmutador 4 después de actualizar sus vectores de estado realiza una notificación a los conmutadores de la etapa anterior (conmutadores 0 y 1) para que actualicen sus respectivos vectores, y estos a su vez lo notifiquen a los nodos de procesamiento. En este caso, los nodos que reciben

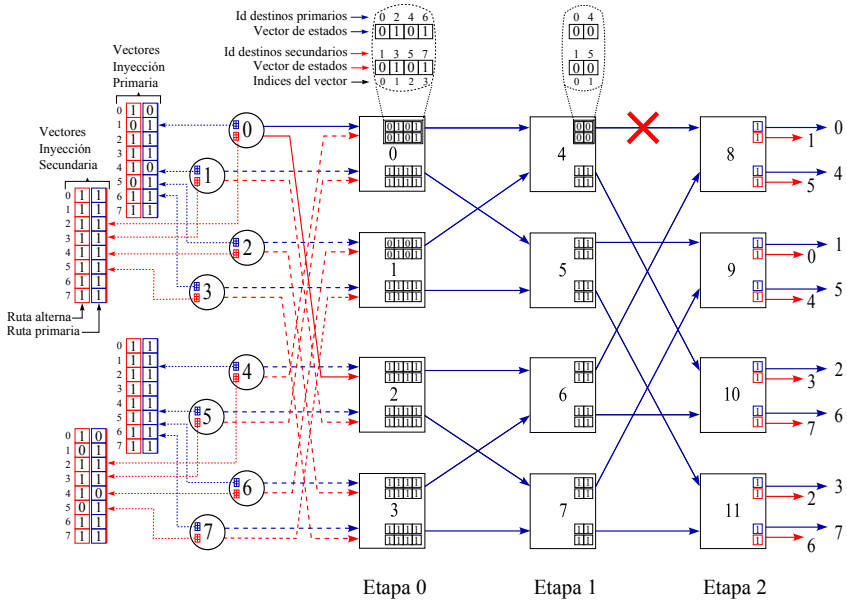


Figura 3.16: FT-RUFT-212 2-ary 3-tree usando vectores de estados para determinar la alcanzabilidad de los nodos destino.

esta notificación son los nodos 0 a 3, en la inyección primaria, y los nodos 4 a 7 en la inyección secundaria.

Con esta información disponible en los nodos de procesamiento podemos determinar por que canal de inyección podemos enviar los mensajes, tomando decisiones basadas en la información de los vectores de estado para equilibrar el tráfico entre las diferentes rutas. Nótese que en el caso de FT-RUFT-212 el fallo si que ha sido soportado.

El funcionamiento de este mecanismo se aplica de la misma forma a FT-RUFT-222, con la diferencia que las notificaciones por parte de los conmutadores solo se realizarán cuando ambos enlaces de red o uno de los enlaces de eyección falle.

En el último ejemplo tenemos la topología FT-RUFT-XL 2-ary 3-tree (Figura 3.17). En este caso hemos marcado con fallos los enlaces que conectan el conmutador 5 con los conmutadores 8 y 9. Como podemos apreciar, los enlaces de ambos conmutadores conectan con los mismos destinos en la última etapa, aunque éstos se encuentran distribuidos de diferente manera. Al igual que

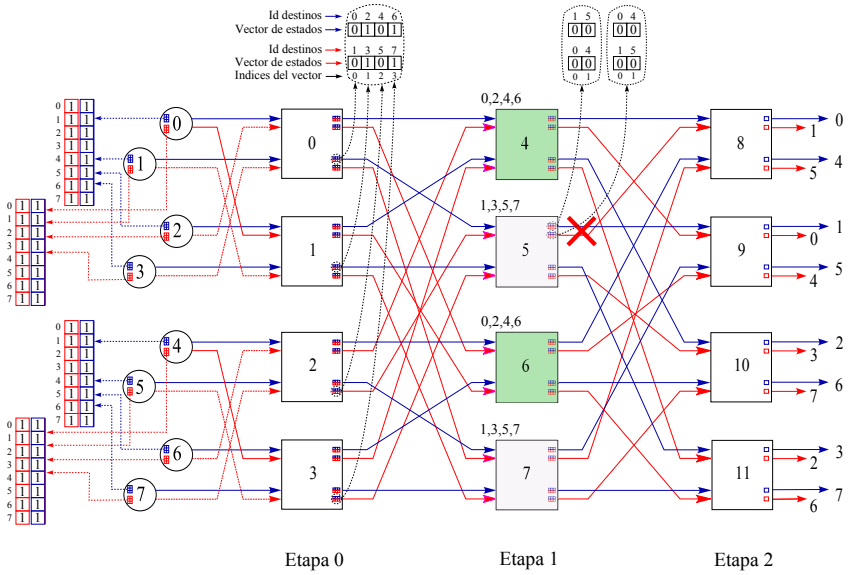


Figura 3.17: FT-RUFT-XL 2-ary 3-tree usando vectores de estados para determinar la alcanzabilidad de los nodos destino.

RUFT-PL y FT-RUFT-222, mientras un conmutador tenga conexión con sus destinos a través de uno de sus enlaces, no se realizará ninguna notificación de fallo a los otros conmutadores.

Como en este caso ambos enlaces han fallado, el conmutador 5 detecta el fallo y actualiza sus vectores de estado, notificando los destinos inalcanzables a todos los conmutadores que tienen conexión con él (conmutadores 0, 1, 2 y 3). Dado que cada uno de los conmutadores que recibe esta noficación aún pueden alcanzar todos sus destinos a través de los otros enlaces, la notificación no se reenvía a los nodos de procesamiento. Como podemos ver en la figura de este ejemplo, todos los nodos de procesamiento tienen cada elemento de sus vectores establecidos a 1, lo cual indica que cada nodo tiene conectividad con los demás a través de los enlaces de inyección primario y secundario. El resto de canales y conmutadores que no se ven afectados por este fallo también mantienen sus vectores de estado establecidos a 1.

La conectividad de esta topología evita que el mensaje de fallo se propague hasta los nodos de procesamiento, debido a que los paquetes pueden tomar rutas alternas en cada etapa de la red.

3.7. Conclusiones

A lo largo de este capítulo hemos presentado cuatro topologías unidireccionales (UMIN) que utilizan una cantidad de hardware similar al de la topología fat-tree, ofreciendo un buen nivel de tolerancia a fallos, a la vez que mantiene el coste de la red aquilatado.

La topología fat-tree utiliza conmutadores de $k \times k$ puertos bidireccionales, es decir, $2k \times 2k$ canales unidireccionales (un canal de subida y un canal de bajada, por puerto). Por otra parte, RUFT es una topología unidireccional que simplifica el hardware usado por fat-tree, usando menos de la mitad de elementos de cómputo requeridos por éste, implementando conmutadores de $k \times k$ puertos unidireccionales. Esta topología ofrece unas prestaciones de red similares a la del fat-tree, pero a un coste más bajo. Sin embargo, el inconveniente que presenta RUFT es que no proporciona tolerancia a fallos.

Para proporcionar tolerancia a fallos e incrementar las prestaciones de la red, hemos propuesto una nueva familia de topologías derivada de RUFT, las cuales hemos llamado RUFT-PL, FT-RUFT-212, FT-RUFT-222 y FT-RUFT-XL.

Nuestra primera propuesta, RUFT-PL, es una topología unidireccional que duplica el número de enlaces utilizados por RUFT, tanto en los enlaces de red como en la inyección/eyección. La forma de conectar estos enlaces sigue el mismo patrón de interconexión que la topología RUFT, es decir, cada par de enlaces comunican con el mismo conmutador en la siguiente etapa y con los nodos de procesamiento. Este incremento de hardware, junto al algoritmo de encaminamiento adecuado, permite reducir el *HoL Blocking* y obtener altas prestaciones, además de ofrecer cierto grado de tolerancia a fallos en la red, como veremos en los análisis llevados a cabo en el Capítulo 4.

Por otra parte, FT-RUFT-212 es una topología que implementa un nuevo modelo de conexión entre los nodos de procesamiento y los conmutadores de la primera y última etapa, pero manteniendo el mismo hardware dentro de la red, al igual que RUFT. El nuevo modelo de inyección/eyección que siguen los nodos de procesamiento, y la forma de distribuir el tráfico en los conmutadores de la primera etapa proporcionan a la topología cuatro rutas disjuntas dentro de la red, para cualquier par de nodos origen-destino. Esta

diversidad de rutas ofrece a la topología la posibilidad de equilibrar el tráfico entre los diferentes enlaces, y por lo tanto incrementar las prestaciones de la red. Además, a diferencia de RUFT-PL, esta topología también ofrece cierto grado de tolerancia a fallos de conmutadores, tal como veremos en el Capítulo 4.

FT-RUFT-222 es una topología derivada de RUFT-PL y de FT-RUFT-212 que combina las mejores propiedades de ambas propuestas. Al hacer uso de una inyección/eyección disjunta seguimos manteniendo cuatro rutas disjuntas dentro de la red, pero al utilizar dobles enlaces de red, el número de rutas se incrementa hasta ocho, permitiendo reducir aún más el *HoL Blocking* que pudiese existir en la fase de subida. Esta topología, al seguir el mismo modelo de interconexión que FT-RUFT-212, también soporta fallos de conmutadores.

RUFT-PL y FT-RUFT-222 son topologías que utilizan enlaces paralelos dentro de la red. En cada etapa, el conmutador con el que comunican los enlaces paralelos en la siguiente etapa siempre será el mismo. Por lo tanto, si el siguiente conmutador ha fallado, el paquete debe ser descartado, ya que no existe una ruta alterna para enviar el paquete.

Es por este motivo que nuestra última propuesta, FT-RUFT-XL, se centra en solucionar este inconveniente. Esta topología, al igual que RUFT-PL y FT-RUFT-222, también utiliza dobles enlaces de red, pero utilizando un nuevo modelo de interconexión entre los conmutadores que permite que los paquetes tomen diferentes rutas en cada conmutador de la red, incrementando considerablemente el nivel de tolerancia a fallos ofrecido por las otras propuestas, tanto a nivel de enlaces de red como de conmutadores, a la vez que mantiene unas altas prestaciones de red.

Por último, el uso de vectores de estado para implementar nuestro mecanismo de tolerancia a fallos nos permite conocer la alcanzabilidad de todos los nodos de procesamiento en cualquier punto de la red. Estos vectores, al trabajar a nivel de bit, nos permiten reducir el consumo de memoria requerido por las tablas de envío tradicionales (*forwarding tables*), y además soportar fallos de enlaces en la inyección/eyección, y fallos de conmutadores en cualquier etapa de la red.

Capítulo 4

Evaluación

"Millones de personas vieron una manzana caer, pero Newton fue el único que preguntó ¿por qué?"

Bernard M. Baruch

El objetivo de este capítulo es realizar una amplia evaluación de las topologías propuestas en esta tesis, para analizar su comportamiento bajo diferentes configuraciones y condiciones de carga. En particular evaluamos la tolerancia a fallos, las prestaciones y el coste de las topologías de red propuestas, comparándolas con la topología RUFT y el fat-tree con encaminamiento adaptativo (FTA).

La evaluación de prestaciones se divide en dos partes. Primero evaluamos el rendimiento de la red sin considerar ningún tipo de fallo, es decir, una red completamente funcional, y segundo, evaluamos el rendimiento a medida que se introduce un conjunto de fallos en los enlaces de red, para observar la degradación de prestaciones que sufre la red de interconexión. Para este análisis, se ha considerado un modelo de fallos estático, lo cual significa que, una vez el fallo es detectado, la actividad del sistema se detiene, se toman las acciones necesarias para manejar el fallo, y luego, la actividad del sistema se reanuda. Cabe mencionar que en la evaluación de degradación de prestaciones no se evaluará la topología RUFT, dado que ésta no soporta ningún tipo de fallo. Por lo tanto su evaluación no puede ser llevada a cabo.

Este capítulo se encuentra organizado de la siguiente manera. En la Sección 4.1 describimos el entorno de simulación utilizado para evaluar las topologías propuestas. En la Sección 4.2 realizamos un análisis para obtener el nivel de tolerancia a fallos que ofrece cada topología, tanto a nivel de enlaces de red como de conmutadores. En la Sección 4.3 se lleva a cabo la evaluación de prestaciones, donde se analizan en detalle cada una de las topologías propuestas considerando diversos parámetros que influyen en el rendimiento de la red. En la Sección 4.4 realizamos un análisis de degradación de prestaciones para ver como decae el rendimiento de la red a medida que se introducen fallos en los enlaces de red. A continuación, en la Sección 4.5, realizamos un análisis de coste/rendimiento sobre las topologías evaluadas, y por último, en la Sección 4.6, presentamos algunas conclusiones acerca de las evaluaciones llevadas a cabo a lo largo de este capítulo.

4.1. Entorno de simulación

Para analizar las nuevas topologías de red y obtener los resultados mostrados en este capítulo se han desarrollado dos herramientas de simulación, una para medir la tolerancia a fallos de la red y otra para el análisis de prestaciones.

Para la evaluación de tolerancia a fallos hemos desarrollado una herramienta que introduce fallos en la red, tanto a nivel de enlace como de conmutador. Para cada combinación de fallos dada, evalúa las rutas disponibles entre cada par origen-destino a fin de obtener el grado de tolerancia a fallos que ofrece la topología evaluada. Si la topología puede proporcionar al menos una ruta libre de fallos para cada origen y cada destino para esa combinación de fallos entonces la combinación de fallos es tolerada. Por el contrario, si existe al menos un par origen-destino que no tiene una ruta disponible para esa combinación de fallos, entonces dicha combinación de fallos se considera no tolerada.

En el análisis de tolerancia a fallos, el número de combinaciones a evaluar dentro de la topología puede llegar a ser demasiado alto, ya que este valor se incrementa conforme el número de fallos a evaluar aumenta y también con el tamaño de la red. Para evaluar un alto número de combinaciones de fallos y analizar todos los pares origen-destino y todas las rutas disponibles en la red se requiere un alto poder de cómputo. Por ello, la herramienta ha sido

diseñada para ejecutarse en un coprocesador dedicado al procesamiento de gráficos, también conocido como GPGPU o GPU. En aquellos entornos de trabajo donde se disponga de más de una GPU, la herramienta obtiene un mayor rendimiento, ya que ésta puede trabajar con varias GPUs a la vez, lo cual nos permite optimizar el tiempo de análisis.

Para una red y un número de fallos dado, hemos analizado todas las combinaciones de fallos, con un límite de hasta 1 millón de combinaciones de fallos en las redes más grandes, para limitar el tiempo de ejecución de la herramienta. En este último caso, hemos utilizado una distribución uniforme aleatoria para generar los fallos en la red.

Por otra parte, para analizar las topologías de red propuestas y comparar su rendimiento frente a FTA y RUFT, hemos implementado un detallado simulador basado en eventos, el cual modela diversas redes de interconexión indirectas con *Virtual Cut-Through*, como técnica de conmutación. Este simulador está basado en la versión anterior del simulador de redes de interconexión desarrollado en el Grupo de Arquitecturas Paralelas (GAP) [4], de la Universidad Politécnica de Valencia.

El simulador admite diversos parámetros de configuración. En particular, se puede seleccionar la topología, el encaminamiento, en su caso, así como el tamaño de la red a evaluar. Por otra parte, soporta multiplexación de canales virtuales.

Como hemos visto en el Capítulo 2, cada red está compuesta de una serie de nodos de procesamiento, conmutadores y enlaces. Cada canal de entrada y salida de un conmutador dispone de varios canales virtuales, cada uno con capacidad para almacenar 2 paquetes. Es bien conocido que el uso de canales virtuales permite mejorar notablemente las prestaciones de la red de interconexión. Por ese motivo, en nuestro análisis hemos realizado pruebas con distinto número de canales virtuales. Cabe decir que una red con 1 canal virtual equivale a una red que no implementa canales virtuales.

Por otra parte, el simulador permite modelar el retardo asociado a diferentes componentes del conmutador. Los valores de retardo utilizados en el conmutadores se especifican en la Tabla 4.1, los cuales se pueden interpretar de la siguiente manera:

Parámetros	Cantidad
Canales virtuales	1, 2, 3, 4
Tamaño cola canal entrada	2 Paquetes
Tamaño cola canal salida	2 Paquetes

Otros parámetros	Ciclos
Tiempo de link	1
Tiempo de fly	1
Tiempo de long fly	$Fly * stages + 1$
Tiempo de crossbar	1
Tiempo de routing	4

Tabla 4.1: Parámetros generales usados en la simulación.

- *Tiempo de Link*: es el número de ciclos que hay entre dos *phits*[1] transmitidos por el enlace. Este valor esta directamente relacionado con la inversa del ancho de banda.

- *Tiempo de Fly*: es el tiempo que le cuesta a un *phit* viajar por el enlace desde el canal de salida del conmutador origen hasta el canal de entrada del conmutador destino.

- *Tiempo de Long Fly*: representa el tiempo que le lleva a un *phit* en viajar por el enlace desde un canal de salida de un conmutador de la última etapa hasta el nodo de procesamiento. Este parámetro se ha establecido en un valor igual al tiempo de vuelo (*Fly*) multiplicado por el número de etapas de red más uno, con el fin de simular un enlace de mayor longitud. Este parámetro solo aplica a la topología RUFT y a las derivadas de ella, como todas las propuestas presentadas en este trabajo.

- *Tiempo de Routing*: es el tiempo que le lleva a un paquete en ser encaminado en cada conmutador. Este valor se ha establecido en cuatro ciclos, asumiendo un modelo de router segmentado en cuatro etapas.

- *Tiempo de Crossbar*: es el tiempo que tarda un *flit* en cruzar el crossbar, una vez se ha tomado la decisión de encaminamiento.

[1] Un *phit* (*physical digit*) es usualmente la cantidad de información que es enviada en un ciclo de reloj a través de un enlace de la red. Uno o más *phits* son combinados para formar un *flit*.

Por último, el simulador utiliza un control de flujo basado en créditos. El número de créditos equivale al número de paquetes que se pueden almacenar en cada canal de entrada/salida.

También es importante mencionar que el simulador permite generar tráfico sintético de diversos tipos para realizar las respectivas simulaciones. Los patrones de tráfico utilizados se explicarán en la Sección 4.3.1.

Para el análisis de degradación de prestaciones ante la presencia de fallos, cada topología ha sido evaluada inyectando una serie de fallos en los enlaces de red y evaluando sus prestaciones. Los valores de fallos analizados se encuentran en el rango [0 - 100]. Para cada red y número de fallos analizados, se han seleccionado 50 combinaciones de fallos de forma aleatoria, para finalmente calcular la media de prestaciones y obtener la degradación de prestaciones para el número de fallos dado. Aunque solo hemos considerado fallos de enlace, un fallo de conmutador puede ser considerado como el fallo de todos sus enlaces.

4.2. Evaluación de tolerancia a fallos

Se dice que una red es tolerante a fallos si en presencia de cierto tipo de fallos ésta puede continuar proporcionando una ruta para cualquier par de nodos origen-destino. Muchas redes de interconexión ofrecen una única conexión entre los nodos de procesamiento y la red. En estos casos, si el único conmutador o enlace de inyección/eyección falla, el nodo de procesamiento quedaría desconectado de la red, lo cual quiere decir que la red no es tolerante a fallos.

Para la evaluación de tolerancia a fallos hemos realizado diferentes análisis. Primero consideramos el máximo número de fallos de enlaces de red tolerados para cada red (Tabla 4.2). El segundo análisis (Figura 4.1) muestra el porcentaje de combinaciones de fallos de enlaces de red no tolerados para cada topología. El tercer análisis (Figura 4.2) presenta un punto de vista alternativo, mostrando el porcentaje de pares origen-destino que pueden comunicarse para cada topología a medida que se incrementa el número de fallos de enlaces en la red. Por último realizamos un análisis para mostrar el número de combinaciones de fallos de conmutadores no tolerados para cada topología.

La Tabla 4.2 presenta el máximo número de fallos de enlace tolerados por

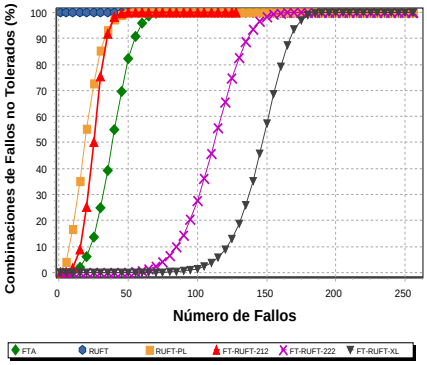
Aridad	FTA	RUFT	RUFT PL	FT-RUFT 212	FT-RUFT 222	FT-RUFT XL
2	1/0	0/0	1/1	3/1	7/1	7/1
4	3/0	0/0	1/1	3/1	7/1	7/1
8	7/0	0/0	1/1	3/1	7/1	7/1
16	15/0	0/0	1/1	3/1	7/1	7/1

Tabla 4.2: Número de fallos de enlaces de red tolerados (izquierda) y enlaces de inyección/eyección (derecha).

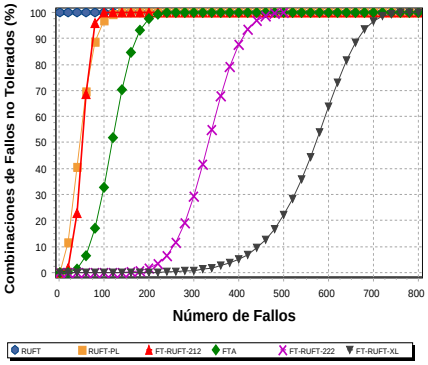
cada topología, variando la aridad de los conmutadores utilizados en la red. Para ese número de fallos, todas las combinaciones de fallos son toleradas, lo que significa que todos los pares origen-destino son capaces de comunicarse, para todas las combinaciones de ese número de fallos. En la tabla consideramos fallos tanto en los enlaces de red (valor de la izquierda) como en los enlaces de inyección/eyección (valor de la derecha). Como podemos notar, RUFT no soporta fallos en ningún punto de la red sin importar la aridad del conmutador. FTA ofrece encaminamiento adaptativo en la fase de subida, ofreciendo a cada paquete tantas opciones de encaminamiento como número de canales de subida. Por lo tanto, FTA puede tolerar tantos fallos en los enlaces de red como la aridad del conmutador menos uno $(k - 1)$ y ningún fallo en los enlaces de inyección/eyección. RUFT-PL, debido al patrón de conexión de sus enlaces, solo puede tolerar un fallo en los enlaces de red y un fallo en la inyección/eyección. FT-RUFT-212 puede tolerar tres fallos en los enlaces de red y un fallo en los enlaces de inyección/eyección. Por último, FT-RUFT-222 y FT-RUFT-XL soportan siete fallos en los enlaces de red y un fallo en la inyección/eyección. Además, FT-RUFT-212, FT-RUFT-222 y FT-RUFT-XL gracias a que proporcionan una inyección/eyección disjunta, pueden soportar fallos de conmutadores, como veremos en el último análisis de ésta sección.

En la Figura 4.1 presentamos el porcentaje de combinaciones de fallos no toleradas para diferentes tamaños de redes a medida que incrementamos el número de fallos en los enlaces de red. Las redes analizadas han sido seleccionadas para analizar el efecto de cambiar la aridad del conmutador y el número de etapas en la red.

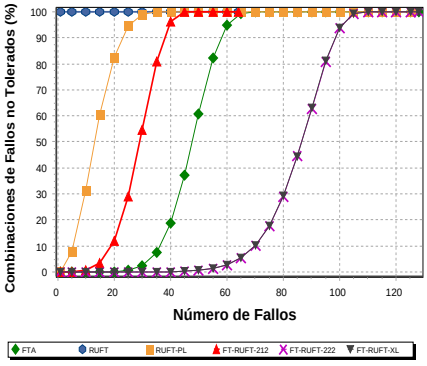
Como ya hemos mencionado, RUFT no proporciona tolerancia a fallos, ya que solo provee una ruta para cada par origen-destino, de modo que, si



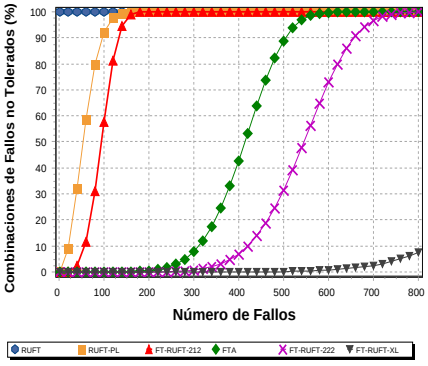
(a) 4-ary 3-tree



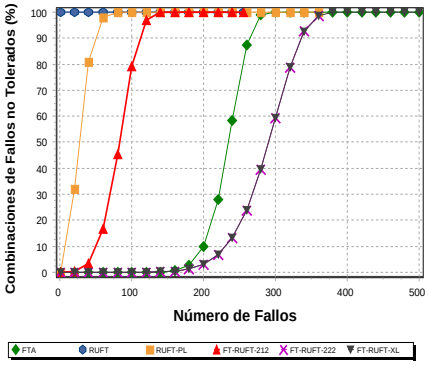
(b) 4-ary 4-tree



(c) 8-ary 2-tree



(d) 8-ary 3-tree



(e) 16-ary 2-tree

Figura 4.1: Porcentaje de combinación de fallos no tolerados para diferentes tamaños de redes.

se presenta algún fallo en la red éste no puede ser evitado, interrumpiendo la conectividad del sistema. Aunque RUFT-PL tiene el mismo número de enlaces de red que FTA[2], solo soporta un fallo, dejando el enlace paralelo disponible para comunicar los conmutadores. Sin embargo, dado que ambos enlaces paralelos se conectan al mismo conmutador, cuando ambos enlaces fallan, la red deja de ser tolerante a fallos. Por este motivo el porcentaje de fallos tolerados es menor que las demás topologías.

Con respecto a FT-RUFT-212, la topología es capaz de tolerar más fallos que RUFT-PL a pesar de que usa la mitad de enlaces de red que ésta última. Aunque ambas topologías tienen doble enlace en la conexión con los nodos, FT-RUFT-212 incrementa el número de fallos tolerados con solo implementar una inyección/eyección disjunta en la primera y última etapa de la red, permitiendo alcanzar los nodos destino a través de diferentes puntos de conexión.

Por otra parte, cuando la aridad del conmutador es igual a cuatro (Figura 4.1a y 4.1b), aunque FTA soporta el mismo número de fallos que FT-RUFT-212, FTA puede encaminar paquetes a través de más rutas en la red que FT-RUFT-212. Como consecuencia, tolera un alto porcentaje de combinaciones de fallos.

En cuanto a FT-RUFT-222, esta topología es capaz de tolerar un mayor número de fallos con respecto a las topologías anteriores. Gracias al uso de enlaces paralelos dentro de la red, y al esquema de doble inyección/eyección, la topología puede proporcionar hasta ocho rutas dentro de la red para cada par origen-destino, tolerando hasta siete fallos y también un alto número de combinaciones de fallos. Por ejemplo, en cada uno de los resultados de la Figura 4.1 podemos ver que la topología soporta un alto porcentaje de fallos de enlaces de red. En la Figura 4.1a apreciamos que FT-RUFT-222 puede soportar casi todas las combinaciones de fallos para 60 fallos simulados, a pesar de ser una red de tamaño pequeño. Sin embargo, cuando se incrementa el número de etapas de la red, Figura 4.1b, el número de combinaciones de fallos no tolerados disminuye considerablemente, tolerando casi todas las combinaciones de 180

[2]Recuerde que FTA usa enlaces bidireccionales mientras que las topologías derivadas de RUFT implementan enlaces unidireccionales. De modo que, FTA dobla el número de enlaces de red con respecto a RUFT y FT-RUFT-212, y tiene el mismo número de enlaces de red que RUFT-PL, FT-RUFT-222 y FT-RUFT-XL.

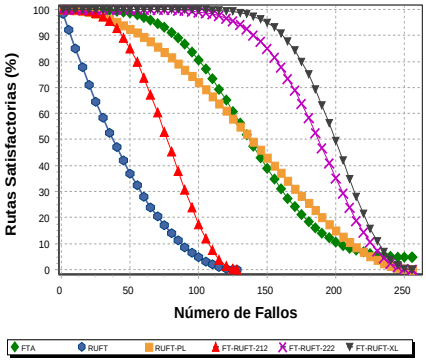
fallos. Este efecto también lo podemos observar a medida que aumenta la aridad del conmutador, como se aprecia en las Figuras 4.1c y 4.1e.

Con respecto a FT-RUFT-XL, la topología también implementa una doble inyección/eyección disjunta, y gracias al uso de sus enlaces cruzados con los conmutadores intercambiables, como hemos visto en la Sección 3.5.1, podemos incrementar considerablemente la tolerancia a fallos en cualquier red. A diferencia de FT-RUFT-212 y FT-RUFT-222, FT-RUFT-XL incrementa el número de rutas a medida que aumenta el número de etapas de la red.

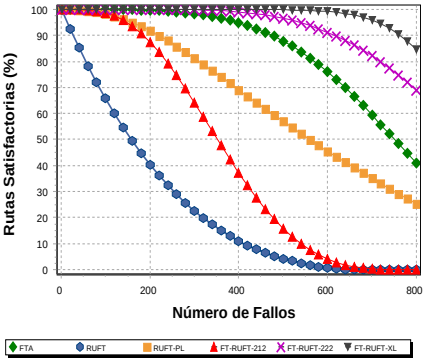
En redes de dos etapas, Figura 4.1c y 4.1e, la tolerancia a fallos en FT-RUFT-XL es igual que la ofrecida por FT-RUFT-222, debido a que el bajo número de etapas de la red no permite crear demasiadas conexiones para cada par origen-destino. En redes de más etapas, como la mostrada en la Figura 4.1a, podemos notar que casi todas las combinaciones de 100 fallos son toleradas. Además, si pasamos de una red 4-ary 3-tree (Figura 4.1a) a una 4-ary 4-tree (Figura 4.1b) el número de combinaciones de fallos tolerados incrementa a tal punto que se sitúa cerca de los 300 fallos, con un valor cercano al 1 % para ese número de fallos.

Otro claro ejemplo del incremento del nivel de tolerancia a fallos al aumentar la aridad del conmutador y el número de etapas de la red lo podemos ver en la Figura 4.1d, donde FT-RUFT-XL soporta casi todas las combinaciones de 600 fallos. Por último, FTA también logra obtener un buen nivel de tolerancia a fallos, gracias a su algoritmo de encaminamiento adaptativo que le permite usar cualquier puerto de salida disponible en la fase ascendente. Sin embargo, la diferencia con las otras topologías es que cuando consideramos los enlaces de inyección/eyección, ni FTA, ni RUFT soportan fallos en estos puntos de conexión, mientras que RUFT-PL y las variantes de FT-RUFT pueden soportar un fallo en estos enlaces.

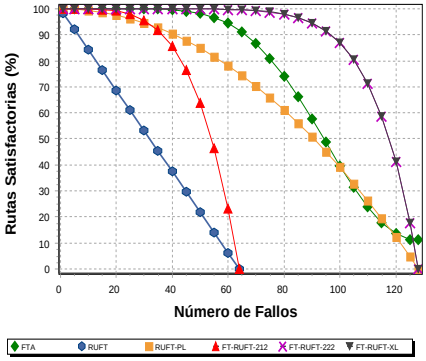
Para ofrecer un punto de vista alternativo a este análisis, hemos decidido mostrar como decae la comunicación entre los nodos de procesamiento a medida que se introducen fallos en la red. En la Figura 4.2 mostramos el número de pares origen-destino que pueden comunicarse en cada topología para diferentes número de fallos. En vez de mostrar las combinaciones que no son toleradas, mostramos el porcentaje de pares de nodos que pueden comunicarse para las combinaciones de fallos analizadas para ese número de fallos.



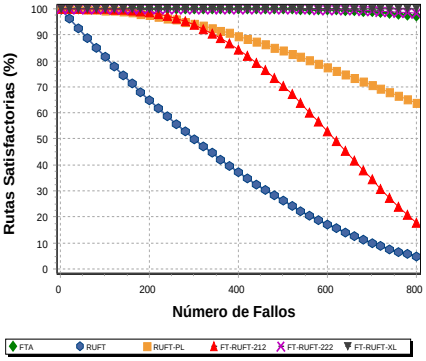
(a) 4-ary 3-tree



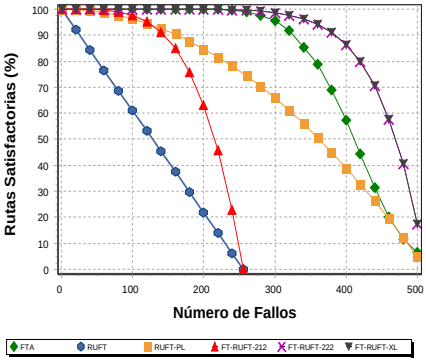
(b) 4-ary 4-tree



(c) 8-ary 2-tree



(d) 8-ary 3-tree



(e) 16-ary 2-tree

Figura 4.2: Porcentaje de rutas origen-destino que pueden comunicarse bajo diferentes números de fallos y redes de diferentes tamaños.

Como se puede apreciar, FT-RUFT-212 obtiene mejores resultados que RUFT, incrementando considerablemente el porcentaje de rutas ofrecidas, agregando solo unos pocos enlaces en la primera y última etapa de la red. Por otro lado, RUFT-PL puede comunicar un alto porcentaje de pares de nodos origen-destino gracias a los enlaces paralelos de la red; teniendo en cuenta que, mientras ambos enlaces no fallen, siempre existirá comunicación entre los conmutadores, proporcionando una ruta para cada par de nodos. En el caso de FTA, el porcentaje de rutas disponibles entre todos los pares de nodos incrementa, ya sea con el número de etapas y/o la aridad del conmutador, gracias a su encaminamiento adaptativo, tal como hemos explicado previamente. Aun así, FT-RUFT-222 y FT-RUFT-XL ofrecen mejores resultados que FTA, a pesar de utilizar los mismos recursos de red.

En particular, estas dos topologías pueden hacer frente a un altísimo número de fallos, mientras mantienen un alto porcentaje de comunicación entre todos los nodos de procesamiento. Por ejemplo, en la Figura 4.2d tenemos el FTA con un 97,36 % de comunicación entre los nodos, mientras que FT-RUFT-222 alcanza un 99,31 % y FT-RUFT-XL un 99,34 %, para una red con 800 fallos. Es decir, que aún con este nivel de fallos estamos garantizando cerca del 100 % de comunicación entre todos los nodos de procesamiento. Nuevamente hacemos hincapié en que FT-RUFT-XL incrementa su nivel de tolerancia a fallos a medida que incrementa el número de etapas de la red.

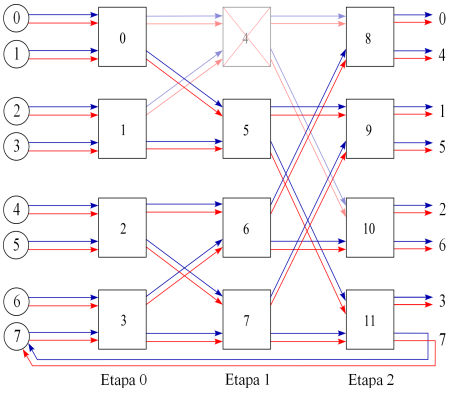
En el análisis realizado previamente hemos visto cómo afectan los fallos de los enlaces a la comunicación de la red. Sin embargo, no hemos tenido en cuenta el fallo de conmutadores, ya que el fallo de un conmutador se puede considerar como el fallo de todos sus enlaces. El fallo de un conmutador afecta de forma directa a todos sus enlaces, perjudicando en mayor medida la comunicación de la red. Por este motivo, en nuestro último análisis, evaluaremos el porcentaje de combinaciones de fallos de conmutadores que no son tolerados, a medida que se introducen fallos de conmutadores en la red.

Como punto de partida de este análisis, hemos representado algunos fallos de conmutadores en una red 2-ary 3-tree (Figura 4.3), para las diferentes topologías presentadas en este trabajo. Los enlaces afectados por el fallo de estos componentes también se pueden visualizar en la imagen. Los conmutadores y enlaces que han fallado son aquellos que se muestran en un color más claro.

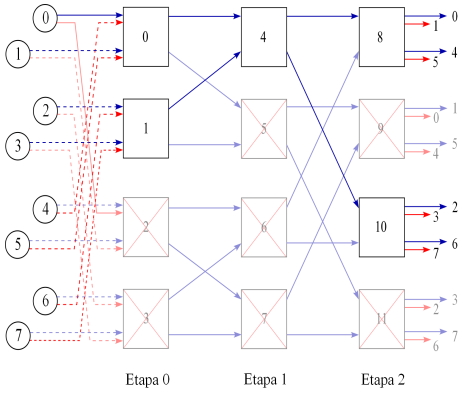
En la Figura 4.3a tenemos la topología RUFT-PL. En este caso hemos supuesto que el conmutador 4 de la red ha fallado, y por lo tanto, los enlaces que conectan con él también dejan de estar disponibles. Aunque RUFT-PL soporta un fallo de enlace en cualquier punto de la red, no soporta ningún fallo de conmutador dado que ambos enlaces paralelos (en la inyección, red y eyección) conectan con el mismo conmutador o nodo de procesamiento. En este caso, al fallar cualquiera de los conmutadores de la red perdemos conexión entre los nodos de procesamiento, en menor o mayor medida, dependiendo de la etapa donde se encuentre el conmutador.

Por otra parte, como podemos ver en las Figuras 4.3b y 4.3c, FT-RUFT-212 y FT-RUFT-222 soportan el mismo número de fallos de conmutadores, ya que ambas topologías presentan el mismo patrón de interconexión entre todos los elementos de la red, con la diferencia que FT-RUFT-222 usa enlaces paralelos/dobles para interconectar los conmutadores entre sí. Gracias a la inyección/eyección disjunta que implementan estas dos topologías, se incrementa el número de rutas para cada par origen-destino, y por lo tanto, también el número de conmutadores que pueden fallar.

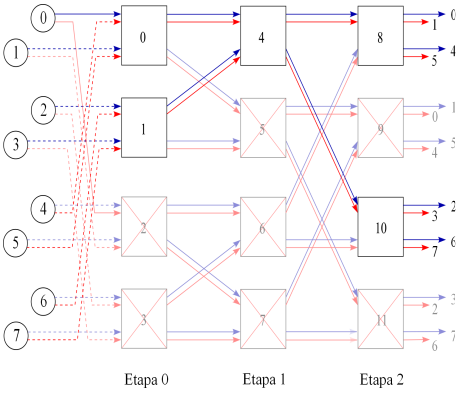
La conexión de los nodos de procesamiento con diferentes conmutadores de la primera etapa, en diferentes subárboles de la red, permite que más de la mitad de dichos conmutadores falle, sin perder conexión entre los nodos de la red, teniendo en cuenta que dos conmutadores no conecten con el mismo nodo de procesamiento. En la Figura 4.3b y 4.3c hemos supuesto que los conmutadores 2 y 3 de la primera etapa han fallado. Sin embargo, los nodos 4 a 7 siguen conectados a la red a través de los conmutadores 0 y 1, utilizando sus enlaces de inyección secundarios. En la última etapa de la red tenemos un comportamiento similar a la primera etapa, es decir, que la mitad de los conmutadores también pueden fallar gracias a la eyección disjunta utilizada por la topología, de modo que, siempre y cuando dos conmutadores no conecten con los mismos destinos existirá comunicación con todos los nodos de procesamiento. De esta manera, cuando un conmutador falla, el conmutador vecino cubrirá la conexión del conmutador que ha caído. Refiriéndonos de nuevo a la figura de este ejemplo, observamos que los conmutadores 9 y 11 han fallado, pero los conmutadores 8 y 10 siguen ofreciendo conexión a los nodos destino que no se encuentran disponibles desde los conmutadores 9 y 11.



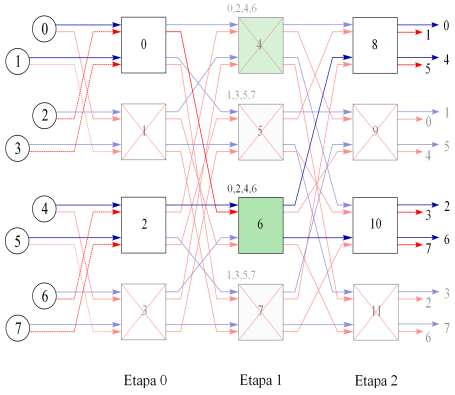
(a) RUFT-PL



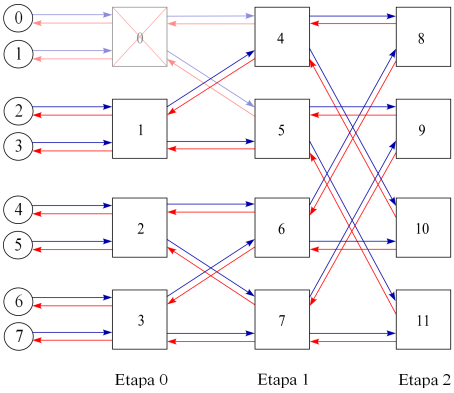
(b) FT-RUFT-212



(c) FT-RUFT-222



(d) FT-RUFT-XL



(e) FTA

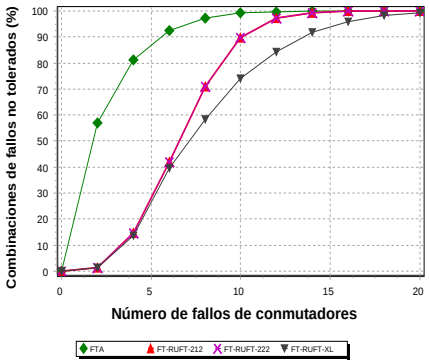
Figura 4.3: Fallos de conmutadores en una red 2-ary 3-tree bajo diversas topologías de red.

En cuanto a la topología FT-RUFT-XL, ésta presenta similitudes con FT-RUFT-212 y FT-RUFT-222 dado que también implementa una doble inyección/eyección; por lo tanto, el número de conmutadores que pueden fallar en la primera y última etapa es el mismo que en las topologías explicadas previamente, aunque en la primera etapa la distribución de los conmutadores que pueden fallar es diferente, dado que la conexión de los nodos de procesamiento con los conmutadores de la primera etapa no se realiza de la misma manera, sino implementando la variación presentada en la Sección 3.5. Como hemos explicado anteriormente, a medida que incrementa el número de etapas de la red también lo hace el nivel de conectividad que existe entre los componentes de esta topología, gracias a los conmutadores intercambiables. De esta manera, el número de rutas entre cada par de nodos y el número de fallos soportados por FT-RUFT-XL se incrementa de forma considerable con cada etapa que se agregue a la red.

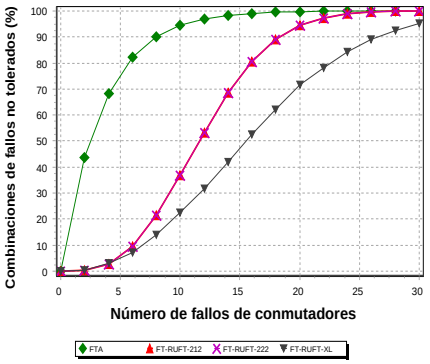
Con respecto a FTA, aunque la topología tiene la posibilidad de enviar el tráfico a través de los diferentes puertos de salida, ésta no soporta fallos en los conmutadores de la primera etapa, tal como podemos apreciar en la Figura 4.3e, debido a que los nodos de procesamiento solo disponen de una única conexión con la red, por lo tanto cualquier fallo que se produzca en este punto no será soportado por la topología.

Aunque el FTA y las topologías derivadas de FT-RUFT soportan el fallo de algunos conmutadores, esto no se cumple para todas las combinaciones de estos fallos. Al igual que hemos hecho para el análisis de combinaciones de fallos de enlace tolerados por cada red, en la Figura 4.4 mostramos el porcentaje de combinaciones de fallos de conmutadores no tolerados. En este análisis la topología RUFT y RUFT-PL no se evalúan, ya que éstas no soportan este tipo de fallos, y por lo tanto el porcentaje de fallos no tolerados será siempre del 100 %.

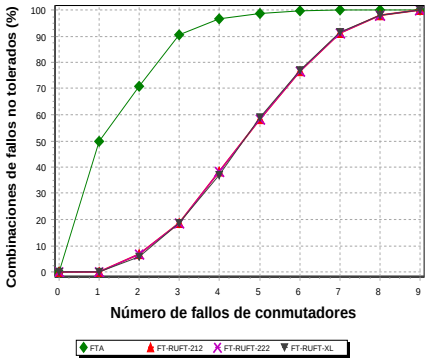
Como podemos ver en la Figura 4.4, tanto en FT-RUFT-212 como FT-RUFT-222 el porcentaje de combinaciones de fallos de conmutadores no tolerados es el mismo, debido a que el patrón de interconexión entre los elementos de la red es igual para ambas topologías. Aun así, en estas dos topologías, el número de fallos de conmutadores soportados es mucho mayor que el FTA, gracias a la inyección/eyección disjunta.



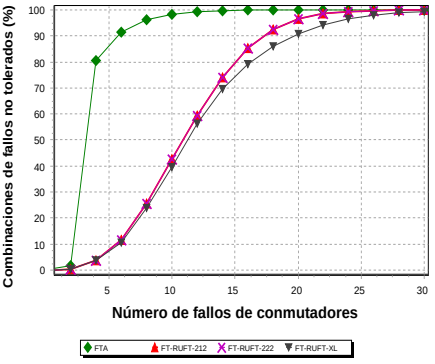
(a) 4-ary 3-tree



(b) 4-ary 4-tree



(c) 8-ary 2-tree



(d) 8-ary 3-tree

Figura 4.4: Porcentaje de combinación de fallos de conmutadores no tolerados para diferentes tamaños de redes.

Por otra parte, FT-RUFT-XL siempre ofrece mejores resultados que las demás topologías. En redes de dos etapas (Figura 4.4d), ésta soporta el mismo número de fallos de conmutadores que FT-RUFT-212 y FT-RUFT-222, pero conforme incrementa el número de conmutadores fallidos, el porcentaje de combinaciones de fallos no tolerados es menor, gracias al mayor nivel de interconexión que existe entre los componentes de la red. Como podemos apreciar en la Figura 4.4b, a medida que se incrementa el número de etapas, FT-RUFT-XL incrementa su nivel de tolerancia a fallos, permitiendo que la topología sea menos susceptible a los fallos. Por último, la topología FTA es la más afectada por el fallo de conmutadores. A pesar de que ésta utiliza encaminamiento adaptativo y cuenta con un alto grado de conectividad, cualquier combinación de fallos de conmutadores que involucre un conmutador de la primera etapa hará que dicha combinación no sea tolerada por la topología, independientemente de la aridad del conmutador que utilice la red. Por este motivo, el FTA es la que ofrece los peores resultados en esta evaluación, a pesar de la cantidad de recursos de hardware que utiliza la red.

4.3. Evaluación de prestaciones

La productividad y la latencia de una red de interconexión no solo van asociados a la topología implementada sino también a otros factores de red como su tamaño, canales virtuales o la carga de la red, como puede ser el patrón de tráfico o el tamaño del paquete. Todos estos factores afectan de una u otra forma las prestaciones que puede alcanzar el sistema. Por ello, en esta sección realizaremos un amplio análisis de prestaciones de las topologías propuestas (RUFT-PL, FT-RUFT-212, FT-RUFT-222 y FT-RUFT-XL), RUFT y de FTA, para analizar su comportamiento bajo diferentes condiciones de carga.

4.3.1. Patrones de tráfico

Los patrones de tráfico representan la distribución espacial y temporal de los mensajes que circulan por la red. Para evaluar el comportamiento de estos patrones de tráfico sobre las redes de interconexión se suelen utilizar trazas de entrada/salida asociadas a *clusters* reales o generadores de tráfico sintético.

Las trazas recogidas de los *clusters* suelen representar un comportamiento real del sistema, sin embargo éstas varían dependiendo de la aplicación que las genere, lo cual solo permite analizar cómo se comporta una red bajo unas condiciones específicas.

Por otra parte, los patrones de tráfico sintético son ampliamente usados en la evaluación de redes de interconexión, dado que permiten evaluar el comportamiento de la red de forma general.

Teniendo en cuenta que todas las redes de interconexión no responden igual ante el mismo tipo de distribución de mensajes, hemos considerado evaluar las topologías propuestas usando diversos patrones de tráfico sintético para analizar detalladamente su comportamiento bajo diferentes escenarios. Los patrones de tráfico que utilizaremos a lo largo de la presente evaluación son: *Uniform*, *Hot-Spot*, *Bit-Reversal*, *Complement* y *Shuffle*. A continuación describiremos cada uno de ellos:

- *Tráfico Uniform*: es el patrón de tráfico más ampliamente usado en la evaluación de redes de interconexión. En esta distribución, la probabilidad de un nodo i que envía un mensaje a un nodo j es la misma para todo i y j, con $i \neq j$ [55].

- *Tráfico Hot-Spot*: evalúa aquellos casos donde una cantidad considerable de tráfico es enviada a un destino específico. En nuestro caso, evaluamos un *Hot-Spot* al 5 % y 15 %. Dicho porcentaje de nodos inyecta tráfico dirigido al mismo destino; el tráfico restante que circula por la red es de tipo *Uniform*.

- *Tráfico Bit-Reversal*: en éste patrón de tráfico cada nodo solo envía mensajes al nodo destino cuyos bits se correspondan con los bits del nodo origen invirtiendo el orden de éstos. Por ejemplo, en una red 2-ary 3-tree el nodo 4 $\langle 100 \rangle$ solo enviará mensajes al nodo 1 $\langle 001 \rangle$.

- *Tráfico Complement*: es un patrón de tráfico donde cada nodo solo envía tráfico al destino que es obtenido invirtiendo los bits del nodo origen. Por ejemplo, en una red 2-ary 3-tree, el nodo 3 $\langle 011 \rangle$ solo enviará mensajes al nodo 4 $\langle 100 \rangle$. De esta manera, en las topologías BMIN, los paquetes son forzados a alcanzar siempre los conmutadores de la última etapa.

- *Tráfico Shuffle*: se caracteriza por enviar mensajes a un nodo destino cuyos bits corresponden a los bits del nodo origen pero rotados a la izquierda 1 bit. Por ejemplo, en una red 2-ary 3-tree, el nodo 3 $\langle 011 \rangle$ solo enviará mensajes al nodo 6 $\langle 110 \rangle$.

4.3.2. Prestaciones de red

En esta sección evaluaremos las prestaciones de RUFT-PL, FT-RUFT-212, FT-RUFT-222 y FT-RUFT-XL y las compararemos frente a RUFT y FTA.

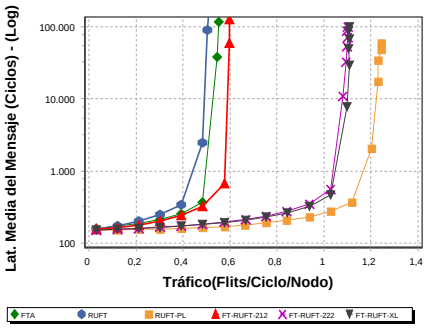
Para poder obtener unas conclusiones generales acerca del comportamiento de la red, realizaremos una amplia evaluación de las topologías propuestas en este trabajo utilizando diferentes parámetros de red, como por ejemplo, variando la aridad de los conmutadores, el número de etapas que compone la red, y más importante aún, el patrón de tráfico utilizado.

Patrón de tráfico *Uniform*

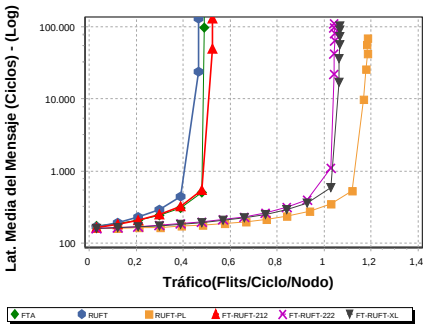
La Figura 4.5 muestra los resultados de prestaciones de diferentes topologías bajo el patrón de tráfico *Uniform* y diversos tamaños de red.

Como podemos ver, en cada uno de los casos, RUFT y FTA obtienen un rendimiento de red muy similar a pesar de que FTA utiliza más del doble de hardware que RUFT. En FTA, a medida que incrementa la aridad de los conmutadores, también incrementa el *HoL Blocking*, dado que cada puerto de salida puede ser demandado por más puertos de entrada. Por otra parte, la ventaja de RUFT es que su encaminamiento determinista permite reducir el *HoL Blocking*, lo cual permite incrementar sus prestaciones. FT-RUFT-212 es capaz de obtener mejores prestaciones que RUFT y FTA, con solo agregar unos pocos enlaces en la inyección/eyección. Además, como ya hemos expuesto, también se dota a la red de tolerancia a fallos.

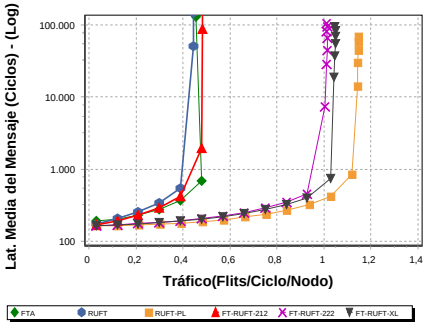
En cuanto a FT-RUFT-222 y FT-RUFT-XL, bajo este patrón de tráfico, la diferencia en la productividad alcanzada por ambos está sujeta al número de etapas que componen la red más que a la aridad del conmutador, debido a que en ambas topologías la utilización de los enlaces es muy similar en redes de pocas etapas. Esto podemos comprobarlo en las Figuras 4.5a, 4.5d y 4.5f, donde el rendimiento de la red es similar para las dos topologías, mientras que



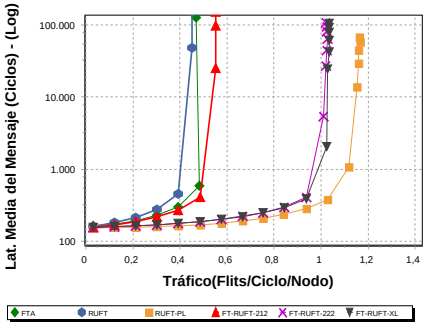
(a) 4-ary 3-tree



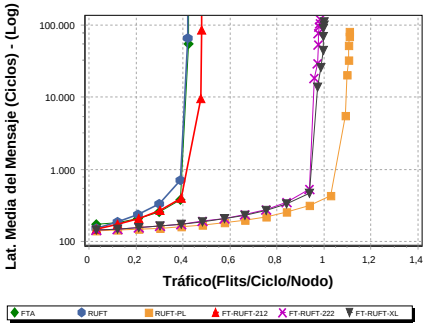
(b) 4-ary 4-tree



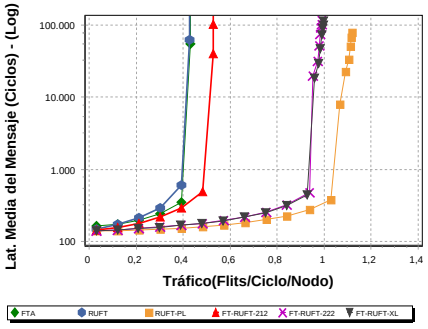
(c) 4-ary 5-tree



(d) 8-ary 3-tree



(e) 8-ary 4-tree



(f) 16-ary 3-tree

Figura 4.5: Latencia desde la generación del paquete versus tráfico aceptado para diferentes tamaños de red, tráfico *Uniform* y un tamaño de paquete de 128B.

en la Figura 4.5c, al aumentar el número de etapas, empezamos a notar un incremento en el rendimiento porque FT-RUTF-XL reduce la congestión al incrementar el número de rutas y de conmutadores intercambiables.

Con respecto a RUFT-PL, las prestaciones obtenidas son superiores a todas las otras topologías, con una ligera diferencia entre FT-RUFT-222 y FT-RUFT-XL, alcanzando más del doble de prestaciones que RUFT, FTA y FT-RUFT-212. La gran ventaja de RUFT-PL, es que ambos enlaces paralelos siempre conectan con el mismo conmutador/nodo, y por lo tanto, el tráfico se mantiene bastante ordenado a medida que fluye por la red, lo que reduce en gran medida el *HoL Blocking*. A nivel general, podemos notar que todas las topologías reducen su productividad a medida que se incrementa el número de etapas de la red, debido a que el número de nodos aumenta, y por ende, también el tráfico y la posibilidad de contención.

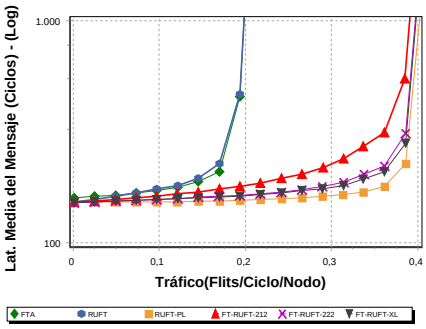
Patrón de tráfico *Hot-Spot*

En el patrón de tráfico *Hot-Spot* un porcentaje del total del tráfico de la red se envía a un único destino (hot-spot), mientras que el resto de tráfico es de tipo uniforme.

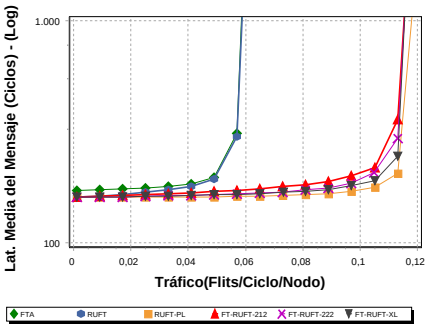
Este patrón de tráfico afecta negativamente a todos los algoritmos de encaminamiento, de manera proporcional al porcentaje de tráfico que se destine al nodo hot-spot, por este motivo, en el presente análisis hemos evaluado *Hot-Spot* al 5 % y al 15 % para ver cómo se comporta la red ante diferentes cargas de tráfico.

En la Figura 4.6 mostramos los resultados para el patrón de tráfico *Hot-Spot* al 5 % para varios tamaños de red. Como se puede apreciar, en todos los casos, las topologías RUFT-PL, FT-RUFT-212, FT-RUFT-222 y FT-RUFT-XL son capaces de doblar el rendimiento de RUFT y FTA, ya que, gracias a la doble inyección/eyección podemos distribuir la concentración de tráfico a través de las diferentes rutas que proveen las topologías. Además, gracias a la doble eyección podemos reducir el cuello de botella generado en la entrega de paquetes, permitiendo de esta forma a cada nodo recibir el doble de *flits*.

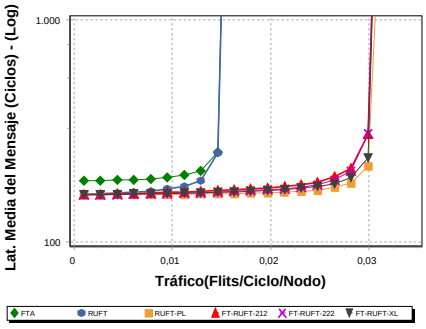
Por otra parte, al incrementar el porcentaje de tráfico hot-spot al 15 % (Figura 4.7), el rendimiento de la red baja considerablemente. Sin embargo, el comportamiento de todas las topologías, en todos los casos, sigue siendo



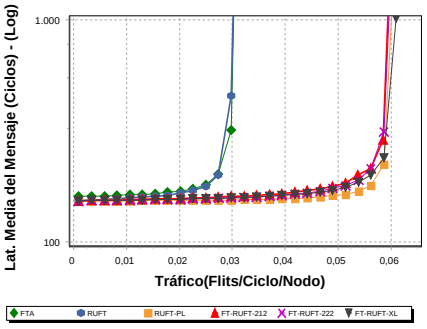
(a) 4-ary 3-tree



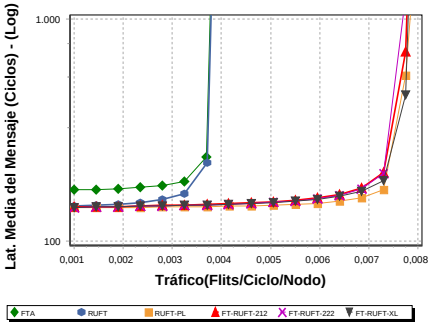
(b) 4-ary 4-tree



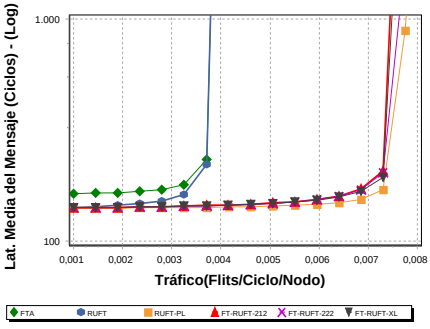
(c) 4-ary 5-tree



(d) 8-ary 3-tree

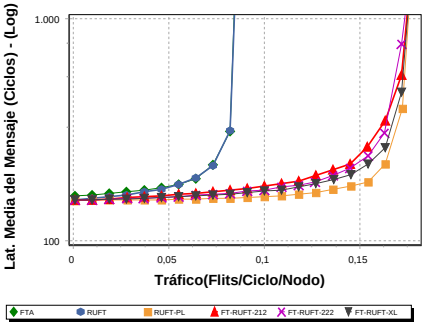


(e) 8-ary 4-tree

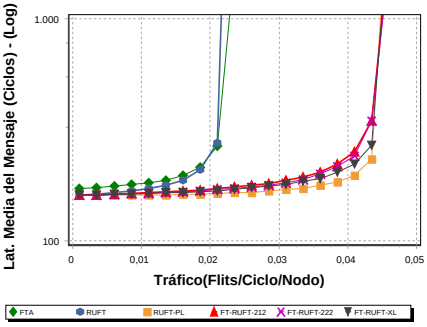


(f) 16-ary 3-tree

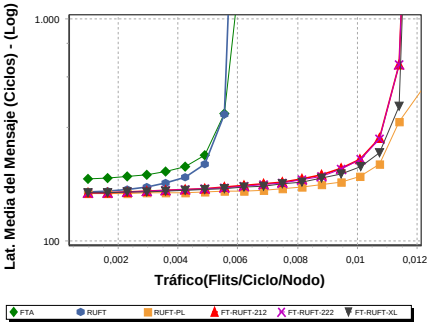
Figura 4.6: Latencia desde la generación del paquete versus tráfico aceptado para diferentes tamaños de red, tráfico *Hot-Spot* (5 %) y un tamaño de paquete de 128B.



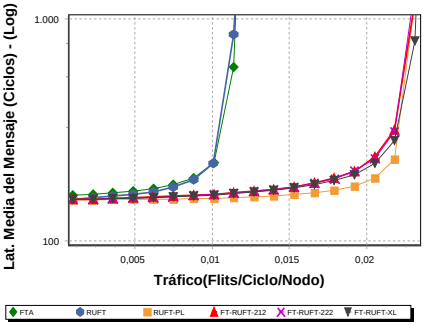
(a) 4-ary 3-tree



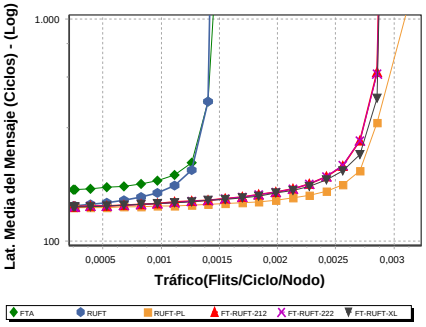
(b) 4-ary 4-tree



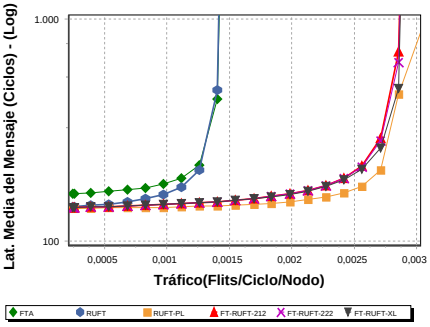
(c) 4-ary 5-tree



(d) 8-ary 3-tree



(e) 8-ary 4-tree



(f) 16-ary 3-tree

Figura 4.7: Latencia desde la generación del paquete versus tráfico aceptado para diferentes tamaños de red, tráfico *Hot-Spot* (15 %) y un tamaño de paquete de 128B.

el mismo, es decir, todas las topologías propuestas alcanzarán el doble de rendimiento de RUFT y FTA, debido a que éstas solo disponen de un único enlace de inyección/eyección por nodo.

Respecto a la latencia, RUFT y las topologías derivadas de ésta ofrecen un bajo tiempo de entrega de la información, como consecuencia de utilizar topologías unidireccionales que eliminan la fase de bajada usada por FTA, y por lo tanto el número de saltos requerido por cada paquete, disminuyendo a su vez el tiempo de espera de los paquetes en los buffers. Sin embargo, tanto en *Hot-Spot* al 5 % como al 15 %, podemos ver que FTA incrementa su latencia de generación con respecto a las otras topologías a medida que aumenta el número de etapas de la red, dado que la red estará más saturada, y los paquetes pasarán más tiempo en los buffers de los puertos antes de alcanzar su destino.

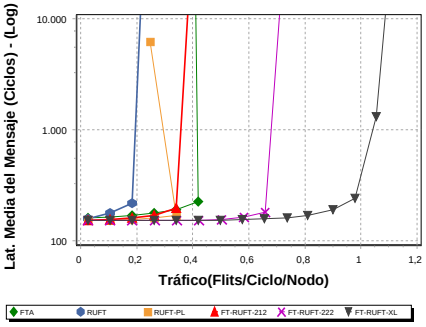
Patrón de tráfico *Bit-Reversal*

Como podemos observar en los resultados obtenidos bajo este patrón de tráfico (Figura 4.8a), RUFT es la topología que obtiene las prestaciones de red más bajas, frente a las otras topologías, ya que solo existe una única ruta para cada par origen-destino.

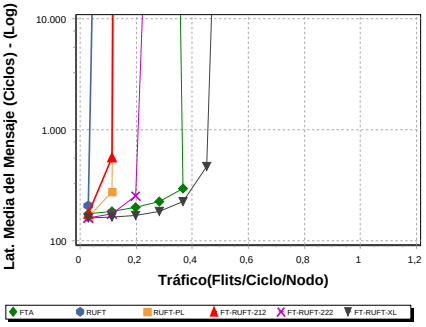
Por otra parte, a pesar de que RUFT-PL ofrecía muy buenos resultados bajo otros patrones de tráfico, en este caso, obtiene bajas prestaciones en comparación con otras topologías. Aun así, alcanza aproximadamente el doble de las prestaciones de RUFT, debido a que la distribución de este tráfico en ambas topologías es la misma.

Respecto a FT-RUFT-212, las prestaciones obtenidas por la topología son superiores a RUFT a pesar de que utilizan los mismos enlaces de red, porque gracias a la doble inyección/eyección y las cuatro rutas que existen para cada par origen-destino podemos distribuir el tráfico en la red, y obtener un rendimiento similar a RUFT-PL.

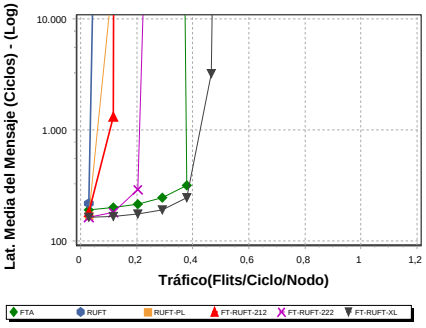
Aunque en otros casos la productividad de FTA ha sido semejante a RUFT, en este caso el rendimiento obtenido es mucho más alto que RUFT, RUFT-PL y FT-RUFT-212, ya que su encaminamiento adaptativo en la fase de subida, permite aprovechar todos los puertos de salida disponibles, haciendo mejor uso de los recursos. En cuanto a FT-RUFT-222, podemos ver que el rendimiento de la red es relativamente alto, en comparación con las topologías anteriores, ya



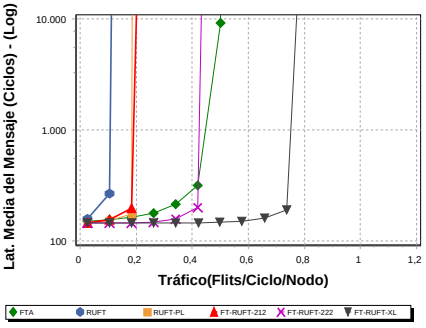
(a) 4-ary 3-tree



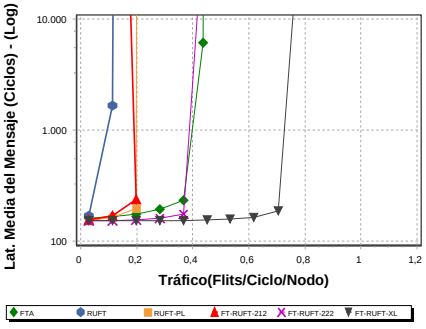
(b) 4-ary 4-tree



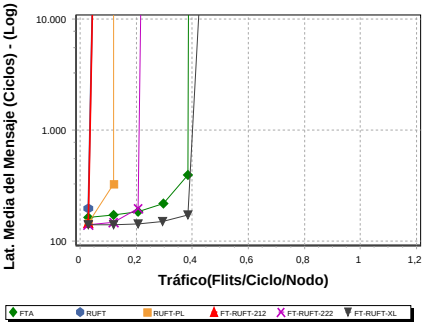
(c) 4-ary 5-tree



(d) 8-ary 2-tree



(e) 8-ary 3-tree



(f) 16-ary 3-tree

Figura 4.8: Latencia desde la generación del paquete versus tráfico aceptado para diferentes tamaños de red, tráfico *Bit-Reversal* y un tamaño de paquete de 128B.

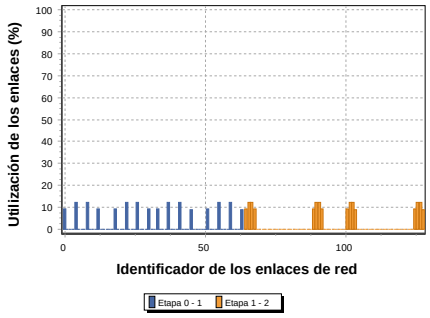
que existen cuatro rutas dobles para cada par origen-destino, incrementando de esta forma el número de enlaces disponibles. Por otra parte, observamos que FT-RUFT-XL supera a todas las otras topologías, gracias a los conmutadores intercambiables que permiten crear más rutas entre cada par de nodos de procesamiento. Además, como se ha mencionado anteriormente, esta topología también ofrece dos rutas en la penúltima etapa para cada destino, lo cual permite aligerar la carga en la entrega de los paquetes.

Analizando una red 4-ary 4-tree (Figura 4.8b) podemos ver que las prestaciones de todas las topologías derivadas de RUFT ha bajado considerablemente. Para entender este comportamiento, vamos a realizar un análisis de utilización de los enlaces sobre RUFT, dado que es la topología base de nuestras propuestas, lo que nos permitirá comprender fácilmente esta situación.

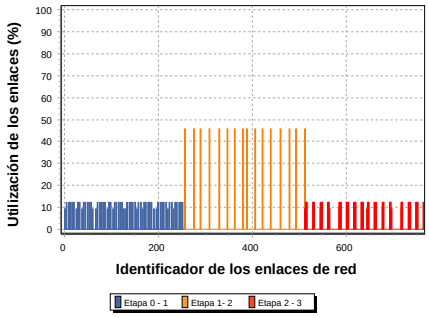
Como sabemos, RUFT es una topología unidireccional que utiliza un encaminamiento determinista, es decir, siempre usa la misma ruta para cada par origen-destino. Además, bajo este patrón de tráfico, cada nodo origen solo envía mensajes a un único destino, teniendo en cuenta que existen nodos que no enviarán mensajes.

Otra particularidad de este patrón de tráfico sobre las topologías RUFT, es que todos los nodos origen que se encuentran agrupados en un mismo conmutador en la primera etapa enviarán mensajes a los nodos destinos agrupados en un mismo conmutador en la última etapa. Esta distribución de tráfico hace que la utilización de los enlaces sea diferente en redes de diferentes etapas, ya que, para alcanzar un destino dado, los enlaces de las etapas intermedias deben ser compartidos por otros mensajes, en mayor o menor medida, creando bloqueos en aquellos puntos donde se sobreutilizan los enlaces, y dejando otros sin utilizar, de ahí que las prestaciones varíen tanto de una red a otra en función del número de etapas.

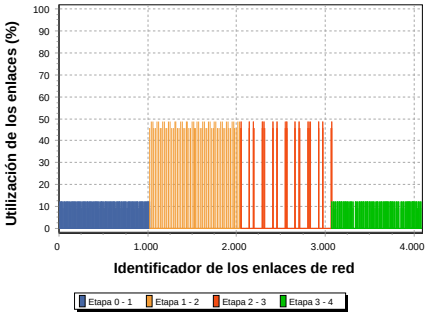
Por ejemplo, en la Figura 4.9a tenemos la utilización de los enlaces de la topología RUFT en una red 4-ary 3-tree, para el primer punto de simulación, es decir, en baja carga. Esta topología consta de 128 enlaces de red, 64 entre las etapas 0-1 (barras de color azul) y 64 entre las etapas 1-2 (barras en color amarillo), los cuales se encuentran identificados en el eje X. De los 128 enlaces de red, solo se están utilizando 32 enlaces (25 %), 16 entre las etapas 0-1 y 16 entre las etapas 1-2, mientras que el 75 % de los enlaces no se utilizan



(a) 4-ary 3-tree



(b) 4-ary 4-tree



(c) 4-ary 5-tree

Figura 4.9: Utilización de los enlaces de red en la topología RUFT, usando redes de igual aridad, diferente número de etapas y tráfico *Bit-Reversal*.

en ningún momento. Como se puede notar, el porcentaje de utilización es aproximadamente el 10 % para cada uno de los enlaces usados.

En la Figura 4.9b tenemos la utilización de los enlaces de RUFT, en una red 4-ary 4-tree, en baja carga. En este caso la topología contiene 768 enlaces de red, distribuidos en tres bloques intermedios, etapas 0-1 (barras azules), 1-2 (barras amarillas) y 2-3 (barras rojas), cada una con 256 enlaces. Para esta red, el número de enlaces usados es mucho menor, ya que a medida que los mensajes avanzan por la red, éstos son compartidos a través de las diferentes etapas. En este caso tenemos que, entre las etapas 1-2 solo se usan 64 enlaces, con un 12 % de utilización, aproximadamente. Entre las etapas 2-3 solo se usan 16 enlaces, pero en este caso la utilización es aproximadamente del 45 %, lo cual nos indica que la mayoría de pares origen-destino comparten estos enlaces en sus rutas,

y por este motivo se incrementa la carga y el *HoL Blocking* en este punto, afectando considerablemente a la productividad de la red. A continuación, el tráfico que llega a la penúltima etapa, nuevamente se distribuye, usando de nuevo 64 enlaces entre las etapas 3-4, restableciendo la carga de los enlaces de nuevo al 12 %, aproximadamente. Al aumentar el número de etapas de la red a 4, como en este caso, el porcentaje de enlaces de red usados es solo del 18,75 %, es decir, un 6,25 % menos con respecto a la red anterior.

Cuando analizamos una red más grande, como una 4-ary 5-tree (Figura 4.9c) podemos notar que los enlaces con más carga de tráfico se centran en aquellos que están presentes en las etapas intermedias, con una gran diferencia en el bajo número de enlaces usados entre las etapas 2-3 (barras rojas). Aunque esta topología tiene 4096 enlaces de red, solo se está usando el 15,62 % de los enlaces, un 9,37 % menos que en la 4-ary 3-tree y un 3,12 % menos que la 4-ary 4-tree, dejando un 84,37 % de enlaces sin usar.

Como hemos podido notar, a través de la explicación anterior, el comportamiento y distribución de tráfico sobre los enlaces varía con cada red, siendo este el motivo por el cual las prestaciones son tan diferentes para cada configuración de la red. En cuanto a las otras topologías derivadas de RUFT, su comportamiento es similar, ya que la distribución de tráfico en la última etapa conserva el mismo patrón base. Por ejemplo, aunque RUFT-PL tiene dobles enlaces en toda la red, la distribución de tráfico es la misma de RUFT. Por lo tanto su comportamiento será el mismo, con la ventaja de que obtendremos el doble de productividad. FT-RUFT-212 y FT-RUFT-222, también siguen el mismo patrón de conexión de RUFT, pero en este caso las rutas disjuntas proporcionadas por la doble inyección/eyección permiten reducir el *HoL Blocking* y aumentar el número de enlaces usados. Nuevamente, FT-RUFT-XL, hace mejor uso de los recursos ya que a diferencia de las otras topologías el número de rutas será mayor, conforme aumenta el número de etapas de la red.

Este patrón de tráfico no afecta de igual manera a FTA, ya que éste utiliza encaminamiento adaptativo y tiene la posibilidad de usar cualquiera de los puertos de salida disponibles en la fase ascendente, lo cual le da cierta ventaja sobre el resto de topologías, es por ello que sus prestaciones no se ven tan afectadas como en los demás casos.

Patrón de tráfico *Complement*

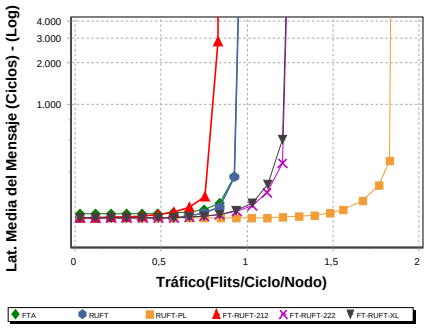
El patrón de tráfico *Complement* es de especial interés, ya que cada nodo de procesamiento solo envía mensajes a un único destino y, además, en las redes bidireccionales (BMIN), los mensajes siempre estan obligados a llegar hasta la última etapa de conmutadores de la red, antes de iniciar la fase descendente, para poder alcanzar su respectivo destino.

En RUFT, este tipo de tráfico suele presentar una gran ventaja, y es que los enlaces de red que forman parte de una ruta entre un nodo origen y un nodo destino, son exclusivos de dicha ruta. Es decir, los mensajes de diferentes nodos, a lo largo de su ruta, nunca compartirán enlaces con otras rutas para alcanzar su destino; siendo éste el caso contrario al patrón de tráfico *Bit-Reversal*.

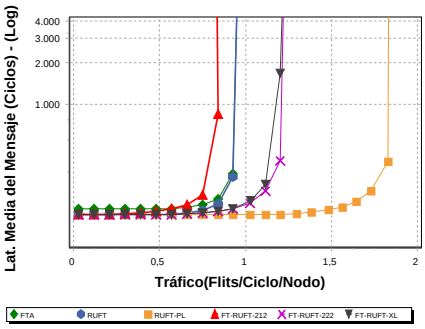
Como podemos ver en la Figura 4.10a, RUFT obtiene el mismo resultado que el ofrecido por FTA, sin embargo, en este último, la latencia será mayor que RUFT dado que todos los paquetes tienen que atravesar más etapas. En cuanto a FT-RUFT-212, la productividad obtenida es menor a la ofrecida por RUFT, a pesar de utilizar más hardware en la primera y última etapa. Esta baja productividad se debe a que la distribución del tráfico se modifica con respecto a RUFT, debido a los dobles enlaces de inyección/eyección, por lo tanto dejan de existir rutas únicas para cada mensaje, incrementando de esta manera el *HoL Blocking* en aquellos puntos donde se comparten enlaces de red.

Aunque FT-RUFT-222 y FT-RUFT-XL obtienen mejores prestaciones que las topologías anteriores, éstas también ven afectada su productividad, ya que la distribución de tráfico también cambia con respecto a RUFT, generando bloqueos en diferentes puntos de la red. Por otra parte, RUFT-PL es la topología que alcanza el mayor rendimiento, dado que el patrón de interconexión de la topología es igual al de RUFT, con la ventaja de que cada ruta va a disponer exclusivamente de dos enlaces paralelos, y no van a existir bloqueos en ningún punto de la red. Por este motivo podemos recibir el doble de *flits* por ciclo/nodo, y aumentar las prestaciones de la red.

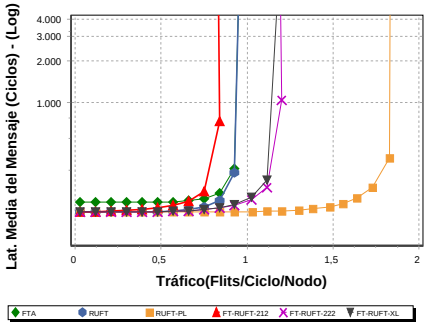
Analizando el resto de tamaños de red, Figura 4.10b a 4.10f, podemos notar que todas las redes, independientemente de su tamaño, tienen un comportamiento muy similar. RUFT y RUFT-PL siempre obtendrán la misma



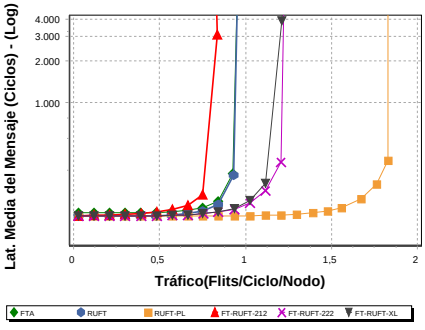
(a) 4-ary 3-tree



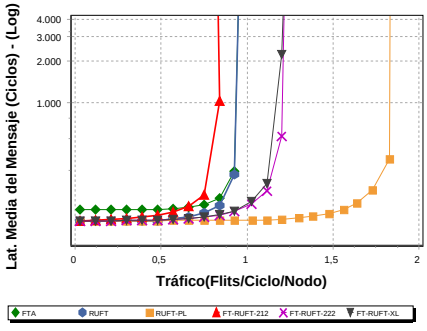
(b) 4-ary 4-tree



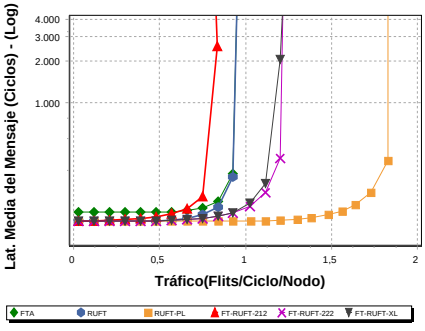
(c) 4-ary 5-tree



(d) 8-ary 3-tree



(e) 8-ary 4-tree



(f) 16-ary 3-tree

Figura 4.10: Latencia desde la generación del paquete versus tráfico aceptado para diferentes tamaños de red, tráfico *Complement* y un tamaño de paquete de 128B.

productividad bajo diferentes configuraciones, ya que éstas ofrecen rutas exclusivas para cada mensaje y eliminan cualquier bloqueo a lo largo de las rutas. FTA también obtiene resultados similares en cada caso, con la diferencia de que la latencia se incrementará conforme lo haga el número de etapas de la red.

Patrón de tráfico *Shuffle*

En este patrón de tráfico, al igual que *Complement* y *Bit-Reversal*, cada nodo de procesamiento solo envía mensajes a un único destino.

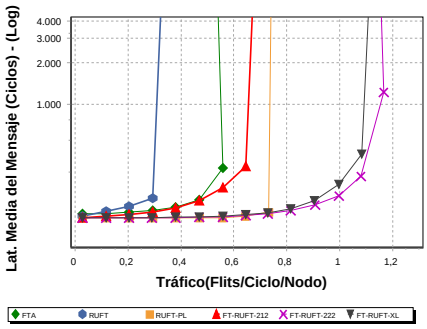
Los resultados de este análisis (Figura 4.11) muestran que RUFT es la topología que obtiene la productividad mas baja, ya que, debido al encaminamiento y a la distribución del tráfico, solo se utilizan la mitad de los enlaces de red, sin importar el tamaño de la red.

Como en anteriores ocasiones, RUFT-PL permite doblar la productividad alcanzada por RUFT, gracias a los enlaces paralelos. En cuanto a FT-RUFT-212, la topología es capaz de obtener mejores prestaciones que RUFT y FTA, como consecuencia de una mejor utilización de los recursos, generando menor cantidad de bloqueos que FTA. Finalmente, como era de esperar, FT-RUFT-222 y FT-RUFT-XL se benefician de la cantidad de rutas disponibles para obtener los mejores resultados.

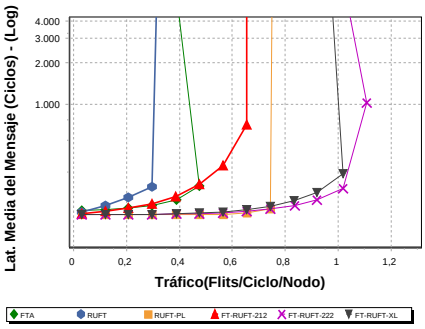
4.3.3. Influencia del tamaño del paquete

En esta sección, analizaremos cómo las diferentes configuraciones analizadas se ven afectadas por el tamaño del paquete utilizado dentro de la red. En particular, cada red se ha evaluado bajo diversos tamaños de paquetes para ver las diferencias que presentan al variar éste parámetro. Las simulaciones llevadas a cabo en esta sección se han realizado con una red 4-ary 3-tree usando tráfico uniforme, un canal virtual en la red y tamaños de paquete de 8B, 32B, 64B, 128B, 512B y 1024B.

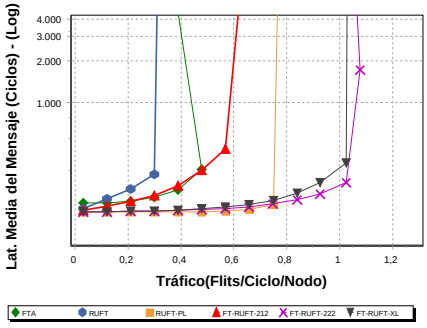
Como cabía esperar, tal y como podemos ver en la Figura 4.12, la latencia de red de cada topología se incrementa conforme lo hace el tamaño del paquete, al igual que la productividad de la red. Con paquetes pequeños (Figura 4.12a), la sobrecarga relativa del encaminamiento aumenta en la latencia del



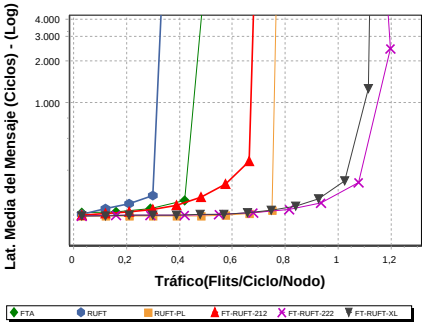
(a) 4-ary 3-tree



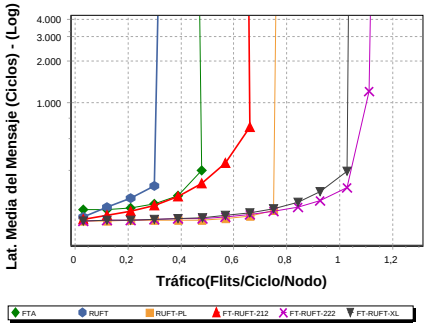
(b) 4-ary 4-tree



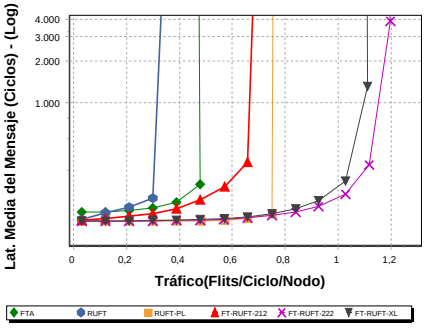
(c) 4-ary 5-tree



(d) 8-ary 3-tree



(e) 8-ary 4-tree



(f) 16-ary 3-tree

Figura 4.11: Latencia desde la generación del paquete versus tráfico aceptado para diferentes tamaños de red, tráfico *Shuffle* y un tamaño de paquete de 128B.

paquete. Por otra parte, para una misma carga, el número de operaciones de encaminamiento es mayor. Ello motiva que se alcance una productividad más baja.

Al utilizar tamaños de paquete muy grandes (Figura 4.12f), podemos amortizar la sobrecarga de encaminamiento y aumentar la productividad de la red a expensas de una alta latencia, la cual se ve más afectada por el tamaño del paquete que por el número de saltos.

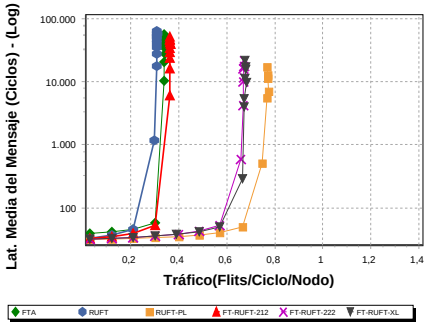
Como podemos ver en cada uno de los resultados mostrados en la Figura 4.12, la posición relativa de los resultados de cada topología se mantiene frente a las demás a medida que aumentamos el tamaño del paquete utilizado; además, podemos observar que a partir de un tamaño de paquete de 128B, las variaciones de productividad que presentan las redes son mínimas. Por este motivo seleccionamos este tamaño de paquete para realizar los análisis en esta tesis.

4.3.4. Influencia de los canales virtuales

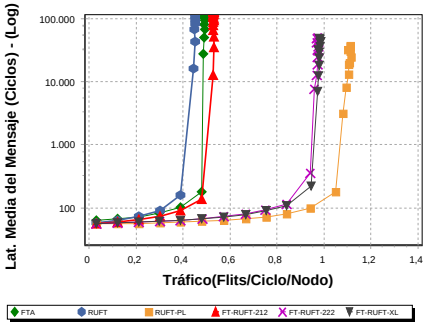
El uso de canales virtuales es uno de los mecanismos más utilizados en redes de interconexión para reducir el *HoL Blocking* y reducir los bloqueos de los mensajes. Por estos motivos, en esta sección analizaremos la influencia de los canales virtuales sobre las prestaciones obtenidas por las redes de interconexión propuestas en esta tesis.

Para llevar a cabo la presente evaluación hemos analizado una red 4-ary 3-tree, con tráfico uniforme, un tamaño de paquete de 128B e implementando 1, 2, 3 y 4 canales virtuales para cada una de las topologías propuestas, comparándolas con RUFT y FTA. La gestión de los canales virtuales dentro de un puerto se realiza de forma dinámica, es decir, un paquete es almacenado en el canal virtual que más espacio disponga para almacenarlo.

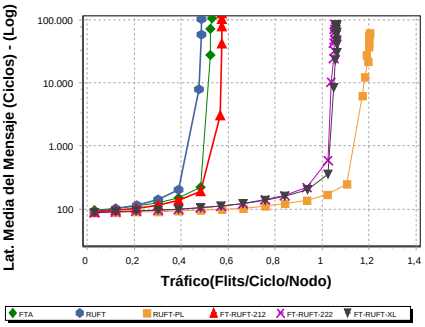
En la Figura 4.13 mostramos el rendimiento de red alcanzado por cada topología, bajo los escenarios descritos previamente. En la Figura 4.13a tenemos todas las topologías con 1 canal virtual (o visto de otra manera, sin canales virtuales). En este caso podemos ver que la productividad alcanzada por RUFT y FTA es muy cercana. Aunque FTA utilice más recursos de hardware, su encaminamiento adaptativo siempre demandará los puertos de salida menos saturados pero, generando el nocivo efecto *HoL Blocking*.



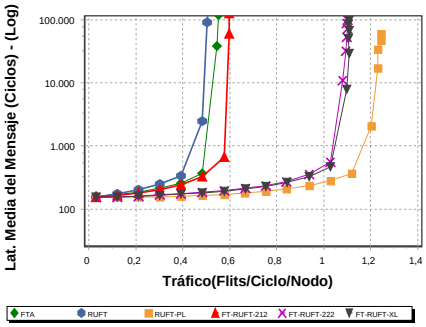
(a) Tamaño del paquete: 8B



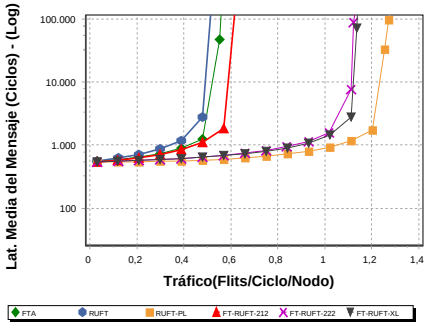
(b) Tamaño del paquete: 32B



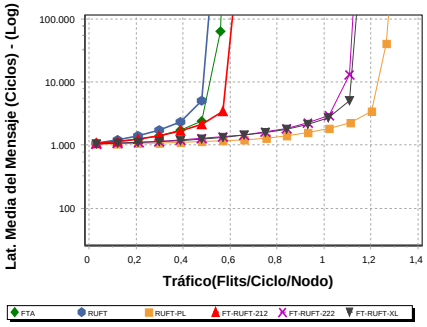
(c) Tamaño del paquete: 64B



(d) Tamaño del paquete: 128B

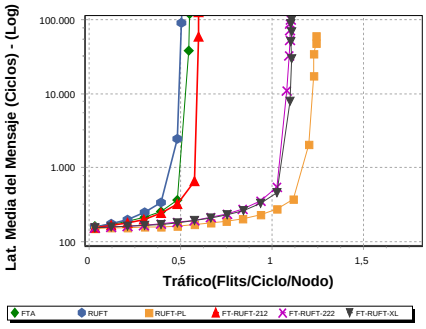


(e) Tamaño del paquete: 512B

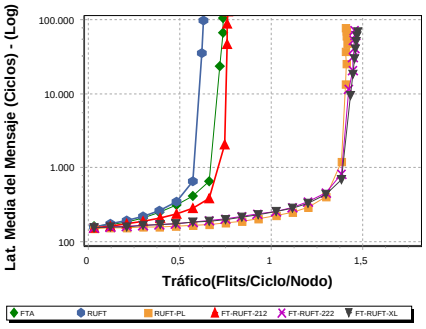


(f) Tamaño del paquete: 1024B

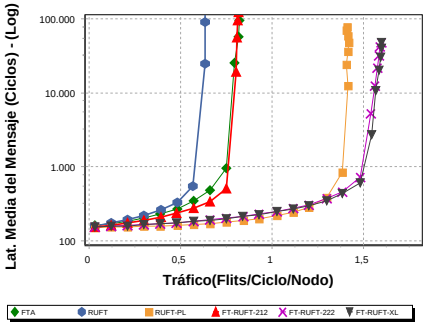
Figura 4.12: Influencia del tamaño del paquete en una red 4-ary 3-tree usando tráfico de tipo *Uniform*.



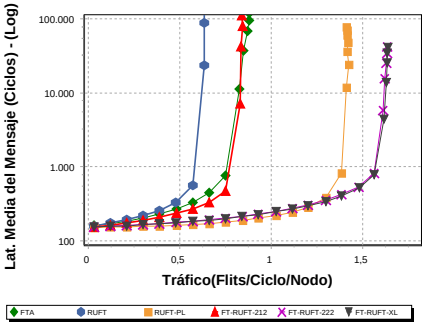
(a) 1 canal virtual



(b) 2 canales virtuales



(c) 3 canales virtuales



(d) 4 canales virtuales

Figura 4.13: Influencia de los canales virtuales sobre el rendimiento de la red, en una red 4-ary 3-tree con diversas topologías, tráfico de tipo *Uniform* y un tamaño de paquete de 128B.

FT-RUFT-212 obtiene un mejor rendimiento que RUFT y FTA, ya que la doble inyección/eyección ofrece más rutas para cada par origen-destino, permitiendo que el tráfico esté mejor distribuido en la red e incrementando su productividad. RUFT-PL obtiene mayor rendimiento que las otras topologías, ya que el tráfico está clasificado, y cada par de enlaces de salida de un conmutador comunican con el mismo conmutador en la siguiente etapa, permitiendo que los paquetes avancen más rápido al poder utilizar cualquiera de los dos canales de salida correspondientes, reduciendo considerablemente el *HoL Blocking* e incrementando la productividad. Por último, FT-RUFT-222 y FT-RUFT-XL obtienen aproximadamente el mismo rendimiento, dado que la distribución de tráfico dentro de esta red es muy similar, debido al bajo número de etapas que la componen. Aunque la productividad de ambas topologías es alta, ésta es menor que la ofrecida por RUFT-PL.

Al agregar un segundo canal virtual a la red, Figura 4.13b, la productividad de la red mejora considerablemente en todos los casos a expensas de un leve incremento en la latencia de red. En este caso, FTA ligeramente sobrepasa la productividad de RUFT ya que el *HoL Blocking* generado por el encaminamiento adaptativo se ve reducido gracias a la existencia de un segundo canal virtual. Sin embargo, en RUFT los paquetes pueden permanecer bloqueados por más tiempo si los paquetes de ambos canales virtuales demandan el mismo puerto de salida, teniendo en cuenta que esta topología es unidireccional y usa un encaminamiento determinista, la única posibilidad de encaminar un paquete es esperando a que el puerto requerido se libere. Éste efecto también se presenta en FT-RUFT-212 dado que el hardware utilizado dentro la red es el mismo (1 enlace de red).

Por otra parte, RUFT-PL también incrementa su productividad. Sin embargo, la mejora obtenida por FT-RUFT-222 y FT-RUFT-XL es superior. De hecho, obtienen las mismas prestaciones que RUFT-PL. El efecto de disponer de más canales virtuales no aporta tanto a RUFT-PL, dado que éste ya disponía de dos canales en paralelo. Sin embargo, en el caso de FT-RUFT-222 y FT-RUFT-XL, el hecho de poder optar entre rutas alternativas en la inyección y en la primera etapa (destino primario o alterno) introducía *HoL Blocking*, el cual queda reducido gracias a los canales virtuales.

Al incluir más canales virtuales en la red, la productividad de la red todavía

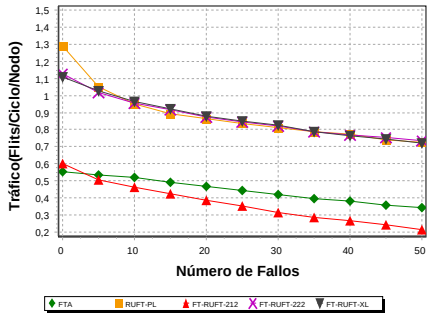
mejora más en todos los casos, tal como podemos ver en las Figuras 4.13c y 4.13d. No obstante, FT-RUFT-222 y FT-RUFT-XL se ven muy beneficiadas en comparación con las otras topologías, por la reducción del efecto *HoL Blocking*. De hecho, obtienen prestaciones superiores a RUFT-PL. Cabe decir que FT-RUFT-XL se vería más beneficiada en redes de más etapas, ya que como hemos comentado, el nivel de rutas entre cada par origen-destino se incrementa cuanto más etapas disponga la red.

4.4. Degradación de prestaciones

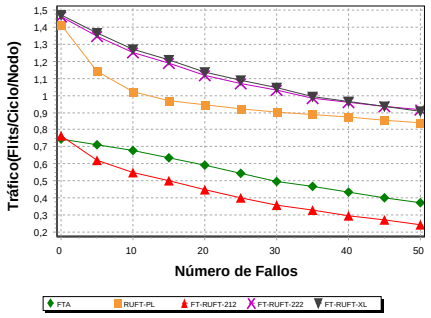
A lo largo de este capítulo hemos mostrado cómo las topologías de red propuestas en el presente trabajo proporcionan un alto nivel de tolerancia a fallos, como vimos en la Sección 4.2, a la vez que ofrecen unas buenas prestaciones de red bajo diversos patrones de tráfico, como se analizó en la Sección 4.3. Sin embargo, también es muy importante saber cómo se comporta la productividad de la red a medida que se van generando fallos en el sistema. Es por ello que en esta sección realizaremos un análisis de la degradación de prestaciones de las topologías propuestas, comparándolas frente al fat-tree con encaminamiento adaptativo. Analizaremos diversos tamaños de redes, usando un tamaño de paquete de 128B, uno y dos canales virtuales, bajo diferentes patrones de tráfico. En esta sección no evaluaremos la topología RUFT dado que ésta no ofrece tolerancia a fallos en ningún punto de la red.

La Figura 4.14 y 4.15 muestra los resultados de degradación de prestaciones de las topologías FTA, RUFT-PL, FT-RUFT-212, FT-RUFT-222 y FT-RUFT-XL bajo tráfico de tipo *Uniform*, para redes 4-ary n-tree y 8-ary n-tree, respectivamente, usando un canal virtual (gráficas de la izquierda) y dos canales virtuales (gráficas de la derecha).

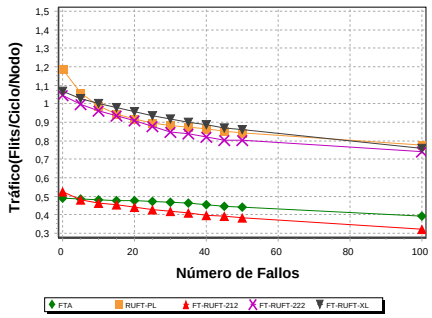
Como podemos ver en cada una de las gráficas, usando un canal virtual, RUFT-PL es la topología que obtiene la mayor productividad de red. Sin embargo, a medida que se van introduciendo fallos en los enlaces de red, la degradación de prestaciones se hace más notable porque la topología no es capaz de hacer frente al tráfico con un número de rutas tan reducido para cada par origen-destino, incrementando el *HoL Blocking* en aquellos puntos donde solo existe un enlace de interconexión para enviar el tráfico, debido a



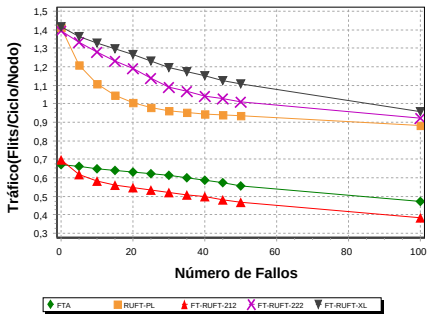
(a) 4-ary 3-tree, 1 canal virtual



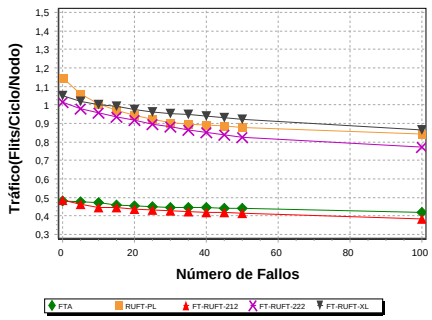
(b) 4-ary 3-tree, 2 canales virtuales



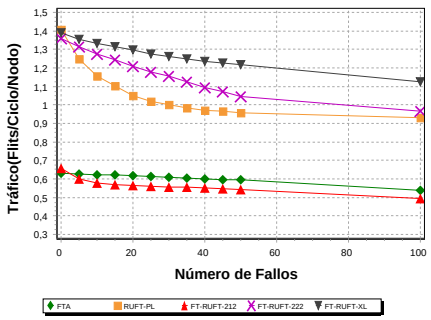
(c) 4-ary 4-tree, 1 canal virtual



(d) 4-ary 4-tree, 2 canales virtuales

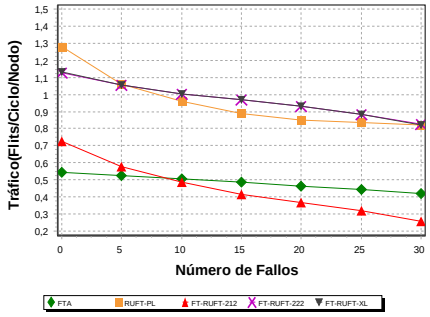


(e) 4-ary 5-tree, 1 canal virtual

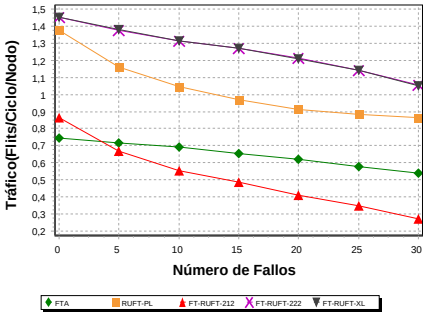


(f) 4-ary 5-tree, 2 canales virtuales

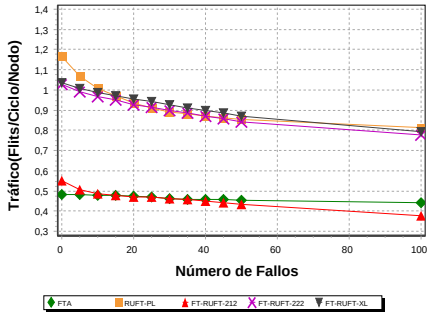
Figura 4.14: Degradación de prestaciones con fallos para redes 4-ary con diferentes etapas, bajo tráfico de tipo *Uniform*, 1 y 2 canales virtuales y un tamaño de paquete de 128B.



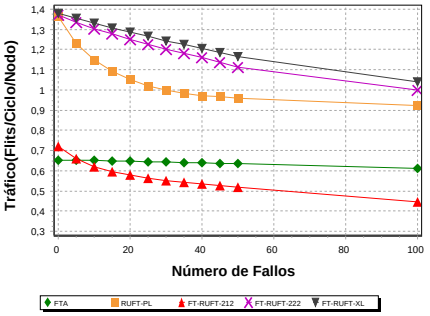
(a) 8-ary 2-tree, 1 canal virtual



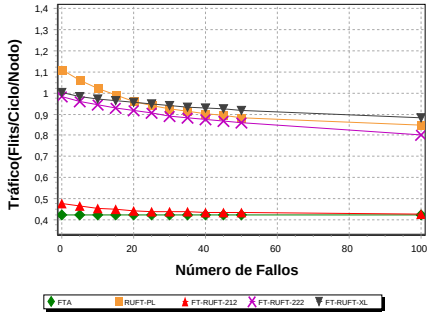
(b) 8-ary 2-tree, 2 canales virtuales



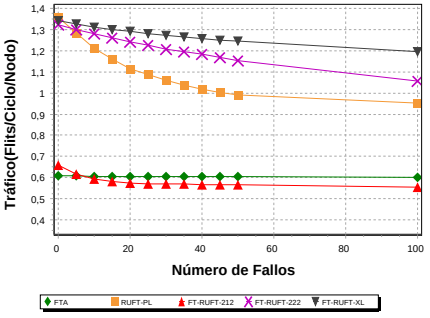
(c) 8-ary 3-tree, 1 canal virtual



(d) 8-ary 3-tree, 2 canales virtuales



(e) 8-ary 4-tree, 1 canal virtual



(f) 8-ary 4-tree, 2 canales virtuales

Figura 4.15: Degradación de prestaciones con fallos para redes 8-ary con diferentes etapas, bajo tráfico de tipo *Uniform*, 1 y 2 canales virtuales y un tamaño de paquete de 128B.

los fallos presentes en el sistema. También podemos notar que, a medida que el tamaño de la red incrementa, la degradación de prestaciones se hace menor, dado que el porcentaje de enlaces afectados es menor para el mismo número de fallos, en comparación con las redes de menor tamaño. Aún así la degradación relativa que presenta RUFT-PL es mayor que las demás.

Al utilizar dos canales virtuales, como ya sabemos, las prestaciones de la red se incrementan notablemente. Sin embargo, en RUFT-PL, conforme se van introduciendo fallos en los enlaces de red la degradación de prestaciones se hace mayor, ya que el número de paquetes que permanecen en los buffers es el doble en comparación con un canal virtual, incrementando de esta manera el *HoL Blocking* a causa del bajo número de rutas disponibles por par origen-destino.

Por otra parte, la degradación de FT-RUFT-212, con un canal virtual, es menor que RUFT-PL, gracias a los dos enlaces de inyección/eyección que permiten distribuir el tráfico entre las diferentes rutas que provee la topología, tomando ventaja de los enlaces que aún quedan disponibles. Ahora, al incrementar el número de canales virtuales, la topología también incrementa su productividad, y a diferencia de RUFT-PL, gracias a las diferentes rutas disjuntas que proporciona la topología la degradación de prestaciones no es tan significativa, sobre todo en las redes más grandes.

FT-RUFT-222 presenta un comportamiento similar a FT-RUFT-212, con la diferencia de que la productividad de la red presenta menor degradación, gracias a los dobles enlaces de red, y a las ocho rutas que la topología provee dentro de la red para cada par origen-destino.

Nuestra última propuesta, FT-RUFT-XL, es la topología que ofrece la menor degradación de prestaciones a pesar de tener un alto número de fallos en la red. Como hemos mencionado antes, esta topología ofrece grandes ventajas sobre las demás debido a su alto nivel de conectividad. Por ejemplo, en la Figuras 4.15a y 4.15b tenemos una red de dos etapas con uno y dos canales virtuales, respectivamente, y podemos ver que FT-RUFT-XL obtiene el mismo nivel de degradación de prestaciones que FT-RUFT-222 debido a que ambas topologías proporcionan el mismo número de rutas para cada par de nodos de procesamiento. Este comportamiento será igual en cualquier red de dos etapas. Sin embargo, a medida que se va incrementando el número de etapas en la red vemos que la productividad de FT-RUFT-XL se mantiene más uniforme, por

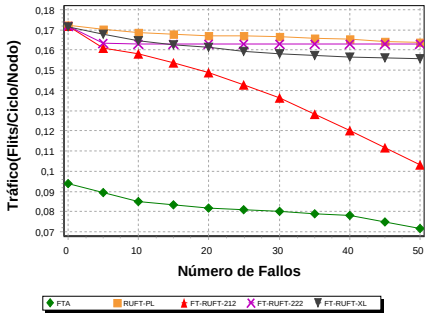
encima de RUFT-PL y FT-RUFT-222, a pesar de utilizar el mismo hardware en cada punto de la red. Conforme se van agregando más etapas a la red, la degradación de prestaciones se hace menor, confirmando que el número de rutas aumenta con cada etapa que se agrega a la red, y por ende, la productividad será más estable a pesar del alto número de fallos presentes en el sistema.

Finalmente, podemos ver que la topología FTA presenta la menor degradación de prestaciones relativa, debido a que su encaminamiento adaptativo permite encaminar los paquetes a través de todos los puertos disponibles, incrementando además el número de rutas disponibles conforme lo hace lo aridad del conmutador y/o el número de etapas de la red. No obstante, solo logra sobrepasar a la topología FT-RUFT-212 que tiene aproximadamente la mitad de componentes de red.

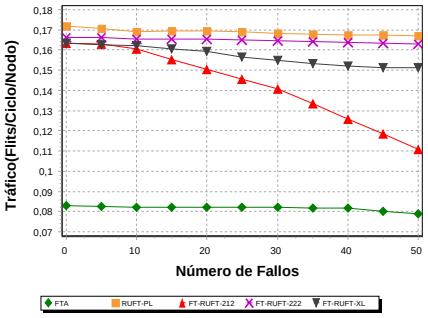
En la Figura 4.16 mostramos la degradación de prestaciones de las topologías propuestas y de FTA, bajo el patrón de tráfico *Hot-Spot* al 15 %. Como podemos apreciar, tanto RUFT-PL, FT-RUFT-222 y FT-RUFT-XL mantienen el tráfico lo más estable posible a pesar del alto número de fallos, gracias a los dobles enlaces dentro de la red y a la doble inyección/eyección que nos permite equilibrar el tráfico entre ellos. Aunque FT-RUFT-212 también dispone de dobles enlaces en la inyección/eyección, la degradación de prestaciones es mayor, en comparación con las otras topologías, debido a la poca cantidad de enlaces dentro de la red (Figura 4.16b), pero a medida que el número de enlaces de red incrementa, la topología se ve menos afectada por los fallos, dando lugar a una baja degradación de prestaciones, como podemos ver en la Figura 4.16d.

Con respecto a FTA, la degradación de prestaciones relativa es mínima, debido al alto número de rutas que ofrece el encaminamiento adaptativo, sin embargo, la productividad máxima alcanzada por ésta es muy baja, comparada con las demás propuestas.

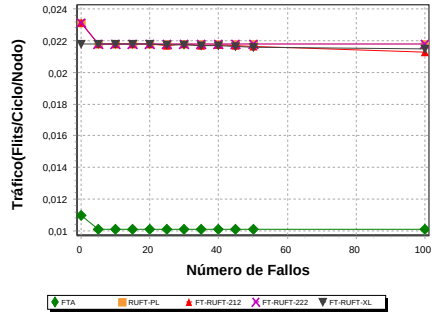
En la Figura 4.17 realizamos el análisis correspondiente al patrón de tráfico *Complement*. Como hemos explicado en la Sección 4.3.2, bajo este patrón de tráfico, RUFT-PL ofrece una única ruta, con dobles enlaces, para cada par origen-destino. Ahora, al introducir fallos en los enlaces de red, en redes pequeñas (Figura 4.17b), la topología sufre una alta degradación de prestaciones dado que ésta no es capaz de gestionar todo el tráfico en aquellos puntos don-



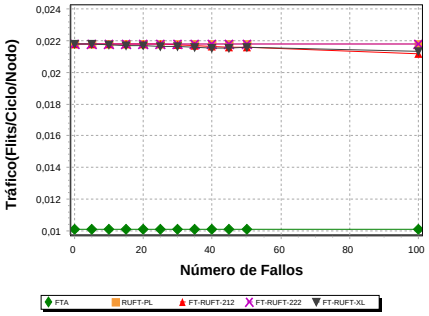
(a) 4-ary 3-tree, 1 canal virtual



(b) 4-ary 3-tree, 2 canales virtuales



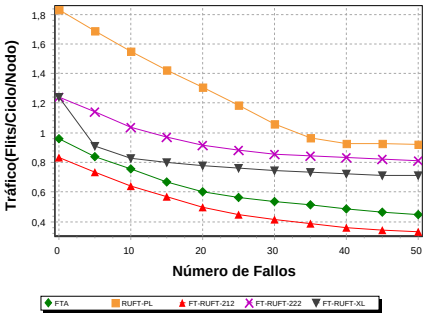
(c) 8-ary 3-tree, 1 canal virtual



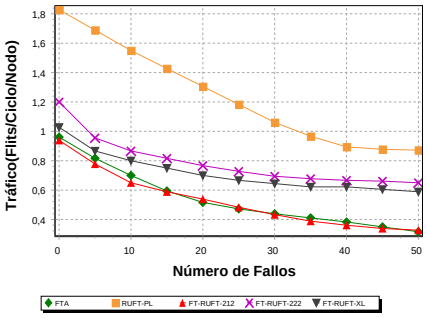
(d) 8-ary 3-tree, 2 canales virtuales

Figura 4.16: Degradación de prestaciones con fallos para diferentes tamaños de redes con tráfico *Hot-Spot* al 15 % y un tamaño de paquete de 128B.

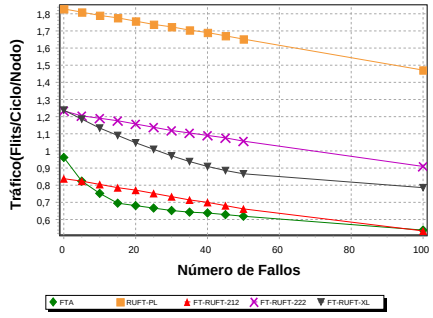
de solo queda disponible un enlace. Respecto a FT-RUFT-212, vemos que en una red pequeña (Figura 4.17b) la degradación de prestaciones es muy similar a FTA, aunque su productividad se encuentra un poco por debajo. Sin embargo, cuando se incrementa el tamaño de la red (Figura 4.17d) vemos que FT-RUFT-212 tiene menor degradación que FTA a pesar del alto número de fallos en la red. A pesar de que FT-RUFT-212 dispone de menos rutas por par origen-destino que el FTA (cuando la aridad del conmutador es mayor a 4), es capaz de superar la productividad alcanzada por éste último. En cuanto a FT-RUFT-222, la degradación relativa que ofrece la topología es baja, en comparación con RUFT-PL. Aunque la distribución de los enlaces dentro de la red es igual a RUFT-PL, con el simple hecho de tener una doble inyec-



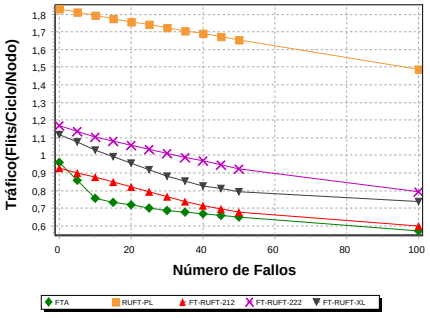
(a) 4-ary 3-tree, 1 canal virtual



(b) 4-ary 3-tree, 2 canales virtuales



(c) 8-ary 3-tree, 1 canal virtual

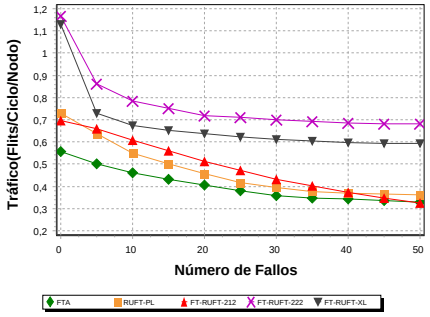


(d) 8-ary 3-tree, 2 canales virtuales

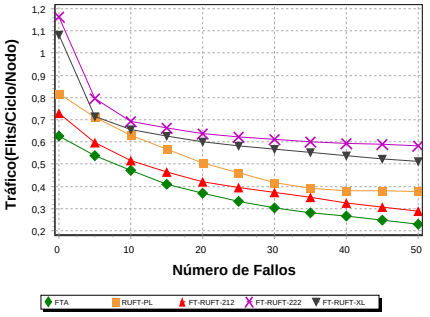
Figura 4.17: Degradación de prestaciones con fallos para diferentes tamaños de redes con tráfico *Complement* y un tamaño de paquete de 128B.

ción/eyección disjunta, podemos equilibrar el tráfico a través de los diferentes enlaces para reducir los bloqueos dentro de la red.

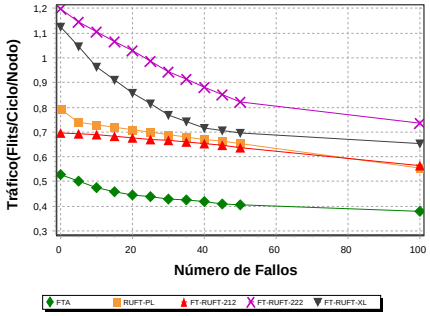
En la Figura 4.18 tenemos el análisis de degradación de prestaciones bajo el patrón de tráfico *Shuffle*. Debido a que en las UMINs este patrón de tráfico deja sin utilizar un alto número de enlaces en la red, de manera similar a *Bit-Reversal*, las topologías suelen verse más afectadas por la presencia de fallos. De manera similar al resto de patrones de tráfico analizado, el hecho de disponer de dos canales virtuales mejora las prestaciones alcanzadas. Como podemos observar, FT-RUFT-222 y FT-RUFT-XL son las topologías que sufren la mayor degradación de prestaciones. Sin embargo, aún así, éstas son capaces de mantener la productividad por encima de FT-RUFT-212 y de FTA,



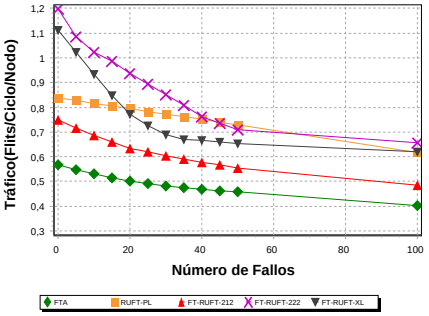
(a) 4-ary 3-tree, 1 canal virtual



(b) 4-ary 3-tree, 2 canales virtuales



(c) 8-ary 3-tree, 1 canal virtual

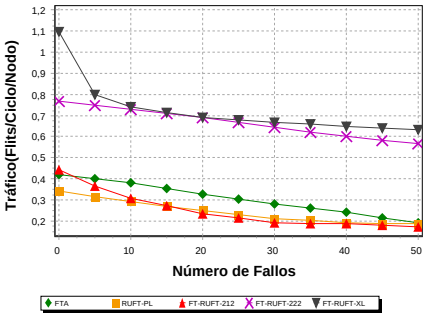


(d) 8-ary 3-tree, 2 canales virtuales

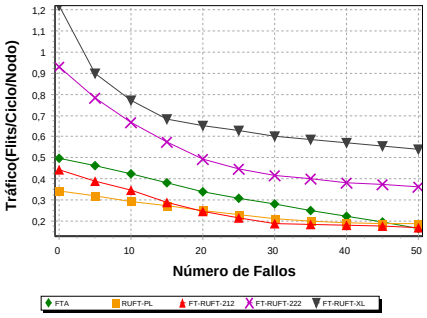
Figura 4.18: Degradación de prestaciones con fallos para diferentes tamaños de redes con tráfico *Shuffle* y un tamaño de paquete de 128B.

a pesar de que esta última utiliza todos sus enlaces de red para encaminar los mensajes. RUFT-PL, con un canal virtual, se ve bastante afectada por los fallos de red, hasta tal punto de obtener un rendimiento cercano a FT-RUFT-212. Al implementar dos canales virtuales aumentan sus prestaciones, pero a medida que van apareciendo fallos en la red, se incrementa el *HoL Blocking* y por lo tanto la degradación de prestaciones es mayor. Con respecto a FT-RUTF-212, podemos apreciar que la topología nos ofrece mejores prestaciones que FTA para cualquier número de fallos, a pesar de utilizar menor cantidad de hardware.

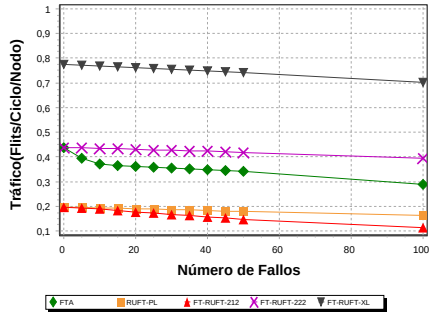
Por último, en la Figura 4.19 presentamos los resultados de degradación de prestaciones para el patrón de tráfico *Bit-Reversal*. En este caso, de forma



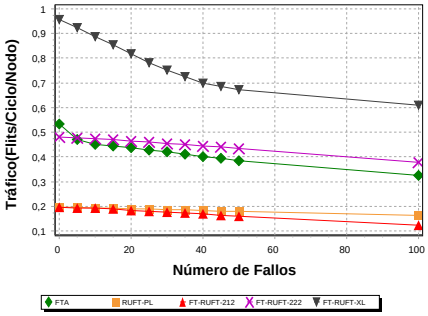
(a) 4-ary 3-tree, 1 canal virtual



(b) 4-ary 3-tree, 2 canales virtuales



(c) 8-ary 3-tree, 1 canal virtual



(d) 8-ary 3-tree, 2 canales virtuales

Figura 4.19: Degradación de prestaciones con fallos para diferentes tamaños de redes con tráfico *Bit-Reversal* y un tamaño de paquete de 128B.

relativa, la degradación de prestaciones de FTA, RUFT-PL y FT-RUFT-212 es mínima, utilizando diferentes canales virtuales. Por una parte FTA puede distribuir el tráfico que va dirigido a un mismo destino a través de todos sus enlaces, lo cual es beneficioso ante la presencia de fallos. RUFT-PL gracias a la clasificación de destinos entre sus enlaces y a las rutas dobles que existen entre cada par de nodos, permiten mantener el tráfico lo más estable posible ante los fallos de enlaces. Con respecto a FT-RUFT-212, su comportamiento es muy similar a RUFT-PL, gracias a la clasificación de destinos que permite distribuir el tráfico de manera eficiente para reducir los bloqueos de mensajes.

Por otra parte, la degradación de prestaciones de FT-RUFT-222 es mayor en redes pequeñas (Figuras 4.19a y 4.19b). Sin embargo, en redes más grandes

la degradación es mínima gracias al nivel de conectividad que ofrece la red. Finalmente, FT-RUFT-XL es la topología que obtiene la mayor degradación relativa de prestaciones. Aun así, es la que obtiene mejores prestaciones en presencia de fallos.

4.5. Evaluación coste/rendimiento

Como hemos visto a lo largo de este trabajo, las topologías de red propuestas requieren diferentes configuraciones en el hardware utilizado. Algunas usan más canales por conmutador o más enlaces entre los elementos de la red, además de una lógica de conmutación añadida que depende del algoritmo de encaminamiento implementado por cada una de ellas.

Todos estos aspectos dificultan la existencia de una métrica estándar para obtener el coste total de un sistema, aunque algunos autores [59], realizan este cálculo basado en la Fórmula 4.1, donde contemplan de forma directa el precio de los conmutadores y de los enlaces requeridos.

$$Coste\_Total = Coste_{conmutador} \times N_{conmutadores} + Coste_{enlace} \times N_{enlaces} \quad (4.1)$$

Aunque esta ecuación expresa de manera global el coste del sistema, se limita más al valor de los elementos en el mercado que al propio coste de diseño. Por este motivo, para analizar y comparar el coste del hardware de cada topología descrita en el presente trabajo, hemos considerado los elementos más destacados en el diseño de una red: el número de enlaces unidireccionales y el número de elementos de conmutación. Este último factor implica el número de conmutadores, su grado y su complejidad interna.

En la Tabla 4.3 presentamos la complejidad de un conmutador para todas las topologías analizadas, medida como el número de elementos de conmutación requeridos para conectar los puertos de entrada y de salida de un conmutador.

Como podemos apreciar, la complejidad de un conmutador no solo va ligada al diseño de la red, sino también al mecanismo de encaminamiento que implementan los conmutadores. En el caso de la topología FTA, debido a que

Topología	Complejidad del conmutador
FTA	$3k^2$
RUFT	k^2
RUFT-PL	$4k^2$
FT-RUFT-212	k^2 (Etapas interm.) | $2k^2$ (Primera y última etapa)
FT-RUFT-222	$4k^2$
FT-RUFT-XL	$4k^2$

Tabla 4.3: Comparación entre la complejidad de un conmutador bajo diferentes topologías de red.

utiliza un encaminamiento adaptativo, el número de puertos que puede demandar un paquete es mayor, lo que lleva a una complejidad del conmutador de $3k^2$ elementos de conmutación, tal como hemos visto en la Sección 2.3. Por otra parte, la complejidad del conmutador de FT-RUFT-212 depende de la etapa de red donde se encuentre dicho conmutador, dado que aquellos que se encuentren en la primera y última etapa requerirán una complejidad de $2k^2$, puesto que son asimétricos, mientras que los conmutadores de las etapas intermedias tienen una complejidad de k^2. El diseño de esta topología favorece el coste de la red, ya que todos los conmutadores de las etapas intermedias son de un coste/complejidad relativamente bajos, como los usados por RUFT, y a medida que aumenta el número de etapas el coste de diseño no se ve muy afectado.

En RUFT-PL, FT-RUFT-222 y FT-RUFT-XL, para tolerar fallos en los enlaces de red es necesario implementar una función de selección dinámica, lo cual incrementa el número de puertos/canales de salida disponibles para cada paquete, y por ende el número de elementos de conmutación a $4k^2$, es decir, cada uno de los $2k$ canales de salida pueden ser demandados por los $2k$ canales de entrada. Por último, RUFT es la topología que requiere menos complejidad de diseño en el conmutador, por lo tanto será también la de menor coste.

Para analizar de manera conjunta el coste y el rendimiento de la red, mediante la Tabla 4.4 mostramos el número de enlaces y elementos de conmutación de cada red junto a los parámetros de prestaciones (productividad y latencia base).

Como hemos mencionado anteriormente, la topología más económica es RUFT, dado que ésta utiliza el menor número de enlaces y el menor número

	Topología	Enlaces[1]	Elementos de conmutación	Productividad[2]	Latencia[3] carga baja
4-ary 3-tree	FTA	384	2304	0,55	161
	RUFT	256	768	0,51	156
	RUFT-PL	512	3072	1,24	152
	FT-RUFT-212	384	1280	0,60	154
	FT-RUFT-222	512	3072	1,10	153
	FT-RUFT-XL	512	3072	1,10	153
8-ary 3-tree	FTA	3072	36864	0,48	162
	RUFT	2048	12288	0,45	159
	RUFT-PL	4096	49152	1,16	153
	FT-RUFT-212	3072	20480	0,55	156
	FT-RUFT-222	4096	49152	1,03	154
	FT-RUFT-XL	4096	49152	1,03	154
16-ary 3-tree	FTA	24576	589824	0,43	165
	RUFT	16384	196608	0,42	147
	RUFT-PL	32768	786432	1,11	140
	FT-RUFT-212	24576	327680	0,52	143
	FT-RUFT-222	32768	786432	1,00	141
	FT-RUFT-XL	32768	786432	1,00	141

[1] Número de enlaces unidireccionales en toda la red.

[2] La productividad de la red está dada en flits/ciclo/nodo.

[3] La latencia de red está dada en ciclos.

Tabla 4.4: Coste/Rendimiento para diferentes tamaños de redes usando tráfico *Uniform*, un tamaño de paquete de 128B y 1 canal virtual.

de elementos de conmutación. Ésta topología utiliza menos de la mitad de recursos que FTA y menos recursos que las otras topologías. Sin embargo, el inconveniente que presenta es que no provee tolerancia a fallos. Por otra parte, FT-RUFT-212 incrementa el número de enlaces y elementos de conmutación requeridos para implementar la topología, en comparación con RUFT, debido a que los conmutadores de la primera y última etapa son asimétricos, pero cabe recalcar que esta modificación incrementa ligeramente las prestaciones de la red y proporciona tolerancia a fallos tanto en los enlaces de red como en los de inyección/eyección, ofreciendo cuatro rutas alternativas dentro de la red para cada par origen-destino. Además, el número de enlaces de red requerido por ésta es igual a FTA, para redes de tres etapas, pero a medida que el número de etapas aumenta el número de enlaces de red se reduce, bajando de esta forma su coste total, ofreciendo también un mayor rendimiento de red en la mayoría de patrones de tráfico estudiados.

	Topología	1 canal virtual	2 canales virtuales	3 canales virtuales	4 canales virtuales
4-ary 5-tree	FTA	61440	245760	552960	983040
	RUFT	20480	81920	184320	327680
	RUFT-PL	81920	327680	737280	1310720
	FT-RUFT-212	28672	114688	258048	458752
	FT-RUFT-222	81920	327680	737280	1310720
	FT-RUFT-XL	81920	327680	737280	1310720

Tabla 4.5: Elementos de conmutación requeridos por una red 4-ary 5-tree usando diferente número de canales virtuales.

Como podemos apreciar en este análisis, RUFT-PL, FT-RUFT-222 y FT-RUFT-XL requieren más enlaces que FTA, debido a los dobles enlaces de inyección/eyección, y más elementos de conmutación debido a la función de selección utilizada en cada conmutador, pero este leve incremento de hardware permite obtener más del doble de prestaciones que el ofrecido por las otras topologías, además la latencia de red también es la más baja.

Es importante mencionar que aunque RUFT-PL, FT-RUFT-222 y FT-RUFT-XL requieren el mismo número de enlaces y elementos de conmutación, la diferencia en cuanto a la tolerancia a fallos es notoria, siendo FT-RUFT-XL la que ofrece mejor tolerancia a fallos, tal como vimos en la Sección 4.2.

Finalmente, en las Tablas 4.5 y 4.6 mostramos el número de elementos de conmutación requerido por cada topología y la productividad alcanzada al usar diferentes canales virtuales en una red 4-ary 5-tree.

Como era de esperar, al incrementar el número de canales virtuales en los conmutadores también lo hace el número de elementos de conmutación requerido por cada topología (Tabla 4.5). Como se explicó anteriormente, el número de elementos de conmutación requerido por FT-RUFT-212 es bastante inferior al de FTA, gracias a que los conmutadores de las etapas intermedias tienen una baja complejidad y el incremento de elementos de conmutación solo se ve reflejado en la primera y última etapa de la red.

Analizando la Tabla 4.6 vemos que el incremento de canales virtuales no beneficia de forma proporcional a las topologías. Por una parte, vemos que la topología RUFT mejora con cada canal virtual añadido, pero aun así sigue siendo la que menor productividad ofrece. En cuanto a FT-RUFT-212, con

	Topología	1 canal virtual	2 canales virtuales	3 canales virtuales	4 canales virtuales
4-ary 5-tree	FTA	0,48	0,63	0,71	0,76
	RUFT	0,44	0,60	0,63	0,63
	RUFT-PL	1,15	1,41	1,42	1,42
	FT-RUFT-212	0,48	0,66	0,73	0,77
	FT-RUFT-222	1,01	1,36	1,50	1,58
	FT-RUFT-XL	1,05	1,39	1,53	1,60

* La productividad de la red está dada en flits/ciclo/nodo.

Tabla 4.6: Productividad alcanzada en una red 4-ary 5-tree usando tráfico *Uniform*, diferente número de canales virtuales y un tamaño de paquete de 128B.

cada canal virtual que se agregue a la red, ésta incrementa su productividad, siendo ligeramente mejor en todos los casos que la ofrecida por FTA, a pesar de que ésta última utiliza más recursos hardware.

En el caso de RUFT-PL vemos que a partir de dos canales virtuales la productividad ofrecida por la topología no varía demasiado respecto a la anterior, a pesar de que el número de elementos de conmutación se eleva con cada canal virtual. Como se ha explicado antes, esto se debe a que los enlaces paralelos ya realizan el aporte que deberían brindar los canales virtuales, en relación a la reducción del *HoL Blocking*.

Por último, FT-RUFT-222 y FT-RUFT-XL son las que mejor se comportan ante la inclusión de los canales virtuales. Aunque el número de elementos de conmutación es igual al de RUFT-PL, con cada canal virtual agregado a la red, la productividad incrementa notablemente llegando a más del doble de la ofrecida por FTA.

4.6. Conclusiones

En este capítulo se ha realizado una amplia evaluación de las topologías RUFT-PL, FT-RUFT-212, FT-RUFT-222 y FT-RUFT-XL y las hemos comparado entre sí y frente a la topología FTA. Para analizar las propiedades, ventajas y desventajas que ofrecen cada una de las topologías desarrolladas se han evaluado aspectos tan importantes como la tolerancia a fallos, las prestaciones de red, la degradación de prestaciones que sufre la red ante la presencia

de fallos y el coste de diseño que supone cada una de ellas.

En la evaluación de tolerancia a fallos se han considerado fallos a nivel de enlace y a nivel de conmutador. En este caso, la topología RUFT-PL, gracias a sus enlaces paralelos, puede soportar algunos fallos de enlaces en cualquier punto de la red; sin embargo no soporta el fallo de ambos enlaces paralelos a la vez, dado que éstos siempre conectan con el mismo conmutador o nodo de procesamiento. Al no ofrecer rutas alternativas, la topología tampoco soporta el fallo de conmutadores en ningún punto de la red. Por otra parte, la topología FT-RUFT-212, utilizando unos pocos enlaces extra únicamente en la inyección/eyección, es capaz de sobrepasar el nivel de tolerancia a fallos ofrecido por RUFT-PL. Además, gracias a las rutas disjuntas que ofrece la topología, también soporta el fallo de conmutadores en cualquier punto de la red.

En cuanto a FT-RUFT-222, gracias a los dobles enlaces de red y a las diversas rutas que ofrece la topología, el nivel de tolerancia a fallos a nivel de enlace se incrementa considerablemente con respecto al FTA y FT-RUFT-212. Por último, FT-RUFT-XL es la topología que ofrece el mejor nivel de tolerancia a fallos, tanto a nivel de enlace como de conmutador, gracias a los conmutadores intercambiables que incrementan el grado de conectividad del sistema a medida que aumenta el tamaño de la red.

Analizando las prestaciones que ofrecen las topologías implementadas en este trabajo, hemos visto que es posible sobrepasar la productividad ofrecida por el FTA, rediseñando las conexiones de la topología RUFT y utilizando un algoritmo de encaminamiento adecuado. En este análisis, bajo los patrones de tráfico *Uniform* y *Hot-Spot*, las topologías RUFT-PL, FT-RUFT-222 y FT-RUFT-XL alcanzan el doble de productividad que el FTA. Bajo los otros patrones de tráfico analizados, las topologías FT-RUFT rompen con el esquema del patrón de tráfico debido a la diversidad de rutas ofrecidas dentro de la red, alcanzando de esta manera un buen nivel de productividad en comparación con el FTA.

El análisis de degradación de prestaciones ha mostrado que las topologías FT-RUFT-212, FT-RUFT-222 y FT-RUFT-XL son capaces de mantener las prestaciones bastante estables ante un alto número de fallos en la red. Por otra parte, en RUFT-PL la degradación de prestaciones es mayor, debido a que la

topología no puede hacer frente a una alta carga de tráfico ante un elevado número de fallos.

En el último análisis presentado en este capítulo podemos ver que nuestras topologías requieren un coste de diseño y una complejidad menor o similar a la de FTA. FT-RUFT-212, gracias a los conmutadores asimétricos de la primera y última etapa, mantiene un coste de diseño menor al de FTA, ya que todos los conmutadores de las etapas intermedias no se ven modificados con respecto a la topología RUFT. RUFT-PL, FT-RUFT-222 y FT-RUFT-XL usan conmutadores simétricos en toda la red, y en comparación con el FTA solo requieren un ligero incremento en la complejidad de sus conmutadores, permitiendo a éstas obtener hasta el doble de productividad frente al FTA.

Capítulo 5

Conclusiones

"Mientras los hombres sean libres para preguntar lo que deben; libres para decir lo que piensan; libres para pensar lo que quieran; la libertad nunca se perderá y la ciencia nunca retrocederá."

Robert Oppenheimer

En el presente capítulo, el cual marca el final de esta tesis, enumeramos las principales conclusiones y aportaciones derivadas del trabajo realizado en esta tesis doctoral. Posteriormente, se describen las posibles líneas de trabajo futuro. Finalmente, se presentan las publicaciones que han sido fruto de la presente investigación.

5.1. Conclusiones

En esta tesis nos hemos centrado en el desarrollo de nuevas topologías para redes de interconexión, las cuales ofrecen una alta productividad a la vez que proporcionan un buen nivel de tolerancia a fallos, manteniendo un coste razonable.

Para desarrollar este trabajo hemos partido de RUFT, una topología de red que reduce a más de la mitad la complejidad de los conmutadores utilizados por la topología fat-tree con encaminamiento adaptativo (FTA). Las prestaciones que ofrece RUFT son muy similares a las de FTA. Sin embargo, su principal

inconveniente es que no ofrece ningún grado de tolerancia a fallos. Este es el punto débil de la topología RUFT.

Como primera medida para dotar a RUFT de tolerancia a fallos se propuso la topología RUFT-PL (RUFT with Parallel Links), la cual emplea el mismo número de conmutadores que la topología fat-tree, pero sin reducir su complejidad. RUFT-PL, al igual que RUFT, usa enlaces unidireccionales, pero en este caso los enlaces se disponen en paralelo, es decir, cada elemento de la red esta unido por dos enlaces paralelos. Estos enlaces permiten que RUFT-PL soporte un fallo de enlace en cualquier punto de la red. Este simple cambio en el diseño de RUFT permite incrementar notablemente la productividad ofrecida por el fat-tree, además de ofrecer un ligero grado de tolerancia a fallos.

A continuación, se desarrolló una nueva topología de red denominada FT-RUFT-212. Esta topología implementa dobles enlaces en la inyección, un enlace en la red para interconectar los conmutadores y dobles enlaces en la eyección. FT-RUFT-212 requiere conmutadores asimétricos en la primera y última etapa, mientras que los conmutadores de las etapas intermedias no requieren ninguna modificación con respecto a RUFT. A diferencia de RUFT-PL, que utiliza sus enlaces en paralelo para conectar los nodos de procesamiento con los conmutadores, en FT-RUFT-212 se ha modificado la conexión de los enlaces de inyección/eyección para ofrecer rutas alternas dentro de la red. Este nuevo diseño ofrece hasta cuatro rutas para cada par origen-destino, las cuales son lo más disjuntas posibles, incrementando de esta manera el nivel de tolerancia a fallos para soportar hasta tres fallos de enlace. A pesar de que esta topología requiere pocos recursos hardware en sus conmutadores, las prestaciones ofrecidas por ésta son mejores que las de fat-tree.

Teniendo en cuenta la productividad ofrecida por RUFT-PL y las propiedades de tolerancia a fallos de FT-RUFT-212, se desarrolló una nueva topología denominada FT-RUFT-222, la cual combina las buenas características de las dos topologías anteriores. Esta topología emplea conmutadores simétricos en toda la red y usa dobles enlaces, al igual que RUFT-PL, pero siguiendo el patrón de conexión utilizado por FT-RUFT-212 para conectar los nodos de procesamiento a los conmutadores de la red. FT-RUFT-222 alcanza un alto nivel de productividad en la red e incrementa el nivel de tolerancia a fallos de las propuestas anteriores, ya que para cada par origen-destino existen ocho

rutas (cuatro rutas dobles). Tanto FT-RUFT-212 como FT-RUFT-222, gracias a las rutas disjuntas proporcionadas por la topología, soportan el fallo de conmutadores en cualquier punto de la red, a diferencia de la topología fat-tree, que no soporta ningún fallo en aquellos conmutadores que comunican con los nodos de procesamiento.

La topología FT-RUFT-222 ofrece una alta productividad y un buen nivel de tolerancia a fallos. Sin embargo, el par de enlaces de red siempre conecta con el mismo conmutador, y esto se puede convertir en un inconveniente en el momento que uno de estos elementos falle, ya que estaría sacrificando el par de rutas asociadas a dicho conmutador. Por ello, para incrementar aún más el nivel de tolerancia a fallos y mantener una alta productividad desarrollamos FT-RUFT-XL, una topología que utiliza conmutadores simétricos en toda la red, al igual que RUFT-PL y FT-RUFT-222. En FT-RUFT-XL también se usan dobles enlaces en cada punto de la red, pero se ha modificado el patrón de conexión de los enlaces de red, inyección y eyección. En particular, utilizamos dos topologías dentro de la misma red (una en los enlaces primarios y otra en los enlaces secundarios), las cuales comparten el mismo encaminamiento, creando diversas rutas disjuntas dentro de la red. Este nuevo diseño eleva considerablemente el nivel de tolerancia a fallos a nivel de enlace y de conmutador.

Aunque cada una de las topologías implementadas en este trabajo ofrece tolerancia a fallos, es necesario utilizar un mecanismo adecuado que actúe en el momento que se detecta un fallo en el sistema. Los mecanismos de encaminamiento con tolerancia a fallos están basados en el uso de tablas y de intervalos de encaminamiento. Sin embargo, las tablas requieren demasiada memoria e introducen un retardo en el encaminamiento derivado de la búsqueda en una tabla de gran tamaño para determinar el puerto de salida que debe tomar un paquete. Por otra parte, los intervalos de encaminamiento no son adecuados cuando se consideran los fallos en los enlaces de inyección, ya que requieren un alto número de registros para poder soportar este tipo de fallos. Por este motivo, hemos desarrollado un mecanismo de tolerancia a fallos estático basado en un vector de estados a nivel de bit que reduce considerablemente la memoria requerida por las tablas, a la vez que ofrece un alto grado de detalle del estado de la red.

Resumiendo, en esta tesis hemos presentado una nueva familia de topologías indirectas, eficientes y tolerantes a fallos, compuesta por cuatro topologías. También hemos presentado un mecanismo de tolerancia a fallos que permite conocer el estado de toda la red, el cual requiere poca memoria para ser implementado. En general, hemos demostrado que es posible desarrollar nuevas topologías de red que ofrezcan una alta productividad de red y un alto nivel de tolerancia a fallos utilizando pocos recursos hardware en los conmutadores.

5.2. Trabajo futuro

El estudio de las redes de interconexión para sistemas de cómputo de alto rendimiento, y en especial de la topología fat-tree, ha abierto muchas oportunidades en el campo de investigación. Una de estas líneas de investigación es la que hemos plasmado en esta tesis.

A pesar del tiempo que se ha dedicado a este trabajo siguen surgiendo algunos retos, los cuales son importantes y que orientan algunas líneas de trabajo futuro. A continuación se mencionan algunas de ellas:

- Las topologías FT-RUFT-212, FT-RUFT-222 y FT-RUFT-XL suelen modificar el comportamiento de los patrones de tráfico dentro de la red. Ante este cambio, es de gran interés optimizar el control de flujo para reducir la congestión ante patrones de tráfico especiales.

- El mecanismo de tolerancia a fallos basado en un vector de estados a nivel de bit debe ser adaptado para que actúe con un modelo de fallos dinámico y de este modo no sea necesario detener la actividad de la red ante la presencia de fallos.

- Nuestra propuesta se ha basado en duplicar el número de enlaces de la red, cambiando asimismo el patrón de interconexión en algunos casos. Sería factible diseñar nuevas topologías que replicaran más (por ejemplo triplicar o cuadruplicar) el número de enlaces, variando asimismo por separado el número de enlaces replicados en la inyección/eyección y en la red.

- El mecanismo de tolerancia a fallos propuesto puede ser extendido a otras topologías MIN que utilicen encaminamiento determinista.

- Las topologías propuestas en este trabajo ofrecen una alta productividad y un buen nivel de tolerancia a fallos. Sin embargo, debemos tener en cuenta que no todos los conmutadores que encontramos en el mercado tienen un número de puertos potencia de 2. Por ello, sería importante extender estas topologías a conmutadores de diferente aridad, es decir, implementar *real-life-fat-trees*.

5.3. Contribuciones

A lo largo de este periodo de investigación, hemos tenido la oportunidad de publicar algunos trabajos relacionados con esta tesis en diferentes congresos y revistas. A continuación se citan las publicaciones relacionadas con esta tesis:

La publicación que describe la propuesta RUFT-PL es [15]:

- D. Bermúdez Garzón, C. Gómez, M.E. Gómez, P. López, and J. Duato. *"Towards an Efficient Fat-Tree Like Topology"*. In Christos Kaklamanis, Theodore Papatheodorou, and Paul G. Spirakis, editors, Euro-Par 2012 Parallel Processing, volume 7484 of Lecture Notes in Computer Science, pages 716-728. Springer Berlin Heidelberg, 2012.

Las publicaciones relacionadas con las propuestas FT-RUFT-212 y FT-RUFT-222 son [16] y [17]:

- D. Bermúdez Garzón, C. Gómez, M.E. Gómez, P. López, and J. Duato. *"FT-RUFT: A Performance and Fault-Tolerant Efficient Indirect Topology"*. In Proceedings of the 22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pages 405-409, Feb 2014.

- D. Bermúdez Garzón, C. Gómez, M.E. Gómez, P. López, and J. Duato. *"A Family of Fault-Tolerant Efficient Indirect Topologies"*. In IEEE Transactions on Parallel and Distributed Systems (TPDS), 2015.

La herramienta desarrollada para evaluar el nivel de tolerancia a fallos de las topologías ha sido presentada en [18]:

- D. Bermúdez Garzón, C. Gómez, P. López, M.E. Gómez. *"Speeding-Up the Fault-Tolerance Analysis of Interconnection Networks"*. International Conference on High Performance Computing & Simulation, Amsterdam, 2015.

Bibliografía

[1] Amazon web services. http://aws.amazon.com/es/hpc/.

[2] Arquitectura InfiniBand Vol. 1, Especificaciones Generales. `https://cw.infinibandta.org/document/dl/7143`.

[3] Extoll GmbH. http://www.extoll.de/.

[4] Grupo de Arquitecturas Paralelas de la Universidad Politécnica de Valencia. http://www.gap.upv.es/.

[5] Mellanox technologies. http://www.mellanox.com/.

[6] Myricom®. `https://www.myricom.com/`.

[7] Nasa supercomputer. http://www.nas.nasa.gov/hecc/resources/pleiades.html.

[8] National university of defense technology. http://english.nudt.edu.cn/.

[9] Sabalcore hpc. http://www.sabalcore.com/index.html.

[10] Top500 supercomputers site. http://www.top500.org.

[11] N.R. Adiga, M.A Blumrich, D. Chen, P. Coteus, A Gara, M.E. Giampapa, P. Heidelberger, S. Singh, B.D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas. Blue Gene/L torus interconnection network. *IBM Journal of Research and Development*, 49(2.3):265–276, March 2005.

[12] R.R. Aggarwal and L. Kaur. Fault-Tolerance and Permutation Analysis of ASEN and its Variant. *Intl. Journal of Computer Science and Information Technologies*, 1(1):24–32, 2010.

[13] E. Bakker, J.V. Leeuwer, and R.B. Tan. Linear Interval Routing. *Tech. Rep. RUU-CS-91-7, Dept. of Computer Science, Utrecht University (1991). Also in: Algorithms review 2*, 2:45–61, 1991.

[14] S.M. Bataineh and B.Y. Allosl. Fault-Tolerant Multistage Interconnection Network. *Telecommunication Systems*, 17(4):455–472, 2001.

[15] D. Bermúdez, C. Gómez, M.E. Gómez, P. López, and J. Duato. Towards an Efficient Fat-Tree like Topology. In Christos Kaklamanis, Theodore Papatheodorou, and P. Spirakis, editors, *Euro-Par 2012 Parallel Processing*, volume 7484 of *Lecture Notes in Computer Science*, pages 716–728. Springer Berlin Heidelberg, 2012.

[16] D. Bermúdez, C. Gómez, M.E. Gómez, P. López, and J. Duato. FT-RUFT: A Performance and Fault-Tolerant Efficient Indirect Topology. In *Proceedings of the 22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 405–409, Feb 2014.

[17] D. Bermúdez, C. Gómez, M.E. Gómez, P. López, and J. Duato. A Family of Fault-Tolerant Efficient Indirect Topologies. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2015.

[18] D. Bermúdez, C. Gómez, P. López, and M.E. Gómez. Speeding-up the Fault-Tolerance Analysis of Interconnection Networks. *International Conference on High Performance Computing & Simulation, Amsterdam, HPCS*, 2015.

[19] V.P. Bhardwaj and N. Nitin. A New Fault Tolerant Routing Algorithm for Advance Irregular Augmented Shuffle Exchange Network. In *Proceedings of the 14th International Conference on Computer Modelling and Simulation (UKSim)*, pages 505–509, March 2012.

[20] F. Bistouni and M. Jahanshahi. Pars Network: A Multistage Interconnection Network with Fault-Tolerance Capability. *Journal of Parallel and Distributed Computing*, 75(0):168 – 183, 2015.

[21] R. Casado, A. Bermúdez, J. Duato, F.J. Quiles, and J.L. Sanchez. A Protocol for Deadlock-Free Dynamic Reconfiguration in High-Speed Local Area Networks. *IEEE Transactions on Parallel and Distributed Systems*, February 2001.

[22] C.W. Chen and C.P. Chung. Designing a Disjoint Paths Interconnection Network with Fault Tolerance and Collision Solving. *The Journal of Supercomputing*, 34(1):63–80, 2005.

[23] C.W. Chen, P.S. Gan, and C.H. Chang. Designing a High Performance and Fault Tolerant Multistage Interconnection Network with Easy Dynamic Rerouting. In Jiannong Cao, LaurenceT. Yang, Minyi Guo, and Francis Lau, editors, *Parallel and Distributed Processing and Applications*, volume 3358 of *Lecture Notes in Computer Science*, pages 1007–1016. Springer Berlin Heidelberg, 2005.

[24] C.W. Chen, N. P. Lu, T.F. Chen, and C.P. Chung. Fault-Tolerant Gamma Interconnection Networks by Chaining. *IEE Proceedings Computers and Digital Techniques*, 147(2):75–81, March 2000.

[25] F.T. Chong and T.F. Knight, Jr. Design and Performance of Multipath MIN Architectures. In *Proceedings of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '92, pages 286–295, New York, NY, USA, 1992. ACM.

[26] P.J. Chuang. CGIN: A Fault Tolerant Modified Gamma Interconnection Network. *IEEE Transactions on Parallel and Distributed Systems*, 7(12):1301–1306, December 1996.

[27] D. Culler, J.P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[28] W. Dally. *VLSI and parallel computation*, chapter Network and Processor Architecture for Message-Driven Computers, pages 140–222. Morgan Kaufmann, San Mateo, CA, USA, 1991.

[29] W. Dally. Virtual-Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, March 1992.

[30] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[31] J. Dongarra. Visit to the National University for Defense Technology Changsha, China - [White Paper]. Technical report, National University of Defense Technology (NUDT), June 2003.

[32] J. Duato, S. Yalamanchili, and N. Lionel. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[33] C.C. Fan and J. Bruck. Optimal Constructions of Fault-Tolerant Multistage Interconnection Networks. *Tech. Rep. Parallel and Distributed Systems Group. California Institute of Technology*, 1996.

[34] P. Fraigniaud and C. Gavoille. Optimal Interval Routing. In Bruno Buchberger and Jens Volkert, editors, *CONPAR*, volume 854 of *Lecture Notes in Computer Science*, pages 785–796. Springer, 1994.

[35] P.T. Gaughan and S. Yalamanchili. Adaptive Routing Protocols for Hypercube Interconnection Networks. *Computer*, 26(5):12–23, May 1993.

[36] I. Gazit and M. Malek. Fault Tolerance Capabilities in Multistage Network-Based Multicomputer Systems. *Computers, IEEE Transactions on*, 37(7):788–798, July 1988.

[37] F. Gilabert, M.E. Gómez, P. López, and J. Duato. On the Influence of the Selection Function on the Performance of Fat-Trees. In Wolfgang E. Nagel, Wolfgang V. Walter, and Wolfgang Lehner, editors, *Euro-Par 2006 Parallel Processing*, volume 4128 of *Lecture Notes in Computer Science*, pages 864–873. Springer Berlin Heidelberg, 2006.

[38] C. Gómez, F. Gilabert, M.E. Gómez, P. López, and J. Duato. Deterministic versus Adaptive Routing in Fat-Trees. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–8, March 2007.

[39] C. Gómez, F. Gilabert, M.E. Gómez, P. López, and J. Duato. RUFT: Simplifying the Fat-Tree Topology. In *Proceedings of the 14th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pages 153–160, December 2008.

[40] C. Gómez, M.E. Gómez, P.J. López, and J.F. Duato. FT^2EI: A Dynamic Fault-Tolerant Routing Methodology for Fat Trees with Exclusion Intervals. *IEEE Transactions on Parallel and Distributed Systems*, 20(6):802–817, June 2009.

[41] M.E. Gómez, P. López, and J. Duato. A Memory-Effective Routing Strategy for Regular Interconnection Networks. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 41b–41b, April 2005.

[42] R.I. Greenberg and C.E. Leiserson. Randomized Routing on Fat-Trees. In *Advances in Computing Research*, pages 345–374. JAI Press, 1996.

[43] C.T. Ho and L. Stockmeyer. A New Approach to Fault-Tolerant Wormhole Routing for Mesh-Connected Parallel Computers. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS)*, pages 427–438, October 2002.

[44] N. Kamiura, T. Kodera, and N. Matsui. Design of a Fault Tolerant Multistage Interconnection Network with Parallel Duplicated Switches. In *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 143–151, 2000.

[45] S. Konstantinidou. The Selective Extra-Stage Butterfly. In *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD)*, pages 502–506, October 1992.

[46] J.V. Leeuwen and R.B. Tan. Routing with Compact Routing Tables. *Tech. Rep. RUU-CS-83-16, Dept. of Computer Science, Utrecht University (1983). Also as: Computer Network with Compact Routing Tables, in G: Rozenberg and A. Salomaa (Eds). The Book of L, Springer-Verlag, Berlín (1986)*, pages 298–307, 1986.

[47] C.E. Leiserson. Fat-trees: Universal Networks for Hardware-Efficient Supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, October 1985.

[48] X.Y. Lin, Y.C. Chung, and T.Y. Huang. A Multiple LID Routing Scheme for Fat-Tree-Based InfiniBand Networks. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, pages 11–, Los Alamitos, CA, USA, April 2004.

[49] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson. F10: A Fault-Tolerant Engineered Network. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, NSDI'13, pages 399–412, Berkeley, CA, USA, 2013.

[50] M.M.K. Martin, M.D. Hill, and D.A. Wood. Token Coherence: Decoupling Performance and Correctness. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 182–193, June 2003.

[51] D.S. Parker and C.S. Raghavendra. The Gamma Network. *IEEE Transactions on Computers*, C-33(4):367–373, April 1984.

[52] F. Petrini and M. Vanneschi. k-ary n-trees: High Performance Networks for Massively Parallel Architectures. In *Proceedings of the 11th International Parallel Processing Symposium*, pages 87–93, April 1997.

[53] R. Rastogi, Nitin, and D.S. Chauhan. 3-Disjoint Paths Fault-Tolerant Omega Multi-stage Interconnection Network with Reachable Sets and Coloring Scheme. In *Proceedings of the 13th International Conference on Computer Modelling and Simulation (UKSim)*, pages 551–556, March 2011.

[54] R. Rastogi, R. Verma, Nitin, and D. Chauhan. 3-Disjoint Paths Fault-Tolerant Multi-Stage Interconnection Networks. In Ajith Abraham, Jaime Lloret Mauri, JohnF. Buford, Junichi Suzuki, and SabuM. Thampi, editors, *Advances in Computing and Communications*, volume 190 of *Communications in Computer and Information Science*, pages 21–33. Springer Berlin Heidelberg, 2011.

[55] D.A. Reed and D.C. Grunwald. The Performance of Multicomputer Interconnection Networks. *Computer*, 20(6):63–73, June 1987.

[56] N. Santoro and R. Khatib. Routing without routing tables. *Tech. report SCS-TR-6, School of Computer Science, Carleton University*, 1982.

[57] S.L. Scott and G. Thorson. Optimized Routing in the Cray T3D. In *Proceedings of the First International Workshop on Parallel Computer Routing and Communication*, PCRCW '94, pages 281–294, London, UK, UK, 1994. Springer-Verlag.

[58] F.O. Sem-Jacobsen, T. Skeie, O. Lysne, and J. Duato. Dynamic Fault Tolerance in Fat Trees. *IEEE Transactions on Computers*, 60(4):508–525, April 2011.

[59] F.O. Sem-Jacobsen, T. Skeie, O. Lysne, O. Toerudbakken, E. Rongved, and B. Johnsen. Siamese-Twin: A Dynamically Fault-Tolerant Fat-Tree. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, pages 100b–100b, April 2005.

[60] J. Sengupta and P. K. Bansal. Fault-Tolerant Routing in Irregular MINs. In *TENCON '98. 1998 IEEE Region 10 International Conference on Global Connectivity in Energy, Computer, Communication and Control*, volume 2, pages 638–641, 1998.

[61] H.J. Siegel. Analysis Techniques for SIMD Machine Interconnection Networks and the Effects of Processor Address Masks. *IEEE Transactions on Computers*, February 1977.

[62] C.B. Stunkel, D.G. Shea, B. Abali, M.G. Atkins, C.A. Bender, D.G. Grice, P. Hochschild, D.J. Joseph, B.J. Nathanson, R.A. Swetz, R.F. Stucke, M. Tsao, and P.R. Varker. The SP2 High-Performance Switch. *IBM Systems Journal*, 34(2):185–204, 1995.

[63] C.B. Stunkel, D.G. Shea, D.G. Grice, P.H. Hochschild, and M. Tsao. The SP1 High-Performance Switch. In *Proceedings of the Scalable High-Performance Computing Conference*, pages 150–157, May 1994.

[64] R. Suzuki, S. Fukurnoto, and K. Iwasaki. Adaptive checkpointing for Time Warp Technique with a Limited Number of Checkpoints. In *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops*, pages 95–100, 2002.

[65] M. Valerio, L.E. Moser, and P.M. Melliar-Smith. Fault-Tolerant Orthogonal Fat-Trees as Interconnection Networks. In *Proceedings of the IEEE First International Conference on Algorithms and Architectures for Parallel Processing (ICAPP)*, volume 2, pages 749–754, April 1995.

[66] S. Wei and G. Lee. Extra Group Network: A Cost-Effective Fault-Tolerant Multistage Interconnection Network. In *Proceedings of the 15th Annual International Symposium on Computer Architecture*, volume 16, pages 108–115, May 1988.

[67] G. Zarza, D. Lugones, D. Franco, and E. Luque. A Multipath Fault-Tolerant Routing Method for High-Speed Interconnection Networks. In Henk Sips, Dick Epema, and Hai-Xiang Lin, editors, *Euro-Par 2009 Parallel Processing*, volume 5704 of *Lecture Notes in Computer Science*, pages 1078–1088. Springer Berlin Heidelberg, 2009.