UNIVERSIDAD POLITÉCNICA DE VALENCIA

DEPARTAMENTO DE
INFORMÁTICA DE SISTEMAS Y COMPUTADORES

Mejora de las prestaciones de las redes de estaciones de trabajo con encaminamiento fuente

Memoria que para optar al grado de
Doctor en Informática presenta
José Flich Cardo

Dirigida por
Pedro Juan López Rodríguez
Manuel Pérez Malumbres

Valencia, Enero de 2001

Dedicatoria

A Maria Amparo, la meua dona

Agradecimientos

A mis directores de tesis, Pedro López y Manuel Pérez, por su constante apoyo e ilusión en este trabajo. En especial, a Pedro por la confianza que siempre me ha transmitido. Con él empecé en la investigación y gracias a él disfruto de mi trabajo. A Manolo, por su constante aportación de ideas y ánimos en el trabajo.

Un agradecimiento muy especial a José Duato. Sin sus ideas, ánimos y confianza este trabajo no hubiera existido. Con él he tenido la oportunidad de incrementar mis conocimientos, conocer nuevos enfoques y aprender a superarme profesionalmente. A los tres, gracias por confiar en mí desde el principio.

A Thomas Rokicki por sus ideas geniales que han impregnado esta tesis.

A los compañeros del grupo de arquitecturas paralelas (GAP), en especial a Juan Miguel, Antonio, Elvira, Fede, José Carlos, Juan Carlos y José por el apoyo que me han prestado desinteresadamente siempre que lo he necesitado.

A los compañeros y amigos del departamento, en especial a Juan Carlos, Julio, Félix y Teresa, por sus ánimos e interés por el trabajo. Por los buenos ratos que ayudan a seguir trabajando.

A Salvador Coll por su enorme capacidad de trabajo y desinterés en ayudar. Gracias por todo el trabajo, en especial por la implementación realizada de la principal idea de esta tesis.

A mi familia, en especial a mis padres, Miguel y Encarna, por su constante apoyo en mi carrera profesional.

A mi mujer, María Amparo, por su infinita paciencia y comprensión. Sin ella no habría podido terminar este trabajo.

Por último, al disco de *mesh*, por aguantar hasta el día de hoy. Cuando quieras ya puedes descansar en paz.

Gracias a todos.

Índice General

Índice de Tablas

Índice de Figuras

Objetivos de la tesis

En los últimos años, las redes de estaciones de trabajo (también denominadas *clusters*) se han convertido en una alternativa a los computadores paralelos para conseguir elevadas potencias de cálculo. Esto ha sido posible gracias al creciente aumento de las prestaciones y capacidades de los computadores personales y estaciones de trabajo y, sobre todo, a la aparición de redes de interconexión de altas prestaciones que permiten la interconexión de los elementos de procesamiento, proporcionando el ancho de banda y la reducida latencia necesarios para poder afrontar los requerimientos de las aplicaciones con elevadas demandas en comunicación.

Estas redes utilizan diferentes mecanismos y técnicas para enviar los mensajes desde los nodos fuente a los nodos destino. La transmisión de los mensajes puede realizarse bien utilizando encaminamiento distribuido o bien utilizando encaminamiento fuente. En el encaminamiento distribuido, el mensaje contiene el identificador del destino, y los conmutadores encaminan el mensaje en función del destino y del puerto de entrada. Los conmutadores poseen unas tablas internas de encaminamiento que indican el puerto de salida a partir del destino del mensaje y del puerto de entrada. Ejemplos de redes con este tipo de encaminamiento son ServerNet [Rob96] y ServerNet II [Dav97]. Por otro lado, en el encaminamiento fuente, toda la ruta desde el nodo origen al nodo destino viaja junto al mensaje y los conmutadores sólo se limitan a encaminar el mensaje por la ruta indicada en la cabecera del mensaje. Este tipo de encaminamiento tiene la principal ventaja de que el tiempo utilizado por los conmutadores para encaminar los mensajes es menor. No obstante, el encaminamiento distribuido permite cierta flexibilidad de encaminamiento ausente en el encaminamiento fuente. Un ejemplo de red con encaminamiento fuente es Myrinet [Bod95].

Precisamente, la red Myrinet ha obtenido una gran popularidad en los últimos años debido, entre otras razones, a que posee un procesador específico en el interfaz

de red que ejecuta un programa de control (denominado MCP) que gobierna dicho interfaz en particular y toda la red en general. El código fuente del MCP se encuentra disponible, por lo que numerosos grupos de investigación han adoptado la red Myrinet para diseñar y evaluar nuevas propuestas de encaminamiento.

En los últimos años se ha realizado un profundo análisis de las redes con encaminamiento distribuido. En particular, se ha obtenido que el algoritmo de encaminamiento tradicionalmente utilizado (el algoritmo de encaminamiento *up*/down**) posee ciertas limitaciones (desequilibrado del tráfico, utilización de rutas no mínimas, etc.) que limitan la productividad de la red. Estos estudios han comprobado que la utilización de algoritmos adaptativos permite un aumento considerable de las prestaciones de la red.

Sin embargo, poco esfuerzo se ha invertido para aumentar las prestaciones de las redes con encaminamiento fuente. En este tipo de redes, la utilización de algoritmos de encaminamiento adaptativos no es viable debido a que los conmutadores no pueden reencaminar los mensajes. Por lo tanto, la única vía para mejorar las prestaciones de estas redes ha sido la propuesta de nuevos algoritmos de encaminamiento deterministas. Algunas propuestas en esta línea son el algoritmo de encaminamiento *smart-routing* [Che95] y, más recientemente, el algoritmo de encaminamiento *DFS* [San00].

Por lo tanto, teniendo en cuenta la importancia comercial de las redes con encaminamiento fuente (en especial Myrinet) y observando las escasas propuestas que se han realizado para mejorar sus prestaciones, el objetivo de esta tesis es el de mejorar las prestaciones de las redes con encaminamiento fuente en general y de la red Myrinet en particular. En detalle, los objetivos específicos de la tesis son los siguientes:

1. Identificación de los principales problemas de las redes con encaminamiento fuente que limitan sus prestaciones. Se detectarán las principales causas que impiden que los algoritmos de encaminamiento propuestos hasta el momento para las redes con encaminamiento fuente obtengan elevadas prestaciones.

2. Propuesta de unos mecanismos que eliminen dichos problemas y, por consiguiente, que ayuden a mejorar las prestaciones. Además, dichos mecanismos serán sencillos. Para ello, y teniendo en mente la red Myrinet, se diseñará una implementación *software* compatible con las soluciones existentes. En concreto, las soluciones que se proponen en esta tesis se ubicarán en el MCP de

Myrinet, siendo las siguientes:

- Mecanismo de selección de rutas alternativas. Entre un par origen-destino típicamente existen varias rutas alternativas válidas. Con un mecanismo de selección de rutas en el nodo origen se facilitará cierta adaptatividad a la red, ya que el tráfico generado hacia cualquier destino se repartirá entre las diferentes rutas. Se propondrán diferentes políticas de selección de rutas.
- Mecanismo de detección de contención. Se implementará un mecanismo ubicado en el nodo fuente que, utilizando únicamente información local al nodo, obtendrá una estimación del grado de contención de una ruta. Dicho mecanismo será utilizado junto con el mecanismo de selección de rutas previo para añadir un cierto grado más de adaptatividad.
- Mecanismo de eliminación de dependencias. Se implementará un mecanismo (que hemos denominado *in-transit buffers*) que eliminará las dependencias prohibidas de los algoritmos de encaminamiento. Básicamente, con dicho mecanismo, se absorberán algunos mensajes a lo largo de su recorrido para después reinyectarlos nuevamente a la red. Con esta absorción/reinyección se eliminarán dependencias entre canales prohibidas por algunos algoritmos de encaminamiento tradicionales, como *up*/down** y *DFS*.
- Diseño de nuevos algoritmos de encaminamiento de ruta mínima. Utilizando el mecanismo *in-transit buffers* se diseñarán nuevos algoritmos de encaminamiento que utilizarán rutas mínimas para todos los pares origen-destino. También, dichos algoritmos de encaminamiento (gracias a las propiedades del mecanismo) obtendrán un buen equilibrado de tráfico y una reducción en la contención de red.

3. Evaluación de las prestaciones obtenidas con los mecanismos propuestos. Se analizará, empleando un modelo de simulación, el comportamiento de la red de interconexión con los nuevos mecanismos, comparándolos con las soluciones previas ya existentes para estas redes. El modelo de simulación empleado deberá ser lo más realista posible tanto en el modelo de la red como en la definición de las nuevas técnicas propuestas, con el fin de obtener resultados fiables.

Desarrollo de la tesis

Para cubrir estos objetivos, esta memoria se ha estructurado en seis capítulos. El capítulo 1 justifica el interés de las redes de estaciones de trabajo, así como hace hincapié en la importancia de la obtención de elevadas prestaciones por parte de las redes de interconexión. Asimismo, el capítulo hace un recorrido por las técnicas y mecanismos que se utilizan en las redes de interconexión, así como describe dos redes típicas: Myrinet y ServerNet. A continuación, identifica las principales limitaciones que se han detectado en las redes con encaminamiento fuente, al tiempo que presenta, de una forma general, los posibles mecanismos que se proponen para evitar dichas limitaciones, siendo estos mecanismos los objetivos principales de la tesis.

El capítulo 2 presenta el primero de los mecanismos propuestos, el cual se basa en la selección de una entre las múltiples rutas alternativas existentes para enviar mensajes hacia un destino. Adicionalmente, se presenta un mecanismo de detección de contención a lo largo de una ruta. En dicho capítulo, se especifican todas las combinaciones y parámetros que se van a considerar de ambos mecanismos.

El capítulo 3 describe el mecanismo principal propuesto en la tesis, denominado buffers en tránsito o *in-transit buffers* (ITB) . Con este mecanismo se eliminan las dependencias prohibidas por los algoritmos de encaminamiento tradicionales a la vez que permite la utilización de algoritmos de encaminamiento más agresivos. En concreto, en este capítulo se indican las diferentes metodologías utilizadas para aplicar el mecanismo sobre los algoritmos de encaminamiento *up*/down**, *DFS* y *smart-routing*. También se describen los nuevos algoritmos que han sido expresamente diseñados para aprovechar al máximo las posibilidades ofrecidas por el mecanismo. Por último, en este capítulo se describen las consideraciones a tener en cuenta para obtener una implementación eficiente del mecanismo en la red Myrinet.

El capítulo 4 presenta la metodología de evaluación utilizada, presentando diversos aspectos del simulador, como su funcionamiento, los parámetros de entrada,

los resultados que obtiene, etc.

En el capítulo 5 se realiza mediante simulación la evaluación de los nuevos algoritmos de encaminamiento obtenidos como resultado de la aplicación de cada uno de los mecanismos propuestos (y sus combinaciones). Se han tomando en consideración diversos valores de los parámetros de diseño de la red (topologías regulares, topologías irregulares, varios tamaños de red, etc.), y de la carga (varios niveles de tráfico, diversas distribuciones de destinos de los mensajes, etc.). También se analiza un problema originado por la utilización del mecanismo de buffers en tránsito, así como posibles soluciones al mismo. Por último, se realiza un análisis de sensibilidad del mecanismo de buffers en tránsito observando la influencia de diferentes parámetros de diseño sobre el comportamiento del mecanismo.

Finalmente, las conclusiones, aportaciones y líneas de trabajo futuras quedan recogidas en el capítulo 6.

En el apéndice A se incluye la implementación realizada del mecanismo de buffers en tránsito en Myrinet sobre el sistema de paso de mensajes GM.

Capítulo 1

Redes de estaciones de trabajo

1.1 Introducción

Durante la última década, los computadores han experimentado un gran incremento en sus prestaciones y capacidades. El motivo principal de estos incrementos ha sido la evolución de la tecnología VLSI lo que permite el incremento de las frecuencias de reloj y el aumento en el número de componentes en un mismo chip.

A medida que aparecen computadores con mayores capacidades de cómputo, aparecen a su vez aplicaciones más complejas que demandan más potencia de cálculo. Sin embargo, el incremento en las prestaciones de los computadores tiene un límite físico.

Por lo tanto, la alternativa natural a dichos computadores para proveer de mayores potencias de cálculo es la de utilizar múltiples procesadores en un sistema, coordinándolos para que solucionen el problema de una forma concurrente. Estos sistemas son conocidos como computadores paralelos [Cul99]. En estos sistemas, una tarea es dividida en diferentes subtareas que serán solucionadas en paralelo por todos los procesadores del sistema.

Diferentes arquitecturas de computadores paralelos han sido propuestas durante las últimas décadas, existiendo multitud de implementaciones. Con los computadores paralelos se alcanza la potencia de cálculo requerida para solucionar problemas de gran coste computacional, como pueden ser el modelado global del cambio climático durante largos periodos de tiempo, la evolución de las galaxias, la estructura atómica de los materiales, la eficiencia de la combustión de un motor, el flujo del aire sobre superficies de vehículos, etc.

Sin embargo, estos sistemas tienen una serie de inconvenientes que cabe resaltar:

- **Poco volumen de ventas**. Los computadores paralelos son de gran ayuda en la ejecución de aplicaciones altamente paralelizables. Sin embargo, estas aplicaciones son un conjunto reducido de las necesidades de los usuarios. Existe un gran número de aplicaciones interactivas que no son fácilmente paralelizables e incluso que no es necesaria su paralelización. Por lo tanto, tan sólo un conjunto reducido de aplicaciones obtienen el mejor partido de estos sistemas, por lo que su mercado es reducido.

- **Elevado coste de mantenimiento por nodo**. Los computadores paralelos tienen un elevado coste de mantenimiento. Estos sistemas están construidos a partir de componentes caros, debido a su reducido volumen de ventas. Además, el personal encargado de mantenimiento de un computador paralelo debe estar altamente cualificado, lo cual también incrementa el coste del mantenimiento.

- **Mayor esfuerzo en ingeniería**. El diseño de un sistema paralelo requiere de un mayor esfuerzo en ingeniería que redunda en un mayor retardo en el desarrollo de nuevos sistemas.

- **Mayor coste de *software***. Al tener un reducido mercado de ventas, el *software* de estos sistemas también tiene un reducido mercado lo que conlleva a unos elevados costes.

Estos inconvenientes limitan el uso de estos sistemas, quedando relegados a un sector pequeño y específico del mercado.

1.1.1 Clusters

Por otra parte, las estaciones de trabajo[1] están obteniendo elevadas potencias de cálculo en los últimos años. Las prestaciones de estos sistemas se duplica cada 18 o 24 meses [Buy99]. Además, estos sistemas tienen un reducido coste, decrementándose el ratio precio/prestaciones en un 80% cada año, mientras que el de los computadores paralelos lo hace entre un 20% y un 30% [And87]. El motivo de este decremento en la relación precio/prestaciones de las estaciones de trabajo es el elevado volumen de ventas y la elevada producción de sus componentes.

[1] Con el término *estaciones de trabajo* englobamos tanto las estaciones de trabajo como los computadores personales.

Por otro lado, en los últimos años las redes de interconexión de estaciones de trabajo han evolucionado hacia las redes de medio dedicado basadas en conmutadores. Al igual que los primeros computadores paralelos tomaron prestada la tecnología de las redes de área local, esta nueva tecnología ha sido desarrollada a partir de las redes de multicomputadores y multiprocesadores. Ejemplos de estas redes son Myrinet [Bod95], ServerNet [Rob96], ServerNet II [Dav97] y SCI [SCI].

Debido a la evolución de las estaciones de trabajo y a la aparición de redes de interconexión de altas prestaciones, los *clusters* (redes de estaciones de trabajo) constituyen una alternativa de bajo coste a los computadores paralelos. Básicamente, un *cluster* [Buy99] es una colección de estaciones de trabajo y/o computadores personales interconectados por una red. Para propósitos de computación paralela, un *cluster* estará formado por estaciones de trabajo y redes de altas prestaciones. Los *clusters* son conocidos también por los términos de redes de estaciones de trabajo (*Network of Workstations*, NOWs) y *clusters* de estaciones de trabajo (*cluster of Workstations*, COWs). Los dos términos son sinónimos. En adelante, utilizaremos indistintamente los términos redes de estaciones de trabajo y *clusters*.

Ventajas frente a los computadores paralelos

Los *clusters* no deben ser vistos como una amenaza para los computadores paralelos. De hecho, no son el medio ideal para solucionar grandes problemas computacionales. El uso en la computación paralela de estos sistemas está en las aplicaciones de complejidad media. Para ello, los *clusters* aportan una serie de ventajas que los hacen más atractivos frente a los computadores paralelos. Algunas de sus ventajas son:

- Las estaciones de trabajo y computadores personales son baratos y tienen una elevada disponibilidad. Si una estación de trabajo se estropea, con un bajo coste es sustituida por una estación de trabajo de igual o mejores prestaciones.

- Los recursos en una red de estaciones de trabajo pueden crecer en número fácilmente. La capacidad del nodo puede ser ampliada añadiendo más memoria, actualizando el procesador o añadiendo incluso más procesadores. A su vez, más estaciones de trabajo pueden ser añadidas al sistema conectándolas a la red de interconexión, aumentando la potencia de cálculo global del sistema.

- Además, estos nuevos nodos añadidos son más fáciles de integrar en sistemas existentes que en los computadores paralelos. Los computadores paralelos suelen estar conectados formando una topología regular. Añadir unos pocos nodos más al sistema puede conllevar la alteración de la topología por lo que se incrementa su complejidad. Por otra parte, en los *clusters* pueden añadirse nodos al sistema debido a la posibilidad de formar topologías irregulares.

Componentes de un cluster

En la figura 1 podemos ver la arquitectura típica de un *cluster*. Un *cluster* es un sistema paralelo o distribuido formado por una colección de estaciones de trabajo, conectadas mediante una red trabajando conjuntamente como un único sistema integrado [Buy99]. Cada nodo puede estar formado por un computador con un sólo procesador (PC o estación de trabajo) o por un sistema multiprocesador simétrico (SMP). Cada nodo tiene su memoria y dispositivos de entrada/salida, ejecutando su propia versión de sistema operativo.

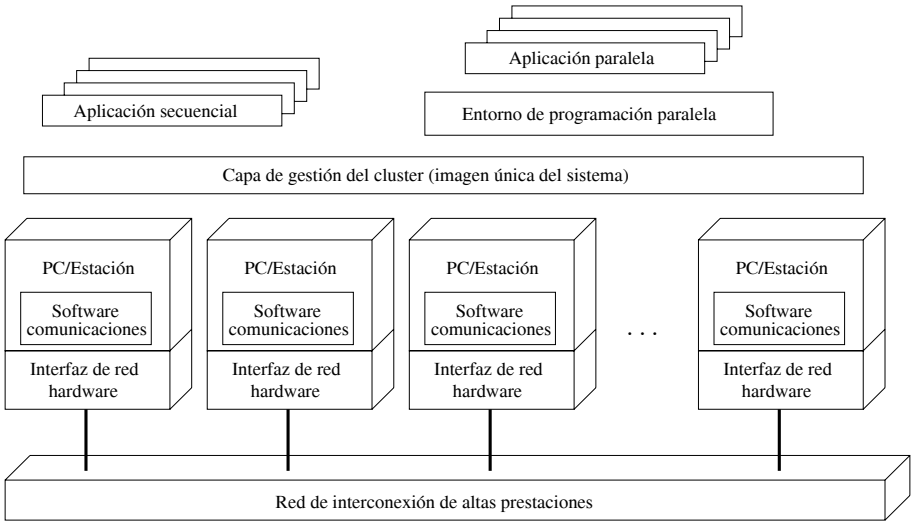


Figura 1: Arquitectura de un *cluster*.

La red de interconexión permite la comunicación entre todos los procesadores del *cluster*, siendo la responsable de transmitir los paquetes de datos entre los nodos. El interfaz de red debe proveer una interacción rápida y eficiente entre la red de interconexión y el computador local, siendo gestionado eficientemente por el *software*

de comunicaciones. En la actualidad están apareciendo procesadores de red ubicados en los interfaces de red que liberan al procesador principal de las tareas propias de la red. Tal es el caso del procesador RISC ubicado en los interfaces de red de Myricom [MYRI].

En algunos protocolos de comunicación se permite el acceso directo por parte del usuario al interfaz de red para evitar la sobrecarga del sistema operativo y las múltiples copias del mensaje a enviar/recibir. Tal es el caso de los protocolos de comunicación a nivel de usuario como BIP [Pry98], Fast Messages [Lau98] y VIA [VIA].

Los nodos del *cluster* pueden trabajar conjuntamente integrados como un sólo recurso o pueden trabajar individualmente. La capa intermedia del *cluster* (*cluster middleware*) es la responsable de ofrecer la ilusión de un sistema único (*single system image*). Esta capa también es la responsable de ofrecer una alta disponibilidad de los computadores conectados al sistema.

Los entornos de programación pueden ofrecer herramientas de programación de aplicaciones paralelas portables entre sistemas. Estos entornos incluyen librerías de paso de mensajes (como MPI [Mpi94] y PVM [Gei94]), depuradores, etc. No obstante, no hay que olvidar que los *clusters* también pueden ejecutar concurrentemente múltiples aplicaciones secuenciales de los usuarios.

1.1.2 Redes de interconexión de clusters

La red de interconexión es el componente más crítico en las prestaciones de un *cluster*. Las posibilidades y prestaciones de la red de interconexión influyen directamente sobre las prestaciones globales de todo el sistema.

Las redes de interconexión tradicionales (Ethernet [IEEE85], FDDI [Ross89]) no son una buena solución para construir un *cluster*. Estas tecnologías de red ofrecen latencias de transmisión que oscilan entre 0.1 mseg y 1 mseg, ofreciendo a su vez reducidos anchos de banda entre 1 y 100 Mb/seg. Una limitación adicional de estas redes es la fragmentación del ancho de banda entre todos los procesadores del sistema. Así, en una red Ethernet de 100 Mb/seg con 8 procesadores, un procesador puede obtener solamente un ancho de banda medio efectivo de 12.5 Mb/seg disponible para su uso. Por lo tanto, estas redes no permiten aumentar el número de procesadores en el *cluster* sin repercutir en las prestaciones globales del sistema. De hecho, la red de interconexión se convierte en el cuello de botella del sistema.

Con las nuevas tecnologías Ethernet (Ethernet 100 [IEEE95] y Gigabit Ethernet [She98]) se ha incrementado notablemente el ancho de banda debido, sobre todo, a la utilización de conmutadores de red. No obstante, las latencias de transmisión continúan siendo elevadas (en el rango de cientos de microsegundos).

En la actualidad están apareciendo nuevas tecnologías de redes de interconexión basadas en conmutadores con capacidades que varían de 1 a varios Gb/seg ofreciendo bajas latencias de transmisión. Ejemplos de estas redes son Myrinet [Bod95] y ServerNet [Rob96]. Estas prestaciones se obtienen gracias al empleo de los mecanismos de conmutación utilizados en las redes de interconexión de los computadores paralelos.

Además del ancho de banda ofrecido y de las reducidas latencias de transmisión, también existen otros factores que afectan en el diseño de estas redes de interconexión. Estos factores son:

- **Precio/Prestaciones**. Aún existen grandes diferencias en coste entre las redes con un ancho de banda moderado como Fast Ethernet (entre 50$ y 100$ para un adaptador de red) y redes de altas prestaciones como Myrinet y ServerNet (mas de 1000$). Esto es debido a los bajos volúmenes de producción de las redes de altas prestaciones.

- **Escalabilidad**. La escalabilidad es una propiedad crucial. Se refiere a la posibilidad de la red de crecer linealmente con el número de nodos sin perder las prestaciones ofrecidas. Las redes de interconexión en los computadores paralelos tradicionales tienen una topología de red fija (mallas, hipercubos). Por otra parte, los *clusters* son más dinámicos. Se pueden implementar *clusters* con pocos nodos y cuando se requiere mayor potencia de computación, se añaden más nodos al sistema. La red debe soportar el incremento de carga y ofrecer prácticamente las mismas prestaciones (el mismo ancho de banda y latencias de transmisión) en *clusters* pequeños (8 a 32 nodos) que en *clusters* grandes (cientos de nodos). Hay que tener también presente que la escalabilidad de un sistema está influenciada por otros condicionantes, entre ellos, la topología de la red y el algoritmo de encaminamiento utilizado.

- **Topologías irregulares**. Debido al bajo coste de los *clusters*, estos sistemas son atractivos para pequeñas y medianas empresas u organizaciones con una demanda de computación paralela. Típicamente, estas empresas conectan

todos sus computadores distribuidos utilizando una red de altas prestaciones. Debido a restricciones de espacio, la topología puede ser completamente irregular. Por lo tanto, las redes de interconexión deben ser diseñadas para soportar topologías irregulares. Asimismo, los algoritmos de encaminamiento utilizados también deben soportar topologías irregulares. No obstante, cuando la potencia de cálculo es el principal factor de diseño, estas redes de interconexión también se utilizan para formar *clusters* con topologías regulares. Tal es el caso del CPLANT (Computational PLANT) [Rie99] implementado en los laboratorios Sandia [SAN].

- **Fiabilidad**. En determinadas aplicaciones (típicamente transacciones) debe garantizarse la ausencia de errores o corrupción de mensajes. Para garantizar la fiabilidad de los datos, las redes de área local y área ancha tradicionales (LANs y WANs) generan mediante protocolos *software* CRCs, reconocimientos de mensajes y retransmisión de datos erróneos. Estos protocolos añaden una elevada latencia a los mensajes. En los *clusters* esta sobrecarga debe ser minimizada. Para ello, el cálculo de CRCs es realizado al mismo tiempo (*on-the-fly*) por el interfaz de red. El interfaz también se ocupa de almacenar temporalmente los mensajes y retransmitir el mensaje en el caso de que se detecte algún error, liberando así al procesador del nodo.

- **Reconfiguración**. Algunos nodos, conmutadores o enlaces de los *clusters* pueden desconectarse debido a apagados de máquinas, mal funcionamiento, etc. Por lo tanto, la topología de la red puede cambiar. Las redes de interconexión y los algoritmos de encaminamiento deben detectar estas situaciones y reconfigurarse a la nueva topología. Estos aspectos de reconfiguración son especialmente importantes en sistemas que deben ofrecen calidad de servicio.

Como conclusión, los *clusters* son una alternativa barata a los computadores paralelos para la resolución de problemas con un coste computacional medio. Las redes de interconexión deben ser eficientes tanto en términos de anchos de banda como en reducidas latencias para poder soportar el elevado tráfico de mensajes demandado por dichas aplicaciones.

Es por ello que, en la presente tesis, nos centramos en el estudio y mejora de las redes de interconexión para *clusters*. En los apartados restantes de este capítulo se analizarán diversos aspectos de implementación de estas redes y se marcarán las

ideas básicas a seguir para mejorar sus prestaciones.

1.2 Aspectos de implementación

Antes de describir ejemplos de redes de interconexión y de definir los objetivos de la presente tesis, es necesario introducir ciertos aspectos de implementación de las redes de interconexión para *clusters*. Los más importantes son: protocolo de enlace, control de flujo, conmutación, canales virtuales, encaminamiento y algoritmos de encaminamiento. Para obtener más detalles se recomienda [Dua97].

1.2.1 Protocolo de enlace

Con el término protocolo de enlace se hace referencia al formato de los mensajes y a la interacción entre dos puntos de comunicación. La figura 2 muestra dos puntos de comunicación. Estos puntos de comunicación pueden formar parte de un conmutador o de un interfaz de red. Estos puntos están conectados por dos canales unidireccionales para el envío y recepción de mensajes.

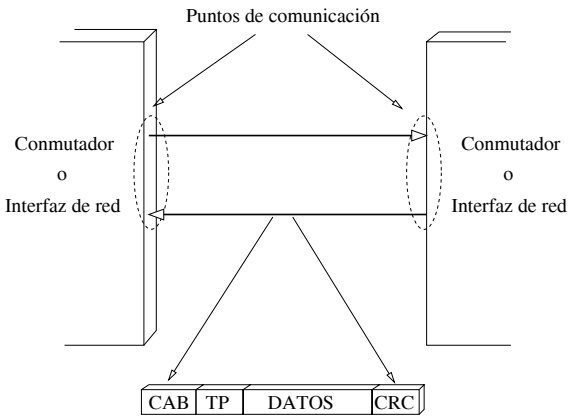


Figura 2: Protocolo de enlace.

La figura también muestra el formato general de un mensaje. El formato típico de un mensaje comprende una cabecera, un identificador de tipo de mensaje y los datos. La cabecera incluye información de encaminamiento que los conmutadores utilizan para encaminar el mensaje. El identificador de tipo de mensaje es utilizado en el nodo destino para identificar la aplicación destinataria del mensaje. Al tipo

de mensaje le siguen los datos. Por último, algunos formatos de mensajes también incluyen información de control de errores (CRC).

Tradicionalmente, los mensajes están delimitados por símbolos de control especiales que son utilizados para detectar el inicio/fin de los mensajes. Estos símbolos de control pueden también ser utilizados para activar eventos en el protocolo de enlace (por ejemplo, cuando el receptor no puede aceptar más datos, etc.).

1.2.2 Control de flujo

El control de flujo es una de las funciones básicas del protocolo de enlace y es utilizado para evitar desbordamientos en los *buffers* internos de los conmutadores. En los computadores paralelos, el control de flujo es implementado con señales de control adicionales. Esto es posible ya que tradicionalmente los enlaces en estos sistemas son cortos y prácticamente todos de la misma longitud. Sin embargo, en los *clusters* no sucede así. Los enlaces suelen ser largos (hasta 18 metros en Myrinet) e incluso hay largos y cortos en un mismo sistema. Por lo tanto, debido a los retardos variables que estos enlaces introducen, se utilizan otros mecanismos de control de flujo.

Una solución es implementar un mecanismo de control de flujo basado en créditos [IRFC] donde cada conmutador recibe varios créditos para enviar mensajes a un conmutador vecino. En cada transmisión de un paquete, el conmutador emisor consume un crédito. El conmutador detiene la emisión de paquetes cuando consume todos sus créditos. Después de liberarse espacio en el conmutador receptor, se restablece la transmisión. Para ello, el conmutador receptor envía más créditos al conmutador emisor a través de paquetes de control que son enviados por el canal de vuelta.

Otra solución adoptada es el control de flujo por medio de marcas *Stop & Go* [Bod95]. Por ejemplo, cuando un conmutador receptor detecta que el *buffer* está próximo a desbordarse, envía por el canal de vuelta un símbolo de *STOP* para que el conmutador emisor detenga el envío de paquetes. Cuando el conmutador receptor tiene espacio suficiente en el *buffer*, envía al conmutador emisor por el canal de vuelta un símbolo de *GO* para que se restablezca el envío de paquetes. En la sección 1.3.1 se describe con detalle el control de flujo *Stop & Go* utilizado en Myrinet.

Hay que hacer constar que la transmisión de los datos por los enlaces suele ser segmentada [Sco94] por lo que múltiples bytes pueden encontrarse en vuelo en el

enlace al mismo tiempo. Por lo tanto, para la implementación del mecanismo de control de flujo se deberán tener en cuenta dichos bytes en vuelo.

1.2.3 Conmutación

La conmutación comprende la conexión entre los puertos de entrada y los puertos de salida de un conmutador y la forma en que se realiza la transferencia de información entre ellos. Existen tres técnicas de conmutación principales utilizadas en las redes actuales: *store&forward*, *virtual cut-through* y *wormhole*.

En la conmutación *store&forward* [Tanen96] (figura 3) el mensaje que llega a un conmutador se almacena (*store*) en un *buffer* asociado al puerto de entrada y una vez ha llegado completamente se reenvía (*forward*) al siguiente conmutador o al nodo destino. Este mecanismo tiene el inconveniente de que la latencia del mensaje es proporcional a la distancia entre el origen y el destino. Otro problema adicional en la conmutación *store&forward* es la limitación del tamaño de los mensajes. Puesto que se debe almacenar un mensaje por completo en un conmutador y el tamaño de los *buffers* asociados a los puertos de entrada son finitos, el tamaño máximo de un mensaje está limitado al tamaño del *buffer*. La red ATM [Mar93] utiliza este mecanismo de conmutación.

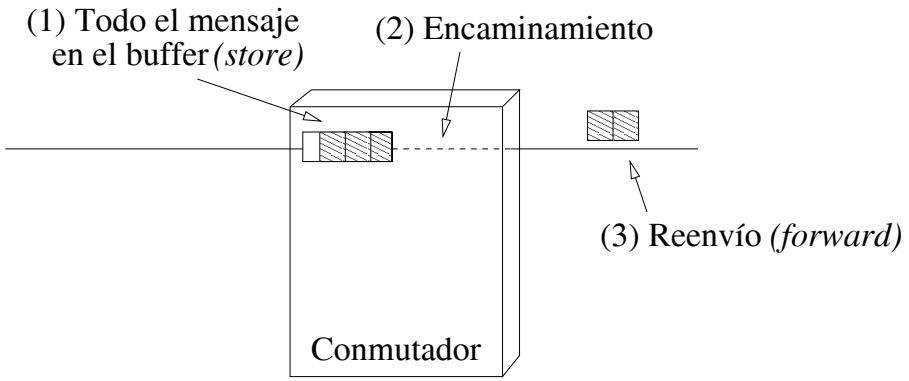


Figura 3: Conmutación *store&forward*.

En la conmutación *virtual cut-through* [Ker79] un paquete es retransmitido hacia el siguiente conmutador tan pronto como la cabecera es decodificada, sin necesidad de que se reciba previamente todo el mensaje. No obstante, si el canal de salida

seleccionado está ocupado, el mensaje se almacena por completo en el *buffer* de entrada del último conmutador hasta que la salida esté disponible. Por lo tanto, la conmutación *virtual cut-through* se comporta como *store&forward* cuando el canal de salida está ocupado. Con este mecanismo de conmutación, en ausencia de contención, la latencia del mensaje se ve poco influenciada por la distancia al destino. Debido a que todo el mensaje se debe almacenar en un *buffer* (en caso de que el canal de salida esté ocupado) el tamaño de los mensajes está limitado (al igual que en el mecanismo *store&forward*). El chaos router [Kon91] utiliza el mecanismo de conmutación *virtual cut-through*.

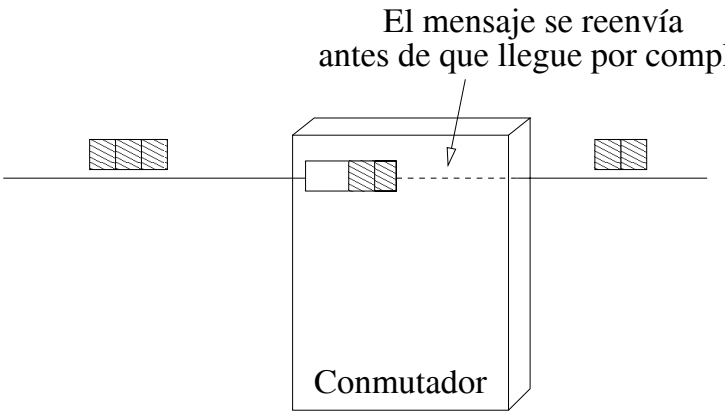


Figura 4: Conmutación *virtual cut-through*.

En la conmutación *wormhole* [Dal87] (también conocida como *cut-through*), al igual que en la conmutación *virtual cut-through*, el mensaje es reenviado inmediatamente hacia el siguiente conmutador antes de que llegue todo el mensaje. No obstante, la diferencia con el mecanismo *virtual cut-through* es que cuando un canal de salida está ocupado, el mensaje no se almacena en el *buffer* del último conmutador visitado sino que el mensaje se queda almacenado a lo largo de todos los *buffers* de los conmutadores visitados. Por lo tanto, el mensaje se queda bloqueado a lo largo de toda la ruta. La principal ventaja de este mecanismo es que el tamaño del mensaje no está limitado, por lo que se necesita poco espacio de almacenamiento (*buffers*) en los puertos de entrada de los conmutadores.

Sin embargo, la productividad de las redes de interconexión que utilizan el mecanismo de conmutación *wormhole* es, a menudo, baja. Esto se debe a una situación que aparece cuando un mensaje no puede avanzar debido a que el canal de salida

está ocupado. El mensaje ha reservado una serie de *buffers* y canales, y los mantiene durante todo el tiempo que está bloqueado. Como consecuencia, si otros mensajes desean utilizar algún canal de los reservados por el mensaje bloqueado, se producirán nuevos conflictos, que, a su vez, originarán más conflictos con otros mensajes. Todo ello conlleva a una pobre utilización de los canales, lo que se traduce en una baja productividad de la red de interconexión. En la figura 5 observamos como el mensaje a se detiene ocupando canales entre los tres conmutadores que ha visitado. Estos canales no pueden ser utilizados por los mensajes b y c hasta que el mensaje bloqueado siga su camino hacia el destino final. Adicionalmente, los mensajes b y c bloquearán otros mensajes.

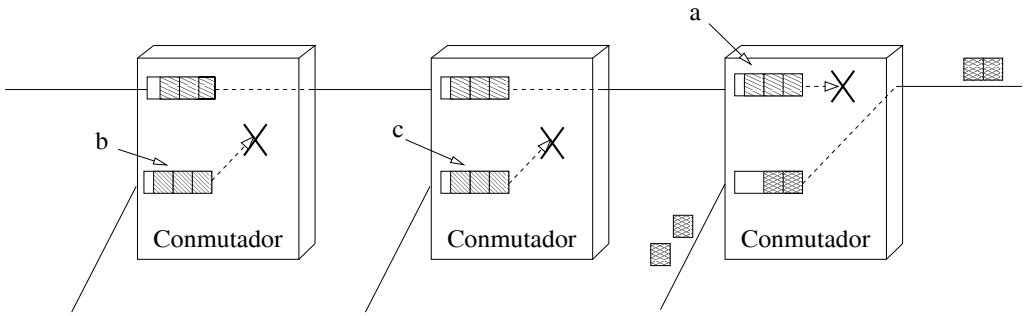


Figura 5: Conmutación *wormhole*. Ejemplo de mensajes bloqueados.

Las redes de altas prestaciones como Myrinet y ServerNet utilizan el mecanismo de conmutación *wormhole*.

Canales virtuales

Para evitar el problema de la baja productividad en el mecanismo de conmutación *wormhole*, en [Dal92] se propuso una solución. En primer lugar, en vez de asociar un único *buffer* a cada canal físico, se asocian múltiples *buffers* (figura 6). Adicionalmente, en lugar de asignar simultáneamente *buffers* y canales a los mensajes, se asignan por separado estos recursos. De este modo, los *buffers* se asignan igualmente a los mensajes conforme la cabecera va avanzando hacia su destino. Sin embargo, los canales físicos se comparten entre todos los *buffers* con información lista para transmitir. Es evidente que si un mensaje está bloqueado, los bytes almacenados en los *buffers* reservados no están listos para ser transmitidos, puesto que no pueden

avanzar, dejando los canales totalmente libres para que otros mensajes los utilicen. Por supuesto, este mecanismo también permite que varios mensajes utilicen de forma compartida un mismo canal físico, avanzando todos ellos a menor velocidad. Al conjunto formado por cada *buffer* junto con la información que caracteriza su estado se le denomina canal virtual.

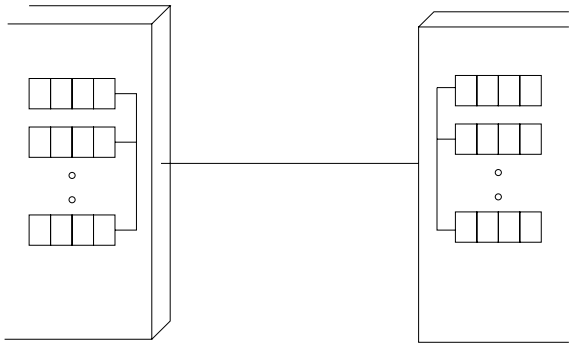


Figura 6: Canales virtuales.

La utilización de canales virtuales permite aumentar notablemente la productividad de la red, a costa de aumentar también la lógica de control asociada a cada canal físico. El inconveniente de este mecanismo es que se reduce el ancho de banda disponible para cada mensaje. Además, sólo con que uno de los canales que atraviesa el mensaje esté multiplexado, la totalidad del mensaje avanzará a menor velocidad.

1.2.4 Encaminamiento

Como se ha comentado anteriormente, la cabecera del mensaje contiene información para el encaminamiento. Esta información es leída en los conmutadores, y a partir de ella se selecciona el canal de salida que se utilizará para encaminar el mensaje hacia el destino final. Existen principalmente dos mecanismos [Dua97] para encaminar mensajes: encaminamiento fuente y encaminamiento distribuido.

En el encaminamiento fuente, la ruta a seguir por el mensaje se calcula en el nodo que lo genera, escribiéndola en la cabecera del mensaje. Cada conmutador lee el primer elemento de la cabecera del mensaje y lo utiliza para seleccionar el canal de salida. Una vez encaminado el mensaje, se elimina el primer elemento de la cabecera. Por lo tanto, ahora el primer elemento de la cabecera contiene la

información de encaminamiento para el siguiente conmutador. La figura 7 ilustra el mecanismo de encaminamiento fuente.

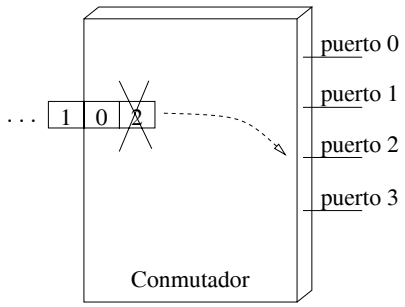


Figura 7: Encaminamiento fuente.

Por otra parte, en el encaminamiento distribuido la cabecera del mensaje contiene únicamente el identificador del nodo destino. A partir de esta información y del puerto de entrada del mensaje, el conmutador calcula el puerto de salida. Habitualmente, los conmutadores poseen unas tablas de encaminamiento internas que, a partir del puerto de entrada de un mensaje y de su destino devuelven el puerto de salida que debe utilizarse. Cuando se utilizan tablas en los conmutadores el encaminamiento se denomina encaminamiento distribuido basado en tabla. La figura 8 ilustra el mecanismo del encaminamiento distribuido basado en tabla.

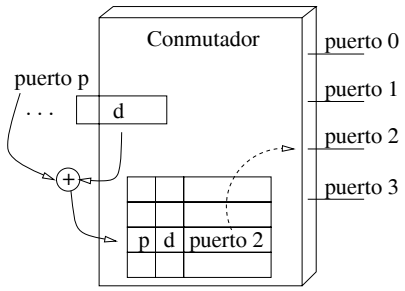


Figura 8: Encaminamiento distribuido basado en tabla.

Tradicionalmente, los conmutadores basados en encaminamiento fuente son más rápidos que los basados en encaminamiento distribuido basado en tabla ya que en los segundos la selección del canal de salida se realiza accediendo a la tabla mientras

que en los primeros el puerto de salida a utilizar se encuentra en la cabecera del mensaje, por lo que el tiempo de encaminamiento es menor. Además, el tamaño de la tabla de encaminamiento debe ser proporcional al número de nodos, lo que puede limitar la escalabilidad del sistema al utilizar encaminamiento distribuido basado en tabla.

Una desventaja del encaminamiento fuente es que el tamaño de la cabecera depende de la distancia al destino. Este tamaño puede ser considerablemente grande en relación con el número de bytes a transmitir (especialmente en mensajes cortos). Por otra parte, en el encaminamiento distribuido el tamaño de la cabecera es pequeño y constante (uno o dos bytes).

Por otra parte, los conmutadores con encaminamiento distribuido tienen la ventaja de que pueden decidir el canal de salida en función del estado del tráfico local detectado en los puertos de salida. En este caso, el conmutador ofrece diferentes salidas de encaminamiento para un determinado par *puerto de entrada-nodo destino*, seleccionando de entre las opciones alternativas en función del estado de los canales de salida (libre u ocupado). Por lo tanto, los conmutadores con encaminamiento distribuido pueden adaptarse a las condiciones de tráfico. Los conmutadores con encaminamiento fuente no tienen esta posibilidad ya que el mensaje tiene la ruta prefijada en la cabecera desde el origen y no puede ser cambiada por los conmutadores.

Myrinet utiliza encaminamiento fuente mientras que ServerNet utiliza encaminamiento distribuido basado en tabla.

1.2.5 Algoritmos de encaminamiento

Multitud de algoritmos de encaminamiento se han propuesto para las redes de interconexión. No obstante, muchos de ellos no son aplicables a los *clusters*, debido a que dichas redes pueden adoptar cualquier topología (típicamente irregular). Por lo tanto, en la presente sección, solamente nos centramos en los algoritmos de encaminamiento que pueden ser utilizados sobre cualquier topología.

Uno de los aspectos más importantes en los algoritmos de encaminamiento es el problema del bloqueo[2]. Tradicionalmente, se utilizan técnicas de evitación de bloqueos que evitan ciclos en el grafo de dependencias de canales (GDC) [Dal87]. No obstante, hay algoritmos de encaminamiento menos restrictivos que permiten

[2]En inglés *deadlock*.

ciclos en el GDC siendo aún libres de bloqueo. Para ello, estos algoritmos de encaminamiento deben proveer rutas alternativas de escape [Dua93, Dua95] para evitar el bloqueo.

Sin embargo, los algoritmos de encaminamiento que permiten ciclos en el GDC solamente pueden ser utilizados en redes con encaminamiento distribuido (son los conmutadores los que reenvían los mensajes por las rutas de escape en caso necesario). Por otro lado, los algoritmos de encaminamiento sin ciclos en el GDC pueden ser utilizados tanto en redes con encaminamiento fuente como en redes con encaminamiento distribuido basado en tablas.

El algoritmo de encaminamiento más conocido para topologías irregulares es el *up*/down**. Este algoritmo de encaminamiento fue utilizado en el prototipo DEC AN1 [Sch90] y actualmente en Myrinet [Bod95]. Se basa en una asignación de etiquetas de dirección a los enlaces. Para ello, se calcula un árbol en profundidad a partir de un conmutador raíz. La dirección *up* se asigna al sentido que va desde un conmutador de un nivel a otro conmutador en un nivel más cercano al conmutador raíz (en sentido ascendente en el árbol). Si los dos conmutadores se encuentran en el mismo nivel del árbol, entonces se asigna una dirección en función de los identificadores de los conmutadores (por ejemplo, dirección *up* de un conmutador con un identificador mayor a un conmutador con un identificador menor).

Como resultado de esta asignación, cada ciclo en la red tendrá por lo menos un enlace en la dirección *up* y un enlace en la dirección *down*. El algoritmo utiliza la siguiente regla para evitar dependencias cíclicas entre canales: los mensajes no podrán utilizar canales en la dirección *up* después de haber utilizado canales en la dirección *down*. Básicamente, los mensajes podrán recorrer el árbol en sentido ascendente y después en sentido descendente, pero no podrán volver a subir el árbol una vez empiezan a bajarlo. En la figura 9 podemos ver la asignación de canales en una red irregular según el algoritmo de encaminamiento *up*/down**. Debido a que el algoritmo de encaminamiento *up*/down** no introduce ciclos en el GDC, puede ser utilizado tanto en redes con encaminamiento fuente como en redes con encaminamiento distribuido.

El algoritmo de encaminamiento *adaptive trail* [Qia96] se basa en encontrar una ruta euleriana en la red, la cual establece un orden de dependencias para todos los canales. Ciertos canales (denominados *shortcuts*) se añaden a la ruta euleriana original para reducir las longitudes de las rutas. Los conmutadores utilizarán los *shortcuts* si están libres, en caso contrario utilizarán la ruta euleriana como ruta de

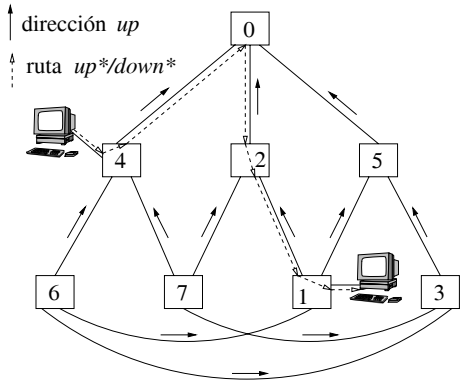


Figura 9: Asignación de direcciones a los canales en *up*/down**.

escape. Como vemos, el *adaptive trail* fue propuesto para redes con encaminamiento distribuido, no siendo posible su utilización en redes con encaminamiento fuente.

El principal inconveniente del *adaptive trail* es que puede ser difícil encontrar una ruta euleriana en algunas topologías irregulares. Este inconveniente es especialmente importante cuando la topología cambia frecuentemente debido a reconfiguraciones, como es el caso de los *clusters*.

El algoritmo de encaminamiento *smart-routing* [Che95] primero calcula todas las posibles rutas entre los pares origen-destino, construyendo al mismo tiempo el GDC. Después, busca a través del grafo la existencia de ciclos. Un proceso iterativo rompe ciclos eliminando dependencias (tomando en cuenta una función de coste que intenta equilibrar el tráfico de los canales). El proceso finaliza cuando el GDC no tiene ciclos. Aunque *smart-routing* distribuye mejor el tráfico que otros algoritmos de encaminamiento, tiene el inconveniente de su elevado coste computacional, ya que utiliza un algoritmo de programación lineal para equilibrar el tráfico mientras elimina ciclos. El *smart-routing* puede ser utilizado tanto en redes con encaminamiento fuente como en redes con encaminamiento distribuido, ya que no introduce ciclos en el GDC.

El algoritmo de encaminamiento *minimal adaptive* [Sil97b] multiplexa cada enlace en dos canales virtuales, denominados *adaptive* y *escape*, respectivamente. El canal *adaptive* es utilizado para encaminar mensajes a través de rutas mínimas sin restricciones, mientras que el canal *escape* es utilizado como ruta de escape por los

mensajes cuando el canal *adaptive* está ocupado. Los canales *escape* deben ser utilizados siguiendo las reglas del algoritmo de encaminamiento *up*/down**, lo cual garantiza un GDC acíclico de los canales *escape*. El algoritmo *minimal adaptive* es útil para redes con encaminamiento distribuido y utilizando canales virtuales. Sin embargo, en redes con encaminamiento fuente no puede ser utilizado.

El algoritmo *DFS* [San00] es muy parecido al algoritmo *up*/down**, estando basado también en la construcción de un árbol, pero en este caso en amplitud. Los canales restantes no incluidos en el árbol son añadidos para proveer rutas mínimas, lo que conduce a ciclos en el GDC. Al igual que otros algoritmos, los ciclos son eliminados introduciendo restricciones de encaminamiento. Sin embargo, los canales son etiquetados utilizando una heurística que reduce el número de restricciones. Este algoritmo de encaminamiento puede ser utilizado tanto en redes con encaminamiento fuente como en redes con encaminamiento distribuido.

1.3 Ejemplos de redes de interconexión

Diferentes redes de interconexión han aparecido en el mercado durante los últimos años. Ejemplos de estas redes son HiPPI [HIP], ATM [Mar93], SCI [SCI], ServerNet [Rob96], ServerNet II [Dav97], Myrinet [Bod95], MemoryChannel [Fil97] y Synfinity [Lar98].

Cada red de interconexión tiene unas características determinadas. No obstante, podemos realizar una clasificación atendiendo a unos pocos parámetros como son el ancho de banda, el mecanismo de conmutación y el encaminamiento utilizado. En la tabla 1 se muestran las principales características de algunas redes[3]. Como podemos observar, la mayoría de las redes de interconexión utilizan el encaminamiento distribuido basado en tabla. Solamente Myrinet y Synfinity utilizan el encaminamiento fuente. Por otro lado, la mayoría de las redes utilizan conmutación *store&forward*.

En la siguientes secciones describimos con mayor detalle las redes Myrinet y ServerNet. Estas redes han sido seleccionadas por tratarse de dos ejemplos utilizados en la actualidad y por ser representativos de las redes con encaminamiento fuente (Myrinet) y con encaminamiento distribuido (ServerNet).

[3]Información obtenida de [Buy99].

Red	Ancho de banda uni.	Conmutación	Encaminamiento
Fast Ethernet	100 Mbit/s	*store&forward*	basado en tabla
Gigabit Ethernet	1 Gbit/s	*store&forward*	basado en tabla
Myrinet	1.28 Gbit/s	*wormhole*	fuente
ServerNet II	125 Mbyte/s	*wormhole*	basado en tabla
Memory Channel	100 Mbyte/s	*store&forward*	basado en tabla
Synfinity	1.6 Gbyte/s	*wormhole*	fuente
SCI	400 Mbyte/s	*store&forward*	basado en tabla
ATM	155 Mbit/s	*store&forward*	basado en tabla
HiPPI	800 Mbit/s	*store&forward*	basado en tabla

Tabla 1: Principales características de las redes de interconexión.

1.3.1 Myrinet

La red Myrinet surge de dos proyectos de investigación previos: El CalTech Mosaic Project [Sei93] y el proyecto USC/ISI ATOMIC LAN [Fel94]. A partir de estos dos proyectos surge la empresa Myricom [MYRI] con el fin de comercializar la red Myrinet. Myrinet es ampliamente utilizada en la actualidad como red de interconexión en sistemas de *clusters* de PCs y estaciones de trabajo. La red Myrinet está compuesta de enlaces, conmutadores e interfaces de red. Estos tres elementos pueden ser interconectados entre sí formando topologías arbitrarias. A continuación se describen algunas de las características más relevantes de estos elementos.

Enlaces

Un enlace (figura 10) está compuesto de dos canales funcionando como *full-duplex*, denominándose puerto a la conexión del enlace a un sistema (un conmutador o un interfaz de red).

Un canal Myrinet lleva 9 bits de información, pudiendo ser datos de 8 bits o símbolos de control de 8 bits. El noveno bit permite distinguir los datos de los símbolos de control. Los símbolos de control son intercalados entre los datos para realizar operaciones de delimitación de paquetes, control de flujo y otras funciones. Por ejemplo, la secuencia mostrada en la figura 11 corresponde a la transmisión de un paquete de 4 bytes (d0, d1, d2 y d3) el cual es delimitado por el símbolo *GAP*. Los símbolos *GO* y *STOP* son intercalados en esta secuencia como soporte para el mecanismo de control de flujo *Stop & Go* del canal en sentido contrario. El

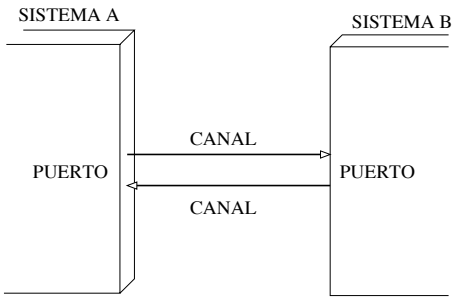


Figura 10: Enlaces Myrinet.

símbolo *IDLE* es utilizado para rellenar ciclos en los que no se transmite ningún dato ni símbolo de control pudiendo aparecer entre datos de un paquete o entre dos paquetes.

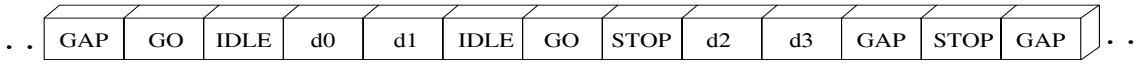


Figura 11: Secuencia de símbolos en Myrinet.

Existen dos tipos de cables: SAN y LAN. La longitud de los cables SAN (figura 12.a) es de 3 metros. El cable SAN lleva dos enlaces, cada uno de ellos soportando dos canales funcionando como *full-duplex*. Debido a su reducida longitud, estos cables están destinados para implementar sistemas empotrados. Los cables SAN no están apantallados y soportan anchos de banda de 1.28 Gb/s (160MB/s) en cada canal.

Respecto al cable LAN (figura 12.b) la longitud máxima es de 10.66 metros. El cable LAN está apantallado soportando un enlace con dos canales (*full-duplex*). Estos cables soportan anchos de banda de 1.28 Gb/s (160MB/s) en cada canal. Existen otros cables LAN de mayor longitud (12-18 metros) pero solamente soportan anchos de banda de 0.64 Gb/s (80MB/s) en cada canal. Los cables LAN son utilizados para formar topologías sobre sistemas situados a relativas distancias.

El medio de transporte físico del canal tiene un retardo denominado tiempo de vuelo. Debido a este retardo, en un canal pueden haber varios bytes en vuelo al mismo tiempo. En función del retardo del cable, de su longitud y del ancho de banda soportado existirán más o menos bytes en vuelo. Por ejemplo, para un cable de 10

(a) (b)

Figura 12: Cables Myrinet: (a) SAN y (b) LAN.

metros con un retardo de 4.92 ns/m y un ancho de banda de 160 MB/s existirán 8 bytes en vuelo.

Formato de los mensajes

Los mensajes en Myrinet pueden ser de cualquier tamaño y están compuestos por una cabecera, un identificador de tipo, los datos del mensaje, un código CRC y un símbolo de control (*GAP*) que delimita el fin del mensaje. En la figura 13 podemos ver el formato típico de un mensaje.

Myrinet utiliza encaminamiento fuente. Por lo tanto, la cabecera incluye todos los bytes de encaminamiento para llegar al destino. Adicionalmente, también incluye el tipo del mensaje (2 bytes). Los bits más significativos de los bytes de encaminamiento de la cabecera se utilizan para la detección de errores de encaminamiento. Cuando el primer bit de un byte de encaminamiento en la cabecera está a 1 significa que realmente es un byte de encaminamiento que debe ser utilizado por un conmutador. Si el bit está a 0 significa que es el primer byte que debe llegar al interfaz de red (primer byte del tipo). Por lo tanto, si un conmutador detecta el primer byte de la cabecera con un 0 en el bit más significativo, interpretará que es un error de encaminamiento (por ejemplo debido a una ruta mal formada) y desechará el

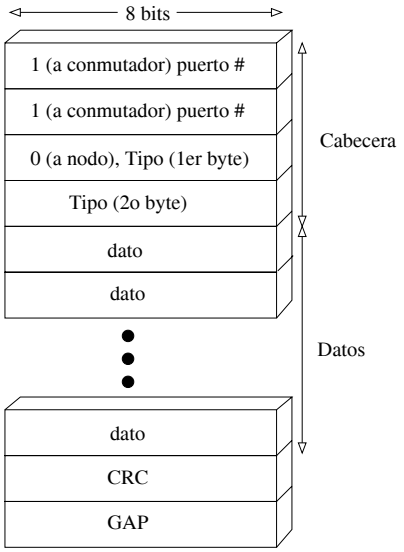


Figura 13: Formato de los mensajes en Myrinet.

mensaje. Del mismo modo, si un interfaz de red detecta que el bit más significativo del primer byte recibido está a 1 interpretará también que ha ocurrido un error de encaminamiento por lo que también desechará el mensaje.

Los bytes de tipo de mensaje identifican en el nodo destino la aplicación que debe recibir el mensaje. Después de la cabecera siguen los datos del mensaje y el CRC. El CRC es comprobado en el nodo destino. Por último, el símbolo de control *GAP* indica el fin del mensaje.

Conmutadores

Los conmutadores están implementados con un *crossbar* entre sus canales de entrada y salida. Existen conmutadores de 8 puertos y conmutadores de 16 puertos. Adicionalmente, Myricom suministra conmutadores de mayor tamaño construidos a partir de conmutadores básicos de 8 y 16 puertos.

Los conmutadores utilizan conmutación *wormhole* y encaminamiento fuente para encaminar los mensajes. Para encaminar un mensaje que entra por un canal de entrada, el conmutador utiliza el primer byte de la cabecera del mensaje. La información de la cabecera son desplazamientos a partir del canal de entrada. Por lo tanto, el conmutador suma el byte de la cabecera al identificador del canal de

entrada para obtener el identificador del canal de salida.

Si el canal de salida está libre en el momento del encaminamiento, entonces la cabecera avanza por el *crossbar* hacia el canal de salida tan pronto como se va recibiendo el mensaje. El primer byte es eliminado de la cabecera. La conexión en el *crossbar* entre el canal de entrada y el canal de salida se mantiene hasta que se recibe la cola del mensaje (símbolo de control *GAP*). Cuando un conmutador recibe este símbolo rompe la conexión entre el canal de entrada y el canal de salida en el conmutador.

Por otra parte, si el canal de salida está ocupado por otro mensaje, entonces el mensaje se bloquea. Cada canal de salida posee un árbitro que selecciona de entre los mensajes bloqueados en diferentes canales de entrada cual de ellos será encaminado una vez se libera el canal de salida. Estos árbitros utilizan un mecanismo *round-robin* para seleccionar el mensaje que será encaminado.

Según [Bod95] la latencia máxima por un conmutador de 8 puertos es de 550 ns. No obstante, en la actualidad dichas latencias son menores y dependientes del tipo de cable conectado en ambos puertos (entrada y salida). Por ejemplo, el conmutador M2FM-SW8 tiene 4 puertos LAN y 4 puertos SAN. La latencia (tiempo de encaminamiento) entre puertos LAN es de 300 ns, entre puertos LAN-SAN de 200 ns y entre puertos SAN de 100 ns. Estos parámetros son aproximados según [SWCH].

Interfaces de red

En la figura 14 podemos ver la arquitectura del interfaz de red.

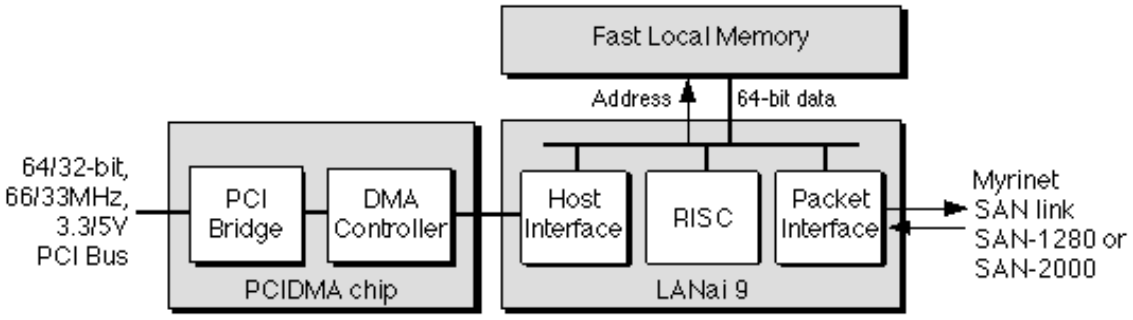


Figura 14: Arquitectura del interfaz de red de Myrinet.

El interfaz de red consta de los siguientes elementos:

- **Interfaz con el bus PCI**. El interfaz de red se comunica con el *host* a través del bus PCI. Este interfaz con el bus PCI incluye un dispositivo DMA para realizar transferencias directamente desde la memoria del *host* hacia la memoria del interfaz (o viceversa).

- **Memoria del interfaz**. El propio interfaz tiene una memoria donde se ubica el *software* de control denominado Myrinet Control Program (MCP) y donde se almacenan temporalmente los mensajes que se van a enviar o los que se reciben de la red. En las implementaciones actuales la memoria alcanza los 8MB.

- **LANai 9**. El LANai 9 es el corazón del interfaz de red. Incluye un procesador encargado de gestionar el acceso por parte del *host* a la red y viceversa. Este procesador es un procesador RISC específico a 132MHz. El procesador ejecuta el MCP ubicado en la memoria local del interfaz.

- **Packet Interface**. Este dispositivo ubicado en el LANai es el que finalmente envía/recibe los datos a/de la red. También avisa al procesador del interfaz (mediante registros de estado) de actividad en la red (recepción, finalización de envío, etc.). Adicionalmente, también dispone de dos dispositivos de DMA que envían/reciben información a/de la red desde/hacia la memoria local del interfaz. Estos dispositivos se denominan S-DMA y R-DMA.

La implementación del interfaz de red por medio de un procesador que ejecuta cierto código de control (MCP) y con dispositivos DMA de envío y recepción permiten una gran versatilidad. De hecho, el código MCP es el encargado de enviar/recibir mensajes, reconfigurar la red, calcular las rutas a utilizar, enviar mensajes de reconfiguración, etc. Dicho código es de dominio público existiendo herramientas de desarrollo gratuitas. Por lo tanto, el código MCP puede ser diseñado incluso para implementar nuevos mecanismos o mejorar los presentes. Este hecho ha propiciado la popularidad de Myrinet en el ámbito universitario ya que permite la implementación de nuevos protocolos y mecanismos por parte de los grupos de investigación.

Control de Flujo

El control de flujo es implementado mediante el envío de los símbolos *STOP* y *GO* por el canal opuesto. Para ello, cada canal receptor tiene asociado un *buffer* denominado *slack buffer*. En este *buffer*, se definen dos marcas k_s y k_g para generar los símbolos *STOP* y *GO*, respectivamente. Este *buffer* es de tipo FIFO de tamaño $r = k_g + h + k_s$, siendo h la histéresis entre los dos umbrales que se definirán a partir de las marcas. El *slack buffer* puede ser visto como un depósito de agua. El conmutador receptor genera un símbolo de control *STOP* cuando el nivel de agua (f) del *slack buffer* supera la marca $r - k_s$ y genera un símbolo de control *GO* cuando el nivel de agua (f) se decrementa y es superado por k_g.

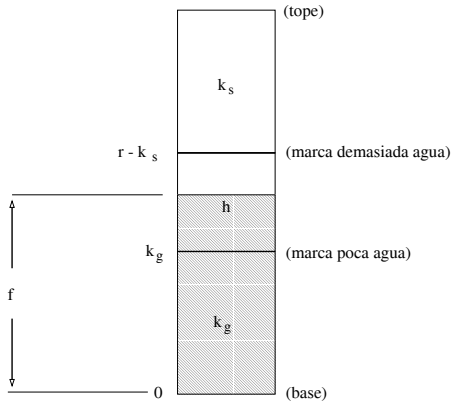


Figura 15: *Slack buffer* de Myrinet.

Los niveles k_s y k_g deben ser calculados teniendo en cuenta el retardo de los cables y los posibles bytes en vuelo. El nivel k_s es el espacio disponible para detener el control de flujo sin pérdida de información. En el peor caso k_s debe ser lo suficientemente grande como para almacenar los bytes restantes que se recibirán desde que se envía el símbolo *STOP* hasta que realmente el conmutador emisor detiene el control de flujo. Para un ancho de banda $B = 80\ MB/s$ y un retardo de $D = 140\ ns$ se recibirán $2 * D * B = 23$ bytes desde que se envía el símbolo *STOP* hasta que realmente se detiene la recepción de bytes. Por lo tanto, k_s debe ser mayor o igual a 23.

De forma análoga el valor mínimo de k_g debe ser elegido para que el *buffer* no se vacíe tras emitir un evento *GO* antes de recibir nuevos bytes del conmutador emisor.

Para los mismos datos B y D anteriores, k_g debe ser mayor o igual a 23. Por último, el parámetro h (histéresis) debe ser diseñado para minimizar el número de símbolos *STOP* y *GO* enviados.

Reconfiguración y bloqueos

Para detectar fallos de enlaces o incluso de conmutadores e interfaces de red, los conmutadores e interfaces de red envían periódicamente símbolos de control. A su vez, existe un proceso denominado *mapper* ubicado en un nodo determinado que es el encargado de detectar los cambios en la topología. Cuando se detecta un fallo de enlace, conmutador o interfaz, el *mapper* envía mensajes de control a todos los nodos descubriendo la nueva topología. El *mapper* calcula las nuevas rutas a partir de la nueva topología y las distribuye a todos los nodos. Una característica importante de Myrinet es que en el caso de que el nodo con el *mapper* falle, la red automáticamente genera otro *mapper*. Incluso pueden crearse varios *mappers* en el caso de que la red se divida por completo en dos subredes completamente independientes.

Debido a que existe la posibilidad de que un nodo envíe mensajes por una ruta errónea, se pueden producir bloqueos en la red. Para evitar estas situaciones, los conmutadores utilizan *timeouts* para detectar los bloqueos. Cuando un mensaje permanece bloqueado más de 50 ms, el conmutador lo desecha de la red generándose una señal (*FRESET*) que vacía todos los *buffers* de los enlaces utilizados por el mensaje.

Recientemente, Myricom ha anunciado la disponibilidad de nuevos conmutadores (*Myrinet-2000*), enlaces (*Myrinet SAN-2000*, *Myrinet-2000 serial* y *Myrinet-2000 Fiber*) e interfaces de red (*M3M-PCI64B* y *M3M-PMC64B*) que incrementan el ancho de banda de la red a 2 Gb/s. Para una mayor información consultar [MYR2K].

1.3.2 ServerNet

En 1995, Tandem introduce su primera implementación comercial de ServerNet [Rob96]. El objetivo principal de Tandem en el diseño de ServerNet era el de obtener unos mayores anchos de banda en sus sistemas de entrada/salida. Por lo tanto, esta red se originó con la idea de conectar procesadores con dispositivos de entrada/salida. No obstante, la red evolucionó rápidamente para interconectar también procesadores.

En 1998 ServerNet fue mejorada apareciendo una nueva versión de la red denominada ServerNet II [Dav97]. Esta nueva versión aumenta el ancho de banda y añade nuevas características manteniendo la total compatibilidad con ServerNet. De hecho, los sistemas pueden ser construidos a partir de la utilización de componentes de ambas redes.

Los principales objetivos de diseño en ServerNet son la escalabilidad y la fiabilidad. La red está compuesta de nodos y dispositivos de entrada/salida todos ellos con interfaces de red. Todos estos elementos están conectados por medio de conmutadores.

Una característica importante en ServerNet es la posibilidad de realizar transferencias directamente entre dispositivos de entrada/salida, por lo que se libera a los nodos de estas transferencias. Esta característica permite la realización de múltiples transferencias de entrada/salida en el sistema simultáneamente.

En la figura 16 podemos ver una configuración típica de una red ServerNet.

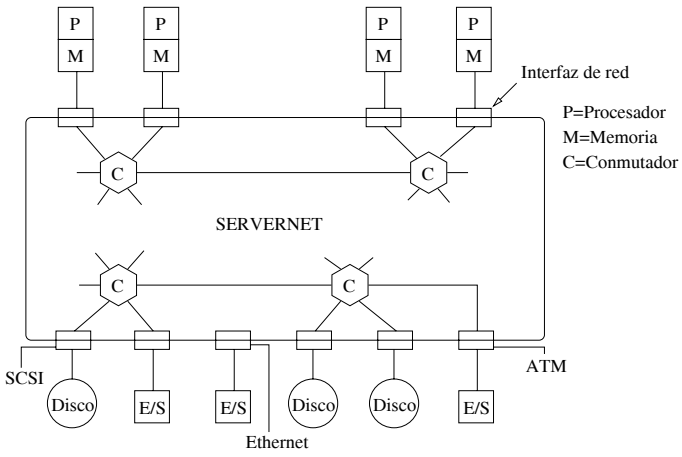


Figura 16: Red ServerNet.

Los interfaces de red están conectados al bus del sistema del computador (generalmente el bus PCI). Tan sólo los interfaces de red de la serie Himalaya de Tandem se conectan al bus de memoria del computador.

Enlaces

Los enlaces ServerNet están formados por dos canales en *full-duplex*. ServerNet utiliza enlaces de 9 bits en cada dirección trabajando a 50 MB/s, mientras que

ServerNet II eleva el ancho de banda a 125 MB/seg.

Los enlaces operan de forma asíncrona utilizándose símbolos de control *SKIP* para evitar el desbordamiento de los *buffers* de entrada. Estos símbolos de control son desechados por el receptor. El control de flujo es implementado por medio de símbolos especiales.

El mecanismo básico de transferencia por la red es a través de lecturas/escrituras en memoria remota por medio de dispositivos DMA. Un nodo puede leer/escribir un paquete de hasta 64 bytes (hasta 512 bytes en ServerNet II) desde/hacia una posición de memoria remota.

La dirección de un paquete consta de un identificador de 20 bits y de 32/64 bits de dirección del destino. Con el identificador, ServerNet identifica unívocamente un sólo nodo y la ruta a utilizar hacia el destino. La dirección puede ser vista como una dirección virtual ServerNet. Los 12 bits menos significativos son desplazamientos de página, mientras que los bits más significativos son un índice a una tabla AVT (*Address Validation & Translation Table*). Con esta indirección, el receptor puede comprobar permisos de lectura/escritura del emisor tal y como se indica en la figura 17.

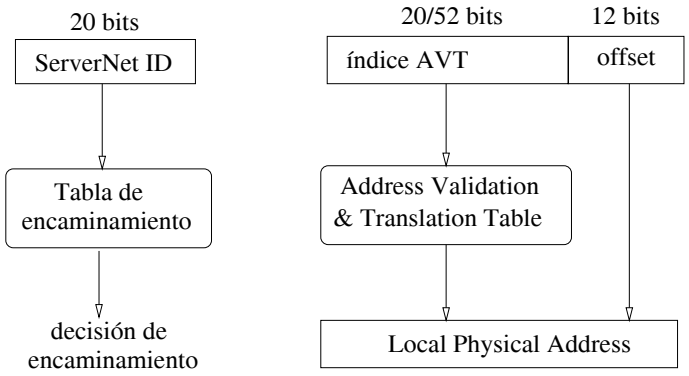


Figura 17: Espacio de direccionamiento en ServerNet.

Una característica importante de ServerNet es su tolerancia a fallos. ServerNet posee un mecanismo de detección y corrección de errores. Al nivel de enlace, cada conmutador comprueba el CRC del mensaje recibido. A su vez, cada enlace es comprobado mediante intercambio periódico de símbolos de control. Por último, cada paquete es reconocido por mensajes de reconocimiento de vuelta al emisor. En el caso de error, se invocan rutinas para su tratamiento.

Conmutadores

ServerNet posee conmutadores de 6 puertos que pueden ser conectados formando cualquier topología. La siguiente generación de conmutadores de ServerNet (Router II) incrementa el número de puertos a 12. En este último caso, el conmutador posee un *crossbar* interno de 13x13. El puerto adicional es utilizado para inyectar o extraer paquetes de control. Cada conmutador posee un interfaz de procesador para servicios de mantenimiento y depuración.

Una característica especial de ServerNet es la posibilidad de formar los denominados *Fat Pipes*. Algunos enlaces físicos pueden ser utilizados para formar un enlace lógico (conectando dos elementos). Los conmutadores pueden ser configurados para elegir dinámicamente uno de los enlaces. Con esto se consigue obtener una mejor utilización de los enlaces en alta carga.

ServerNet utiliza conmutación *wormhole* para encaminar mensajes a través de la red. El encaminamiento de los mensajes es distribuido y se realiza mediante tablas. Para ello, los mensajes llevan el destino en la cabecera. Esta información es utilizada para buscar en tablas internas del conmutador el puerto de salida a utilizar.

1.4 Limitaciones en el encaminamiento fuente

Durante los últimos años, en el grupo de arquitecturas paralelas (GAP) de la UPV se ha realizado un profundo análisis de las redes de interconexión con encaminamiento distribuido para *clusters* [Sil97a, Sil97b, Sil97c, Sil97d, Sil00]. En estos estudios se han evaluado distintos algoritmos de encaminamiento, así como consideraciones acerca de los mecanismos de control de flujo.

En lo referente a los algoritmos de encaminamiento se ha propuesto un nuevo algoritmo de encaminamiento adaptativo denominado *minimal adaptive* (ver sección 1.2.5). Como ya se ha descrito, el algoritmo de encaminamiento *minimal adaptive* necesita de canales virtuales para su implementación, utilizando un canal virtual para encaminar mensajes por rutas mínimas y el otro como vía de escape.

El algoritmo *minimal adaptive* ha sido comparado con el algoritmo *up*/down** [Sil00]. Incluso se ha evaluado dicho algoritmo de encaminamiento bajo carga real [Fli99a]. Los resultados de dichas evaluaciones indican que el algoritmo *minimal adaptive* mejora sustancialmente en prestaciones al algoritmo *up*/down**. Estas mejoras son debidas a que el algoritmo *minimal adaptive* utiliza en la mayoría de los

casos rutas mínimas y equilibra mejor el tráfico de red aportando a la red una mayor adaptatividad. Sin embargo, el algoritmo de encaminamiento *minimal adaptive* no puede ser implementado en redes con encaminamiento fuente pues, en estas redes, los conmutadores no pueden reencaminar los mensajes por las vías de escape. De hecho, poco esfuerzo se ha realizado en las redes con encaminamiento fuente para proveerlas de cierta adaptatividad y de rutas mínimas. Algunos estudios en este sentido se han traducido en los algoritmos *DFS* [San00] y *smart-routing* [Che95].

Por lo tanto, cabe pensar que la naturaleza de las redes con encaminamiento fuente limita las posibilidades de encaminamiento que tienen los algoritmos. Por consiguiente, vamos a identificar los principales problemas derivados del encaminamiento fuente.

1.4.1 Falta de adaptatividad

En el encaminamiento fuente los conmutadores encaminan mensajes a partir de la información que encuentran en la cabecera del mensaje. Esta información viene prefijada desde el nodo origen, por lo que los conmutadores no tienen la posibilidad de reencaminar los mensajes por rutas menos congestionadas en ese momento. Si una zona de la red está congestionada, un mensaje no podrá ser reencaminado para evitar la zona congestionada. Más aún, se incrementará dicha congestión. Por otra parte, en las redes con encaminamiento distribuido, los conmutadores sí tienen esta posibilidad, por lo que pueden desviar los mensajes por rutas alternativas, evitando posibles zonas congestionadas de la red e incluso aliviándolas.

En la figuras 18 y 19 podemos ver un ejemplo. Cuando un mensaje sale del nodo origen en una red con encaminamiento fuente (figura 18), el mensaje sigue una ruta predeterminada y no puede ser cambiada. Vemos que el mensaje entra en una zona congestionada. Por otra parte, en una red con encaminamiento distribuido (figura 19), el mensaje puede ser reencaminado dinámicamente por los conmutadores evitando la zona congestionada.

Otro problema adicional es que en el encaminamiento fuente los nodos suelen calcular solamente una ruta para cada par origen-destino. No obstante, el poder utilizar varias rutas alternativas hacia un mismo destino podría conllevar el obtener un mejor equilibrado, ya que todo el tráfico generado por un nodo hacia un destino podría ser distribuido entre varias rutas.

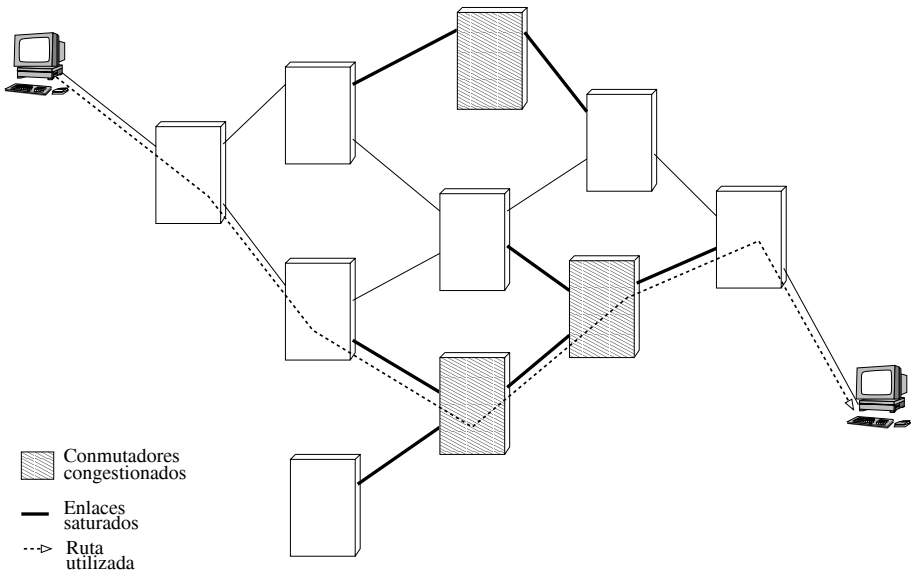


Figura 18: Problema de la adaptatividad en las redes con encaminamiento fuente.

1.4.2 Restricciones de encaminamiento

En las redes con encaminamiento fuente los algoritmos de encaminamiento con un GDC cíclico y rutas de escape [Dua93, Dua95] no pueden utilizarse, puesto que en el momento de calcular la ruta no se sabe si se tendrán que utilizar las rutas de escape. Por lo tanto, la única forma de garantizar la ausencia de bloqueo es mediante un algoritmo de encaminamiento con un GDC acíclico. Por ello, en la fase de cálculo de rutas, los algoritmos de encaminamiento deben garantizar que el conjunto de rutas resultante no introduzca ciclos en el GDC. Para conseguirlo, los algoritmos introducen más restricciones de encaminamiento que los algoritmos diseñados para redes con encaminamiento distribuido, en los cuales sí se pueden introducir ciclos (siempre y cuando se faciliten rutas de escape).

Por ejemplo, en la figura 20 podemos ver las restricciones (transiciones *down-up*) que aparecen en una red irregular cuando aplicamos el algoritmo de encaminamiento *up*/down**. Vemos que aparecen 14 restricciones de encaminamiento en la red (recuérdese que se utilizan enlaces bidireccionales). Al tener 3 enlaces conectados a cada conmutador, tenemos 6 posibles dependencias entre canales en cada conmutador (6 combinaciones entre los 3 canales). Al tener 8 conmutadores, tenemos 48 dependencias posibles. De ellas, 14 no las podemos utilizar, por lo que el algoritmo

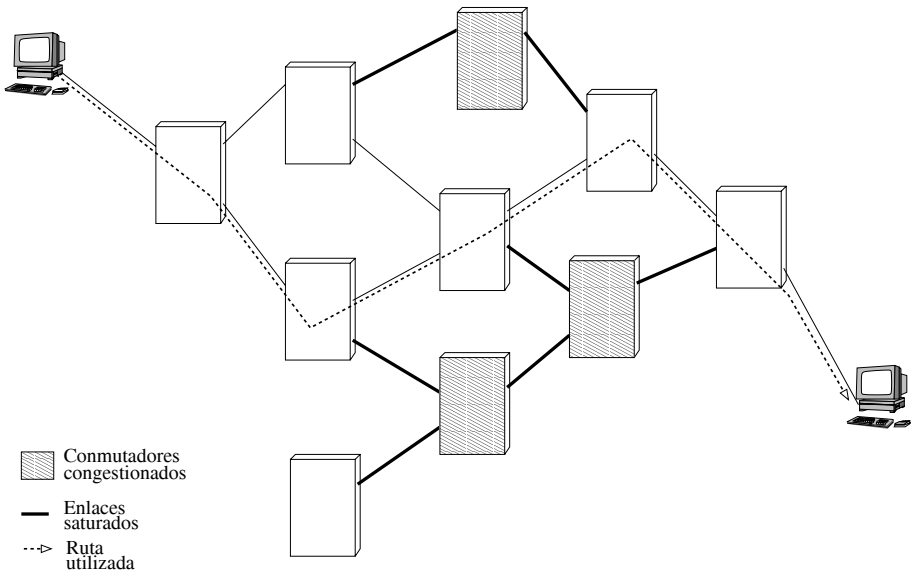


Figura 19: Adaptatividad en las redes con encaminamiento distribuido.

de encaminamiento *up*/down** en este ejemplo tiene un 30% de restricciones.

Además, pueden aparecer algunos conmutadores que no encaminen ningún mensaje, en nuestro ejemplo los conmutadores 6 y 7. Estos conmutadores tan sólo recibirán mensajes de los otros conmutadores que van destinados a nodos conectados directamente al conmutador.

Las restricciones de encaminamiento dependen en gran medida del algoritmo de encaminamiento utilizado. Por ejemplo, el algoritmo *DFS* mejora en prestaciones al algoritmo *up*/down** ya que introduce menos restricciones [San00]. No obstante, siempre van a perjudicar más las restricciones al utilizar algoritmos con GDC acíclicos que en los algoritmos con GDC cíclicos con rutas de escape ya que en los últimos, las restricciones se aplican solamente a las rutas de escape.

1.4.3 Desequilibrado del tráfico

Al existir más restricciones de encaminamiento, el algoritmo de encaminamiento tiende a desequilibrar el tráfico. Debido a que el algoritmo *up*/down** está basado en un árbol, tiende a saturar la zona cercana al nodo raíz, desequilibrando el tráfico. La figura 21 muestra la utilización de los enlaces asumiendo una distribución uniforme de destinos. Se muestran con diferentes grosores de línea el número de rutas

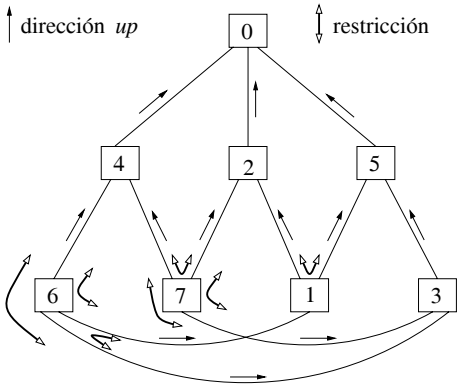


Figura 20: Restricciones de encaminamiento con el algoritmo *up*/down**.

que pasan por cada enlace teniendo en cuenta una ruta (la más corta posible según *up*/down**) para cada par origen-destino. En particular, los enlaces conectados a los conmutadores 6 y 7 tienen poco tráfico ya que estos conmutadores no pueden encaminar mensajes debido a las restricciones de encaminamiento. También observamos como los enlaces conectados al nodo raíz están altamente utilizados. El efecto del desequilibrado es mucho más importante cuando se combina con el mecanismo de conmutación *wormhole* y su efecto de contención. Más aún, al aumentar el tamaño de la red, el desequilibrado es mayor, ya que van a existir más restricciones.

Por otra parte, existen otros algoritmos de encaminamiento, como es el caso del algoritmo *smart-routing*, que obtienen un buen equilibrado del tráfico, pero por contra, su algoritmo de cálculo de rutas es muy costoso, no siendo posible aplicarlo para tamaños de redes elevados.

1.4.4 Rutas no mínimas

Debido a las restricciones introducidas por los algoritmos de encaminamiento, las rutas suelen ser más largas que la ruta mínima. En la figura 22 podemos ver como, para enviar mensajes desde el nodo conectado al conmutador 4 al nodo conectado al conmutador 1 (o viceversa) se utiliza la ruta *up*/down** válida que atraviesa los conmutadores 4-0-2-1. No obstante, esta ruta no es una ruta mínima ya que la ruta mínima es 4-6-1. Sin embargo, esta ruta no es válida para el algoritmo de encaminamiento *up*/down**. Esto conlleva que los mensajes van a utilizar por término medio más recursos en la red ya que el mensaje va a estar ocupando más canales y *buffers*

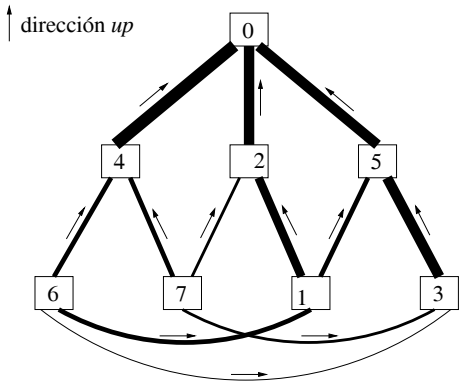


Figura 21: Desequilibrado de tráfico con el algoritmo *up*/down**.

asociados. Debido a esto, menos mensajes van a poder circular simultáneamente por la red al mismo tiempo con la consiguiente pérdida en prestaciones de la red. Nuevamente, al aumentar el tamaño de la red, este efecto será mayor ya que existirán más restricciones.

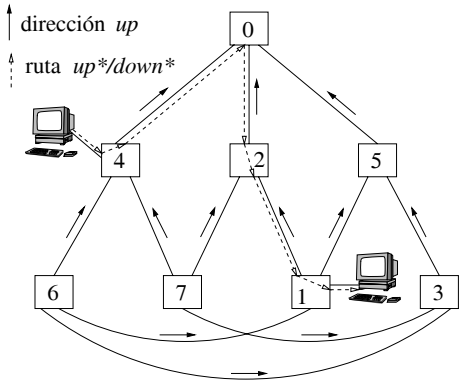


Figura 22: Ruta no mínima con el algoritmo *up*/down**.

1.4.5 Elevada contención

Al utilizarse conmutación *wormhole*, los mensajes, cuando se bloquean temporalmente en la red, se detienen ocupando muchos recursos (enlaces y *buffers*). Este bloqueo se propaga muy rápidamente, por lo que con relativamente poca carga la

red puede llegar a saturarse rápidamente. Este problema también ocurre en las redes con encaminamiento distribuido. Una solución sencilla a este problema es el uso de canales virtuales. Sin embargo, muchas de las implementaciones reales de redes para *clusters* no permiten el uso de canales virtuales.

1.4.6 Resumen

Como hemos visto, las redes con encaminamiento fuente sufren de varias limitaciones que afectan de una forma directa a las prestaciones. Muchas de ellas vienen derivadas por el elevado número de restricciones de encaminamiento impuestas por los algoritmos de encaminamiento para garantizar la ausencia de bloqueo. En las redes con encaminamiento distribuido es posible garantizar la ausencia de bloqueo imponiendo menos restricciones de encaminamiento.

En este trabajo nos centramos en el estudio de las redes con encaminamiento fuente. Teniendo en cuenta las limitaciones detectadas en las redes con encaminamiento fuente y con el objetivo de mejorar sus prestaciones, nos planteamos las siguientes preguntas:

- ¿Es posible implementar cierta adaptatividad?
- ¿Es posible reducir (o incluso eliminar) las restricciones de encaminamiento y asegurar al mismo tiempo la ausencia de bloqueos?
- ¿Es posible encontrar un algoritmo de equilibrado de carga escalable para redes con encaminamiento fuente?
- ¿Es posible la utilización de rutas mínimas para todos los pares origen-destino garantizando a la vez la ausencia de bloqueos?
- ¿Es posible reducir la contención de la red?
- ¿Cómo afectaría cada una de estas posibles mejoras a las prestaciones de las redes con encaminamiento fuente?

En la presente tesis vamos a tratar de responder a cada una de estas preguntas. Para ello vamos a implementar ciertos mecanismos que nos permitan mejorar las prestaciones de las redes con encaminamiento fuente. Un objetivo fundamental será obtener una solución sencilla y a la vez económica. La solución más económica pasa

por una implementación de los posibles mecanismos en *software* sin modificar el *hardware* actual de estas redes. En concreto, los mecanismos que propondremos se ubicarán en el interfaz de red, suponiendo la existencia de un procesador de red que ejecuta cierto código de control de la red (como ocurre en la red Myrinet) a la vez que ejecuta los mecanismos. A su vez, vamos a buscar una solución que se adapte fácilmente a cualquier red actual con encaminamiento fuente.

1.5 Objetivos de la tesis

Una vez vistos todos los problemas asociados a las redes con encaminamiento fuente, vamos a describir los objetivos de la presente tesis así como los mecanismos y técnicas que se proponen para alcanzar tales objetivos.

Con el fin de ofrecer una visión clara e intuitiva de las mejoras que se proponen en esta tesis, vamos a identificar previamente en un primer punto las diferentes etapas en que puede ser dividido el cálculo de rutas de los algoritmos de encaminamiento desarrollados para redes con encaminamiento fuente. En cada etapa identificaremos las limitaciones actuales y las mejoras que se van a proponer con el fin de aumentar las prestaciones de las redes con encaminamiento fuente.

1.5.1 Etapas en el cálculo de rutas de los algoritmos de encaminamiento tradicionales

La figura 23 muestra las diferentes etapas en que puede ser dividido el proceso de cálculo de rutas de los algoritmos de encaminamiento tradicionales. Por algoritmos de encaminamiento tradicionales nos referimos a los algoritmos que no introducen ciclos en el GDC, siendo pues válidos para redes con encaminamiento fuente (*up*/down**, *DFS* y *smart-routing*). En una primera etapa, y a partir de la topología de la red, el algoritmo de encaminamiento calcula el conjunto de rutas posibles entre todos los pares origen-destino. A continuación, en una segunda etapa, el algoritmo de encaminamiento selecciona, entre todo el conjunto, una ruta para cada par origen-destino. Este conjunto de rutas será el que se utilice para enviar mensajes por la red.

En ambas etapas existen una serie de limitaciones y condiciones que influyen en el cálculo y selección de rutas. Una primera condición que deben cumplir los algoritmos de encaminamiento en la etapa de cálculo de rutas, es el hecho de calcular

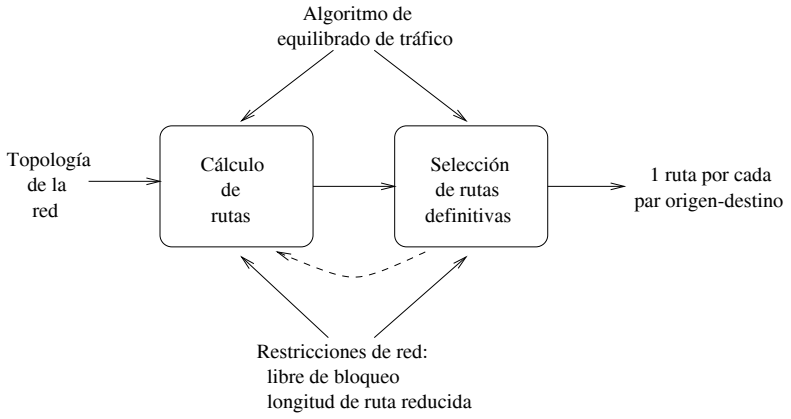


Figura 23: Etapas de los algoritmos de encaminamiento tradicionales para redes con encaminamiento fuente.

rutas lo más cortas posibles. No obstante, algunos algoritmos de encaminamiento puede que seleccionen rutas más largas en algún momento dado. Esto puede suceder cuando den más prioridad al equilibrado del tráfico que a la longitud de la ruta. El algoritmo de encaminamiento *smart-routing* es un ejemplo. Seleccionar rutas más largas también puede ser debido a las restricciones de encaminamiento. Los algoritmos *up*/down** y *DFS* se ven forzados muchas veces a seleccionar rutas más largas debido a que las rutas cortas no son válidas ya que necesitan utilizar dependencias no permitidas (restricciones).

En lo relativo a la limitación de que el conjunto de rutas final sea libre de bloqueo (GDC acíclico), los algoritmos *up*/down** y *DFS* cumplen esta limitación en la primera etapa, es decir, solamente calculan rutas que son libres de bloqueo. Sin embargo, no todos los algoritmos de encaminamiento imponen esta restricción en esta etapa. Este es el caso del algoritmo *smart-routing*. El algoritmo *smart-routing* primero calcula todas las rutas sin tener en cuenta la condición de red libre de bloqueo. Ya en la etapa de selección de rutas, el algoritmo seleccionará un conjunto de rutas que no introduzca ciclos en el GDC.

Por otra parte, en lo concerniente al equilibrado del tráfico, típicamente los algoritmos de encaminamiento seleccionan el conjunto de rutas finales intentando que bajo un hipotético tráfico (principalmente aleatorio) las rutas utilizarán todos los enlaces de una forma homogénea. Esta condición se suele atender en la segunda etapa (selección de rutas).

No obstante, el algoritmo de encaminamiento *smart-routing* utiliza otra aproximación más sofisticada para equilibrar el tráfico. El algoritmo *smart-routing* calcula en la primera etapa todo el conjunto de rutas (a ser posible rutas cortas) que hay entre cada par origen-destino. En la segunda etapa de selección de rutas busca entre las rutas calculadas aquel conjunto de rutas que equilibren el tráfico de una forma óptima. Sin embargo, dado que entonces pueden aparecer ciclos en el GDC, el algoritmo *smart-routing* es recursivo, en el sentido de que vuelve a la etapa anterior para calcular y seleccionar nuevas rutas que no introduzcan ciclos. Esta vuelta atrás entre las dos etapas hace que el algoritmo *smart-routing* equilibre muy bien el tráfico puesto que busca todas las posibles soluciones. No obstante, el tiempo de cálculo de rutas es prohibitivo al aumentar el tamaño de la red. De hecho, en recientes estudios, no se consiguieron rutas *smart-routing* para redes de más de 32 conmutadores debido al excesivo tiempo de cálculo [Fli00f].

Finalmente, los algoritmos de encaminamiento obtienen una ruta para cada par origen-destino. Este conjunto de rutas es libre de bloqueo y ha sido seleccionado teniendo en cuenta un hipotético equilibrado de tráfico.

Seguidamente describimos las diferentes mejoras que se proponen en la tesis para las redes con encaminamiento fuente. La descripción de estas propuestas se realiza también a través de diferentes etapas en el cálculo y selección de rutas.

1.5.2 Rutas alternativas

Como hemos visto, los algoritmos de encaminamiento tradicionales obtienen sólo una ruta por cada par origen-destino. No obstante, para un par origen-destino pueden existir varias rutas diferentes. Por lo tanto, una primera mejora podría ser el poder utilizar más de una ruta para cada par origen-destino. La figura 24 muestra las etapas de los algoritmos de encaminamiento utilizando esta primera mejora.

Ahora, los algoritmos de encaminamiento, en vez de seleccionar sólo una ruta para cada par origen-destino, obtienen diferentes rutas (n rutas). La forma de obtener las n rutas será la misma que la utilizada hasta este momento por el algoritmo de encaminamiento utilizado. A su vez, el conjunto final de rutas debe mantener la condición de un GDC acíclico.

Debido a que ahora un nodo va a tener diferentes rutas alternativas para enviar mensajes a un destino, es necesario que, en tiempo de utilización de rutas, el nodo seleccione una ruta para enviar el mensaje. Por lo tanto, es necesario utilizar un

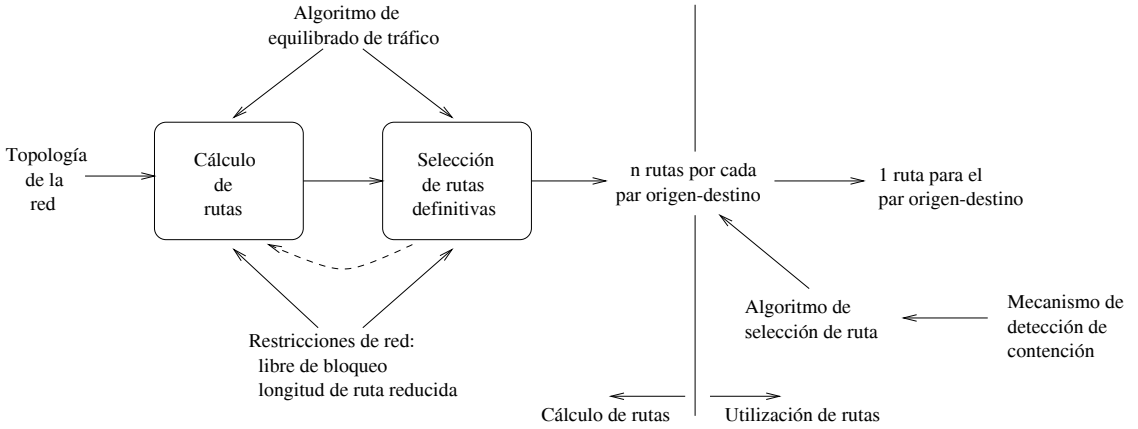


Figura 24: Primera mejora: varias rutas alternativas.

algoritmo de selección de rutas. Una primera posibilidad que se obtiene de utilizar varias rutas es la de poder distribuir el tráfico en la red de una forma más homogénea entre las rutas alternativas. Para ello, el algoritmo de selección de rutas debe tratar de distribuir el tráfico destinado a un determinado destino entre todas sus rutas alternativas. Por lo tanto, el algoritmo seleccionará las rutas de forma aleatoria o de forma ordenada (por ejemplo con una política de selección tipo *round-robin*).

Por otra parte, utilizar varias rutas también nos puede permitir el obtener cierta adaptatividad en la red. Para ello, el algoritmo de selección debería tener cierta información del estado actual del tráfico en la red para poder seleccionar una ruta u otra, con el fin de poder evitar posibles zonas congestionadas en la red (adaptatividad). No obstante, los nodos no tienen tal información (la tienen los conmutadores). Sin embargo, a partir de información local al nodo, vamos a diseñar un mecanismo que nos aportará cierta información de la contención existente en una ruta determinada. Por lo tanto, con este mecanismo de detección de contención, el algoritmo seleccionará las rutas en función del tráfico de la red. En el capítulo 2 se describen con detalle dichos algoritmos y mecanismos.

1.5.3 Eliminación de dependencias

No obstante, incluso utilizando varias rutas alternativas, vamos a seguir sufriendo de los principales problemas que acucian a las redes con encaminamiento fuente (rutas no mínimas, desequilibrado de tráfico y elevada contención). Debido a que estos

problemas están directamente relacionados con el problema de las restricciones de encaminamiento para evitar ciertas dependencias, una segunda mejora pasa por la eliminación de estas dependencias.

Básicamente, para garantizar la ausencia de bloqueo, los algoritmos de encaminamiento evitan la aparición de ciclos en el GDC impidiendo la utilización consecutiva de algunos pares de canales. Estos dos canales originan una dependencia que, de incluirse en el GDC, formarían un ciclo. Por lo tanto, decimos que entre estos canales hay una dependencia prohibida. Es decir, las rutas se calculan de tal forma que ninguna ruta utilizará consecutivamente los dos canales que generan dependencias prohibidas en el GDC. Vamos a proponer un mecanismo que romperá las dependencias entre canales. Brevemente, el mecanismo va a consistir en que los mensajes que necesiten utilizar dos canales que forman una dependencia prohibida van a ser redirigidos a ciertos nodos conectados en los conmutadores donde se encuentra la dependencia. Dichos nodos extraerán el mensaje de la red y posteriormente lo reinyectarán hacia el destino final. Debido a que el mensaje entra por completo en un nodo, la dependencia entre los dos canales se elimina. Este mecanismo lo denominaremos ITB (*in-transit buffers*) haciendo referencia a los elementos de almacenamiento (*buffers*) que se utilizan en los nodos intermedios (*in-transit*). En la capítulo 3 describimos con detalle el mecanismo.

Utilizando el nuevo mecanismo ITB, los algoritmos podrán tener las etapas mostradas en la figura 25. Como podemos observar, la principal diferencia con las etapas de los algoritmos tradicionales (figura 23) es que ahora, los algoritmos pueden calcular rutas que antes no eran válidas (ahora lo son ya que se utilizan ITBs). El equilibrado de rutas es el mismo que el utilizado en los algoritmos tradicionales. Sin embargo, el algoritmo de selección de rutas puede cambiar ya que ahora tiene un factor adicional: priorizar la utilización de rutas con ITBs o sin ellos. En el capítulo 5 veremos diferentes criterios de selección de rutas en función de la utilización de ITBs.

1.5.4 Nuevo algoritmo de encaminamiento

La utilización de ITBs sobre algoritmos de encaminamiento tradicionales elimina las dependencias prohibidas por dichos algoritmos. Sin embargo, el mecanismo ITB nos permite una mayor flexibilidad, ya que permite utilizar cualquier conjunto de rutas, eliminando los ciclos en el GDC insertando los ITBs necesarios. Por lo tanto,

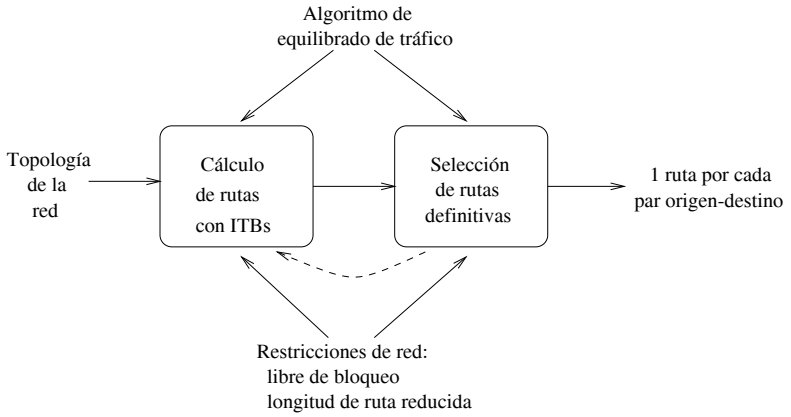


Figura 25: Segunda mejora: eliminando restricciones con ITBs.

podemos diseñar un nuevo algoritmo de encaminamiento que aproveche por completo el mecanismo propuesto.

Suponiendo el mecanismo ITB para eliminar las dependencias que originan ciclos en el GDC, nuestro nuevo algoritmo de encaminamiento podría tener las etapas mostradas en la figura 26.

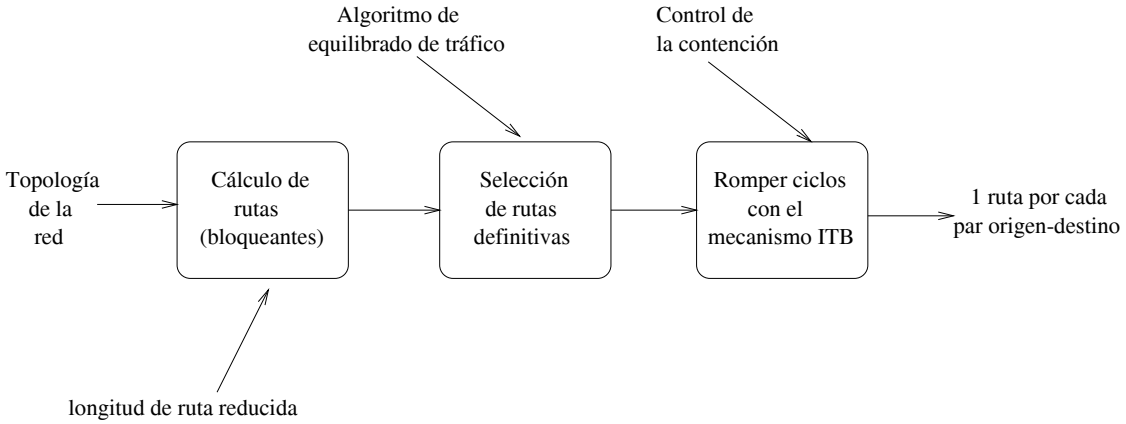


Figura 26: Tercera mejora: nuevo algoritmo de encaminamiento con ITBs.

Ya que vamos a disponer de un mecanismo para romper ciclos en el GDC, en la primera etapa de cálculo de rutas ahora tenemos la posibilidad de calcular muchas más rutas alternativas para cada par origen-destino, ya que no tenemos la restricción

de que sean libres de bloqueo (en una etapa posterior se eliminarán los posibles ciclos). Además, gracias a que no tenemos tal restricción, podemos calcular rutas más cortas, con la consiguiente mejora implícita en la menor contención de red que obtendremos, ya que los mensajes utilizarán, por término medio, menos recursos. Por consiguiente, esta primera etapa solamente está influenciada por la condición de que las rutas calculadas deben ser cortas (mínimas a ser posible).

En la segunda etapa, se seleccionan las rutas definitivas que vamos a utilizar. En este caso, vamos a seleccionar una ruta por cada par origen-destino. En esta etapa se aplica el algoritmo de equilibrado de tráfico. Puesto que vamos a tener más rutas diferentes, vamos a poder equilibrar mucho mejor sin la necesidad de la recursividad que necesita el algoritmo de encaminamiento *smart-routing*, ya que no tenemos la restricción de que el conjunto final de rutas no tenga ciclos en el GDC como en *smart-routing*. Por lo tanto, eliminamos la vuelta atrás en el cálculo y selección de rutas equilibradas, reduciendo el tiempo de cálculo de rutas. A su vez, podremos equilibrar más el tráfico que en los algoritmos *up*/down** y *DFS* ya que el equilibrado en estos algoritmos está muy influenciado por las restricciones de encaminamiento impuestas.

Una vez tenemos el conjunto de rutas definitivas, entramos en la etapa de rotura de ciclos. Mediante la utilización de un algoritmo específico se romperán los ciclos utilizando el mecanismo propuesto (ITB) de eliminación de dependencias, de tal forma que no haya ciclos en el GDC.

Debido a que el mecanismo ITB extrae temporalmente mensajes de la red, es lógico pensar que el mecanismo influye también en la contención de red ya que los mensajes liberan los recursos que están ocupando y, por consiguiente, permiten que otros mensajes los utilicen, avanzando así a través de la red. Por lo tanto, existe una relación directa entre la contención de red y el número de ITBs que pondremos en la red. Por consiguiente, el algoritmo de rotura de ciclos actuará también sobre la contención mientras rompe ciclos (insertando más ITBs o menos).

Al final, el algoritmo obtiene un conjunto de rutas libre de bloqueo con una ruta por cada par origen-destino.

Como podemos observar, gracias al mecanismo de eliminación de dependencias (ITB) podemos proponer un nuevo algoritmo de encaminamiento que trata de evitar los principales inconvenientes de las redes con encaminamiento fuente. Primero, las rutas van a ser cortas ya que la condición de ausencia de bloqueo no es tenida aún en cuenta. Segundo, el equilibrado no va a estar influenciado por las restricciones

de encaminamiento, por lo que podemos obtener un buen equilibrado. Y tercero, la contención va a poder ser disminuida gracias a la naturaleza del mecanismo, que extrae temporalmente los mensajes de la red.

1.5.5 Resumen de las propuestas de mejora

Como hemos visto, en la presente tesis buscamos mejorar las prestaciones de las redes de interconexión con encaminamiento fuente con distintas soluciones. Resumiendo, las propuestas que se plantean en esta tesis para mejorar las prestaciones de tales redes son:

- Cálculo y selección de rutas alternativas para una mejor distribución del tráfico en la red y obtención de cierta adaptatividad combinando la disponibilidad de rutas alternativas con un mecanismo de detección de contención.
- Eliminación de las restricciones de encaminamiento con un nuevo mecanismo (*in-transit buffers*; ITB). Este mecanismo vamos a utilizarlo para diversos fines:
  - Aplicación del mecanismo sobre algoritmos tradicionales como *up*/down**, *DFS* y *smart-routing*. Con ello, conseguimos eliminar las restricciones impuestas por estos algoritmos de encaminamiento.
  - Desarrollo de un nuevo algoritmo de encaminamiento que aproveche todas las posibilidades de encaminamiento del mecanismo ITB. Por lo tanto, el nuevo algoritmo estará centrado en obtener rutas mínimas, equilibrar eficientemente el tráfico y reducir la contención de red.

Por último se propondrá una implementación eficiente del mecanismo ITB. En los siguientes capítulos se detallan los nuevos mecanismos y algoritmos que posteriormente serán evaluados en el capítulo 5.

Capítulo 2

Encaminamiento con múltiples rutas alternativas

En este capítulo vamos a presentar los mecanismos propuestos con el fin de añadir cierta adaptatividad a las redes con encaminamiento fuente. Estos mecanismos estarán ubicados en los interfaces de red y se basarán en la selección de rutas de entre un conjunto de rutas alternativas.

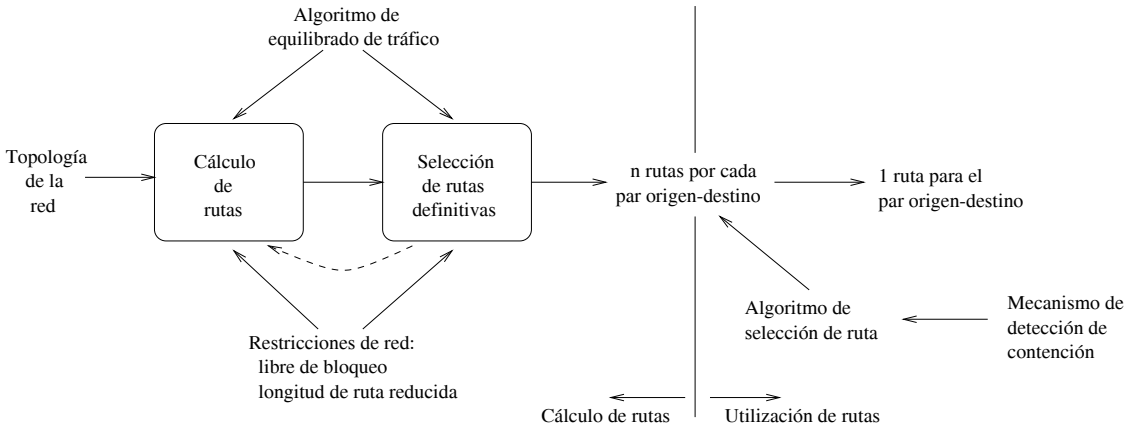


Figura 27: Etapas en los algoritmos con múltiples rutas alternativas.

Adicionalmente, se presenta un mecanismo de detección de contención ubicado también en el interfaz de red. Este mecanismo se utilizará para seleccionar la ruta definitiva en tiempo de ejecución.

2.1 Problema de la falta de adaptatividad

En las redes con encaminamiento distribuido, los conmutadores tienen la posibilidad de reencaminar mensajes. La decisión la toman en función del tráfico que circula en ese momento por la red, estimándolo a partir de información local al conmutador. Habitualmente, se realiza una simple comprobación del estado actual de los canales de salida. El estado puede ser libre u ocupado. Los conmutadores poseen internamente una tabla de encaminamiento en la que tienen el conjunto de canales de salida que pueden utilizar para un mensaje con un destino concreto y que ha entrado por un canal de entrada determinado. En función del estado de los canales de salida encontrados en la tabla se selecciona uno de estos.

En las redes con encaminamiento fuente, los conmutadores no tienen posibilidad alguna de reencaminar mensajes. Tan sólo se limitan a aplicar las instrucciones de encaminamiento que encuentran en las cabeceras de los mensajes. Estas instrucciones de encaminamiento son prefijadas por el nodo que envía el mensaje. Los nodos poseen tablas de encaminamiento en las que figuran las rutas a seguir para cada destino.

Por lo tanto, asumiendo que no modificamos el *hardware* de la red, los únicos elementos con poder de encaminamiento van a ser los propios nodos, por lo que la adaptatividad en redes con encaminamiento fuente debe ser implementada en estos.

Vamos a describir a continuación las diferentes etapas (figura 27) que contemplan los nuevos mecanismos: cálculo y selección de rutas disponibles, algoritmo de selección de rutas y mecanismo de detección de contención.

2.2 Cálculo y selección de rutas disponibles

Los algoritmos de encaminamiento tradicionales solamente almacenan una ruta para cada destino en las tablas de encaminamiento. No obstante, en una red pueden existir múltiples opciones de encaminamiento. La figura 28 ilustra este hecho. Para ir del nodo conectado al conmutador 4 al nodo conectado al conmutador 1 existen dos rutas *up*/down** válidas (enlaces 4-0-2-1 y enlaces 4-0-5-1). Como vemos, las dos rutas comparten varios enlaces, en concreto el enlace que une los conmutadores 4 y 0. Si uno (o ambos) de los enlaces 0-2 y 2-1 están muy utilizados (por otros nodos) entonces, la ruta 4-0-5-1 será una mejor alternativa que la ruta 4-0-2-1. No obstante, si el enlace 4-0 tiene una elevada utilización, entonces ninguna de las dos

rutas será mejor que la otra. Por lo tanto, si una ruta alternativa comparte un alto porcentaje de enlaces con otra ruta entonces, a efectos de prestaciones, las dos rutas se comportarán de forma similar.

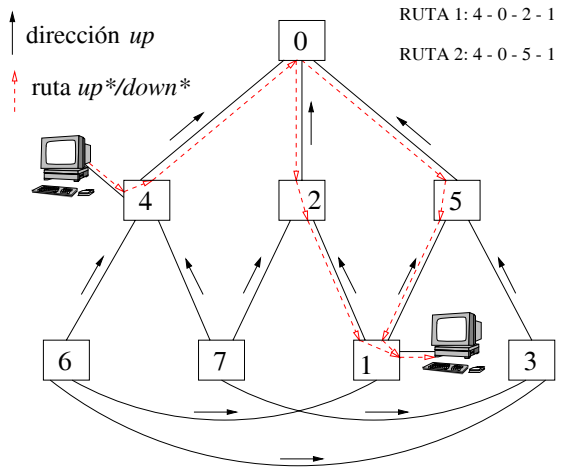


Figura 28: Múltiples rutas *up*/down** en una red irregular.

Encontrar rutas que sean altamente disjuntas para un par origen-destino depende de la topología de la red y de las restricciones del algoritmo de encaminamiento. En las redes irregulares es difícil encontrar un número elevado de rutas con un grado de compartición reducido, pero por contra, en las redes regulares es mucho más fácil. Por ejemplo, en la figura 29 podemos ver como en una malla 2D, utilizándose el algoritmo de encaminamiento *up*/down**, existen múltiples rutas entre cada par origen-destino. Debido a la regularidad de la topología, estas rutas comparten pocos enlaces, incluso podemos obtener 2 rutas sin ningún enlace compartido[1] para determinados pares origen-destino. Sin embargo, en las redes irregulares, conforme aumenta el tamaño de la red, existen menos rutas disjuntas, ya que la propia irregularidad, las restricciones de encaminamiento y la conveniencia de utilizar rutas cortas tiende a unificar las rutas alternativas. Por lo tanto, hay un mayor grado de compartición en las redes irregulares. Debido a que se pretende mejorar las prestaciones de las redes con encaminamiento fuente en cualquier topología (regulares e irregulares), en la presente tesis no vamos a tener en cuenta reducir el grado de compartición de las rutas en el proceso de cálculo y selección de las rutas disponibles,

[1] Considerando únicamente enlaces entre conmutadores.

realizándose dicha selección de forma aleatoria, en su caso.

Asimismo, para cada par origen-destino pueden existir multitud de diferentes rutas alternativas de diferente longitud. Sin embargo, utilizar rutas de mayor longitud que las rutas más cortas posibles para un par origen-destino conlleva elevar la contención de red. Por lo tanto, en el proceso de cálculo y selección de rutas alternativas nos centraremos solamente en las rutas más cortas posibles. Por último, para determinados pares origen-destino puede existir un número elevado de rutas alternativas (especialmente en redes regulares). En tales casos, vamos a limitar el número de rutas posibles a 10.

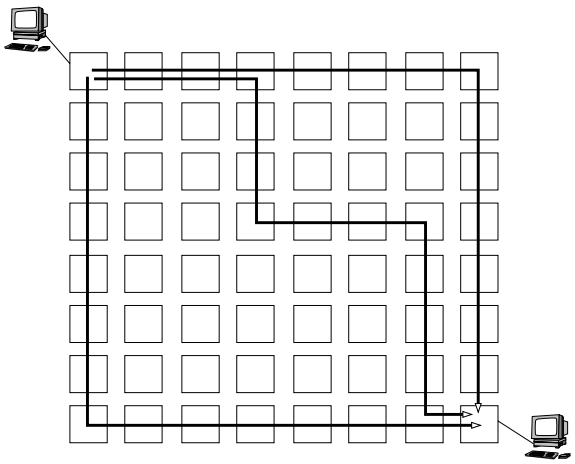


Figura 29: Múltiples alternativas en una malla 2D.

2.3 Algoritmo de selección de rutas

Con distintas rutas alternativas disponibles, los nodos necesitan, en tiempo de envío de mensajes, de un algoritmo de selección para seleccionar la ruta a utilizar de entre las rutas disponibles. Esta selección de ruta debe ser realizada en función de unos criterios basados en diferentes objetivos. Los objetivos que podemos buscar son:

- **Un mejor equilibrado del tráfico**. Al tener varias rutas alternativas para un mismo destino, un nodo puede utilizar todas las rutas con el fin de que el tráfico inyectado hacia ese destino esté lo más distribuido posible entre

todas las rutas alternativas disponibles. Para conseguir esto podemos utilizar diferentes criterios de selección de rutas:

- **Seleccionar aleatoriamente una ruta**. Cada vez que el nodo tiene que enviar un mensaje seleccionará de forma aleatoria la ruta a utilizar del conjunto de rutas disponibles para el destino del mensaje.
- **Seleccionar equitativamente todas las rutas**. Con este criterio, todas las rutas hacia el destino se seleccionan por turnos, siguiendo una política *round-robin*.

La implementación por *software* de estos dos algoritmos de selección es muy sencilla. Por ejemplo, para el algoritmo de selección aleatorio, cada vez que el nodo tiene que enviar un mensaje a un destino determinado genera un índice aleatorio comprendido entre la primera y la última ruta disponible para encaminar hacia el destino. Seguidamente, utiliza la ruta encontrada en la entrada correspondiente al índice calculado. En lo que respecta al algoritmo de selección *round-robin* es necesario añadir un nuevo campo (*siguiente*) por cada entrada (ruta) en la tabla de encaminamiento. Dicho campo indicará la siguiente ruta a utilizar (del conjunto de rutas hacia el mismo destino). Cada vez que el nodo tenga que enviar a un destino determinado, utilizará la ruta con el campo *siguiente* activado. Seguidamente, (mientras se envía el mensaje) el nodo desmarcará el campo *siguiente* de la ruta utilizada marcando la siguiente ruta a utilizar. En la figura 30 podemos ver la nueva tabla de encaminamiento necesaria para utilizar el algoritmo de selección de rutas *round-robin*.

- **Adaptatividad en la red**. La selección de rutas se realizará teniendo en cuenta la carga de la red. Para ello, necesitamos de un mecanismo que nos indique el estado de la red en un momento dado. Con este mecanismo mediremos un cierto nivel de tráfico para cada ruta que utilicemos. Utilizando el nivel de tráfico obtenido podremos conmutar entre las rutas y seleccionar la ruta que ofrezca un nivel de tráfico menor. En la siguiente sección describimos el mecanismo de detección de contención que utilizaremos para tal fin.

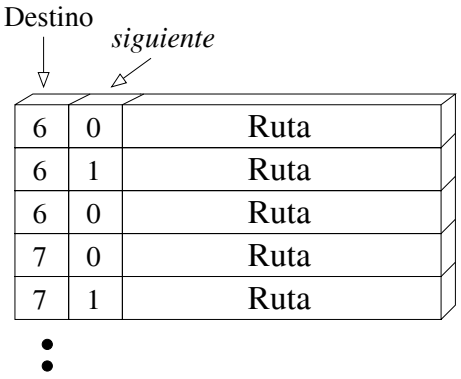


Figura 30: Nueva tabla de rutas para el algoritmo de selección de rutas *round-robin*.

2.4 Mecanismo de detección de contención

Al ser los nodos los que hacen la selección de rutas, será en éstos donde se deba ubicar el mecanismo y por lo tanto, la única información en la que podemos basarnos son los mensajes enviados y recibidos. El mecanismo que proponemos se basa en la medición del tiempo de inyección de los mensajes enviados.

De hecho, cuando el interfaz de red envía un mensaje a través de una ruta puede obtener una idea del estado de la contención en dicha ruta en función del tiempo de inyección del mensaje. En la figura 31 vemos como, cuando sale el mensaje por una ruta determinada, el interfaz de red anota el tiempo de inicio de inyección (T_{ini}). El mensaje va saliendo. Los conmutadores poseen *buffers* a la entrada de cada canal. El tamaño de estos *buffers* está diseñado para desacoplar la entrada de la salida por lo que el mensaje, si no encuentra contención en la red, no se va a detener en su avance. Por lo tanto, si no hay contención en la red, el mensaje va a inyectarse de una forma continuada sin pausas. Cuando el mensaje sale por completo (figura 32) el interfaz de red anota el tiempo de fin de la inyección (T_{fin}).

El periodo de tiempo de inyección ($T_{fin} - T_{ini}$) será utilizado para estimar el grado de contención a lo largo de la ruta. En el caso de que no haya contención, el periodo de tiempo de inyección debería ser:

$$T_{fin} - T_{ini} = \frac{Tam_m}{Tasa_{inyeccion}}$$

donde $Tasa_{inyeccion}$ es la tasa de inyección del nodo (la velocidad a la que se inyecta información en la red) y Tam_m es el tamaño del mensaje.

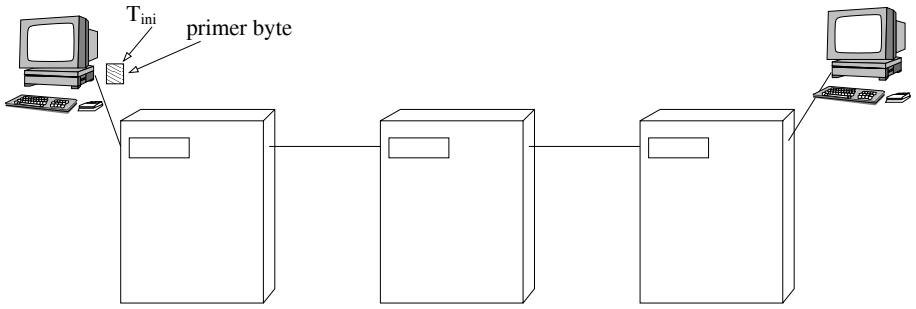


Figura 31: Toma de tiempos de inyección: el mensaje empieza a salir del nodo origen.

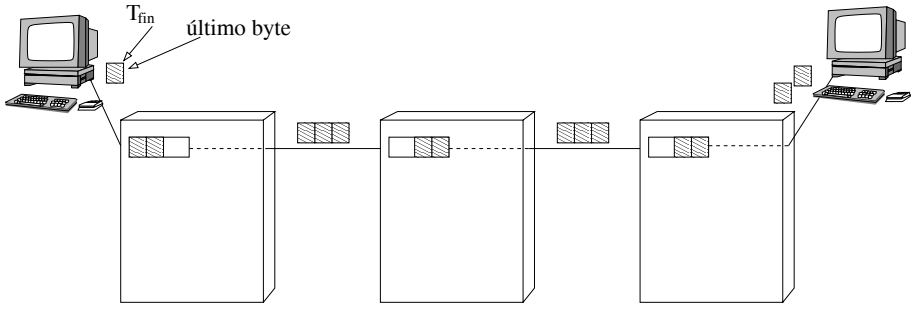


Figura 32: Toma de tiempos de inyección: el mensaje sale por completo del nodo origen.

En el caso de que el mensaje encontrase contención a lo largo de la ruta, el periodo de tiempo de inyección sería mayor. Por ejemplo, en la figura 33 vemos que el mensaje se detiene en la red. De hecho, el mensaje se detiene un cierto tiempo en el segundo conmutador debido a que el canal que solicita está ocupado por otro mensaje (contención). Por lo tanto, el tiempo de inyección del mensaje en este caso será mayor puesto que incluye el tiempo en que el mensaje está detenido, ya que el mensaje no ha salido por completo.

Hay que hacer constar que en el caso de que el mensaje ya haya salido por completo del nodo, la posible contención que encuentre el mensaje a lo largo de la ruta no podrá ser detectada. No obstante, al aumentar la contención en una ruta aumentarán también las probabilidades de detectar dicha contención, ya que los mensajes enviados por dicha ruta irán formando una cola hasta llegar al nodo

origen. Por lo tanto, con una sencilla medición del tiempo de inyección de los mensajes, un nodo puede obtener cierta información sobre la contención en la red a lo largo de una ruta.

A partir de los tiempos de inicio y fin de inyección se calculará un coeficiente de contención C_c, que será anotado junto a la ruta utilizada. Dicho coeficiente de contención deberá ser calculado de forma que el valor obtenido sea independiente del tamaño de mensaje y de la tasa de inyección del nodo. A partir de este coeficiente, el nodo seleccionará la ruta con un coeficiente de contención menor de entre todas las rutas alternativas con el fin de evitar rutas más congestionadas que otras.

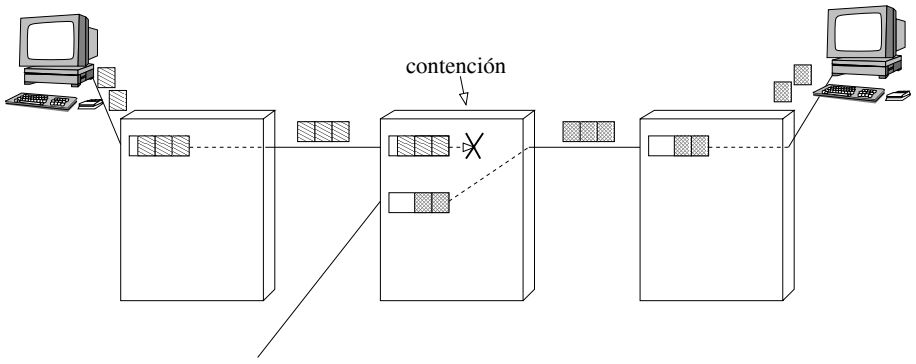


Figura 33: Toma de tiempos de inyección: el mensaje encuentra contención en la ruta.

Por último, hay que hacer constar que las rutas que ofrecen una elevada contención en un momento determinado puede que después de un periodo de tiempo no muy largo vuelvan a ofrecer una baja contención. Por lo tanto, el mecanismo debe permitir a estas rutas con un coeficiente de contención elevado la posibilidad de que reduzcan dicho coeficiente con el fin de que puedan ser utilizadas de nuevo. Para ello, se define una constante de olvido c_o. La constante de olvido será la cantidad en que se reducirá el coeficiente de contención asignado a una ruta. Dicha constante será aplicada a cada ruta cada vez que se encamine un mensaje hacia el destino de dicha ruta. Por lo tanto, en un tiempo finito, todas las rutas reducirán su coeficiente de contención hasta un nivel en donde podrán, de nuevo, ser utilizadas para encaminar mensajes.

Resumiendo, el mecanismo de detección se basa en los siguientes parámetros y acciones:

- **Cálculo del tiempo de inicio de inyección**. Tras cada inicio de envío de un mensaje a través de una ruta, el mecanismo anotará junto a la ruta el tiempo de inicio de la inyección (T_{ini}).

- **Cálculo del tiempo de fin de inyección**. Al finalizar la inyección de un mensaje, el mecanismo considerará el tiempo actual como tiempo de fin de inyección (T_{fin}) y lo utilizará para calcular el coeficiente de contención de la ruta.

- **Cálculo del coeficiente de contención asociado a una ruta**. El coeficiente de contención se calculará de la siguiente forma:

$$C_c = Tasa_{inyeccion} * \frac{T_{fin} - T_{ini}}{Tam_m}$$

  donde $Tasa_{inyeccion}$ es la velocidad con la que el nodo puede inyectar la información a la red, Tam_m es el tamaño del mensaje inyectado por la ruta y T_{ini} y T_{fin} son los tiempos de inicio y fin de inyección, respectivamente. En nuestro caso asumimos que el nodo inyectará a la red con una tasa de 1 byte por ciclo, por lo que $Tasa_{inyeccion} = 1$. Hay que hacer constar que al dividir el periodo de tiempo de inyección por el tamaño del mensaje, obtenemos un coeficiente de contención independiente del tamaño del mensaje. Como puede observarse, en el caso de que no haya contención, el tiempo de inyección ($T_{fin} - T_{ini}$) coincidirá con el tamaño del mensaje (Tam_m), por lo que el coeficiente de contención (C_c) será la unidad.

  Al finalizar el envío de un mensaje a través de una ruta, el mecanismo calculará el coeficiente de contención (C_c) a partir del tiempo de inicio de inyección (T_{ini}) anotado en la ruta, el tiempo de fin de inyección (T_{fin}) y el tamaño del mensaje (Tam_m). Este coeficiente será anotado junto a la ruta.

- **Selección de ruta**. El algoritmo de selección de ruta seleccionará, de entre todas las rutas hacia el mismo destino, aquella que ofrezca un coeficiente de contención C_c menor.

- **Constante de olvido**. Cada vez que se utilice una ruta para un determinado par origen-destino, el mecanismo decrementará los coeficientes de contención de todas las rutas alternativas hacia el mismo destino en una cierta constante

(c_o). De esta forma, si una ruta que estaba muy congestionada (con un C_c elevado) se libera, tendrá nuevamente oportunidad de ser utilizada. En la evaluación del mecanismo (ver sección 5), se utilizará una constante de olvido de $c_o = 0.1$.

En la figura 34 podemos observar la tabla de rutas que contempla los nuevos campos necesarios (C_c y T_{ini}) para el mecanismo de detección de contención.

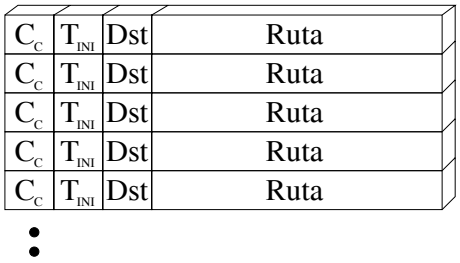


Figura 34: Nueva tabla de rutas para el mecanismo de detección de contención.

---

procedimiento Inicio-Envío(Dst : natural)
var
  i : natural
inicio
  seleccionar ruta i hacia destino Dst con $Rutas[i].C_c$ menor
  anotar T_{actual} en $Rutas[i].T_{ini}$
  enviar mensaje por ruta $Rutas[i].Ruta$
fin procedimiento

Figura 35: Mecanismo de detección de contención: procedimiento de inicio de envío de mensajes.

Por su parte, las figuras 35 y 36 muestran el código del mecanismo de detección de contención. Como podemos observar, el mecanismo se instrumenta en los procedimientos de envío y de fin de envío de un mensaje. En la figura 35 se muestra como el mecanismo, antes de enviar un mensaje, selecciona la ruta a utilizar (i) entre las rutas alternativas para el destino. El algoritmo selecciona la ruta que ofrece un menor coeficiente de contención (C_c). Una vez seleccionada la ruta, el mecanismo

```
procedimiento Fin-Envío(i : ruta, Tam_m : natural)
var
   j : natural
   T_fin : natural
inicio
   T_fin = T_actual
   Rutas[i].C_c = (T_fin − Rutas[i].T_ini) / Tam_m
   para cada ruta alternativa j hacia destino Rutas[i].Dst y j <> i
      Rutas[j].C_c = Rutas[j].C_c − c_o
      si Rutas[j].C_c < 1.0 entonces
         Rutas[j].C_c = 1.0
      fsi
   fpara
fin procedimiento
```

Figura 36: Mecanismo de detección de contención: procedimiento de fin de envío de mensajes.

anota el tiempo actual (T_{ini}) en la tabla de rutas (en la entrada asociada a la ruta i). A continuación, envía el mensaje utilizando la ruta i.

Una vez se ha enviado por completo el mensaje (figura 36), el mecanismo actualiza el coeficiente de contención C_c de la ruta recientemente utilizada (i). A continuación, el mecanismo procede a decrementar los coeficientes de contención de todas las rutas alternativas con la constante de olvido c_o con el fin de que puedan entrar nuevamente en juego.

Capítulo 3

Buffers en tránsito (ITBs)

En este capítulo presentamos el mecanismo de eliminación de dependencias denominado ITB (*in-transit buffers*). En primer lugar, introducimos el mecanismo, viendo en detalle su funcionamiento y cómo elimina las dependencias. Posteriormente, vemos un ejemplo de aplicación sobre el algoritmo de encaminamiento *up*/down**. Asimismo, analizamos también los posibles inconvenientes del mecanismo.

Seguidamente, analizamos cómo se puede aplicar el mecanismo sobre los distintos algoritmos de encaminamiento tradicionales, así como presentamos un algoritmo de encaminamiento totalmente nuevo basado en el uso exclusivo del mecanismo. Por último, se describen aspectos de implementación del mecanismo sobre la red Myrinet [Bod95] y en particular sobre el *software* de comunicaciones GM de Myricom [GM].

3.1 Mecanismo de buffers en tránsito

Para que los algoritmos de encaminamiento en redes con encaminamiento fuente sean libres de bloqueo, no deben introducir ciclos en el grafo de dependencias de canales (GDC).

En la figura 37 vemos una situación de bloqueo en una red con mecanismo de conmutación *wormhole*. En esta situación, hay cuatro mensajes m1, m2, m3 y m4 destinados a los nodos 3, 4, 1 y 2, respectivamente ocupando cada uno un *buffer* (canal) en la red y esperando a que el siguiente *buffer* (canal) a utilizar se libere. El ciclo se forma cuando los cuatro mensajes están esperando a que se libere un *buffer* utilizado por otro de los mensajes. Los mensajes se quedan esperando indefinidamente a que se liberen los *buffers*.

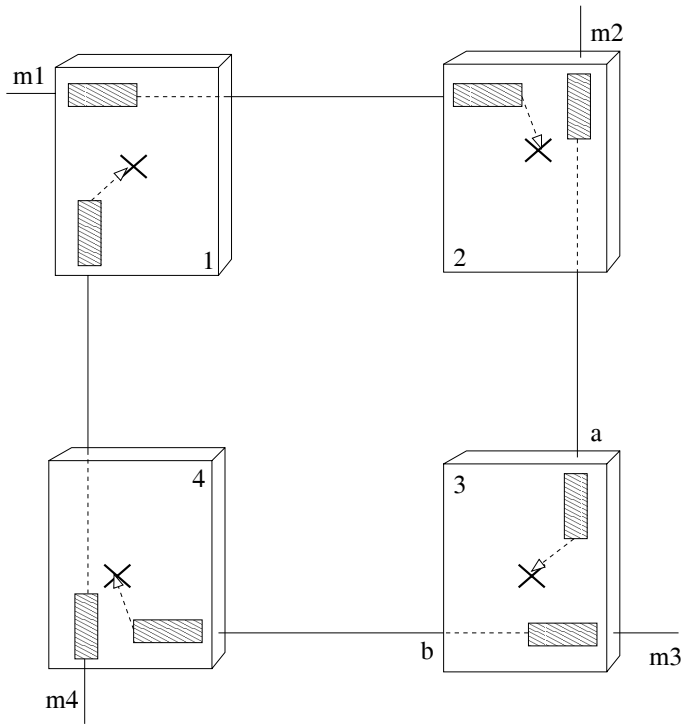


Figura 37: Bloqueo en una red.

Para evitar esta situación, los algoritmos de encaminamiento aplican algunas restricciones en la utilización de canales. Por ejemplo, una restricción que elimina el ciclo de la figura 37 es que ningún mensaje que utilice el canal a asociado al conmutador 3 pueda pedir el canal b en dirección al conmutador 4. Con esto se consigue que no se forme el ciclo y que en un tiempo finito todos los mensajes lleguen a sus destinos.

Básicamente, las restricciones eliminan dependencias entre canales. Hay una dependencia entre dos canales x e y, si existe un mensaje que utilizando el *buffer* asociado al canal x solicita el *buffer* asociado al canal y. En la figura 37 hay dependencias entre todos los canales, las cuales forman un ciclo entre ellas.

3.1.1 Eliminación de dependencias

El mecanismo que proponemos va a eliminar las dependencias entre canales sin restringir el encaminamiento.

La idea principal del mecanismo es la de extraer el mensaje temporalmente de la red y después reintroducirlo. En la figura 38 podemos ver como uno de los mensajes que antes se bloqueaba (m2), es redirigido a un nodo conectado al conmutador 3. Este nodo reinyectará el mensaje posteriormente a la red hacia el destino final. Este nodo lo denominamos *nodo en tránsito*. El mecanismo recibe el nombre de ITB (*in-transit buffer*) debido a que almacena temporalmente mensajes que están en tránsito por la red.

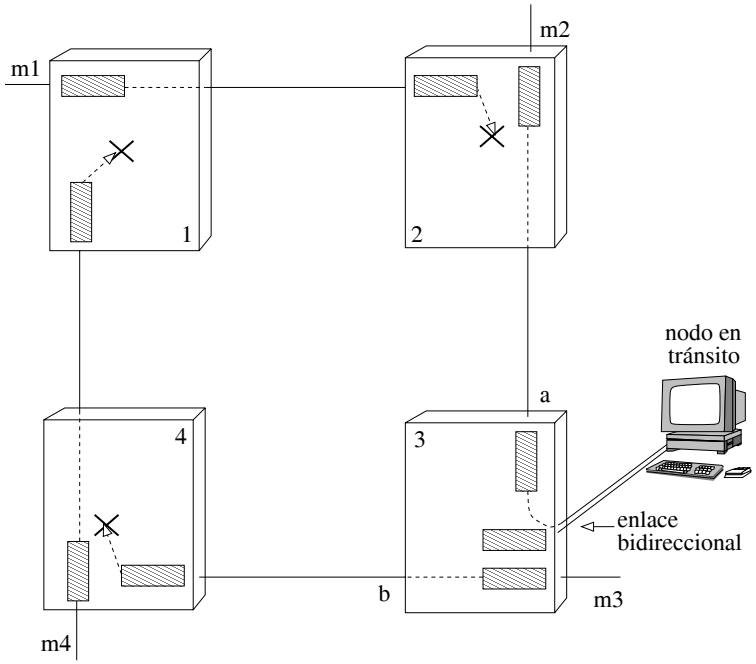


Figura 38: Rotura de ciclo en una red.

Hay que hacer constar que el espacio de almacenamiento para los mensajes en tránsito no tiene porqué estar ubicado en la memoria local del nodo de procesamiento. Si el interfaz de red dispone de cierta capacidad de memoria, esta puede ser utilizada con este fin. De hecho, esta aproximación es más eficiente al no interferir el mecanismo con el procesador local.

Como podemos observar, hemos introducido dos nuevos canales entre los canales que tenían la dependencia: el canal que va hacia el *nodo en tránsito* y el canal que proviene del *nodo en tránsito*. No obstante, para eliminar totalmente la dependencia entre los dos canales, se debe asegurar que el mensaje pueda salir por completo de

la red incluso si el segundo canal de la dependencia (b) permanece ocupado por un tiempo prolongado. Por lo tanto, en la implementación de este mecanismo, el nodo seleccionado deberá reservar espacio para poder almacenar todo el mensaje, si fuera necesario.

Aún más, el nodo debe asegurar un espacio infinito de almacenamiento para que el mecanismo sea totalmente libre de bloqueo. Esto es debido a que el segundo canal de la dependencia podría estar muy saturado y durante este tiempo podrían llegar más mensajes que tuvieran que ser extraídos de la red. Estos mensajes se almacenarían temporalmente en el *nodo en tránsito*. Si llegara un momento en que un mensaje no cupiese en el *nodo en tránsito*, este se bloquearía ocupando el primer canal de la dependencia (canal a en la figura 38). En ese momento se podría producir nuevamente un bloqueo en la red. No obstante, debido a que vamos a utilizar elementos de procesamiento como *nodos en tránsito*, podemos utilizar la memoria local del nodo en el caso de que la memoria para almacenar buffers en tránsito en el interfaz de red se agote (aunque esto puede resultar crítico en términos de prestaciones).

3.1.2 Ejemplo de aplicación

Veamos un ejemplo concreto de aplicación de este mecanismo. Para ello, vamos a aplicar el mecanismo sobre el algoritmo de encaminamiento *up*/down**. Como sabemos, en este algoritmo se evitan los ciclos impidiendo que los mensajes puedan utilizar canales etiquetados como *up* después de haber utilizado canales etiquetados como *down* (dependencias *down-up*). En la figura 39 podemos ver la asignación de etiquetas de dirección a los canales y la ruta calculada desde el nodo conectado al conmutador 4 hasta el nodo conectado al conmutador 1. Vemos que la ruta sigue las reglas de encaminamiento *up*/down**. La ruta atraviesa los conmutadores 4-0-2-1. Sin embargo, esta ruta no es una ruta mínima. La ruta que atraviesa los conmutadores 4-6-1 sí lo es. Esta ruta mínima no está permitida por el algoritmo *up*/down** ya que existe una restricción de encaminamiento (dependencia entre canales *down-up* no permitida) en el conmutador 6.

Vamos ahora a aplicar el mecanismo ITB a esta ruta. En la figura 40 vemos que ahora, gracias al mecanismo, para encaminar un mensaje desde el nodo conectado al conmutador 4 al nodo conectado al conmutador 1 se utiliza la ruta mínima que

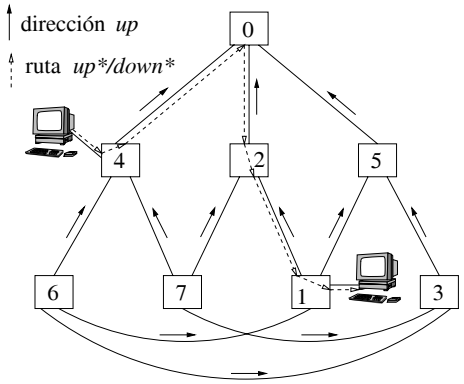


Figura 39: Ruta *up*/down** válida.

atraviesa los conmutadores 4-6-1. Para eliminar la dependencia *down-up* en el conmutador 6 se selecciona uno de los nodos conectados al mismo. Los mensajes que se envíen por esta ruta serán enviados directamente al *nodo en tránsito*. Este nodo recibirá el mensaje y detectará que el mensaje es un mensaje en tránsito, por lo que, en cuanto pueda, reenviará el mensaje hacia el destino final (nodo conectado al conmutador 1). Con esto conseguimos eliminar la dependencia entre los dos canales y que la ruta mínima 4-6-1 se pueda utilizar.

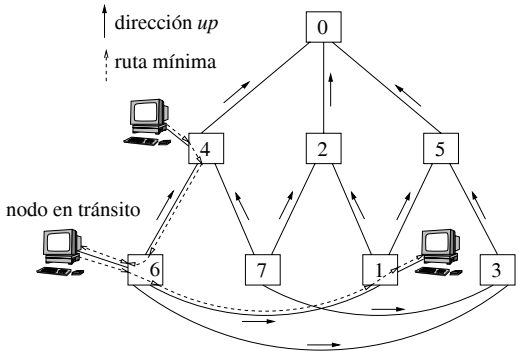


Figura 40: Aplicación del mecanismo ITB a *up*/down**.

3.1.3 Selección del nodo en tránsito

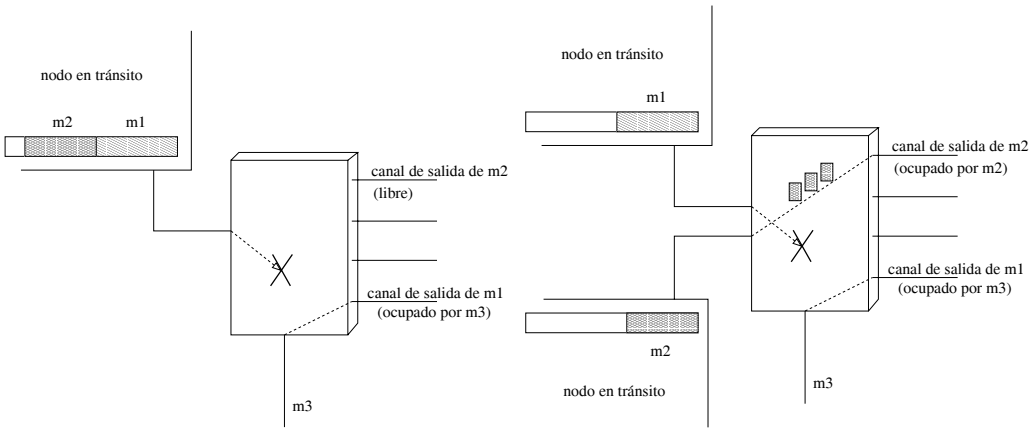
Como hemos visto, el mecanismo se basa en la utilización de un *nodo en tránsito* conectado a un conmutador. No obstante, en un conmutador pueden existir varios nodos conectados. Asimismo, pueden existir multitud de rutas que requieran un ITB en dicho conmutador, cada una entrando por cualquier canal de entrada (de tipo *down* en *up*/down**) y saliendo por cualquier canal de salida (de tipo *up* en *up*/down**). Por lo tanto, es necesario realizar a priori una selección del nodo que atenderá cada ruta con necesidad de ITB en un conmutador.

Esta selección de nodo puede influir en las prestaciones generales del mecanismo, ya que un uso desmesurado de un *nodo en tránsito* (con muchos mensajes en tránsito utilizando dicho nodo) podría ocasionar el desbordamiento de la memoria disponible en el nodo para mensajes en tránsito. Una solución sencilla, para minimizar el posible desbordamiento de los *buffers* en un nodo, es la de repartir equitativamente todas las rutas con necesidad de ITBs en un conmutador entre todos los nodos conectados a dicho conmutador.

No obstante, la asignación de rutas con ITBs a los nodos de un conmutador también podría afectar a las prestaciones globales del mecanismo. Esto es debido a que el mecanismo no puede procesar simultáneamente más de un mensaje. Es decir, cuando un *nodo en tránsito* recibe dos mensajes en tránsito, el primer mensaje sale del conmutador (inyectado por el *nodo en tránsito*) antes que el segundo mensaje. Es más, el segundo mensaje empieza a salir del *nodo en tránsito* una vez ha salido por completo el primer mensaje del *nodo en tránsito*.

En la figura 41.a podemos ver como esta propiedad del mecanismo puede ser un inconveniente cuando dos mensajes utilicen el mismo *nodo en tránsito* y después salgan por dos canales distintos del conmutador. Si el primer mensaje encuentra contención en el canal de salida, dicha contención afectará a los dos mensajes. Por otra parte, si los dos mensajes fueran atendidos por dos *nodos en tránsito* distintos conectados al mismo conmutador (figura 41.b), entonces se podrían procesar simultáneamente y el segundo mensaje no estaría afectado por la contención encontrada por el primer mensaje.

Por lo tanto, puede parecer que en la asignación del *nodo en tránsito* a una ruta debe considerarse el canal de salida del mensaje, con el fin de evitar que el *nodo en tránsito* actúe como un cuello de botella para distintos mensajes que utilizan diferentes canales de salida del conmutador. Esta asignación la denominados



(a) (b)

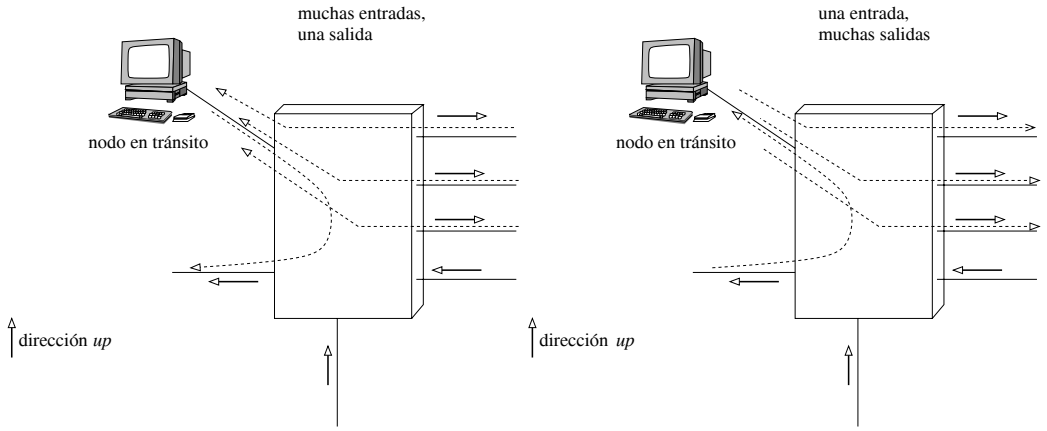
Figura 41: Problema de la contención entre dos mensajes en tránsito.

asignación por flujo de salida. En dicha asignación, un nodo atiende exclusivamente las rutas con necesidad de ITBs en dicho conmutador que salgan por un canal determinado. Es obvio que para evitar completamente el problema que un mismo *nodo en tránsito* procese mensajes que salen por distintos canales del conmutador, es necesario que exista un número mayor o igual de nodos que de canales de salida hacia otros conmutadores.

Sin embargo, esta forma de asignar los nodos a las rutas con ITBs tiene el inconveniente de que puede sobrecargar un nodo con demasiados mensajes en tránsito, ya que puede darse el caso de que todos los ITBs se necesiten en un conmutador para rutas que entran por varios canales (de tipo *down* en *up*/down**) y salgan todos por un único canal (de tipo *up* en *up*/down**) del conmutador. En la figura 42.a podemos ver esta situación.

Por lo tanto, a partir de este inconveniente, se podría considerar que la selección del nodo a una ruta puede estar también en función del canal de entrada utilizado por los mensajes en tránsito al conmutador. Esta asignación la denominamos *asignación por flujo de entrada.* En dicha asignación, un nodo atenderá exclusivamente mensajes que entren por un determinado canal de entrada. Al igual que en la asignación por flujo de salida, es necesario, para un óptimo comportamiento, que exista el mismo número de nodos (o mayor) que de canales de entrada desde otros conmutadores.

Sin embargo, tal y como podemos ver en la figura 42.b pueden existir conmutadores en donde todos los mensajes con necesidad de ITBs entren por un único canal de entrada (de tipo *down* en *up*/down**) y salgan por varios canales de salida (de tipo *up* en *up*/down**) del conmutador, y alguno esté ocupado. Por lo tanto, esta asignación puede suponer la existencia de un cuello de botella en el *nodo en tránsito* y su posible desbordamiento.



(a) (b)

Figura 42: Problema de la asignación de ITBs por flujos de entrada y salida.

Estas dos alternativas (flujo de entrada y flujo de salida) han sido evaluadas junto a una asignación aleatoria del *nodo en tránsito*. Se ha comprobado que las diferencias en prestaciones eran mínimas y que no se producía desbordamiento en la memoria de ningún *nodo en tránsito* en cargas de tráfico por debajo de la saturación de la red. Por lo tanto, se ha estimado que el único criterio a tener en cuenta para seleccionar el *nodo en tránsito* es el de distribuir aleatoriamente todas las rutas con necesidad de ITB entre todos los nodos del conmutador.

3.1.4 Prioridades entre los mensajes locales y los mensajes en tránsito

En los *nodos en tránsito* existirán dos colas de inyección de mensajes. En una primera cola tendremos los mensajes locales inyectados por el *host* mientras que en una segunda cola tendremos los mensajes en tránsito que se van recibiendo y

hay que reinyectar. Por lo tanto, debe existir un mecanismo que seleccione en cada momento el mensaje que debe ser reinyectado. O lo que es lo mismo, se debe definir la prioridad de una cola respecto de la otra.

Por un lado, los mensajes en tránsito deben ser inyectados con la mayor rapidez posible para minimizar un posible incremento en su latencia. Por otro lado, si los mensajes en tránsito tienen la máxima prioridad, la inyección de los mensajes locales se verá perjudicada incluso llegando a la situación de que ningún mensaje local pueda ser inyectado, por lo que se penalizará severamente al *host* del *nodo en tránsito*. Por lo tanto, se debe definir un criterio para seleccionar el mensaje a inyectar de entre las dos colas en cada momento.

El criterio utilizado es el de asignar diferentes niveles de prioridad a los mensajes en tránsito en función del número de mensajes locales encolados. Es decir, ante una situación en que existan pocos mensajes locales encolados, los mensajes en tránsito tendrán la máxima prioridad. Sin embargo, si el número de mensajes se incrementa, se reducirá la prioridad de los mensajes en tránsito, por lo que los mensajes locales empezarán a inyectarse intercalados con los mensajes en tránsito. En el caso extremo de que la cola de mensajes locales sea muy grande, los mensajes locales tendrán la misma prioridad que los mensajes en tránsito por lo que se inyectarán a la red con la misma periodicidad.

La implementación del mecanismo de arbitraje entre mensajes locales y mensajes en tránsito se basará en un contador de los mensajes en tránsito inyectados desde que se inyectó el último mensaje local. Con este contador y con el número de mensajes locales encolados, el mecanismo de arbitraje seleccionará el tipo de mensaje (local o en tránsito) que será inyectado. La figura 43 muestra el código del mecanismo de arbitraje utilizado. La función *Enviar-Mensaje-Local* devuelve *Verdadero* si, ante la existencia de mensajes en tránsito, se puede enviar un mensaje local. Por lo tanto, dicha función se llamará siempre y cuando exista algún mensaje en tránsito. En caso de que la cola de mensajes en tránsito esté vacía, se inyectará un mensaje local (si existe) sin necesidad de llamar a esta función. En dicha función se utiliza la cuenta de mensajes en tránsito enviados desde el último mensaje local enviado (N_{itb}). Si el número de mensajes locales es igual o inferior a 50 se inyectará un mensaje local cada 4 mensajes en tránsito. Por el contrario, si el número de mensajes locales está entre 50 y 100 se inyectará un mensaje local cada 2 mensajes en tránsito. Finalmente, si el número de mensajes locales encolados es mayor de 100, se inyectará un mensaje local y un mensaje en tránsito de forma alternada (con la misma probabilidad).

```
función Enviar-Mensaje-Local(N_itb : natural) : lógico
inicio
    si size(buffer_local) > 0
       si size(buffer_local) <= 50 Y N_itb > 3
          N_itb = 0
          devolver Verdadero
       sino
          si size(buffer_local) <= 100 Y N_itb > 1
             N_itb = 0
             devolver Verdadero
          sino
             si N_itb > 0
                N_itb = 0
                devolver Verdadero
             fsi
          fsi
       fsi
    fsi
    N_itb = N_itb + 1
    devolver Falso
fin función
```

Figura 43: Selección de tipo de mensaje (local o en tránsito) a inyectar.

Con este mecanismo damos la máxima prioridad a los mensajes en tránsito cuando el número de mensajes locales es reducido y contemplamos la posibilidad de que los mensajes locales se acumulen en exceso, situación en la que el mecanismo permite una mayor inyección de mensajes locales.

3.1.5 Inconvenientes potenciales del mecanismo

El mecanismo de buffers en tránsito introduce algunos inconvenientes. En primer lugar, los mensajes que utilizan el mecanismo van a utilizar más recursos en la red. Utilizarán dos canales adicionales además de atravesar un mismo conmutador dos veces. Adicionalmente, consumirán recursos del *nodo en tránsito* (memoria y procesamiento).

Por otra parte, el mecanismo añade cierta sobrecarga a los mensajes que pasan por el *nodo en tránsito*, por lo que la latencia media de los mensajes puede sufrir un

aumento. Por ello, deberá realizarse una implementación eficiente del mecanismo para minimizar este impacto en la latencia. La principal consideración a tener en cuenta para minimizar la latencia añadida por el mecanismo es la de poder reinyectar el mensaje tan pronto como se detecte que es un mensaje en tránsito, incluso antes de que el mensaje llegue por completo al *nodo en tránsito*. No obstante, no hay que olvidar que, aún así, el mensaje debería poder extraerse totalmente de la red en caso necesario. En la sección 3.4 se proponen pautas de implementación sobre Myrinet para minimizar dichos inconvenientes en latencia.

Además, para soportar el mecanismo en un conmutador determinado, debe garantizarse que haya nodos conectados al conmutador. No obstante, esta es una situación típica en muchas redes donde se distribuyen todos los nodos entre todos los conmutadores para obtener una utilización uniforme en toda la red. Por otra parte, si algún conmutador no posee nodos conectados, el algoritmo de encaminamiento puede no insertar ITBs en ese nodo o bien utilizar las rutas originales sin ITBs que pasen por esos conmutadores.

Sin embargo, pese a estos posibles inconvenientes, que deben ser evaluados, no hay que olvidar las ventajas que introduce el mecanismo. Estas ventajas se detallan a continuación en la siguiente sección.

3.2 Aplicación de los ITBs

En esta sección vamos a comentar los beneficios y alternativas de diseño que nos aporta el mecanismo de buffers en tránsito. Básicamente, vamos a tener dos formas de utilizar el mecanismo. En primer lugar, podemos aplicar el mecanismo de buffers en tránsito sobre los algoritmos de encaminamiento tradicionales. La idea es utilizar los buffers en tránsito para eliminar las dependencias prohibidas por los algoritmos de encaminamiento tradicionales. El objetivo que perseguimos con esta utilización de los ITBs es el de proveer rutas mínimas entre los pares de nodos para los cuales los algoritmos tradicionales proporcionan rutas no mínimas. Este será el caso de los algoritmos de encaminamiento *up*/down** y *DFS*.

Por otra parte, también podemos aplicar el mecanismo de buffers en tránsito sobre *smart-routing*. En este caso, la idea es ligeramente distinta. *Smart-routing* proporciona un conjunto de rutas que equilibran mucho el tráfico por la red y además son libres de bloqueo. Sin embargo, el inconveniente es el elevado tiempo de cálculo

de dichas rutas, debido a que en muchas ocasiones la decisión tomada para equilibrar el tráfico es incompatible con la de ausencia de bloqueo y hay que volver a calcular (*backtracking*). En este caso, la idea es que *smart-routing* proporcione rutas en las que se haya intentado optimizar el equilibrado de carga, pero ignorando el efecto de los bloqueos. Posteriormente, se calculan las dependencias entre los canales y se traza el GDC. En caso de que existan ciclos, se rompen eliminando alguna(s) dependencia(s), insertando para ello buffers en tránsito en algún(algunos) conmutador(es). De esta forma, al reducir la complejidad del cálculo realizado por *smart-routing*, al equilibrar la carga e ignorar el efecto sobre los bloqueos, se elimina su principal inconveniente: el elevado tiempo de cálculo que hace imposible la obtención de rutas para redes de tamaño medio-alto.

Otra forma de utilizar el mecanismo ITB es diseñar un algoritmo de encaminamiento completamente nuevo, basado exclusivamente en el mecanismo y que intente obtener las máximas prestaciones. Dicho algoritmo tendrá como objetivos la utilización de rutas mínimas, la obtención de un buen equilibrado del tráfico y la reducción de la contención en la red.

3.2.1 ITBs sobre *up*/down** y *DFS*

Como se ha comentado anteriormente, el objetivo principal del mecanismo ITB es la eliminación de dependencias prohibidas por los algoritmos de encaminamiento tradicionales. Por lo tanto, las rutas prohibidas en estos algoritmos de encaminamiento podrán ahora ser utilizadas.

Una gran ventaja de utilizar estas rutas anteriormente prohibidas es el hecho de garantizar la existencia de rutas mínimas para todos los pares origen-destino de la red. Con ello, los mensajes utilizarán el mínimo número de recursos de la red (sin tener en cuenta los recursos utilizados por el propio mecanismo).

Por lo tanto, cuando se utilizan ITBs en los algoritmos de encaminamiento tradicionales, se deben calcular las rutas de una forma diferente. Ahora, los algoritmos deben calcular el conjunto de rutas mínimas para cada par origen-destino sin tener en cuenta las restricciones de encaminamiento. Posteriormente, los algoritmos aplicarán los ITBs necesarios para que todas las rutas calculadas sean libres de bloqueo. Todo este proceso correspondería a la etapa del cálculo de rutas con ITBs de la figura 44.

Por ejemplo, el algoritmo de encaminamiento *up*/down** con ITBs calculará

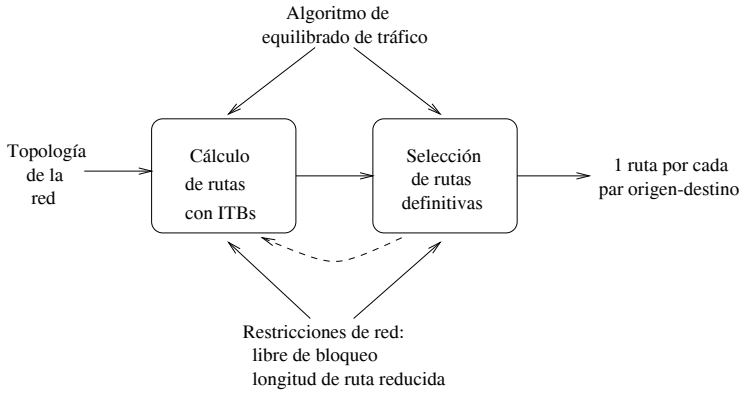


Figura 44: Etapas de los algoritmos tradicionales con ITBs.

rutas mínimas sin tener en cuenta las transiciones *down-up*. Si alguna ruta no cumple en algún punto de la ruta con las reglas de encaminamiento (transición *down-up*), entonces el algoritmo insertará un ITB para eliminar la dependencia prohibida.

Algunas rutas calculadas no necesitarán ITBs (serán rutas *up*/down** válidas), otras sí. Por lo tanto, una vez calculadas las rutas, pueden existir varias rutas alternativas con y sin ITBs para un determinado par origen-destino. En el momento de calcular las tablas de encaminamiento hay que seleccionar una u otra en función de algún criterio. Por lo tanto, tenemos ahora un nuevo criterio para la selección de rutas (segunda etapa de la figura 44). Podemos distinguir en este caso dos alternativas:

- **Utilizar el mínimo número de ITBs que garanticen rutas mínimas**. Con esta alternativa, el algoritmo de encaminamiento seleccionará para un determinado par origen-destino una ruta con ITBs si y sólo si no hay ninguna ruta para el mismo par origen-destino sin ITBs que sea mínima (ruta válida para el algoritmo de encaminamiento). Con esto aseguramos que los ITBs se utilizarán solamente cuando faciliten el uso de rutas mínimas.

- **Utilizar más ITBs**. Con esta alternativa, se van a utilizar más ITBs que los necesarios para asegurar ruta mínima para cada par origen-destino. Para ello, el algoritmo seleccionará de las rutas calculadas para un determinado par origen-destino una ruta al azar, pudiéndose seleccionar una ruta mínima con o sin ITBs. Hay que hacer constar que, con esta alternativa, pueden

existir rutas mínimas sin ITBs para un determinado par origen-destino y, sin embargo, utilizarse una ruta con ITBs.

Con la primera alternativa se pretende minimizar el efecto de la sobrecarga añadida por el mecanismo. De hecho, cuantos menos ITBs se utilicen, menor latencia añadirá el mecanismo (en términos medios). Por otra parte, con la segunda alternativa se pretende dar más flexibilidad al algoritmo de encaminamiento ya que ante dos rutas de la misma longitud, una sin ITBs y la otra con ITBs, puede que se utilice la ruta con ITBs (en la primera alternativa se utilizaría la ruta sin ITBs). Por ejemplo, podemos ver en la figura 45 como existen dos rutas de la misma longitud (ambas son rutas mínimas) para ir del nodo conectado al conmutador 4 al nodo conectado al conmutador 2. La ruta que pasa por el conmutador 7 necesita de un ITB para romper la transición *down-up*. Sin embargo, la ruta que pasa por el conmutador 0 no necesita ningún ITB ya que es una ruta *up*/down** válida. Con la primera alternativa se utilizará siempre la ruta que pasa por el conmutador 0 (conmutador raíz), sin embargo, con la segunda aproximación la ruta que pasa por el conmutador 7 con ITB puede que se utilice (se seleccionaría en el proceso de selección de rutas definitivas). En el caso de que sea seleccionada, descongestionará un poco el conmutador raíz ya que esta ruta no pasará por él. Por lo tanto, con la segunda alternativa menos rutas cumplirán con el algoritmo original *up*/down** y, por lo tanto, menos rutas pasarán por el nodo raíz por lo que se obtendrá un mejor equilibrado.

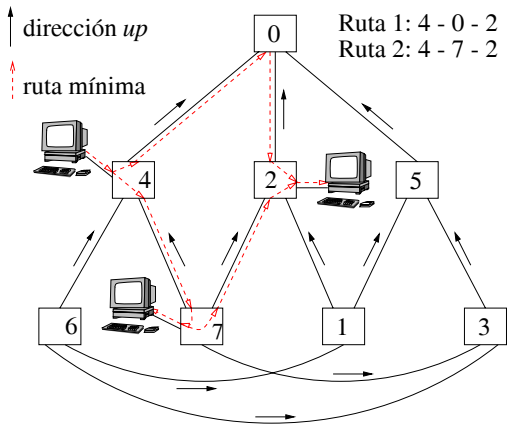


Figura 45: Dos rutas mínimas para un par origen-destino.

Estas dos alternativas serán evaluadas en el capítulo 5. En las figuras 46 y 47 podemos ver las dos alternativas aplicadas al algoritmo de encaminamiento *up*/down** con ITBs. Para el algoritmo de encaminamiento *DFS* se utilizará únicamente la segunda alternativa. El algoritmo de encaminamiento *DFS* con ITBs seguirá las mismas pautas que el algoritmo de la figura 47, pero eliminando ahora las restricciones impuestas por el algoritmo de encaminamiento *DFS*.

```
procedimiento Cálculo-Selección-Rutas()
var
   r_i : ruta
inicio
   para cada par origen-destino
      calcular rutas mínimas
   fpara
   para cada ruta r_i
      repetir
         si existe transición down-up
            insertar ITB
         fsi
      hasta no existe transición down-up
   fpara
   para cada par origen-destino
      si existe ruta r_i sin ITBs
         seleccionar ruta r_i
      sino
         seleccionar ruta r_i al azar
      fsi
   fpara
fin procedimiento
```

Figura 46: Cálculo y selección de rutas con ITBs mínimos sobre *up*/down**.

3.2.2 ITBs sobre *smart-routing*

Para aplicar el mecanismo sobre *smart-routing* se va a proceder de forma diferente. Con *smart-routing* también vamos a partir de las rutas calculadas, pero en este caso, teniendo en cuenta que disponemos del mecanismo ITB para romper dependencias entre canales, vamos a simplificar el algoritmo que *smart-routing* emplea para el

```
procedimiento Cálculo-Selección-Rutas()
var
   r_i : ruta
inicio
   para cada par origen-destino
      calcular rutas mínimas
   fpara
   para cada ruta r_i
      repetir
         si existe transición down-up
            insertar ITB
         fsi
      hasta no existe transición down-up
   fpara
   para cada par origen-destino
      seleccionar ruta r_i al azar
   fpara
fin procedimiento
```

Figura 47: Cálculo y selección de rutas con ITBs sobre *up*/down**.

cálculo de rutas. Es decir, con el mecanismo de buffers en tránsito, el algoritmo *smart-routing* calculará las rutas definitivas sin tener en cuenta la restricción de que el GDC tenga que ser acíclico. Posteriormente, los ciclos serán eliminados aplicando ITBs.

En la figura 48 podemos ver el algoritmo utilizado para calcular las rutas. En primer lugar, se calcula una ruta para cada par origen-destino con el algoritmo *smart-routing* sin tener en cuenta la condición de que el conjunto final de rutas sea libre de bloqueo. Después, para cada ruta calculada, se buscan y eliminan los posibles ciclos que pueda introducir en el GDC.

La detección y posible eliminación de ciclos a partir de una ruta es realizado por el procedimiento *Detectar-Eliminar-Ciclo*. Dicho procedimiento se muestra en la figura 49. En particular, se comprueba la existencia de posibles ciclos a lo largo de toda la ruta. En la figura 50 podemos ver un ejemplo de su aplicación. Una ruta se puede ver como el conjunto de pares conmutador-canal de salida que utiliza. Una ruta puede introducir ciclos a partir de cada par conmutador-canal. Existirá

```
procedimiento Cálculo-Rutas()
var
   r_i : ruta
inicio
   para cada par origen-destino
      calcular ruta smart con posibles ciclos
   fpara
   para cada ruta r_i
      Detectar-Eliminar-Ciclo(r_i)
   fpara
fin procedimiento
```

Figura 48: Cálculo de rutas *smart* con ITBs.

un ciclo a partir de un par conmutador-canal si partiendo de dicho par conmutador-canal y recorriendo todas las dependencias de todas las rutas que parten desde esta, llegamos en algún momento al mismo par conmutador-canal de partida. Por lo tanto, se tienen que analizar todos los pares conmutador-canal de la ruta. Si la función detecta algún ciclo a partir de un determinado par conmutador-canal, entonces inserta un ITB en dicho conmutador antes de utilizar el canal de salida, por lo que se rompe el ciclo detectado.

Como se puede observar, el algoritmo de cálculo de rutas (y posterior eliminación de ciclos) es muy sencillo. Además, dicha solución no utiliza el *backtracking* utilizado por el algoritmo original *smart-routing*. También cabe resaltar que el algoritmo de detección y eliminación de ciclos no minimiza el número de ITBs introducidos, ya que el número de ciclos detectados depende del orden en que se eliminan los ciclos y sobre todo de por donde se elimina el ciclo (en que par conmutador-canal del ciclo se ubica el ITB). Un algoritmo que garantizase el mínimo número de ITBs sería mucho más complejo (con *backtracking*). No obstante, este algoritmo introduce un número de ITBs reducido.

3.2.3 Nuevo algoritmo de encaminamiento

Como hemos visto, el mecanismo ITB puede ser utilizado para proveer rutas mínimas para todo par origen-destino. Sin embargo, este uso de los ITBs no explota todas las posibilidades que nos ofrece el mismo. Como hemos visto anteriormente, el

```
procedimiento Detectar-Eliminar-Ciclos(r : ruta)
inicio
    para cada conmutador c_i y canal e_i de r
        si existe ciclo a partir de c_i y e_i
            insertar ITB en r (c_i, e_i)
        fsi
    fpara
fin procedimiento
```

Figura 49: Algoritmo de búsqueda y eliminación de ciclos.

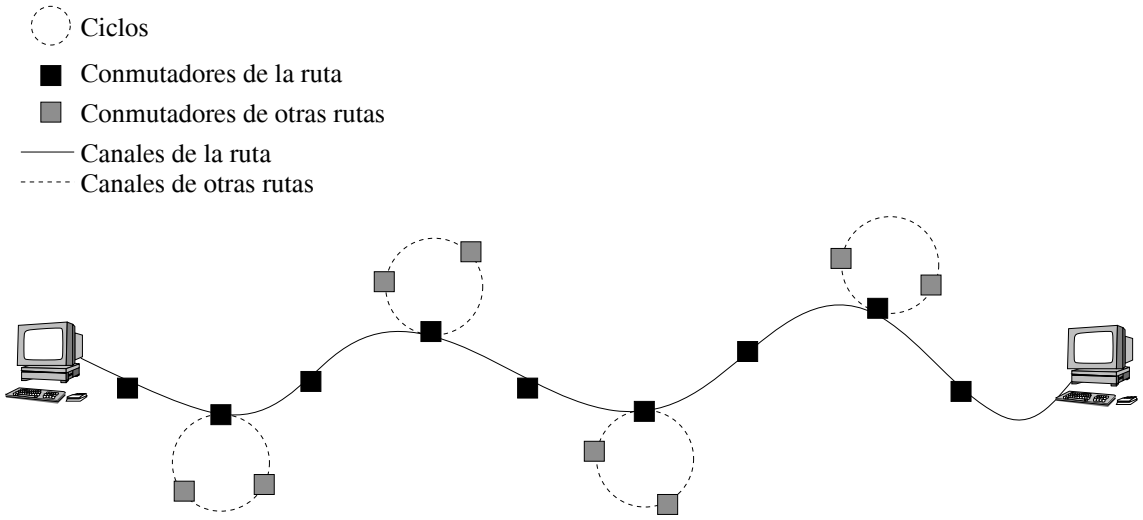
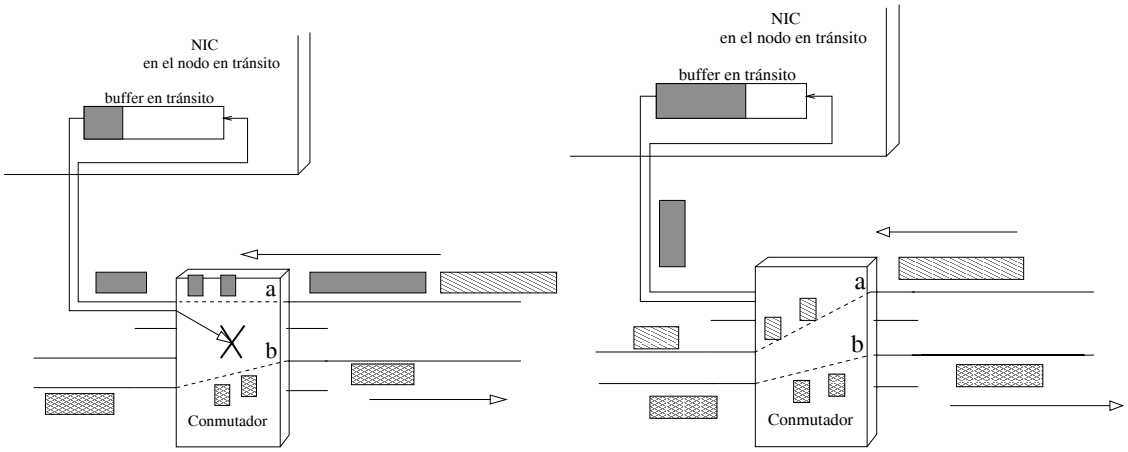


Figura 50: Ejemplo de ciclos a lo largo de una ruta.

mecanismo puede influir también en la contención de la red al extraer mensajes temporalmente de la misma. En efecto, al extraer un mensaje, se liberan todos los canales reservados por el mensaje en tránsito. Estos canales podrán ser utilizados por otros mensajes, los cuales podrán avanzar por la red. En la figura 51.a vemos como un mensaje está entrando en un conmutador por el canal de entrada *a*. Este mensaje es un mensaje en tránsito, por lo que se dirige a un nodo conectado al conmutador. Este mensaje es reinyectado a la red tan pronto como se reconoce como mensaje en tránsito. Sin embargo, el canal que solicita (*b*) está ocupado, por lo que tiene que esperarse a que se libere. El resto del mensaje, mientras la cabecera se espera, se recibe por completo en el interfaz de red del *nodo en tránsito*

liberando así el canal de entrada *a*. Esto permite, como vemos en la figura 51.b, que otro mensaje que se recibe por el canal de entrada *a* del conmutador pueda seguir su camino. Por lo tanto, el utilizar un ITB en dicho conmutador ha evitado que la contención encontrada por el mensaje en tránsito (sobre el canal de salida *b*) repercuta en la contención total que encontrará en la red el segundo mensaje que entra por el canal *a*.



(a) (b)

Figura 51: Reducción de la contención al utilizar ITBs.

Adicionalmente, el mecanismo puede ser aplicado a cualquier conjunto de rutas, convirtiéndolas en rutas libres de bloqueo. Por lo tanto, podemos generalizar el uso de los ITBs de tal forma que se apliquen a cualquier conjunto de rutas calculadas siguiendo cualquier método. Por lo tanto, en este punto avanzamos un paso más en el uso de ITBs, desmarcándonos de los algoritmos de encaminamiento tradicionales y definiendo un nuevo algoritmo de encaminamiento que pretende aprovechar todas las ventajas que nos ofrecen los ITBs. La figura 52 indica las etapas que seguirá el nuevo algoritmo de encaminamiento.

En primer lugar, vamos a definir los objetivos que va a tener el nuevo algoritmo de encaminamiento. Estos objetivos son:

- Utilización de rutas mínimas para todo par origen-destino.
- Equilibrado del tráfico en la red.

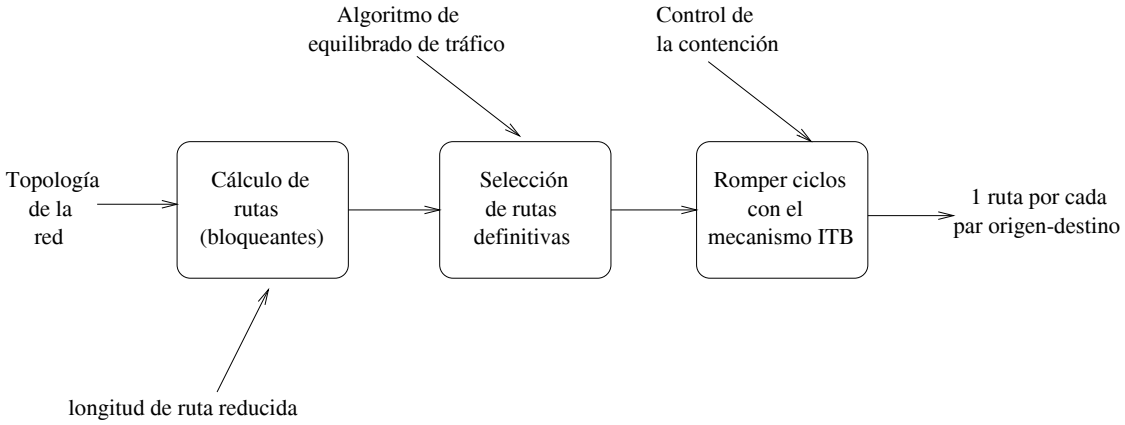


Figura 52: Etapas del nuevo algoritmo de encaminamiento.

- Reducción de la contención de red.
- Bajo coste computacional en el cálculo de las rutas definitivas.

Como podemos observar, son diferentes los objetivos del nuevo algoritmo de encaminamiento. No obstante, el nuevo algoritmo va a intentar alcanzarlos todos, utilizando para ello ITBs.

Rutas mínimas

Debido a que los ciclos serán eliminados en una fase posterior con la inserción de ITBs, el algoritmo de encaminamiento puede calcular y seleccionar cualquier conjunto de rutas. Por lo tanto, el primer objetivo es fácilmente cumplido. El algoritmo se restringe al cálculo de rutas mínimas.

Equilibrado del tráfico

Para obtener un buen equilibrado debemos, partiendo de multitud de rutas, efectuar la selección de las más adecuadas para todos los pares de nodos de la red. No obstante, también tenemos como objetivo el de un coste computacional reducido. Debido a que puede existir un número elevadísimo de rutas mínimas entre un par origen-destino (principalmente en redes regulares), el algoritmo sólo calculará como máximo hasta 10 rutas alternativas entre cada par origen-destino.

Típicamente, los algoritmos de encaminamiento equilibran el tráfico (seleccionan las rutas definitivas) suponiendo un tráfico posterior con una distribución uniforme de destinos. En realidad, la distribución del tráfico de la red cambia en el tiempo y depende del tipo de aplicaciones que se estén ejecutando en cada momento. No obstante, la distribución de destinos uniforme se considera como una distribución a medio camino entre las distribuciones con una elevada localidad y las distribuciones con una localidad mucho menor (por ejemplo, la distribución *hot-spot*). Por lo tanto, en ausencia de más información sobre el tráfico que circulará por la red, el nuevo algoritmo de encaminamiento considerará un futuro tráfico con una distribución uniforme de destinos.

Frecuentemente, las rutas se seleccionan intentando que el número de rutas que atraviesan un canal sea parecido en todos los canales. Es decir, se pretende distribuir uniformemente la utilización de los canales. Nuestro algoritmo de selección de rutas se basará en la misma aproximación. Para ello, el algoritmo seleccionará las rutas definitivas en función de la desviación típica del número de rutas que pasan por cada canal.

En la figura 53 vemos el formato de la tabla de rutas necesario para la selección de rutas en función de la desviación típica. Por cada ruta existirá un campo adicional que contendrá el valor de la desviación típica de los canales (dt) en el caso de que la ruta en cuestión sea eliminada. En la figura 54 podemos ver el algoritmo de equilibrado. Dicho algoritmo se basa en una tabla adicional ($mapa$) que contiene en cada momento el mapa de pesos de cada canal, esto es, el número de rutas que pasan por cada canal, considerando todas las rutas que aún no han sido eliminadas por el algoritmo. El algoritmo elimina una ruta del conjunto de rutas disponibles en cada iteración. Para ello, se calcula la desviación típica resultante de la posible eliminación de cada ruta. En concreto, el algoritmo elimina temporalmente cada ruta y calcula la desviación típica (a partir del mapa de pesos) de la utilización de los canales, asociándola a dicha ruta (dt). A continuación, elimina definitivamente la ruta que tiene asociada una menor desviación típica (al mismo tiempo que actualiza el mapa de pesos). Hay que hacer constar que una ruta será eliminada si y sólo si aún hay rutas alternativas para el par origen-destino determinado. El algoritmo de equilibrado finaliza cuando no hay más rutas que eliminar, esto es, cuando solamente queda una ruta para cada par origen-destino.

Como vemos, a diferencia del algoritmo de encaminamiento *smart-routing*, en cada iteración, el algoritmo de equilibrado elimina una ruta, la cual ya no entrará en

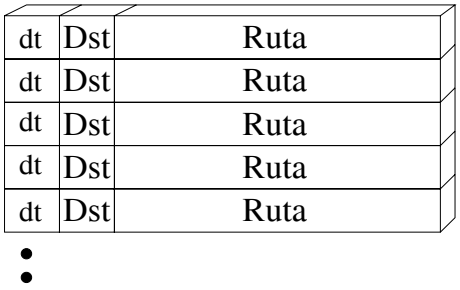


Figura 53: Tabla para la selección de rutas con el algoritmo de equilibrado basado en la desviación típica.

juego nunca más. Por lo tanto, el algoritmo de equilibrado tiene un coste polinómico con el número de rutas alternativas calculadas. Cabe recordar que el algoritmo de encaminamiento *smart-routing* tiene un coste exponencial, lo que le confiere una dudosa utilidad en redes con elevado número de rutas alternativas.

Eliminación de ciclos y control de la contención

Una vez se han calculado y seleccionado las rutas definitivas para su utilización, se entra en la fase de eliminación de ciclos, insertando ITBs en las rutas apropiadas. Como se ha comentado anteriormente, el mecanismo ITB también reduce la contención de red ya que los mensajes que lo utilizan salen de la red temporalmente. Por lo tanto, cuantos más ITBs pongamos menor contención aparecerá en la red. No obstante, como también sabemos, utilizar más ITBs también incrementará la latencia media de los mensajes en el sistema.

Por lo tanto, debe existir una solución de compromiso entre el nivel de reducción de contención en red y la sobrecarga en la latencia de los mensajes. Por consiguiente, vamos a proponer diversas alternativas, variando el número de ITBs que se ubicarán en la red.

El caso más favorable con relación a la contención es poner ITBs en todos los saltos de todas las rutas. Así, los mensajes salen temporalmente de la red en cada conmutador visitado. Esta solución es libre de bloqueo ya que no hay dependencias entre canales que conectan conmutadores. Tan sólo hay dependencias entre un canal que conecta dos conmutadores y un canal que conecta un conmutador a un nodo (y viceversa). No obstante, con esta solución, la latencia introducida por el mecanismo

```
procedimiento Selección-Rutas(mapa : mapa-pesos)
var
   p : ruta
   dt_min : real
inicio
   repetir
      dt_min = max(real)
      p = nil
      para cada ruta r_i
         si existen rutas alternativas para Rutas[r_i].dst
            Eliminar-Ruta-Mapa(r_i, mapa)
            Rutas[r_i].dt = DT(mapa)
            Añadir-Ruta-Mapa(r_i, mapa)
            si Rutas[r_i].dt < dt_min
               dt_min = Rutas[r_i].dt
               p = r_i
            fsi
         fsi
      fpara
      si p <> nil
         Eliminar-Ruta-Mapa(p, mapa)
         Eliminar-Ruta(p)
      fsi
   hasta p==nil
fin procedimiento
```

Figura 54: Algoritmo de equilibrado del tráfico del nuevo algoritmo de encaminamiento.

será máxima.

Por otra parte, el caso más favorable con relación a la sobrecarga introducida en la latencia es poner el mínimo número de ITBs que aseguren que la red es libre de bloqueo. Para ello, el algoritmo buscará ciclos en el GDC insertando ITBs para romperlos. El algoritmo buscará y romperá ciclos siguiendo el algoritmo ya introducido de la figura 49. Nótese que este algoritmo no garantiza la introducción del mínimo número de ITBs que obtienen un GDC acíclico, ya que el número de ciclos a romper depende del orden en el que se examinan las rutas y por donde se rompe cada ciclo. No obstante, el algoritmo insertará un número reducido de ITBs. Con esta solución existirá una elevada contención de red, pero el incremento de latencia

será menor.

Entre estas dos alternativas extremas podemos proponer otras alternativas que intenten compensar ambos efectos (latencia adicional de los mensajes y nivel de contención en la red). En concreto, proponemos tres alternativas adicionales que ubicarán en la red como media un 33%, 50% y 66% de ITBs, respectivamente. Hay que recordar que, al igual que antes, estas tres nuevas alternativas deben asegurar a su vez que la red sea libre de bloqueo (GDC acíclico).

Para poner un 33 % de ITBs a la vez que aseguramos que no existirán ciclos en el GDC seguimos las siguientes reglas. Primero, asignamos un identificador aleatorio a cada conmutador, cuidando de que se asignen identificadores únicos. Después, para cada sub-ruta que cruza tres conmutadores en orden secuencial (conmutadores A, B y C con identificadores a, b y c) pondremos un ITB si

$$(b > a) \;\; Y \;\; (b > c) \qquad (1)$$

Teniendo en cuenta que los identificadores son generados aleatoriamente y que son únicos, podemos obtener seis ordenaciones de los identificadores de los conmutadores (*abc, bac, bca, cba, cab, acb*) según su valor. Dos ordenaciones (*cab,acb*) de las seis siguen la regla (1). Esto es, hay un 33 % de probabilidad de poner un ITB entre dos enlaces conectados a un conmutador. Por lo tanto, podemos concluir que con estas reglas ubicaremos ITBs en un 33 % de todas las posibles posiciones. Adicionalmente, con estas reglas aseguramos que el GDC será acíclico, ya que en cada posible ciclo habrá como mínimo un conmutador con un identificador mayor que el identificador de sus dos vecinos en el ciclo y por tanto tendrá un ITB.

Para poner un 66 % de ITBs utilizamos unas reglas similares. Utilizando identificadores únicos y aleatorios, ponemos un ITB siempre que el identificador del conmutador B es mayor que alguno de los identificadores de los conmutadores vecinos A o C. Esto es,

$$(b > a) \;\; O \;\; (b > c) \qquad (2)$$

Las ordenaciones *abc*, *cba*, *cab* y *acb* siguen la regla (2). Por lo tanto, la probabilidad de poner un ITB es del 66 %. Con estas reglas, también evitamos ciclos en el GDC ya que siempre habrá, en un ciclo, un conmutador con un identificador mayor que el de uno de sus dos vecinos en el ciclo, y este tendrá un ITB.

Por último, para poner un 50 % de ITBs, ubicaremos un ITB en los conmutadores que tengan un identificador menor que el siguiente. Esto es, en una subruta que cruza dos conmutadores A y B en orden secuencial, pondremos un ITB si

$$a < b \qquad (3)$$

Obviamente, el algoritmo de encaminamiento resultante será también, en este caso, libre de bloqueo.

Todas estas alternativas serán evaluadas en el capítulo 5 junto con el algoritmo de equilibrado presentado anteriormente.

3.3 Control de la latencia

El principal inconveniente del mecanismo ITB es el posible incremento en latencia que puedan sufrir los mensajes que utilicen el mecanismo. Este inconveniente solamente será importante en mensajes cortos y con bajas cargas de tráfico (ver capítulo 5). Aún así, para minimizar este inconveniente, vamos a proponer diferentes soluciones basadas todas ellas en la limitación de la utilización de rutas con ITBs. Para ello, volvemos a utilizar la selección de rutas en tiempo de envío de mensajes. Dichas soluciones son:

- **Utilización de ITBs en función del tamaño de los mensajes**. Dado que el efecto sobre la latencia será más importante en los mensajes cortos (ver capítulo 5), esta alternativa seleccionará rutas en función del tamaño del mensaje, permitiendo la utilización sin restricciones del mecanismo ITB solamente en mensajes largos. En concreto, el algoritmo de encaminamiento calculará dos conjuntos de rutas. El primer conjunto de rutas utilizará ITBs de una forma restringida, mientras que el segundo conjunto de rutas utilizará ITBs sin ninguna limitación. Para los mensajes cortos, el algoritmo de selección utilizará rutas del primer conjunto mientras que para mensajes largos el algoritmo de selección utilizará rutas del segundo conjunto.

- **Utilización de los ITBs en función de la latencia máxima permitida**. Al poder utilizar rutas con y sin ITBs podemos dirigir solamente un tanto por cien del tráfico por las rutas con ITBs con el fin de que la latencia total añadida del mecanismo no supere un cierto nivel. El tanto por cien de mensajes por

rutas con ITBs puede ser calculado en función del nivel de latencia adicional máximo que permitimos al mecanismo.

- **Utilizar el mecanismo ITB a partir de cierto nivel de tráfico**. Dado que el efecto de la latencia será significativo sólo en condiciones de poco tráfico (ver capítulo 5), en esta solución vamos a proponer la utilización de rutas con ITBs a partir de cierto nivel de tráfico. Para ello, utilizaremos el mecanismo de detección de contención propuesto para la selección de rutas (sección 2.4). A partir de esta información, el algoritmo de selección utilizará rutas sin ITBs para condiciones de poco tráfico y rutas con ITBs para condiciones de tráfico medio o elevado.

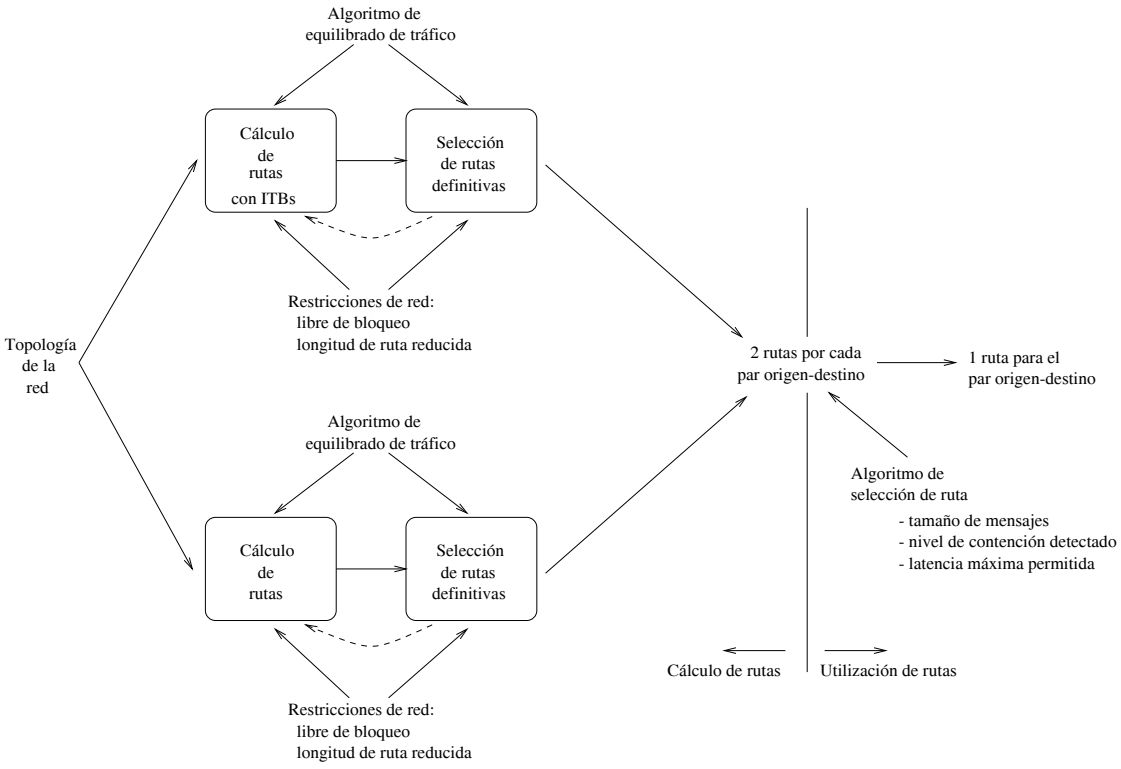


Figura 55: Control de latencia. Utilización de ITBs en función de diferentes criterios.

En la figura 55 podemos ver las etapas en el cálculo de rutas por los algoritmos de encaminamiento que contemplan las soluciones anteriores. Como vemos, el algoritmo de encaminamiento calcula varias rutas entre cada par origen-destino. A partir de

cierta información (tamaño de los mensajes, latencia máxima permitida o tráfico detectado en el nodo local) el algoritmo de selección de rutas, en tiempo de envío de mensajes, seleccionará la ruta adecuada. Hay que tener en cuenta que la unión de los dos conjuntos de rutas a utilizar debe seguir siendo libre de bloqueo. Todas estas soluciones serán evaluadas en el capítulo 5.

3.4 Implementación del mecanismo en Myrinet

En esta sección describimos la implementación del mecanismo en la red Myrinet. En concreto, se describen todos los cambios necesarios para implementar el mecanismo de una forma eficiente, de manera que introduzca la menor latencia posible a los mensajes que utilicen el mecanismo.

Básicamente, la red Myrinet esta controlada por el Myrinet Control Program (MCP) que se ejecuta en el interfaz de red de cada nodo y por el *software* de comunicaciones que se ejecuta en el nivel de sistema operativo de cada nodo. Myricom también facilita un *software* de comunicaciones por paso de mensajes denominado GM [GM]. Este *software* incluye tanto el código MCP de la interfaz de red como el *software* asociado al sistema operativo. Los objetivos principales de GM son: portabilidad, baja latencia, elevados anchos de banda y bajas sobrecargas en los *hosts*. Otras características de GM son:

- Una sobrecarga menor de 1 microsegundo por paquete en el interfaz de red.
- Acceso directo de las aplicaciones de usuario a la memoria del interfaz de red (transferencias *zero-copy*).
- Entrega en orden entre *hosts* incluso en presencia de fallos de red.
- Soporte para construir *clusters* de 10.000 *hosts*.
- Mensajes de longitud máxima de $2^{31} - 1$ bytes.
- Mapeado[1] automático de la red.

Para aprovechar todas las ventajas aportadas por GM, se ha optado por una implementación del mecanismo sobre dicho *software* de comunicaciones. En concreto, el mecanismo se ubicará en el código MCP de GM.

[1] Del inglés *mapping*.

En lo que resta de capítulo describimos las modificaciones y actuaciones globales para implementar de forma eficiente el mecanismo. Posteriormente, en el apéndice A, se describe todo el código de la implementación realizada sobre el MCP de GM.

3.4.1 Formato de la cabecera de los mensajes

Como objetivo principal en la implementación nos proponemos minimizar el efecto que tiene el mecanismo consistente en incrementar la latencia de los mensajes que lo utilizan. Para minimizar la latencia, el *nodo en tránsito* debe detectar un mensaje en tránsito lo antes posible e inyectarlo de inmediato (tan pronto como el canal de salida esté libre) sin que tenga que realizar ninguna tarea adicional. Una buena opción es que la ruta que debe seguir el mensaje desde el *nodo en tránsito* venga ya establecida desde el nodo origen del mensaje, por lo que el *nodo en tránsito* no tiene que recalcular la ruta hacia el destino final.

Por lo tanto, en primer lugar, para que la ruta a utilizar venga ya prefijada es necesario cambiar el formato de la cabecera de los mensajes. En la figura 56 podemos observar el nuevo formato genérico de las cabeceras de los mensajes soportando ITBs. Puede compararse con la figura 13 (página 28) para observar los cambios frente a la cabecera convencional.

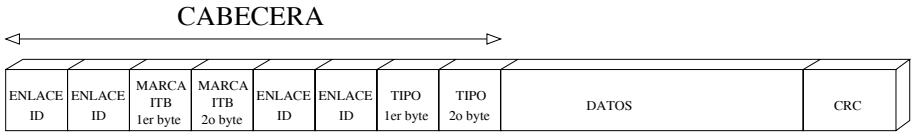


Figura 56: Nuevo formato de cabecera de mensajes con soporte para ITBs.

Toda la ruta se construye en el nodo origen, incluyendo el paso por el *nodo en tránsito*. En la ruta también se introducen dos bytes de marca (*marca ITB*) que sirven para indicar al *nodo en tránsito* que éste es un mensaje en tránsito. Al igual que cada conmutador elimina el primer byte de la cabecera al encaminar un mensaje, el *nodo en tránsito* también deberá eliminar los dos primeros bytes (*marca ITB*). A continuación de la *marca ITB* sigue la subruta hacia el destino final o hacia el siguiente *nodo en tránsito*. De esta forma, el *nodo en tránsito* sólo tiene que leer los dos primeros bytes (*marca ITB*) para detectar que el mensaje es un mensaje en

tránsito. Con esta solución, se reduce la latencia del mecanismo, puesto que el *nodo en tránsito* no tiene que calcular la nueva ruta.

3.4.2 Buffers

La figura 57 muestra la implementación del mecanismo en el interfaz de red Myrinet. En primer lugar, se debe reservar espacio en memoria para ubicar los mensajes en tránsito. Como se ha comentado anteriormente, los nodos deben asegurar suficiente espacio para que cualquier mensaje en tránsito de cualquier tamaño pueda ser completamente almacenado. Además, cada nodo debe tener en cuenta el posible apilamiento temporal de mensajes en tránsito (debido a posible congestión en el canal de salida) por lo que no sólo se debe reservar espacio para un mensaje largo, sino para varios. Actualmente, los interfaces de red Myrinet poseen 8MB de memoria, y el código MCP de GM ocupa alrededor de 128KB, por lo que existe suficiente espacio para muchos mensajes en tránsito.

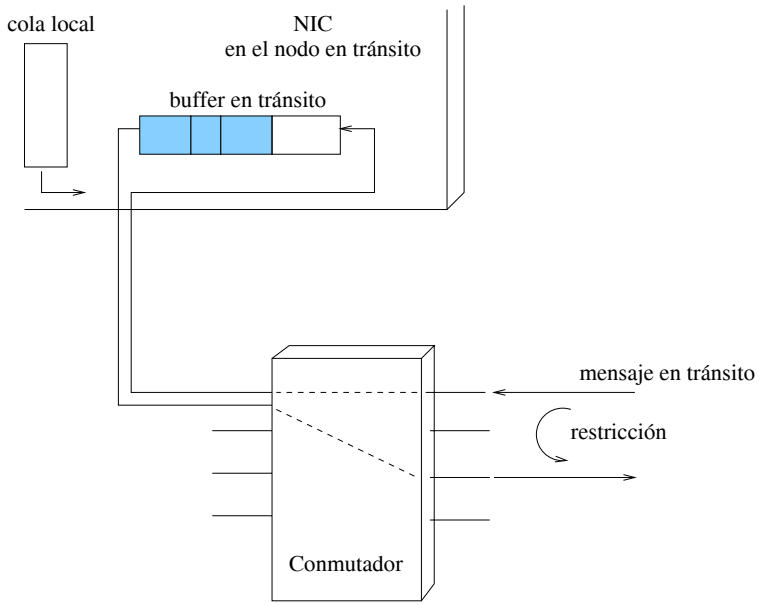


Figura 57: Mecanismo de buffers en tránsito (ITBs).

Como ya hemos indicado, en caso extremo (en el caso de que los buffers en tránsito se agoten), el interfaz de red puede utilizar la memoria del *host*. En este caso, la latencia de los mensajes en tránsito aumentaría de forma excesiva (los

mensajes pasarían a través del bus PCI) por lo que este caso debe ser evitado a toda costa. Además, el contemplar la posibilidad de utilizar memoria del *host* en situaciones de desborde en la memoria local del interfaz elevaría la complejidad del mecanismo. Una solución sencilla puede ser ignorar los mensajes en tránsito que no puedan ser tratados por el interfaz. Con esta solución, los mensajes deberán ser reenviados posteriormente.

3.4.3 Detección de mensajes en tránsito

Para minimizar la latencia del mecanismo ITB, la detección de mensajes en tránsito debe ser prioritaria. Para ello, el *software* MCP debe dar la máxima prioridad a la recepción de mensajes y su posible tratamiento.

En Myrinet existen unos bits accesibles por el MCP que aportan información acerca del estado del *hardware* del interfaz (por ejemplo, informan de actividad por el canal de entrada al interfaz). Los códigos MCP (como es el caso de GM) suelen ser implementados como autómatas gobernados por tablas de eventos con diferentes prioridades. La activación de estos eventos suelen estar relacionados con los bits de estado. La recepción de un nuevo mensaje debe estar asociado a un evento de la máxima prioridad.

Una vez se ha detectado un mensaje, se programa un dispositivo de DMA de recepción denominado R-DMA. Dicho dispositivo introduce todo el mensaje automáticamente en la memoria local del interfaz. Nótese que al no existir canales virtuales en Myrinet, los bytes del mensaje van a ser recibidos sin interrupción (salvo en el caso de que existan las denominadas burbujas que serán analizadas más adelante). Tanto los mensajes normales como los mensajes en tránsito deberán ser recibidos utilizando dicho dispositivo para interrumpir lo mínimo al procesador del interfaz.

3.4.4 Reinyección de mensajes en tránsito

Una vez se detecta la llegada de un mensaje, se debe leer el tipo del mensaje y si es una *marca ITB*, se debe comprobar que el canal de salida esté libre y reenviar el mensaje de inmediato (eliminando los bytes de *marca ITB*).

Con el objeto de minimizar la latencia de los mensajes en tránsito, el mecanismo debe utilizar los dispositivos de acceso directo a memoria disponibles en el interfaz

de red. En concreto debe utilizar de forma encadenada los dispositivos R-DMA y S-DMA que extraen/inyectan mensajes de/a la red. Gracias a estos dispositivos, el interfaz de red puede reinyectar un mensaje al mismo tiempo que está leyéndolo de la red. Con esto, el mecanismo sigue el comportamiento de la conmutación *virtual cut-through* en los *nodos en tránsito*, con la consiguiente reducción de latencia.

En la figura 58 podemos ver cómo se deben utilizar los dos dispositivos de acceso directo a memoria. En un primer paso (figura 58.a) se empieza a recibir un mensaje utilizándose para ello el dispositivo de recepción R-DMA. Tras el tiempo necesario para detectar el mensaje en tránsito (leyendo los dos primeros bytes), el nodo detecta que se trata de un mensaje en tránsito. Tras la comprobación del estado del canal de salida (podría estar enviando un mensaje en ese momento) y, si el canal de salida está libre, programa el envío del mensaje (figura 58.b) utilizando el dispositivo S-DMA. Nótese como el nodo no se espera a recibir todo el mensaje para enviarlo. Finalmente, el mensaje es recibido por completo (figura 58.c), pero el dispositivo de transmisión sigue enviando los bytes restantes. Los dos punteros asociados a ambos dispositivos R-DMA y S-DMA estarán a una distancia en bytes determinada. Esta distancia dependerá del tiempo de detección del mensaje y del tiempo requerido para programar el S-DMA.

3.4.5 Burbujas

Cuanto menor sea la distancia entre los dos punteros, menor latencia sufrirán los mensajes en tránsito. Sin embargo, el mecanismo puede llegar a funcionar de forma incorrecta debido a la posible existencia de burbujas en la red si la distancia entre los dos punteros fuera demasiado pequeña. Una burbuja se forma cuando un nodo emisor de un mensaje detiene temporalmente la transmisión del mensaje por lo que se insertan espacios (burbujas) dentro del mensaje tal y como vemos en la figura 59. Por lo tanto, la recepción de una burbuja lo suficientemente grande podría provocar en un *nodo en tránsito* que el puntero S-DMA adelantase al puntero R-DMA con lo que se enviaría información errónea ya que aún no se ha recibido (en el *nodo en tránsito*, ante una burbuja, el puntero R-DMA se detiene, pero el S-DMA no).

Los únicos elementos que pueden introducir burbujas son los interfaces de red. El interfaz de red posee cuatro dispositivos que pueden acceder a la memoria local del interfaz. Estos son: el procesador del *host* (HOST), el procesador del interfaz de red (LANai), S-DMA y R-DMA. Debido a que la memoria local del interfaz tiene un

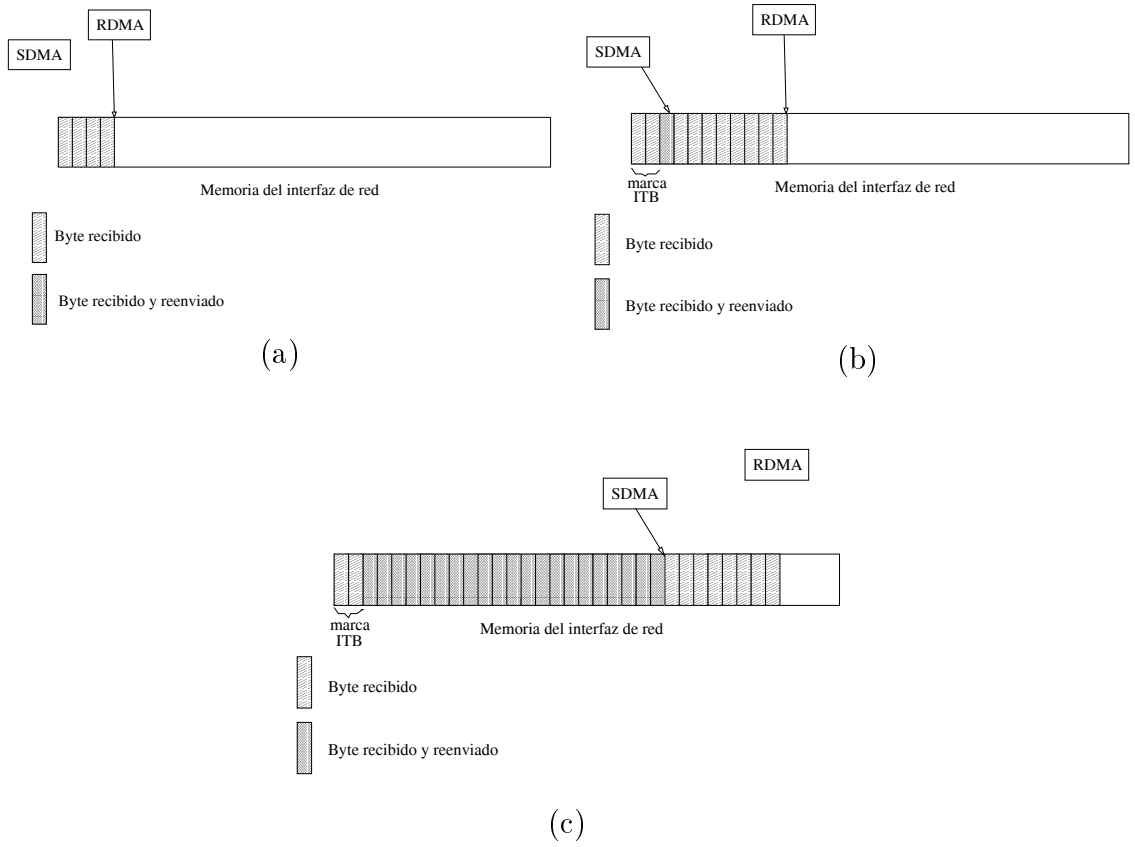


Figura 58: Encadenar dispositivos S-DMA y R-DMA en Myrinet.

ancho de banda limitado, tan sólo dos dispositivos como máximo pueden acceder a la memoria local del interfaz en cada ciclo. Además, existe un orden de prioridades en estos dispositivos para acceder a la memoria local. El orden de prioridades es: HOST, R-DMA, S-DMA y LANai (de mayor a menor prioridad).

Por lo tanto, en el caso de que el HOST esté accediendo a la memoria local del interfaz y el propio interfaz esté recibiendo un mensaje, entonces un posible envío de un mensaje se vería detenido con la consiguiente inclusión de burbujas.

Sin embargo, un diseño del MCP que tomara en cuenta este efecto, podría eliminar las burbujas. Por ejemplo, se puede hacer que el HOST interrumpa lo mínimo al interfaz de red. Típicamente, el HOST accede a la memoria local del interfaz para leer/escribir los mensajes a recibir/enviar. No obstante, se puede forzar a que el HOST escriba solamente la dirección de inicio y la longitud de los mensajes a

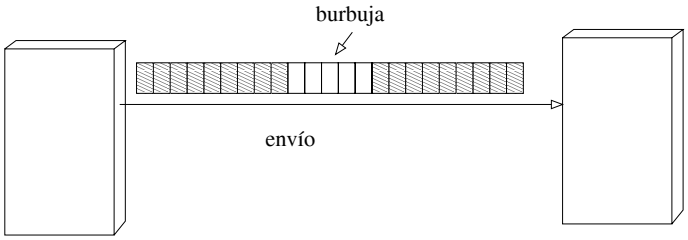


Figura 59: Burbujas en un mensaje.

escribir/leer ubicados en la memoria del *host*, dejando así la responsabilidad de acceder a estos mensajes al MCP cuando no esté enviando un mensaje que pase por un *nodo en tránsito*.

Según fuentes de Myricom, este efecto será eliminado en futuras implementaciones de Myrinet, al aumentar el ancho de banda de la memoria local de los interfaces de red.

No obstante, hay que tener presente que este efecto de las burbujas puede ser mínimo y realmente afecta únicamente a mensajes muy largos[2]. No obstante, es fácil implementar un mecanismo que detecte posibles desbordamientos de los punteros de DMA en el *nodo en tránsito* debidos a burbujas. Una solución muy sencilla es forzar el último byte de cada mensaje a un valor determinado denominado *marca fin mensaje*. En la figura 60 podemos ver que en los *nodos en tránsito*, cuando se detecta un mensaje en tránsito, se ubica en el último byte (antes de que se reciba el correcto) un valor distinto a *marca fin mensaje*. Si el mensaje se recibe correctamente, sin burbujas, el último byte se sobreescribirá con el valor correcto antes de ser retransmitido hacia el nodo destino. En caso contrario, se enviará hacia el nodo destino un valor incorrecto en el último byte del mensaje, por lo que si el nodo destino recibe en un mensaje un valor distinto en el último byte a *marca fin mensaje* entonces sabe que los punteros de DMA en un *nodo en tránsito* se han desbordado por lo que puede iniciar un mecanismo de recuperación del mensaje.

[2]En algunos experimentos hemos comprobado que las burbujas aparecen solamente para mensajes de tamaño mayor de 10k.

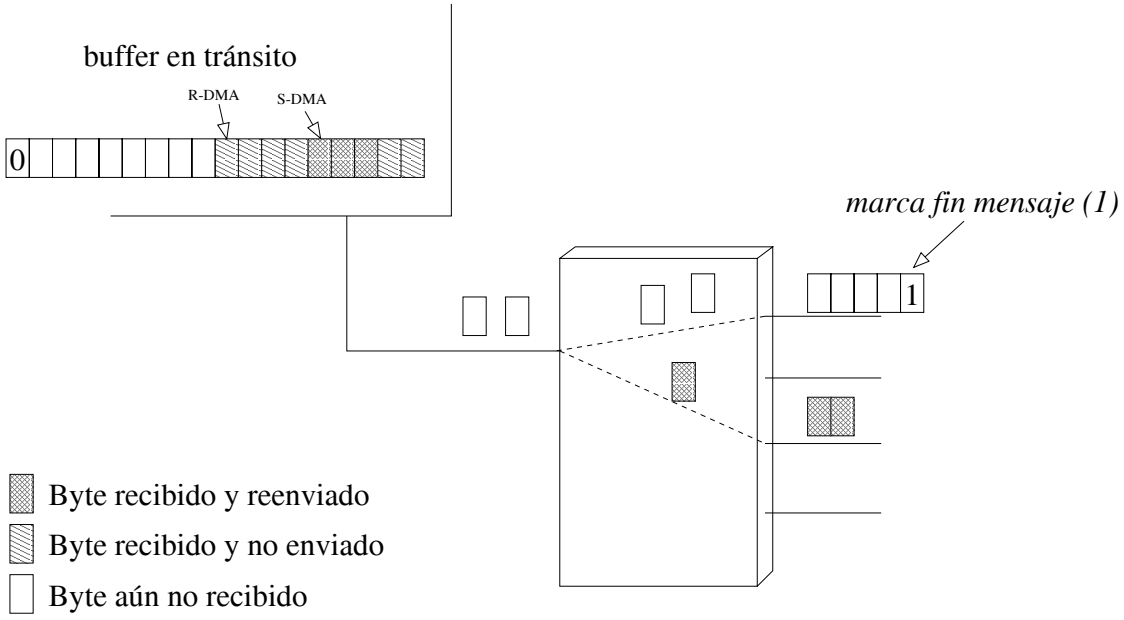


Figura 60: Mecanismo de detección de burbujas.

3.4.6 Interferencias con el bus PCI

Por último, el mecanismo también puede sufrir de burbujas generadas por el bus PCI en el nodo origen del mensaje ya que el bus PCI puede detenerse en cualquier momento. Para evitar este problema hay que asegurarse de que el MCP tiene todo el mensaje en la memoria local del interfaz antes de enviar el mensaje a través de un *nodo en tránsito*.

Típicamente, las implementaciones de *software* de comunicaciones para Myrinet ubican el mensaje en el interfaz antes de enviarlo por lo que se evita dicho inconveniente. Una excepción es Trapeze [Yoc97]. En esta implementación se encadenan los dispositivos DMA del PCI y del interfaz red (S-DMA en el origen y R-DMA en el destino).

Capítulo 4

Método de evaluación

En este capítulo describimos el método empleado para evaluar los mecanismos, justificando su utilización. Adicionalmente, describimos los índices de prestaciones así como los parámetros y topologías utilizados en la evaluación. Por último, validamos el método de evaluación.

4.1 Método de evaluación

Para la evaluación de las prestaciones de un sistema se pueden utilizar diferentes métodos de evaluación [Jai91]:

- **Mediciones reales**. Con este método se toman mediciones reales sobre un sistema ya existente. Este método necesita de un prototipo ya diseñado o del sistema definitivo. Por lo tanto, este método de evaluación requiere de un esfuerzo de desarrollo previo si el sistema aún no ha sido implementado. En nuestro caso, implementar los mecanismos propuestos en la presente tesis tan sólo requiere de modificaciones del código de la tarjeta de interfaz (MCP). No obstante, debido a los escasos recursos disponibles (número de conmutadores e interfaces de red), este método resulta inviable para obtener resultados bajo diferentes topologías y diferentes tamaños de red.

- **Modelos analíticos**. Los modelos analíticos constituyen la alternativa más económica, proporcionando resultados de forma inmediata. Sin embargo, su utilización depende del nivel de detalle requerido para los resultados. Para que el modelo sea susceptible de una evaluación analítica, se debe simplificar

notablemente el modelo del sistema, con la consiguiente pérdida de precisión en los resultados. Por lo tanto, el modelo analítico no se utiliza en la presente tesis debido a que buscamos un alto nivel de detalle en el modelo para así obtener una elevada precisión en los resultados.

- **Simulaciones**. La simulación es un método utilizado ampliamente en muchos campos de investigación. Mediante la simulación por ordenador se puede modelar con precisión cualquier sistema. Estos modelos no pueden ser analizados analíticamente debido a su complejidad. Con la simulación se pueden considerar distintos niveles de detalle del sistema. El principal inconveniente de las simulaciones es el elevado tiempo de ejecución necesario para la simulación del sistema.

  Una vez realizado el modelo de simulación, hay que introducir la carga de prueba con el objeto de obtener los resultados. En los simuladores (en particular, de redes de interconexión) se pueden utilizar diferentes aproximaciones de modelado de la carga del sistema:

  - **Carga sintética**. Esta es la aproximación más utilizada en la evaluación de sistemas con simuladores. Con esta aproximación, el tráfico que circula por la red es generado aleatoriamente. Cada vez que se genera un mensaje, el destino del mensaje es calculado utilizando una función de distribución. Distribuciones típicas utilizadas son: uniforme, local, *bit-reversal*, *hot-spot*, *shuffle*, etc. Con cada función de distribución se genera un patrón de tráfico. En la carga sintética también se define el tiempo entre envíos de los mensajes inyectados por cada nodo y el tamaño de los mismos utilizando alguna función de distribución, con el fin de obtener diferentes tasas de tráfico en la red. La carga sintética es fácil y rápida de obtener. No obstante, hay que remarcar que la carga sintética puede no modelar perfectamente el tráfico real que circulará por la red en el sistema definitivo.
  - **Trazas**. A partir de sistemas reales y ejecuciones de aplicaciones sobre estos sistemas, se obtienen ficheros de trazas. Estos ficheros de trazas contienen la información del tráfico generado por estas aplicaciones. En concreto contienen el destino y tamaño de los mensajes enviados y el tiempo en el que se han generado estos mensajes. Los ficheros de trazas

son inyectados al simulador con el fin de que se simule un tráfico real. El problema de esta aproximación es que el tráfico registrado por la traza es dependiente de muchos factores en el sistema en el que se ha obtenido y al trasladar este tráfico a otras condiciones y parámetros (topologías, retardos, algoritmos de encaminamiento), estas trazas ya no reflejan fielmente el tráfico generado por una aplicación real en el nuevo sistema. Por lo tanto, el uso de ficheros de trazas se desestima en la presente tesis.

– **Carga real de aplicaciones**. En los últimos años ha aparecido un nuevo tipo de simuladores. Éstos son los simuladores dirigidos por ejecución. Estos simuladores ejecutan realmente una aplicación sobre el sistema simulado. La aplicación genera dinámicamente la carga hacia el simulador. Por lo tanto, con estos simuladores se simula con precisión la ejecución de una aplicación real en el sistema simulado. No obstante, aunque ya se han realizado estudios en este sentido [Fli98a, Fli98b, Fli99a] se ha comprobado que aparecen una serie de inconvenientes en estos simuladores aún no solucionados. El primer inconveniente es el elevado tiempo de ejecución. Para que un sistema pueda ser correctamente evaluado, una aplicación paralela de complejidad media-alta debe ser ejecutada por completo en el sistema simulado (típicamente sobre un sistema monoprocesador). A su vez, todas las referencias generadas por la aplicación deben ser simuladas. Por lo tanto, el tiempo de ejecución para un sólo sistema con unos determinados parámetros puede llegar a ser prohibitivo. De hecho, en anteriores trabajos [Fli99a], algunas simulaciones dirigidas por ejecución para evaluar un sistema con un conjunto de parámetros muy reducido llegaban a durar una semana. Como segundo inconveniente, para la utilización de esta aproximación, cabe citar el reducido tráfico generado por tales aplicaciones. Las aplicaciones SPLASH-2 [SPLA2] utilizadas en estos simuladores han sido criticadas, calificándolas como *toy tools*, debido al reducido tráfico que generan. En la presente tesis, vamos a evaluar mecanismos que pretenden mejorar ostensiblemente la productividad de la red. Con las aplicaciones utilizadas en los simuladores dirigidos por ejecución, estos rangos de tráfico elevados no suelen ser alcanzados.

Por lo tanto, en la evaluación de los mecanismos propuestos para la presente tesis, utilizaremos simulaciones con carga sintética. Como ya hemos indicado, el principal inconveniente de esta aproximación es que la carga generada de esta manera puede no reflejar fielmente la carga generada por las aplicaciones reales. Intentaremos soslayar este inconveniente analizando las prestaciones obtenidas utilizando diversas funciones de distribución. Por otra parte, al utilizar simulaciones con carga sintética, existen una serie de condiciones que se deben asegurar y validar para que los resultados resulten fiables. Las condiciones a contemplar son:

- **Cálculo del intervalo transitorio**. Cuando se inicia la inyección de carga en el sistema simulado, el sistema (la red) se encuentra vacío. Durante un cierto periodo de tiempo de simulación, el sistema se encuentra en un estado transitorio hacia el estado permanente, donde el sistema está estabilizado. Por lo tanto, en las simulaciones es necesario desechar el estado transitorio y evaluar el sistema bajo el estado permanente. El problema es que la duración del estado transitorio depende de muchos factores, como por ejemplo del sistema simulado, del rango de tráfico que se está simulando, del tamaño de los mensajes, de la topología, etc.

- **Independencia de los resultados**. Debido a la generación aleatoria de diversos eventos (tiempo de generación de mensajes, destino de los mensajes, etc.) en la carga sintética, los resultados deben ser comprobados con el fin de comprobar que son independientes de los números aleatorios generados. En concreto, se debe comprobar la uniformidad e independencia del generador de números aleatorios. El generador de números aleatorios utilizado ya ha sido comprobado en [Alc96].

- **Intervalo de confianza**. En el proceso de simulación siempre existe un intervalo de error en los resultados obtenidos. Por lo tanto, se deben facilitar intervalos de confianza.

4.1.1 Parámetros de entrada al simulador

Los parámetros de entrada al simulador pueden clasificarse en cuatro grupos:

1. **Parámetros de diseño de la topología**.

- *Definición de la topología.* El simulador trabaja con cualquier topología definida a partir de dos ficheros de configuración. El primer fichero define la topología de la red (ver sección 4.4), indicando el número de conmutadores y el número de *hosts* por conmutador, así como los enlaces entre los conmutadores.
- *Algoritmo de encaminamiento.* En el segundo fichero se indican las tablas de encaminamiento para cada par origen-destino. El fichero de tablas de encaminamiento es creado a partir de programas externos que calculan las rutas en función del algoritmo de encaminamiento empleado. Diferentes algoritmos de encaminamiento serán evaluados. En concreto, se evaluarán los siguientes algoritmos:
  - Algoritmos tradicionales: *up*/down**, *DFS*, *smart-routing.*
  - Algoritmos tradicionales con buffers en tránsito.
  - Nuevo algoritmo de encaminamiento basado en buffers en tránsito.

2. **Parámetros de diseño de la red**.

- *Conmutador*
  - *Retardos del conmutador.* Puede indicarse, medido en ciclos de reloj, el tiempo de encaminamiento y el tiempo de transferencia a través del conmutador.
  - *Tamaño de los buffers de entrada y salida del conmutador.*
  - *Niveles de STOP y GO en los buffers de entrada del conmutador.*
- *Canal*
  - *Tiempo de vuelo de los enlaces.* El tiempo de vuelo indica el número de flits que pueden haber en un determinado enlace en un instante dado. Este parámetro se deduce a partir de la longitud del cable, del retardo de propagación y del ancho de banda.
  - *Tiempo por flit.* Es la inversa de la tasa de transferencia por el canal. Indica la velocidad con que se introduce un flit en el canal.
- *Interfaz de red*
  - *Memoria en los interfaces de red.* El mecanismo ITB necesita reservar memoria en los interfaces de red para almacenar temporalmente

los mensajes en tránsito. El simulador posee un parámetro que indica el tamaño máximo de memoria para almacenar mensajes en tránsito en un nodo.

- *Retardos de los componentes del interfaz de red.* El interfaz de red en Myrinet posee dispositivos de DMA utilizados por el mecanismo ITB. Para modelar el mecanismo ITB el simulador utiliza dos parámetros temporales: el tiempo de detección de un mensaje en tránsito y el tiempo de reprogramación del DMA.

3. **Parámetros que definen la carga de la red**.

   - *Distribución de destinos de los mensajes.* Las distribuciones implementadas son las siguientes: uniforme, uniforme con radio de localidad, *bit-reversal*, *hot-spot* y una distribución combinada de las anteriores.
   - *Longitud de los mensajes.* El simulador permite que coexistan dos tipos de mensajes en la red: cortos y largos, indicando además la contribución de cada tipo de mensajes en la carga de la red. El tamaño de cada tipo de mensajes está uniformemente distribuido entre dos valores introducidos al simulador.
   - *Tasa de generación de mensajes.* El tiempo medio entre envío de mensajes está uniformemente distribuido entre dos valores introducidos al simulador. La inversa de este tiempo es la tasa de generación de mensajes. El producto de la tasa de generación de mensajes por el tamaño de los mensajes es la tasa de generación de flits.

4. **Parámetros específicos del simulador**.

   - *Número de mensajes durante la simulación.* A diferencia de otros estudios en los que el número de ciclos de reloj simulados es constante, en este trabajo se ha modificado dinámicamente el número de ciclos de reloj a simular en función de la carga de la red de interconexión. Todas las simulaciones se han prolongado hasta que el número de mensajes recibidos alcanza un valor dado. Con este sistema se pretende evitar que el número de ciclos simulados no sea suficiente en algunos casos, o que sea excesivo en otros. En efecto, en una red poco cargada, el tiempo necesario para que se reciba un cierto número de mensajes es muy elevado. Sin

embargo, con cargas cercanas a la saturación, en poco tiempo se consigue que esos mensajes hayan atravesado la misma. Adicionalmente, en cada simulación se han desechado los primeros mensajes recibidos, con el objeto de analizar el comportamiento de la red en régimen permanente. En particular, el número de mensajes útiles recibidos en cada simulación es igual a 100000, y el número de mensajes desechados es igual a 50000 en el caso de cargas bajas y 200000 en el caso de cargas elevadas (a partir de tasas de tráfico superiores al 70% de la productividad alcanzada). Estos valores se han determinado experimentalmente y se describen en la sección 4.6.1.

- *Semilla para el generador de números aleatorios.*

4.1.2 Resultados ofrecidos por el simulador

Los resultados ofrecidos por el simulador son los siguientes:

- *Número de ciclos simulados.*
- *Número de mensajes recibidos y pendientes de envío.* El crecimiento desmesurado del número de mensajes pendientes de envío es un signo claro de la saturación de la red de interconexión.
- *Latencia media de la cabecera de los mensajes y su desviación típica.* Mide el tiempo transcurrido desde que la cabecera del mensaje se inyecta en la red, hasta que se recibe en el nodo destino.
- *Latencia media total de los mensajes y su desviación típica.* Mide el tiempo transcurrido desde que la cabecera del mensaje se inyecta en la red, hasta que se recibe la cola del mensaje en el nodo destino.
- *Latencia media desde la generación de los mensajes y su desviación típica.* Mide el tiempo transcurrido desde que el mensaje es generado en el nodo origen, hasta que se recibe la cola del mensaje en el nodo destino. Se supone que todo mensaje generado por un nodo es encolado hasta que se inyecta en la red de interconexión.
- *Tráfico medio.* Mide la tasa de recepción de flits por la red de interconexión.

- *Número medio de canales de salida ocupados.*
- *Ocupación media de las colas de entrada y salida.* Número de colas de entrada y salida llenas y vacías.
- *Distribución del número de mensajes enviados en función de la distancia recorrida.*
- *Distribución de la actividad de cada canal.* Para cada canal que conecta dos conmutadores se registra el tanto por cien en que el canal está inactivo, activo o bloqueado.
- *Número de mensajes en tránsito tratados y número de mensajes en tránsito pendientes de reinyección a la red por cada nodo.*
- *Número de mensajes locales inyectados a la red y número de mensajes locales pendientes de inyección a la red por cada nodo.*

4.2 Índices de prestaciones

Las medidas de prestaciones más importantes de las redes de interconexión son la latencia y la productividad. La latencia es el tiempo requerido para entregar un mensaje en el nodo destino.

El tráfico es la tasa de recepción de flits. Una vez la red alcanza un estado estacionario, la tasa de generación de flits coincide con la tasa de recepción, a no ser que la red se sature. Por este motivo, hay que resaltar que el tráfico no es una variable independiente. En el caso de saturación de la red, ambas tasas difieren y el número de mensajes pendientes de envío crece. La productividad es el valor máximo del tráfico aceptado por la red de interconexión.

La latencia la mediremos en nanosegundos (ns), y el tráfico en flits por conmutador y nanosegundo (flits/ns/conmutador).

4.3 Parámetros de la red

El simulador está dirigido por eventos y utiliza una resolución de ciclo de reloj. Puesto que la red modelada sigue los parámetros de tiempos de la red Myrinet, es necesario fijar el tiempo que dura cada ciclo del simulador. Teniendo en cuenta

que el ancho de banda de Myrinet es de 160 MB/s, y suponiendo que la tasa de transferencia por el canal es de 1 flit/ciclo, el ciclo de reloj lo fijaremos en 6.25 ns ($\frac{1}{160MB/s} = 6.25\ ns$).

Los enlaces se han modelado asumiendo enlaces cortos LAN [lan10m] para interconectar conmutadores y *hosts*. Estos cables son de 10 metros de longitud, ofreciendo un ancho de banda de 160 MB/s, con un retardo de propagación de 4.92 ns/m. Al igual que Myrinet, los flits son 8 bits de ancho al igual que los enlaces. La transmisión de los datos por los enlaces es segmentada [Sco94], por lo tanto, un nuevo flit se puede inyectar en el enlace cada 6.25 ns y habrá un máximo de 8 flits en vuelo en el enlace en un determinado instante.

De igual forma, los conmutadores son modelados siguiendo el modelo de los conmutadores de Myrinet. Según fuentes de Myricom, el primer flit atraviesa el *crossbar* interno del conmutador en 150 ns, mientras que los siguientes flits atraviesan el *crossbar* a la tasa de transferencia de los enlaces (160 MB/s). Por lo tanto, el tiempo de encaminamiento es de $150 - 6.25 = 143.75\ ns$, que equivale a 23 ciclos. El tiempo de transferencia a través del conmutador es de 6.25 ns (1 ciclo).

Respecto del tamaño de los *buffers* de entrada del conmutador, el tamaño del *slack buffer* en Myrinet es de 80 bytes teniendo las marcas de *Stop&Go* en 40 (GO) y 56 ($STOP$) bytes.

Respecto a los parámetros del interfaz de red, el tiempo de detección de un mensaje en tránsito se ha fijado en 275 ns (44 ciclos) y el tiempo de reprogramación del DMA en 200 ns (32 ciclos). Para obtener estos tiempos se ha implementado un MCP básico que detecta mensajes en tránsito y los reinyecta. Con este MCP, se han tomado medidas de tiempo utilizando el registro de tiempo real (RTC) incorporado en el interfaz para más de 1000 mensajes en tránsito. No obstante, en el capítulo 5, también se evaluará la sensibilidad del mecanismo ITB frente a variaciones de los tiempos de detección y reprogramación.

Por último, para el mecanismo ITB también es necesario la reserva de memoria para almacenar mensajes en tránsito. Se ha fijado en 512 KB la disponibilidad de memoria del interfaz para mensajes en tránsito.

4.4 Topologías evaluadas

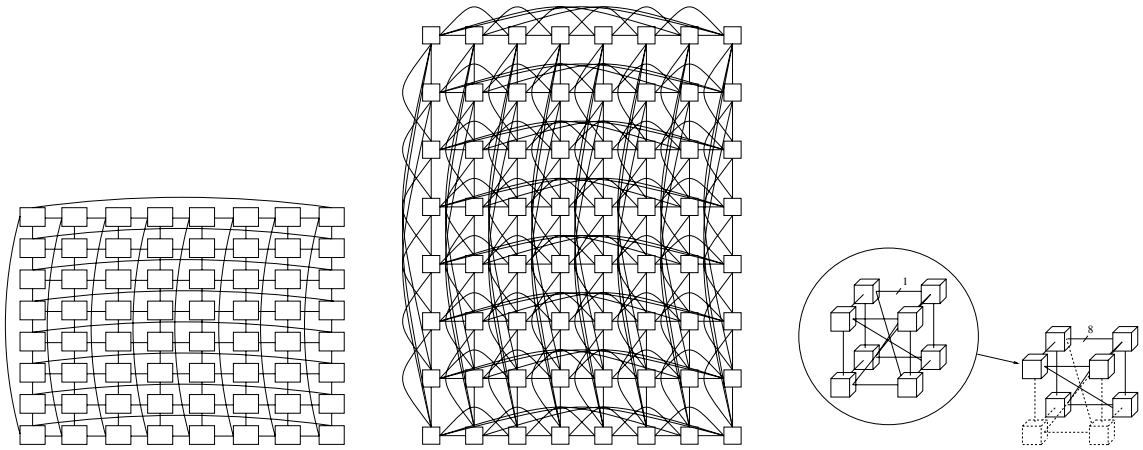
Diferentes topologías irregulares van a ser utilizadas para la evaluación de los mecanismos. Las topologías irregulares son generadas de forma aleatoria, considerando solamente tres restricciones, las cuales han sido ya utilizadas en otros estudios [Sil97a, Sil97b, Sil97c]:

- El número de nodos conectados a un conmutador es constante e igual a 4.
- Todos los conmutadores tienen el mismo tamaño. Utilizamos conmutadores de 8 puertos. Por lo tanto, hay 4 puertos disponibles para conectar con otros conmutadores.
- Dos conmutadores vecinos están conectados a través de un sólo enlace.

A su vez, también serán evaluadas diferentes topologías regulares. Estas topologías se han seleccionado tomando en consideración recomendaciones de Myricom. Algunas topologías (como CPLANT) han sido utilizadas en instalaciones ya existentes equipadas con red Myrinet. Las topologías son las siguientes:

- **Toro 2D**. La red Toro 2D está formada por 64 conmutadores de 16 puertos. Cada conmutador está conectado a cada uno de sus cuatro vecinos a través de un sólo enlace. Hay 8 *hosts* conectados a cada conmutador, por lo que hay 512 *hosts* en el sistema. Hay 4 puertos sin conectar en cada conmutador. La red Toro 2D se muestra en la figura 61.a.
- **Toro 2D con canales *express***. Esta topología es similar al Toro 2D excepto que en esta topología todos los conmutadores se conectan también a sus vecinos de segundo orden utilizando canales *express* [Dal91] (conmutadores que están ubicados a dos saltos en cada dimensión). Cada conmutador tiene 16 puertos. Hay 8 *hosts* conectados a cada conmutador, por lo que hay 512 *hosts* en todo el sistema. Todos los puertos están utilizados en todos los conmutadores. La red Toro-2D con canales *express* se muestra en la figura 61.b.
- **CPLANT**. Esta topología es utilizada en el Computational Plant (CPLANT) en el Sandia National Labs. [Rie99]. Está formada por 50 conmutadores de 16 puertos conectando 400 *hosts* (cada conmutador tiene 8 *hosts* conectados). De estos, 48 conmutadores están agrupados en 6 grupos de 8 conmutadores.

Cada conmutador utiliza 4 puertos para conectarse a otros conmutadores en el mismo grupo y 4 puertos para conectarse con sus conmutadores equivalentes en los restantes grupos. Los seis grupos forman un hipercubo incompleto, que también tiene conexiones entre los conmutadores más alejados. Los dos conmutadores restantes forman un grupo adicional. Por lo tanto, la topología resultante no es completamente regular. La topología utilizada en el CPLANT se muestra en la figura 61.c.



(a) (b) (c)

Figura 61: Topologías regulares: (a) Toro 2D, (b) Toro 2D con canales *express* y (c) red CPLANT de Sandia National Labs.

4.5 Carga de la red

Como ya hemos indicado, utilizaremos un generador de carga sintética para modelar la carga de la red. En particular, hay que decidir cuál es la distribución de destinos de los mensajes, su longitud y la tasa de generación de mensajes.

En lo que respecta a la distribución de destinos de los mensajes, todos los nodos utilizan la misma en cada experimento. Con el objeto de analizar el comportamiento de la red en las más variadas condiciones de carga, hemos considerado cuatro posibilidades: uniforme, con localidad, *hot-spot* y *bit-reversal*.

- **Distribución uniforme de destinos**. En este caso, el destino del mensaje generado en un nodo se elige aleatoriamente de entre todos los nodos de la red.

Este patrón de comunicación es el que más se ha utilizado en otros trabajos [Dal92, Chi92, Bop93, Dua94].

- **Distribución de destinos con radio de localidad**. El destino del mensaje se elige aleatoriamente de entre los nodos ubicados a una distancia en saltos del nodo fuente no superior a un límite en saltos l. En otras palabras, corresponde a una distribución uniforme de destinos entre los nodos incluidos en la zona del nodo fuente. Hemos considerado dos valores para la localidad: $l = 3$ y $l = 5$, intentando modelar tráfico local de corta y media distancia. Este patrón de comunicación es más frecuente en la realidad, ya que se tiende a ubicar cercanos aquellos procesos que más cooperan.

- **Distribución *hot-spot***. En este caso, un tanto por cien de los mensajes es dirigido a un *host* en particular. El tanto por cien se denomina tasa *hot-spot*. El resto del tráfico es distribuido uniformemente entre todos los nodos, siguiendo la distribución uniforme de destinos. Diferentes tasas *hot-spot* serán utilizadas en función del tamaño de las redes que evaluamos. Este patrón de comunicación modela la situación en que existe un *host* (por ejemplo un servidor) frecuentemente accedido en el sistema.

- **Distribución *bit-reversal***. Finalmente, hemos considerado un patrón de comunicación específico entre parejas de nodos. Este patrón se ha seleccionado tomando en cuenta las permutaciones que habitualmente se realizan en los algoritmos numéricos paralelos [Mil91, Kim92]. En concreto, hemos utilizado la distribución *bit-reversal*, en la cual el nodo con identificador en binario $a_{n-1}, a_{n-2}, ..., a_1, a_0$ se comunica con el nodo con identificador $a_0, a_1, ..., a_{n-2}, a_{n-1}$ en binario. En este caso, el nodo destino de los mensajes generados por un nodo determinado es siempre el mismo.

No obstante, con el objeto de evaluar diferentes patrones de tráfico de forma simultánea, también evaluaremos la red con un patrón de tráfico combinado. En este patrón de tráfico, cada nodo utilizará en cada envío de un mensaje una de las cuatro distribuciones anteriores, seleccionándola de forma aleatoria y con las mismas probabilidades.

Por otra parte, hemos analizado el comportamiento de la red de interconexión considerando tamaños de mensaje de 32 bytes y de 512 bytes. Asimismo, hemos evaluado también la combinación de los dos tamaños de mensaje. En concreto, en

este caso, el 70% de los mensajes utilizados han sido de 32 bytes y el resto de 512 bytes. Este tráfico representa el tráfico típico encontrado en un sistema *cluster* en el que los mensajes no son todos del mismo tamaño. Por ejemplo, en los sistemas DSM *software* típicamente coexisten dos tamaños de mensajes: mensajes de datos y mensajes de control de la coherencia del sistema. Los mensajes de control de la coherencia suelen ser mensajes cortos del orden de decenas de bytes con información de control del sistema. Por otra parte, los mensajes de datos suelen ser bloques de memoria utilizados por el propio mecanismo de la coherencia. Estos mensajes suelen ser del tamaño de centenares de bytes.

Finalmente, hemos considerado que la tasa de generación de mensajes es constante e igual para todos los nodos. En cada experimento hemos evaluado todo el rango de tráfico, desde baja carga hasta saturación.

4.6 Validación del método de evaluación

4.6.1 Análisis de la estabilización de la red

En toda simulación existe un periodo transitorio de estabilización del sistema. Durante este periodo no se pueden tomar medidas fiables, ya que el sistema no está estabilizado. Una vez el sistema se estabiliza, entra en el periodo permanente o estacionario.

La duración del periodo transitorio depende de muchos factores, entre ellos, los parámetros del sistema que se está evaluando. En la presente tesis se van a evaluar multitud de sistemas utilizando diferentes parámetros, por lo que un análisis de todos los casos estudiados resultaría inviable. Por ello, en este apartado evaluamos la duración del periodo transitorio en aquellos sistemas que consideramos más representativos de todos los que vamos a evaluar. Para ello, vamos a dar valores fijos a los siguientes parámetros:

- Topologías. Se va a utilizar una topología irregular de 32 conmutadores.
- Tamaño de los mensajes. Vamos a escoger un tamaño de mensaje de 1024 bytes[1].

[1] Aunque en el capítulo de evaluación se utilizan tamaños de mensajes de 32 y 512 bytes, este tamaño se ha escogido porque representa un caso más desfavorable para alcanzar el estado permanente.

- Distribución de destinos. Utilizamos la distribución uniforme de destinos.
- Algoritmo de encaminamiento. Elegimos el algoritmo de encaminamiento *up*/down**.

A su vez, la duración del transitorio también depende del tráfico inyectado. Esto se debe a que un sistema que soporte cargas de tráfico mayores, necesitará de un tiempo mayor para llegar a la cota de tráfico requerida. Normalmente, para cargas de tráfico bajas, el régimen permanente se alcanza rápidamente (del orden de decenas de miles de mensajes inyectados). Sin embargo, al aumentar el tráfico, el régimen permanente se alcanza con un número de mensajes superior. En este análisis, vamos a estudiar cuatro puntos de tráfico diferentes. En la figura 62 se muestra la curva típica de latencia frente a tráfico que se suele obtener, en la cual se han marcado los cuatro puntos de estudio. En el primer punto, se analiza el transitorio para cargas bajas, en el segundo punto se analiza para cargas medias, en el tercer punto se analiza cuando el sistema está entrando en saturación y, por último, en el cuarto punto, se estudia la duración del periodo transitorio en saturación.

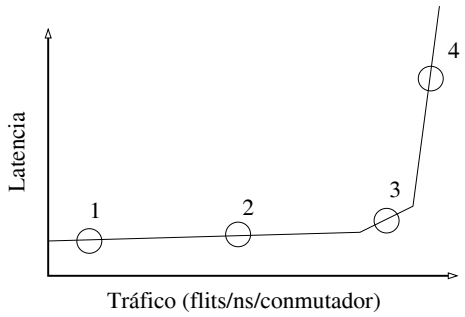
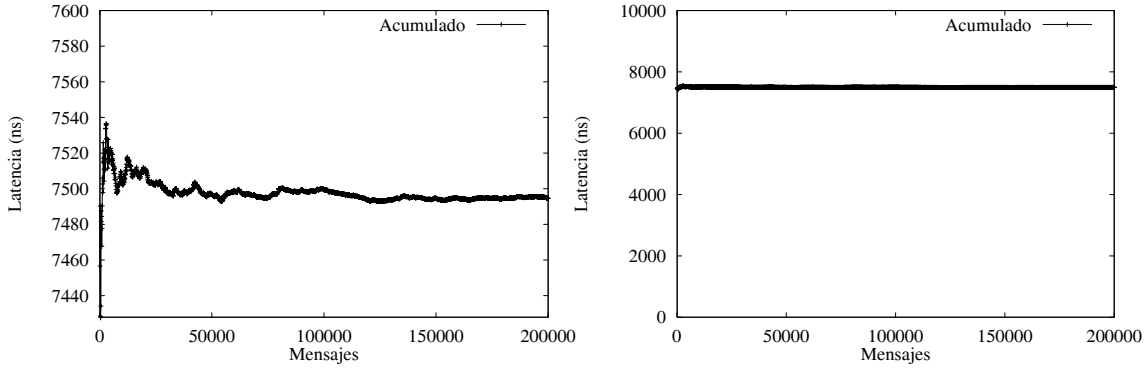


Figura 62: Puntos de análisis del transitorio.

Para analizar el estado transitorio, vamos a analizar los valores obtenidos de la latencia de los mensajes (latencia media de la red expresada en ns) y, en algunos casos, de la latencia de los mensajes desde la generación (incluyendo el tiempo de espera en las colas de inyección). Analizaremos los valores acumulados de la latencia tras promediar una muestra compuesta por los últimos 100 mensajes recibidos. El número total de mensajes simulados ha sido de 200000. Estas muestras las representaremos gráficamente y realizando un sencillo análisis visual, comprobaremos cuándo alcanzamos el estado permanente.

En la figura 63 vemos la evolución de la latencia media de los mensajes con baja carga de tráfico. Observamos que la latencia media se estabiliza muy rápidamente con muy pocos mensajes. Aunque en los primeros mensajes se observan algunas diferencias en los valores de la latencia (figura 63.a), estas diferencias no son significativas (figura 63.b) con respecto a la media (diferencias menores de 20 ns). El valor medio de la latencia se estabiliza rápidamente y las diferencias máximas no superan los 10 ns (que significan el 0.1% con respecto a la media) a partir de los 25000 mensajes.

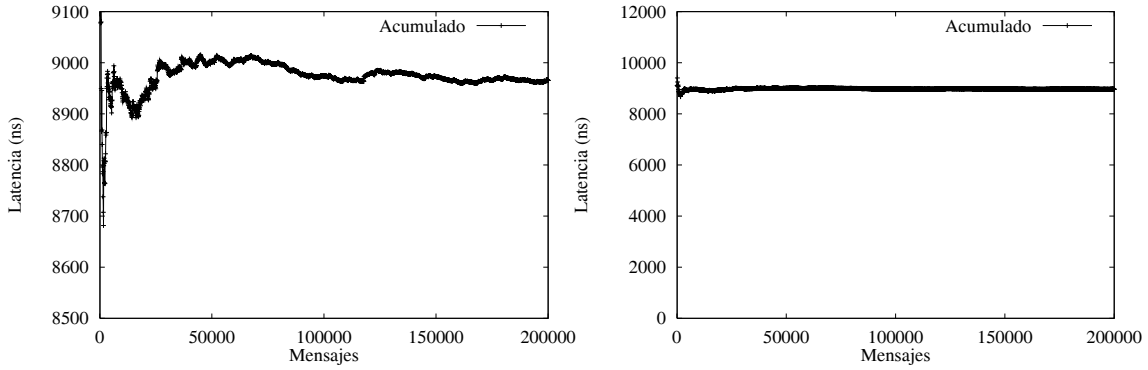


(a) (b)

Figura 63: Evolución de la latencia media acumulada de los mensajes. Baja carga. 32 conmutadores.

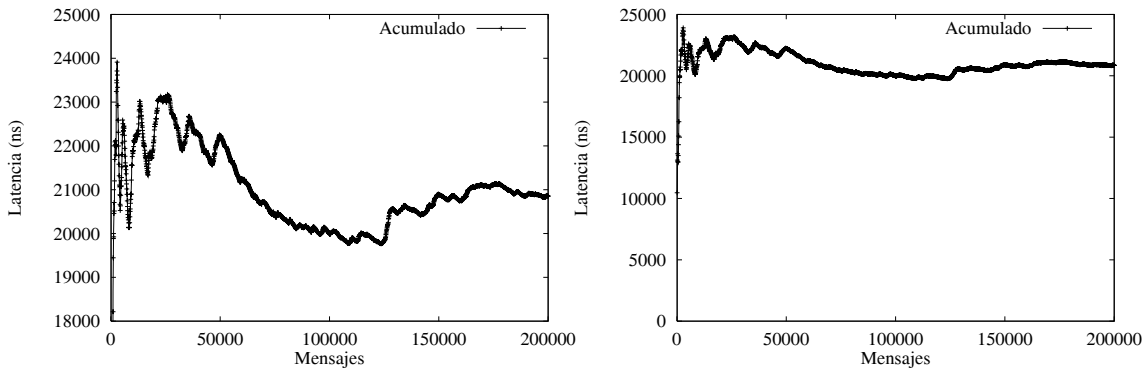
Para cargas de tráfico medias (figura 64.a) observamos como el estado permanente se alcanza un poco más tarde. Aunque existe una mayor diferencia de la latencia acumulada respecto de la media, los valores de latencia no difieren en más de 100 ns con respecto a la media. En la figura 64.b podemos ver cómo esta diferencia es muy poco significativa representando un 1.1% con respecto a la media. Por lo tanto, en condiciones de tráfico bajo o medio, con relativamente pocos mensajes (50000) se alcanza el estado permanente del sistema.

Cuando el sistema está en puntos cercanos a la saturación, el estado transitorio es más largo. En la figura 65.a vemos la variación de la latencia acumulada. Podemos observar como aparentemente la latencia no se estabiliza por debajo de los 200000 mensajes inyectados. Sin embargo, comparando las diferencias con la media, podemos ver en la figura 65.b como realmente el sistema se ha estabilizado a partir de los 100000. A partir de los 100000 mensajes la diferencia de la latencia con la



(a) (b)

Figura 64: Evolución de la latencia media acumulada de los mensajes. Media carga. 32 conmutadores.



(a) (b)

Figura 65: Evolución de la latencia media de los mensajes. Saturando. 32 conmutadores.

media es menor del 4.7%. No obstante, vamos a considerar por precaución, que en estos puntos de tráfico el estado permanente se obtendrá a partir de los 200000 mensajes.

Por último, en la figura 66.a vemos la variación de la latencia cuando la red está saturada. En estos puntos, la red no es capaz de aceptar todo el tráfico inyectado. Como podemos observar, la latencia acumulada no se estabiliza ya que continua incrementándose poco a poco conforme transcurre el tiempo de simulación.

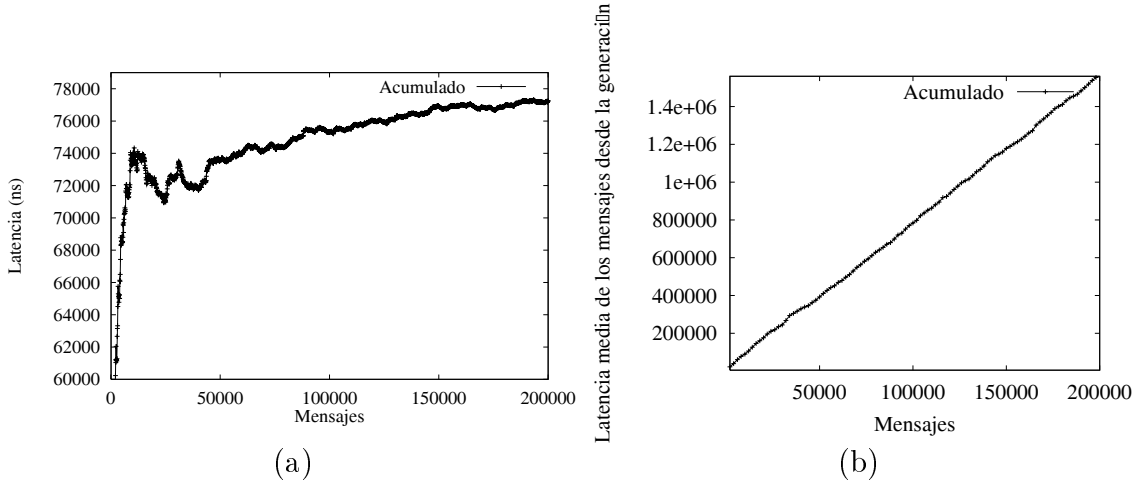


Figura 66: Evolución de la latencia media acumulada de los mensajes. Sistema saturado. 32 conmutadores. (a) Latencia de la red y (b) Latencia desde la generación.

Realmente, en este punto de saturación, las colas de inyección a la red crecen continuamente, pudiendo llegar a desbordarse. La figura 66.b muestra la evolución de la latencia media de los mensajes desde la generación. Como podemos observar, la latencia desde la generación crece linealmente con el número de mensajes inyectados sin llegar a converger. Por lo tanto, en estos puntos de tráfico no podemos alcanzar ninguna conclusión respecto a la latencia. Aunque estos puntos de tráfico podrían ser alcanzados por sistemas reales donde las aplicaciones demanden un tráfico superior al que la red puede soportar, esto sólo ocurre durante un corto intervalo de tiempo, ya que el tráfico generado por las aplicaciones reales suele ser a ráfagas [Fli99a]. En cualquier caso, en esta tesis no estamos especialmente interesados en analizar el comportamiento de la red más allá de su punto de saturación.

Como conclusión, el estado transitorio depende efectivamente del nivel de tráfico solicitado al sistema. Para cargas de tráfico bajas y medias, consideramos que el estado permanente se ha alcanzado a los 50000 mensajes, mientras que en puntos próximos a la saturación consideraremos el inicio del estado permanente a los 200000 mensajes. Con respecto al periodo de tiempo durante el cual se tomarán resultados (duración del estado estacionario) en las simulaciones de la presente tesis, prolongaremos el estado permanente el tiempo necesario para que se reciban 200000 mensajes adicionales, sobre los cuales se tomarán los resultados.

4.6.2 Intervalo de confianza

Con el fin de obtener los intervalos de confianza en las simulaciones utilizaremos el método de las réplicas. Por lo tanto, medimos, para diferentes tasas de inyección de tráfico, los intervalos de confianza del tráfico aceptado y de la latencia, a partir de 20 simulaciones con diferente semilla y considerando un nivel de confianza del 95%. Las simulaciones utilizan los periodos de transitorio y permanente calculados en la sección anterior.

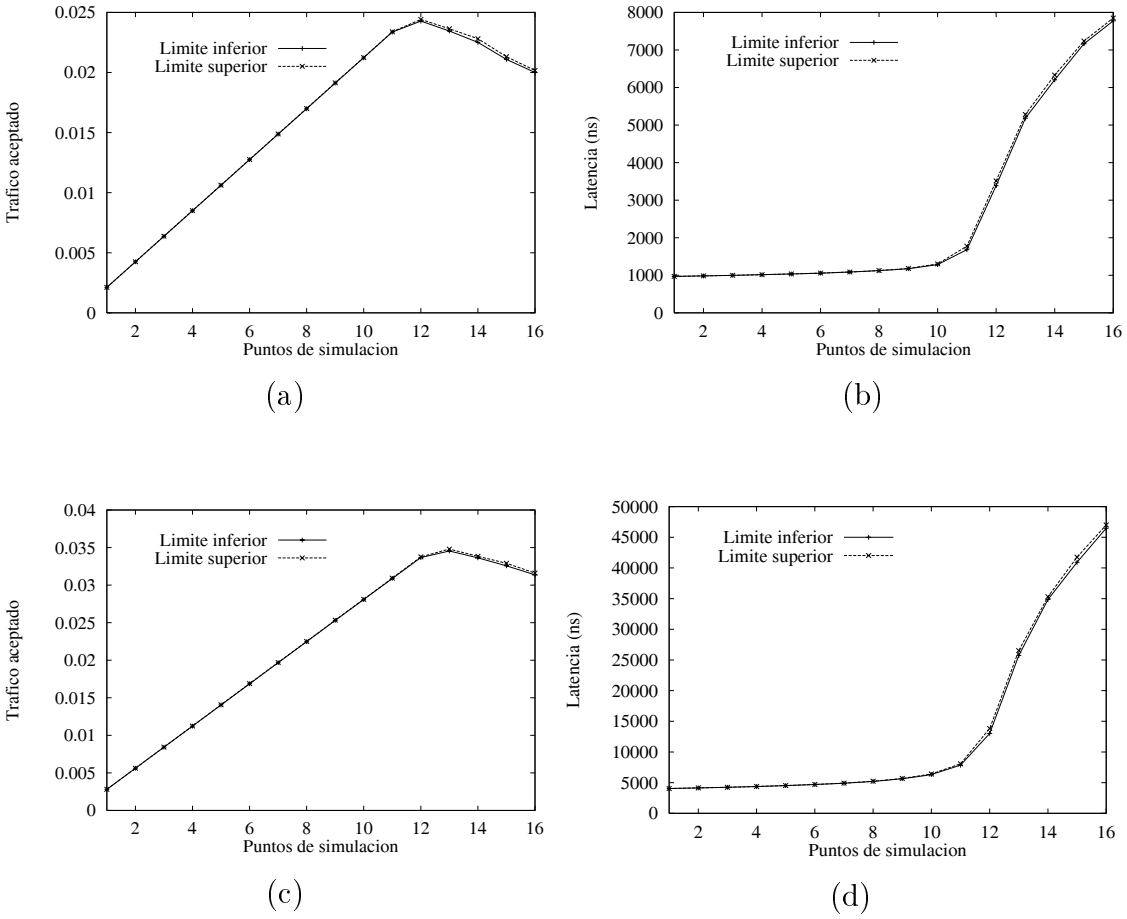


Figura 67: Intervalo de confianza. (a, c) Tráfico y (b, d) Latencia. Tamaño de mensajes de (a, b) 32 bytes y (c, d) 512 bytes. 32 conmutadores. Distribución uniforme de destinos.

Como ejemplo, en esta sección, mostramos los resultados obtenidos para el algoritmo de encaminamiento *up*/down** sobre una red irregular de 32 conmutadores

utilizando una distribución uniforme de destinos y para dos tamaños de mensajes (32 y 512 bytes).

En la figura 67 podemos observar los extremos correspondientes al intervalo de confianza calculado a partir de la media de los resultados medios obtenidos por cada una de las 20 simulaciones. Se muestran tanto los intervalos de confianza del tráfico aceptado (figuras 67.a y 67.c) como de la latencia (figuras 67.b y 67.d), para mensajes de 32 bytes (figuras 67.a y 67.b) y de 512 bytes (figuras 67.c y 67.d).

Como podemos observar, para cargas de tráfico bajas o medias (puntos de simulación de 1 a 10) el intervalo de confianza es muy pequeño. En concreto, los intervalos de confianza obtenidos para el tráfico aceptado son menores de 0,0000195 (0.01%) para mensajes de 32 bytes y menores de 0,0000139 (0.04%) para mensajes de 512 bytes. En cuanto a la latencia, los intervalos de confianza también son muy pequeños. En concreto son menores de 46,92 ns (1.5%) para mensajes de 32 bytes y menores de 112,10 ns (0.44%) para mensajes de 512 bytes.

En cargas de tráfico cercanas a la saturación del sistema (punto de simulación 12) la diferencia entre ambos límites empieza a ser apreciable a simple vista en la figura. No obstante, dicha diferencia entre ambos límites es exigua. En términos de latencia, se obtiene un intervalo de confianza de 63 ns (1%) para mensajes de 32 bytes y de 402 ns (2.68%) para mensajes de 512 bytes.

La obtención de estos intervalos de confianza tan pequeños valida, desde otro punto de vista, la elección de la duración de los periodos transitorios y permanentes seleccionados en el apartado anterior. Por otra parte, dado que los intervalos de confianza obtenidos son tan pequeños, en los resultados de simulación mostrados en esta tesis, no se mostrarán dichos intervalos de confianza.

Capítulo 5

Evaluación

En este capítulo evaluamos todos los mecanismos y algoritmos propuestos en los capítulos anteriores. En primer lugar, evaluamos las propuestas de utilización de rutas alternativas (capítulo 2), tanto para distribuir el tráfico uniformemente (sección 2.3) entre todas las rutas disponibles como utilizando el mecanismo de detección de contención (sección 2.4) como selector de las rutas a utilizar en cada momento. Seguidamente, evaluamos el mecanismo de buffers en tránsito (capítulo 3), aplicándolo en primer lugar sobre los algoritmos de encaminamiento tradicionales (secciones 3.2.1 y 3.2.2) para después evaluar el nuevo algoritmo de encaminamiento basado exclusivamente en ITBs (sección 3.2.3). En este último apartado estudiaremos todas las aproximaciones propuestas para la ubicación de ITBs.

Por último, evaluamos todas las propuestas relacionadas con la disminución de la latencia del mecanismo de buffers en tránsito (sección 3.3), así como algunos aspectos relacionados con la sensibilidad del mecanismo ITB ante cambios en los parámetros de diseño tales como tiempo de detección del mecanismo, tiempo de reprogramación del mecanismo, el efecto del mecanismo sobre los mensajes locales, etc.

5.1 Rutas alternativas

En esta sección evaluamos los algoritmos de selección de rutas propuestos en el capítulo 2. En primer lugar estudiamos el efecto de la selección de rutas alternativas para obtener una mejor distribución del tráfico. Para ello, los nodos seleccionarán las rutas localmente sin tener en cuenta información del tráfico. Por lo tanto, los

nodos utilizarán un algoritmo de selección de rutas. En concreto, vamos a evaluar tres algoritmos de selección de rutas:

- **OSUD (One Shortest Up*/Down* path)**. Cada nodo utilizará solamente una ruta *up*/down** (y siempre la misma) para encaminar los mensajes hacia un determinado destino. La ruta utilizada será seleccionada aleatoriamente en el proceso de cálculo de rutas de entre todas las rutas más cortas posibles para cada destino. Una vez seleccionada la ruta, siempre utilizará la misma ruta para enviar mensajes al destino. Con este algoritmo de selección de rutas modelamos el comportamiento típico en las redes actuales como Myrinet.

- **RSUD (Random Shortest Up*/Down* path)**. Cada nodo utilizará aleatoriamente una ruta *up*/down** de entre las rutas *up*/down** más cortas. Cada vez que el nodo tenga que enviar un mensaje a un destino determinado, seleccionará aleatoriamente una ruta hacia dicho destino.

- **RRSUD (Round-Robin Shortest Up*/Down* path)**. Cada nodo utilizará todas las rutas *up*/down** más cortas posibles utilizando un mecanismo de selección tipo *round-robin*. Por lo tanto, las rutas disponibles se irán seleccionando en orden.

En segundo lugar, vamos a estudiar el efecto de la selección de rutas alternativas en función del tráfico detectado con el mecanismo propuesto en la sección 2.4. Con este mecanismo, se asociará a cada ruta el coeficiente de contención C_c del último mensaje enviado. Con este índice se decidirá qué ruta utilizar al realizar el siguiente encaminamiento hacia el destino. En concreto, se aplicarán los algoritmos de las figuras 35 y 36.

Para todas las evaluaciones que realizamos en esta sección, el cálculo de las rutas alternativas se realizará considerando únicamente las rutas más cortas y limitando a 10 el número de rutas entre cada par origen-destino.

5.1.1 Distribución del tráfico

La figura 68 muestra los resultados obtenidos al utilizar los diferentes algoritmos de selección de rutas para redes de 8, 16, 32 y 64 conmutadores, respectivamente. El tamaño de los mensajes es de 512 bytes y se ha utilizado una distribución uniforme de destinos.

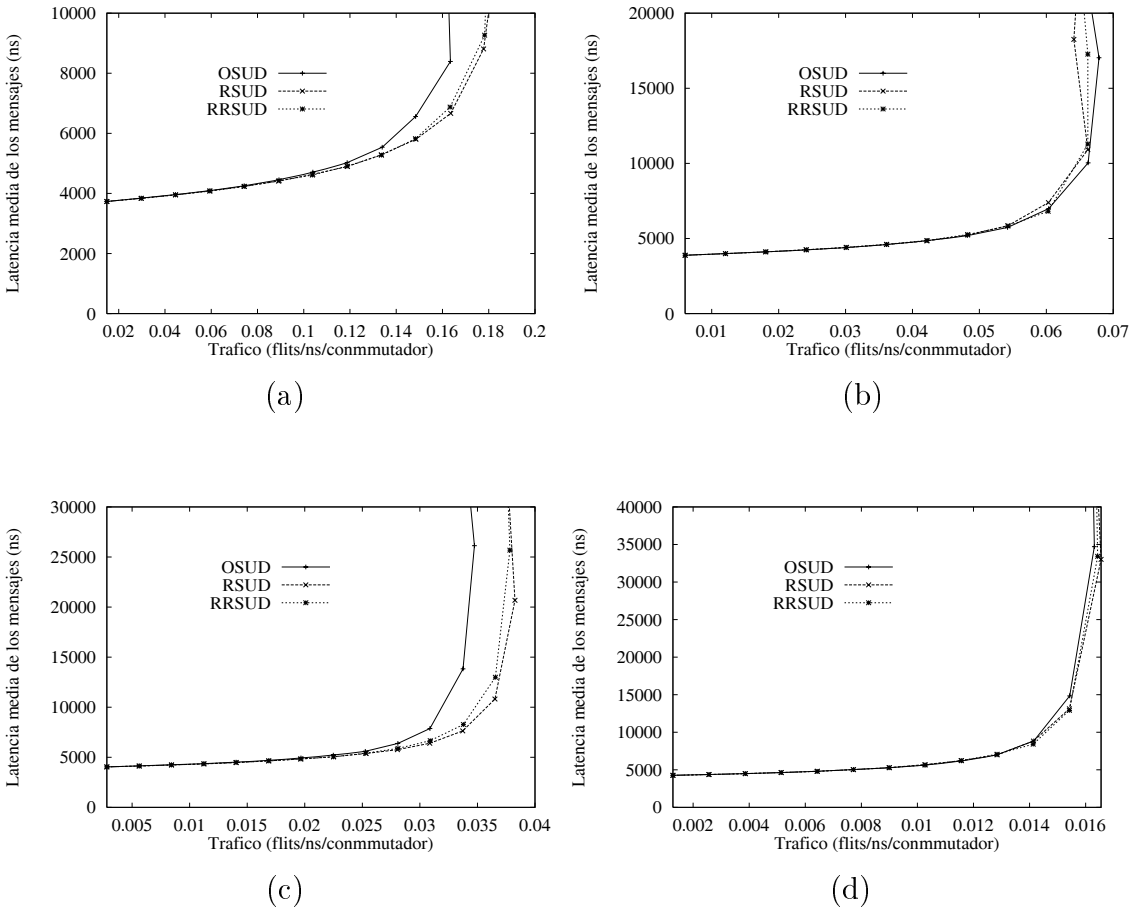


Figura 68: Algoritmos de selección de rutas. Latencia media vs. tráfico. Red de (a) 8, (b) 16, (c) 32 y (d) 64 conmutadores. Tamaño de mensajes de 512 bytes. Distribución uniforme de destinos.

Como podemos observar, en la red de 8 conmutadores (figura 68.a) los algoritmos de selección de rutas RRSUD y RSUD obtienen una productividad mayor que el algoritmo de selección original OSUD. El incremento en productividad es de un 12%. Este incremento en productividad es debido a que los algoritmos RRSUD y RSUD distribuyen el tráfico entre las rutas alternativas, mientras que el algoritmo OSUD siempre utiliza la misma ruta para cada par origen-destino.

Sin embargo, al aumentar el tamaño de la red (figuras 68.b, 68.c y 68.d), podemos observar como el beneficio en términos de productividad obtenido por los algoritmos RRSUD y RSUD no aumenta. Más aún, el incremento en productividad de RRSUD

y RSUD respecto a OSUD disminuye. Para la red de 64 conmutadores, los tres algoritmos de selección de rutas obtienen prácticamente la misma productividad. Al aumentar el tamaño de la red, la longitud media de las rutas también aumenta. Al aumentar la longitud media de las rutas, las rutas alternativas tienden a compartir más enlaces entre ellas, por lo que las rutas alternativas tienden a comportarse como una única ruta y, por consiguiente, los algoritmos de selección de rutas obtienen prácticamente las mismas prestaciones que el algoritmo que utiliza solamente una ruta para cada par origen-destino.

La tabla 2 muestra los factores de incremento de productividad obtenidos por los algoritmos de selección RRSUD y RSUD respecto del algoritmo de selección OSUD. La tabla muestra factores de incremento mínimos, máximos y medios para 10 topologías generadas aleatoriamente por cada tamaño de red evaluado. Como podemos observar, los incrementos medios en productividad obtenidos por los algoritmos RRSUD y RSUD respecto del algoritmo de selección OSUD son muy pequeños y sólo son importantes en redes pequeñas de 8 conmutadores, donde se alcanzan factores de incremento de productividad máximos del 1.25 (25% de incremento). No obstante, en la tabla podemos observar como, al aumentar el tamaño de la red, los incrementos de productividad de los algoritmos de selección RRSUD y RSUD disminuyen. A partir de redes de 16 conmutadores, los algoritmos RRSUD y RSUD no incrementan la productividad en más de un factor de incremento de 1.05 por término medio (5% de incremento).

	RRSUD vs OSUD			RSUD vs OSUD		
Conm	Mín	Máx	Media	Mín	Máx	Media
8	1.00	1.20	1.10	1.01	1.25	1.14
16	0.98	1.10	1.05	0.98	1.12	1.05
32	1.00	1.09	1.04	1.00	1.13	1.05
64	1.00	1.08	1.02	1.00	1.08	1.04

Tabla 2: Algoritmos de selección de rutas. Factor de incremento de productividad al utilizar RRSUD y RSUD respecto de OSUD. Distribución uniforme de destinos. Tamaño de mensajes de 512 bytes.

Por lo tanto, los algoritmos de selección de rutas solamente incrementan la productividad en redes pequeñas (8 conmutadores) donde las rutas alternativas tienen un grado de compartición de enlaces bajo. Sin embargo, para redes más grandes, estos algoritmos de encaminamiento incrementan muy poco la productividad ya que

las rutas alternativas tienden a comportarse como una única ruta debido al elevado grado de compartición de enlaces.

5.1.2 Selección del tráfico en función de la contención

Vamos ahora a utilizar el mecanismo de detección de contención junto con las rutas alternativas. Las rutas calculadas serán las mismas que en la sección anterior. No obstante, ahora las rutas se seleccionarán en función del tráfico local detectado con el mecanismo de detección de contención descrito en la sección 2.4.

A la combinación del mecanismo de detección y la posterior selección de la ruta lo denominamos algoritmo DET. Este algoritmo lo comparamos con el algoritmo OSUD y con el mejor algoritmo de selección de rutas de la sección anterior (el denominado RRSUD). En la figura 69 podemos observar los resultados obtenidos por los algoritmos para diferentes tamaños de red, distribución uniforme de destinos y tamaño de mensajes de 512 bytes.

De la figura podemos concluir que el mecanismo de detección de tráfico (algoritmo DET) apenas ayuda a incrementar las prestaciones obtenidas por el algoritmo de selección de rutas RRSUD. De hecho, se obtienen prácticamente las mismas prestaciones con el algoritmo RRSUD que con el algoritmo DET.

En la tabla 3 podemos ver los factores de incremento en productividad mínimos, máximos y medios del algoritmo DET comparado con los algoritmos OSUD y RRSUD. El mecanismo de detección de tráfico incrementa muy poco las prestaciones en términos de productividad ya obtenidas por el algoritmo de selección RRSUD. El incremento medio nunca supera el 5% respecto RRSUD en cualquier tamaño de red.

	Detección vs OSUD			Detección vs RRSUD		
Conm.	Mín	Máx	Media	Mín	Máx	Media
8	1.01	1.25	1.14	0.92	1.11	1.04
16	1.01	1.14	1.08	0.97	1.07	1.03
32	0.99	1.14	1.05	0.94	1.05	1.01
64	1.00	1.11	1.07	1.00	1.11	1.05

Tabla 3: Detección de tráfico y selección de rutas. Factor de incremento de productividad al utilizar el mecanismo de detección de tráfico respecto de OSUD y RRSUD. Tamaño de mensajes de 512 bytes. Distribución uniforme de destinos.

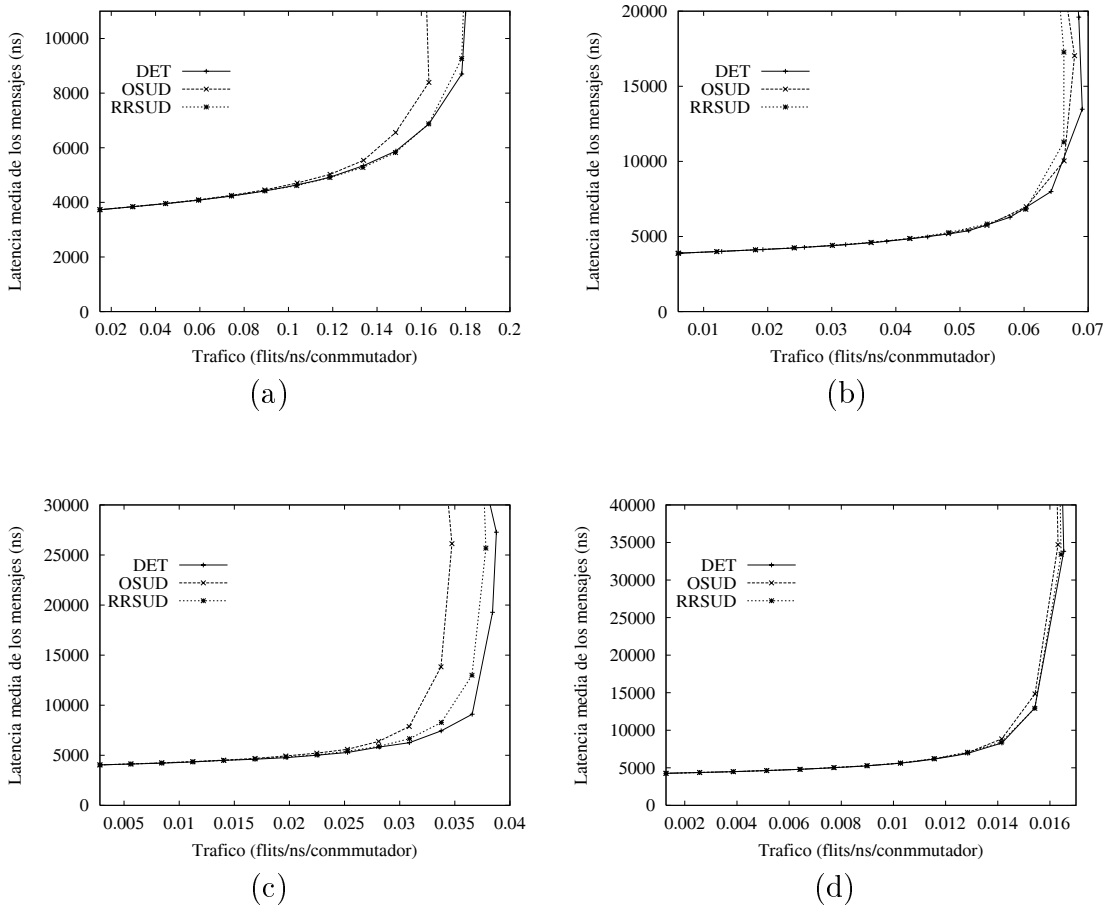


Figura 69: Detección de tráfico y selección de rutas. Latencia media vs. tráfico. Red de (a) 8, (b) 16, (c) 32 y (d) 64 conmutadores. Tamaño de mensajes de 512 bytes. Distribución uniforme de destinos.

Este comportamiento parecido de los dos algoritmos es debido a que el algoritmo DET está utilizando las mismas rutas que el algoritmo RRSUD, por lo que sufre de los mismos problemas derivados de la compartición de enlaces por parte de las rutas alternativas. Por lo tanto, el mecanismo de detección de contención no es operativo para evitar zonas congestionadas ya que no existen rutas verdaderamente alternativas para evitar dichas zonas congestionadas. Por lo tanto, los mecanismos de selección de rutas y de detección de tráfico (y posterior selección de rutas) no son útiles a menos que tengan disponibles rutas realmente alternativas.

Hay que hacer constar que en determinadas topologías regulares (p.ej. malla

2D) van a existir más y mejores rutas alternativas. No obstante, en la presente tesis estamos interesados en mejorar las prestaciones de las redes con cualquier topología de red, considerando a las redes irregulares como las más importantes. Ya que en estas redes irregulares van a existir muchas restricciones de encaminamiento que dificultan el encontrar rutas disjuntas, la selección de rutas alternativas no parece ser la mejor forma de incrementar sustancialmente las prestaciones de dichas redes.

5.2 ITBs sobre los algoritmos tradicionales

Como hemos visto, el utilizar rutas alternativas mejora muy poco las prestaciones de las redes debido a las restricciones de encaminamiento. Por lo tanto, a partir de este punto nos centramos en eliminar las dependencias prohibidas por dichas restricciones de encaminamiento. Para ello, utilizamos el mecanismo de buffers en tránsito (ITBs) propuesto en el capítulo 3. Como ya se ha comentado, dicho mecanismo puede ser utilizado tanto para eliminar las dependencias prohibidas entre canales de los algoritmos de encaminamiento tradicionales como para diseñar nuevos algoritmos de encaminamiento que utilicen de forma exclusiva el mecanismo.

En esta sección nos centramos exclusivamente en la aplicación del mecanismo sobre algoritmos de encaminamiento tradicionales, dejando para la siguiente sección la evaluación del nuevo algoritmo de encaminamiento con ITBs. Los algoritmos sobre los que vamos a aplicar el mecanismo ITB son: *up*/down**, *DFS* y *smart-routing*. Para ello, en primer lugar vamos a evaluar dichos algoritmos de encaminamiento sin utilizar ITBs para poder observar las características y prestaciones de cada uno. Estos algoritmos los denominaremos UD, DFS y SMART, respectivamente. Seguidamente, evaluaremos el mecanismo ITB cuando es aplicado a dichos algoritmos de encaminamiento, analizando cómo influye su uso en las prestaciones. La metodología utilizada para aplicar el mecanismo ITB sobre cada algoritmo de encaminamiento es la descrita en la sección 3.2.1 para los algoritmos *up*/down** y *DFS* y la descrita en la sección 3.2.2 para el algoritmo *smart-routing*.

5.2.1 Algoritmos tradicionales sin ITBs

La figura 70 muestra los resultados obtenidos por los algoritmos de encaminamiento UD, DFS y SMART para una distribución uniforme de destinos, tamaño de mensajes de 512 bytes y para diferentes redes irregulares de 8, 16, 32 y 64 conmutadores,

respectivamente. El algoritmo de encaminamiento SMART no se muestra para redes de 64 conmutadores ya que no se han podido generar las tablas de encaminamiento debido a su prohibitivo tiempo de cálculo.

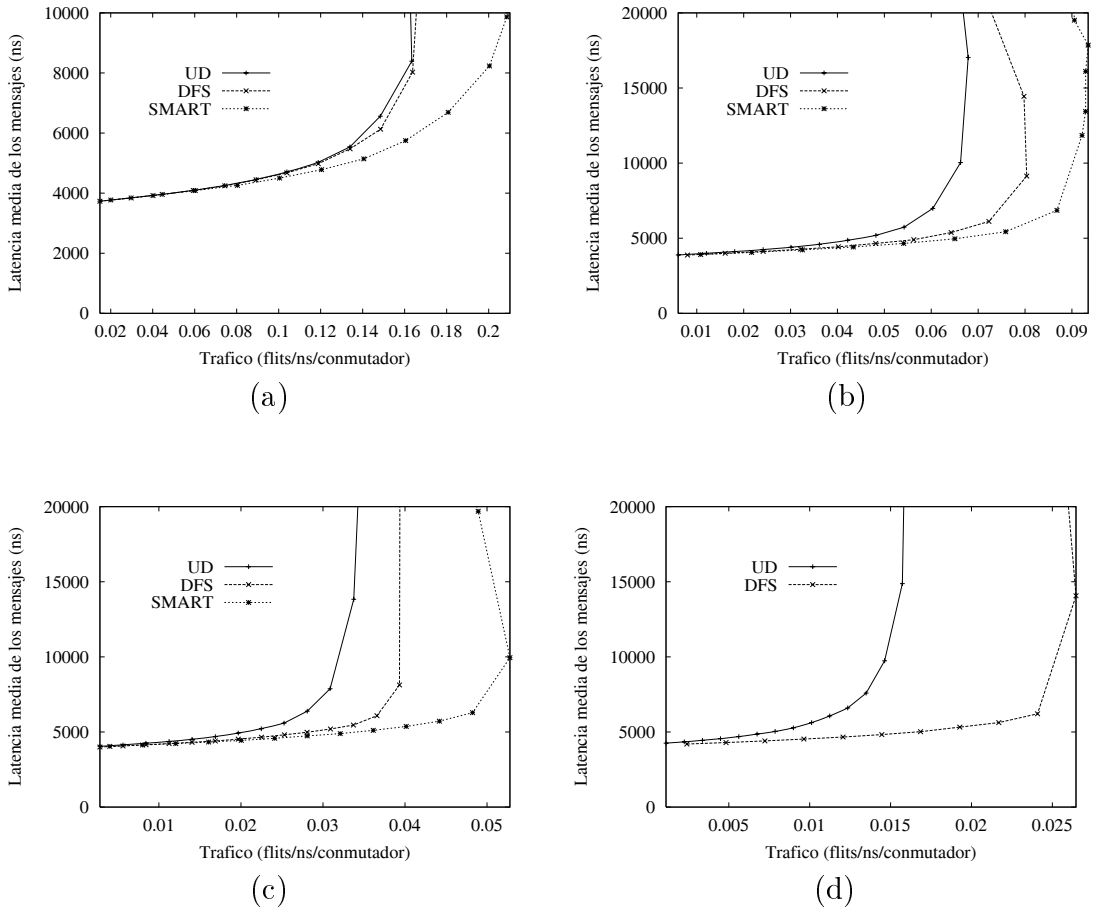
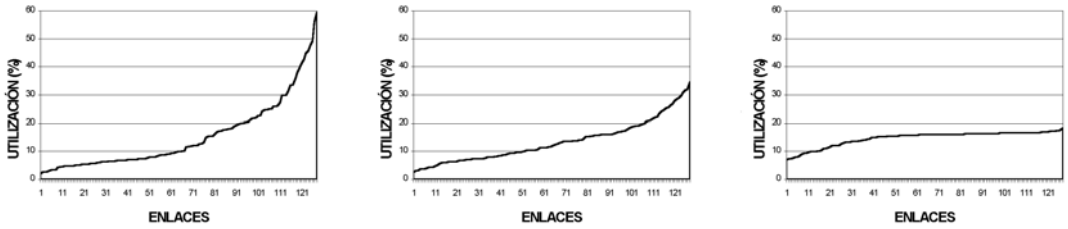


Figura 70: Algoritmos de encaminamiento originales. Latencia media vs. tráfico. Red de (a) 8, (b) 16, (c) 32 y (d) 64 conmutadores. Tamaño de mensajes de 512 bytes. Distribución uniforme de destinos.

Como se puede observar, el mejor algoritmo de encaminamiento es el SMART, obteniendo la mayor productividad en todas las topologías en las que se ha utilizado. En particular, para la red de 16 conmutadores, SMART incrementa la productividad de UD y DFS en factores de 1.39 y 1.16, respectivamente. También podemos observar como al aumentar el tamaño de la red, SMART incrementa la productividad respecto UD y DFS. En la red de 32 conmutadores, SMART incrementa en

factores de 1.53 y 1.33 las productividades de UD y DFS, respectivamente. Por otra parte, para la red de 64 conmutadores, el mejor algoritmo de encaminamiento es DFS (SMART no estaba disponible). De hecho, DFS obtiene mayor productividad que UD en todas las redes, incrementando los beneficios en productividad conforme aumenta el tamaño de la red.

La mayor productividad obtenida por SMART es debida a que SMART obtiene un mejor equilibrado del tráfico que los otros dos algoritmos de encaminamiento. La figura 71 muestra la utilización de los enlaces entre conmutadores en la red de 32 conmutadores al utilizarse los algoritmos de encaminamiento UD, DFS y SMART, respectivamente. Los enlaces están ordenados por utilización y el tráfico es de 0.03 flits/ns/conmutador. En este punto de tráfico, el algoritmo UD está entrando en saturación.



(a) (b) (c)

Figura 71: Algoritmos de encaminamiento originales. Utilización de los enlaces. (a) UD, (b) DFS y (c) SMART. Tráfico de 0.03 flits/ns/conmutador. Red de 32 conmutadores. Tamaño de los mensajes de 512 bytes. Distribución uniforme de destinos.

Podemos observar como con el algoritmo UD (figura 71.a), la mitad de los enlaces están muy poco utilizados (51% de los enlaces con una utilización menor del 10%) mientras que unos pocos enlaces están altamente utilizados (solamente un 11% de los enlaces con una utilización mayor del 30%). De estos pocos enlaces, hay algunos de ellos que están sobreutilizados (3 enlaces con una utilización mayor del 50%). Claramente, el tráfico está desequilibrado entre los enlaces de la red. Por otra parte, el algoritmo DFS (figura 71.b) suaviza este desequilibrado de UD teniendo solamente un 41% de los enlaces con una utilización baja (utilización menor del 10%) y el 3% de los enlaces altamente utilizados (más del 30%). Podemos observar que ningún enlace tiene una utilización superior al 35%.

Sin embargo, el mejor equilibrado lo obtiene SMART (figura 71.c). Podemos observar como, para el mismo punto de tráfico, todos los enlaces tienen una utilización similar (la utilización varía entre un 7.09% y un 18.09%). Incluso, el 72% de los enlaces tienen una utilización entre el 14% y el 18%. Al equilibrar mejor el tráfico, la red puede aceptar más tráfico por unidad de tiempo, por lo que se alcanza una mayor productividad.

El equilibrado del tráfico viene influenciado por las restricciones de encaminamiento. Estas restricciones fuerzan a que el tráfico siga unas rutas predeterminadas. De hecho, los tres enlaces que están más utilizados en UD se corresponden con los enlaces conectados al conmutador raíz, zona por la que fluye la mayoría de las rutas. Como vemos, DFS obtiene un mejor equilibrado ya que disminuye el número de restricciones de encaminamiento. Por su parte, el algoritmo SMART tiene como principal objetivo el equilibrado de tráfico, aunque sacrificando el tiempo de cálculo de rutas.

La tabla 4 muestra los factores de incremento mínimos, máximos y medios de productividad cuando comparamos los algoritmos UD, DFS y SMART para 10 topologías aleatorias de cada tamaño de red. Podemos observar como, el algoritmo de encaminamiento SMART siempre incrementa la productividad con respecto a UD y DFS en todas las redes. Por ejemplo, para redes con 32 conmutadores, como media, el algoritmo de encaminamiento SMART mejora UD y DFS en factores de 1.79 y 1.30, respectivamente. También observamos como al aumentar el tamaño de la red, el algoritmo de encaminamiento DFS también incrementa las diferencias de productividad sobre UD. Para redes grandes de 64 conmutadores DFS mejora la productividad de UD en un factor de 1.58, como media.

	SMART vs UD			SMART vs DFS			DFS vs UD		
Conm.	Mín	Máx	Media	Mín	Máx	Media	Mín	Máx	Media
8	1.22	1.52	1.35	1.10	1.50	1.29	0.91	1.38	1.05
16	1.26	1.80	1.49	1.05	1.35	1.19	1.00	1.48	1.26
32	1.42	2.07	1.79	1.05	1.43	1.30	1.08	1.86	1.39
64	N/D	N/D	N/D	N/D	N/D	N/D	1.29	1.96	1.58

Tabla 4: Algoritmos de encaminamiento originales. Factores de incremento de productividad entre UD, DFS y SMART. Distribución uniforme de destinos. Tamaño de mensajes de 512 bytes.

Sin embargo, también debemos analizar el comportamiento de los algoritmos de

encaminamiento con diferentes patrones de tráfico. Para ello, la tabla 5 muestra los factores de incremento de productividad para diferentes patrones de tráfico y diferentes tamaños de red. Como podemos observar, para todos los patrones de tráfico analizados, el algoritmo SMART mejora en términos medios la productividad de UD y DFS. Por ejemplo, para el tráfico combinado en redes de 32 conmutadores, el algoritmo SMART mejora, en términos medios, la productividad de UD y DFS en factores de 1.50 y 1.28, respectivamente. Por otra parte, el algoritmo DFS siempre mejora la productividad de UD. En particular, con tráfico combinado y en redes de 64 conmutadores, DFS mejora la productividad de UD en un factor de 1.35. Por lo tanto, vemos que el comportamiento de los algoritmos de encaminamiento tradicionales es similar bajo diferentes patrones de tráfico y diferentes topologías irregulares de red.

		SMART vs UD			SMART vs DFS			DFS vs UD		
Distribución	Con	Mín	Máx	Media	Mín	Máx	Media	Mín	Máx	Media
Hot-spot	16	0.98	1.19	1.10	0.86	1.18	1.07	0.87	1.17	1.03
Hot-spot	32	1.00	1.40	1.21	0.98	1.17	1.02	0.86	1.42	1.15
Hot-spot	64	N/D	N/D	N/D	N/D	N/D	N/D	1.09	1.70	1.34
Bit-reversal	16	1.02	2.21	1.46	0.87	1.57	1.18	0.97	1.99	1.26
Bit-reversal	32	1.12	2.01	1.69	1.12	1.61	1.30	1.00	1.66	1.39
Bit-reversal	64	N/D	N/D	N/D	N/D	N/D	N/D	1.20	1.75	1.55
Local	16	1.00	1.54	1.27	1.01	1.51	1.21	0.82	1.36	1.06
Local	32	1.17	1.41	1.34	1.12	1.43	1.24	0.93	1.24	1.09
Local	64	N/D	N/D	N/D	N/D	N/D	N/D	0.93	1.04	1.00
Combinada	16	1.09	1.75	1.34	1.08	1.30	1.19	0.91	1.37	1.13
Combinada	32	1.21	1.73	1.50	1.18	1.37	1.28	0.90	1.29	1.18
Combinada	64	N/D	N/D	N/D	N/D	N/D	N/D	1.06	1.57	1.35

Tabla 5: Algoritmos de encaminamiento originales. Factores de incremento de productividad entre UD, DFS y SMART para diferentes patrones de tráfico. Tamaño de mensajes de 512 bytes.

De la evaluación de los algoritmos tradicionales podemos concluir que el equilibrado de tráfico juega un papel fundamental. El algoritmo SMART obtiene siempre mayor productividad en todas las topologías y bajo todos los patrones de tráfico. Por otra parte, UD y DFS obtienen unos valores de productividad menores, debido principalmente al desequilibrado originado por las propias restricciones de encaminamiento. Comparando UD y DFS, DFS siempre obtiene mayores productividades ya que introduce menos restricciones.

5.2.2 ITBs aplicados a *up*/down** y *DFS*

Una vez analizado el comportamiento de los algoritmos de encaminamiento tradicionales, vamos a aplicar el mecanismo ITB. Para ello, en esta sección evaluamos el mecanismo ITB cuando es aplicado a los algoritmos *up*/down** y *DFS*. Posteriormente, en la siguiente sección, evaluamos el mecanismo ITB cuando es aplicado sobre el algoritmo de encaminamiento *smart-routing*.

Para aplicar el mecanismo ITB sobre *up*/down** vamos a utilizar las dos aproximaciones descritas en la sección 3.2.1. En una primera aproximación, el mecanismo es aplicado solamente para asegurar el encaminamiento de ruta mínima libre de bloqueo. El algoritmo se muestra en la figura 46. El algoritmo resultante lo denominamos UD_MITB (*Up*/Down* with Minimum ITBs*). En la segunda aproximación vamos a utilizar más ITBs de los necesarios que aseguran la condición de rutas mínimas libres de bloqueo. El algoritmo se muestra en la figura 47. Al algoritmo resultante lo denominamos UD_ITB (*Up*/Down* with ITBs*).

En lo que respecta al algoritmo *DFS*, vamos a utilizar sólo la segunda aproximación, esto es, vamos a seleccionar rutas mínimas de forma aleatoria utilizando en los casos necesarios ITBs de forma que se cumplan las reglas de encaminamiento del algoritmo *DFS*. Este algoritmo lo denominamos DFS_ITB (*DFS with ITBs*).

En la figura 72 podemos ver las prestaciones obtenidas por los nuevos algoritmos de encaminamiento que utilizan ITBs (UD_MITB, UD_ITB y DFS_ITB) junto con los algoritmos tradicionales (UD y DFS), en redes de diferentes tamaños (8, 16, 32 y 64 conmutadores). Se ha utilizado la distribución uniforme de destinos y el tamaño de los mensajes es de 512 bytes.

En la red de 8 conmutadores (figura 72.a), el mecanismo apenas incrementa las prestaciones obtenidas por los algoritmos tradicionales. Esto es debido a que en redes pequeñas los algoritmos de encaminamiento tradicionales como UD y DFS no introducen apenas restricciones, por lo que prácticamente todas las rutas son mínimas. Por lo tanto, sobre estas redes, se van a utilizar pocos ITBs por lo que el mecanismo no va a incrementar notablemente las prestaciones. Hay que hacer constar, sin embargo, que el mecanismo no perjudica las prestaciones de la red.

Sin embargo, a partir de redes de 16 conmutadores (figura 72.b), el mecanismo de buffers en tránsito siempre mejora la productividad de los algoritmos de encaminamiento tradicionales. Incluso, al aumentar el tamaño de la red a 32 y 64 conmutadores (figuras 72.c y 72.d), el mecanismo incrementa aún más la productividad de

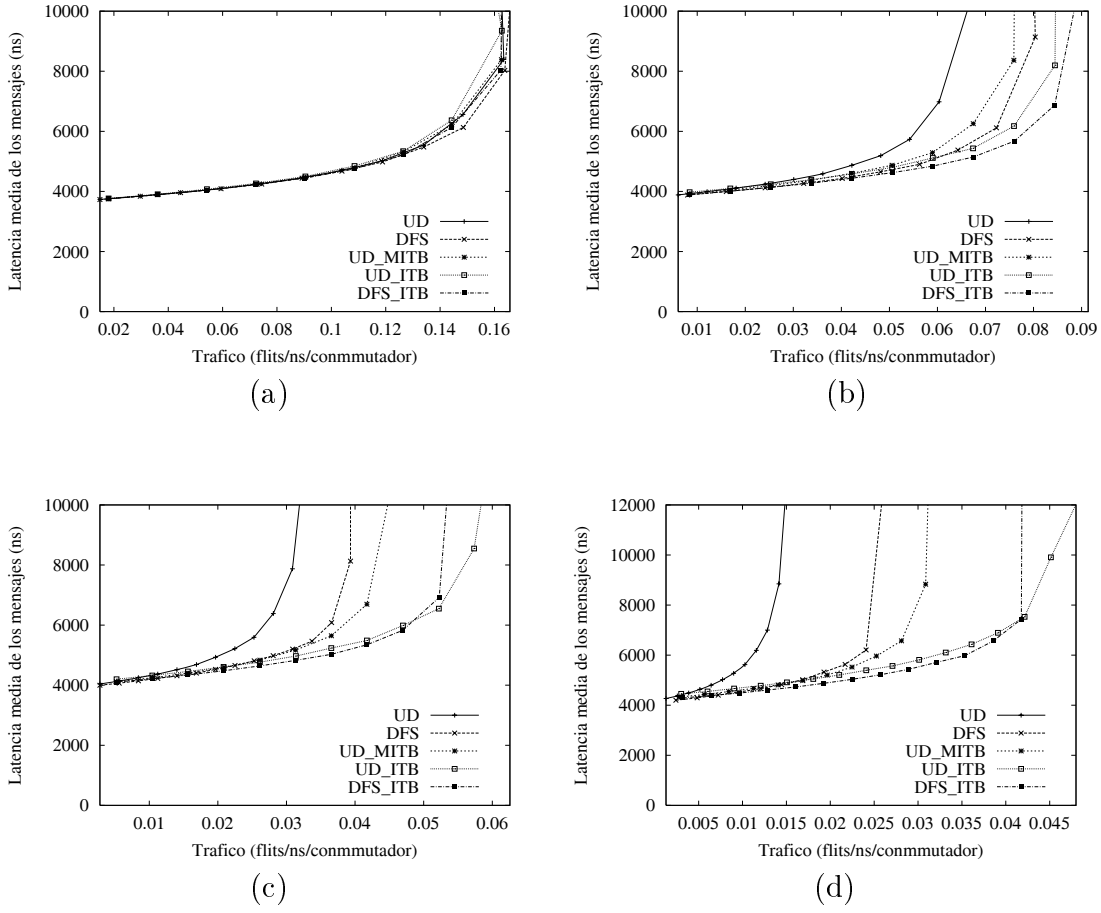


Figura 72: ITBs sobre UD y DFS. Latencia media vs tráfico. Red de (a) 8, (b) 16, (c) 32 y (d) 64 conmutadores. Distribución uniforme de destinos. Tamaño de mensajes de 512 bytes.

los algoritmos tradicionales. Por lo tanto, los beneficios obtenidos al insertar ITBs aumentan conforme aumenta el tamaño de la red. Respecto al algoritmo UD, podemos observar como el algoritmo UD_MITB incrementa la productividad en factores de 1.26, 1.49 y 2.40 en la red de 16, 32 y 64 conmutadores, respectivamente. También podemos observar como, al utilizarse más ITBS, el algoritmo UD_ITB mejora aún más la productividad de UD. En concreto, UD_ITB mejora la productividad de UD en factores de 1.40, 2.04 y 3.52 para las redes de 16, 32 y 64 conmutadores, respectivamente. Respecto a DFS, DFS_ITB también incrementa su productividad en factores de 1.05, 1.33 y 1.58, respectivamente.

Cuando comparamos entre sí los algoritmos de encaminamiento que utilizan más ITBs de los necesarios para asegurar rutas mínimas (UD_ITB y DFS_ITB), podemos observar que obtienen prácticamente la misma productividad. Ambos algoritmos de encaminamiento utilizan las mismas rutas mínimas (seleccionadas aleatoriamente). La única diferencia entre los dos algoritmos radica en dónde se utilizan los ITBs y en cuántos se necesitan. También se observa como el algoritmo DFS_ITB obtiene una latencia ligeramente menor que UD_ITB. Esto es debido a que el algoritmo de encaminamiento DFS es menos restrictivo que UD y, por consiguiente, DFS_ITB necesita menos ITBs. El algoritmo DFS_ITB utiliza como media 0.3 ITBs por mensaje para la red de 64 conmutadores, mientras que el algoritmo UD_ITB utiliza como media 0.55 ITBs por mensaje.

Una vez analizados los incrementos en productividad que produce la aplicación del mecanismo sobre los algoritmos de encaminamiento UD y DFS, vamos a ver los motivos de este incremento. Para ello, vamos a evaluar los beneficios que aporta el uso del mecanismo. Estos son: rutas mínimas y un mayor equilibrado del tráfico[1].

Los algoritmos de encaminamiento UD y DFS se calculan a partir de árboles. Uno de los principales inconvenientes de esta metodología es que al aumentar el tamaño de la red, aparecen más restricciones de encaminamiento por lo que se obtiene un menor porcentaje de rutas mínimas. Para la red de 8 conmutadores, la práctica totalidad de las rutas calculadas por UD son rutas mínimas. No obstante, para la red de 16 conmutadores, el 89% de las rutas calculadas por UD son mínimas. Ya para redes más grandes de 32 y 64 conmutadores, el porcentaje de rutas mínimas decrece ostensiblemente a 71% y 61%, respectivamente. Para el algoritmo de encaminamiento DFS ocurre algo similar. El tanto por cien de rutas mínimas calculadas por DFS es 99%, 94%, 81% y 70% para redes de 8, 16, 32 y 64 conmutadores, respectivamente. Por otra parte, cuando utilizamos ITBs, todas las rutas calculadas por los algoritmos de encaminamiento (UD_MITB, UD_ITB y DFS_ITB) son mínimas. Por lo tanto, al aumentar el tamaño de la red, el mecanismo ITB obtendrá mayores beneficios ya que facilitará el uso de más rutas mínimas (con respecto a los algoritmos originales) por lo que los mensajes utilizarán, por término medio, menos recursos de la red.

[1] Obviamente el mecanismo permite también una reducción en la contención de red. Sin embargo, este efecto lo estudiaremos al aplicar ITBs sobre *smart-routing* y en el nuevo algoritmo de encaminamiento.

Por otra parte, tal y como hemos visto en la sección anterior, el principal inconveniente de los algoritmos de encaminamiento basados en árbol es el desequilibrado del tráfico. Al aumentar el tamaño de la red, estos algoritmos de encaminamiento tienden a sobreutilizar los enlaces cercanos al conmutador raíz, llegándose rápidamente a un desequilibrado del tráfico. Al utilizarse ITBs, se permite utilizar rutas alternativas prohibidas por el algoritmo original. Debido a la utilización de estas rutas alternativas, el tráfico no es forzado a pasar a través de la zona cercana al conmutador raíz, obteniéndose un mejor equilibrado del tráfico y por lo tanto unas mejores prestaciones de la red. En las figuras 73.a, 73.b y 73.c podemos ver la utilización de los enlaces entre conmutadores al utilizar los algoritmos UD_MITB, UD_ITB y DFS_ITB, respectivamente, para la red de 32 conmutadores. El tráfico de red es 0.03 flits/ns/conmutador (igual que en la figura 71), punto en el cual el algoritmo UD está entrando en zona de saturación.

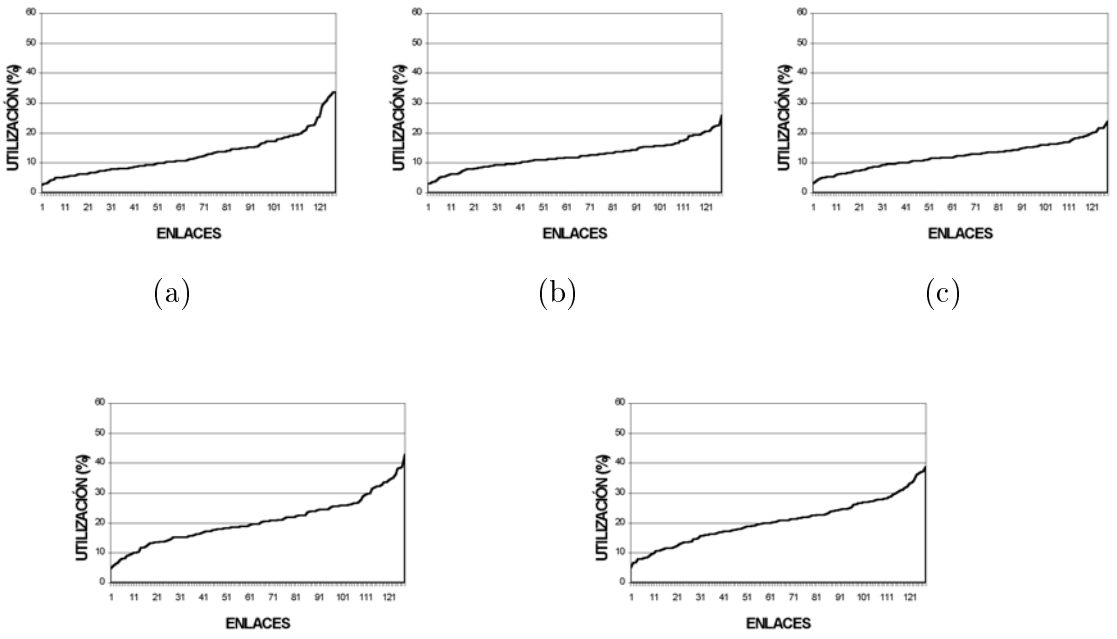


Figura 73: ITBs sobre UD y DFS. Utilización de los enlaces. (a) UD_MITB, (b, d) UD_ITB y (c, e) DFS_ITB. Tráfico es (a, b, c) 0.03 flits/ns/conmutador y (d, e) 0.052 flits/ns/conmutador. Red de 32 conmutadores. Tamaño de mensajes de 512 bytes. Distribución uniforme de destinos.

Podemos observar como, el algoritmo UD_MITB (figura 73.a) obtiene un mejor equilibrado del tráfico que el algoritmo UD (figura 71.a). Solamente el 42% de los enlaces tienen ahora una utilización menor del 10% y solamente el 5% de los enlaces es utilizado más del 30% del tiempo. Sin embargo, el algoritmo UD_MITB solamente utiliza los ITBs necesarios para asegurar encaminamiento mínimo. Debido a esto, un elevado porcentaje de rutas aún son forzadas a pasar a través de la zona cercana al conmutador raíz, hecho que apreciamos en el desequilibrado de tráfico que aún presenta UD_MITB.

Por otro lado, el algoritmo de encaminamiento UD_ITB (figura 73.b) obtiene un mejor equilibrado del tráfico ya que utiliza más ITBs y por lo tanto tiene más rutas que evitan las cercanías del nodo raíz. El algoritmo UD_ITB tiene todos los enlaces con una utilización inferior al 26% del tiempo y solamente el 31% de los enlaces se utilizan menos de un 10% del tiempo. De la misma forma, el algoritmo DFS_ITB (figura 73.c) obtiene prácticamente el mismo equilibrado que el algoritmo de encaminamiento UD_ITB (las rutas son las mismas, lo que cambia es la ubicación y el número de ITBs).

El equilibrado obtenido por UD_ITB y DFS_ITB se mantiene para cualquier punto de tráfico. En la figura 73.d y 73.e podemos observar el equilibrado obtenido en los puntos de saturación de UD_ITB y DFS_ITB (0.052 flits/ns/conmutador). Como se puede apreciar, el equilibrado del tráfico se mantiene en estos puntos de elevado tráfico. Por lo tanto, al utilizar ITBs, la red acepta un mayor tráfico ya que se obtiene un mejor equilibrado. Este hecho permite que se obtenga una mayor productividad.

Vamos a generalizar los resultados obtenidos para más topologías de red. Para ello, la tabla 6 muestra los incrementos en productividad mínimos, máximos y medios obtenidos para 10 topologías aleatorias de cada tamaño de red evaluado.

Como se aprecia en la tabla, los resultados medios son similares a los obtenidos en las topologías anteriores. UD_MITB obtiene unos incrementos en productividad medios de 1.27, 1.60 y 2.13 en redes de 16, 32 y 64 conmutadores, respectivamente. Al utilizar más ITBs, obtenemos mejores resultados (en términos de productividad). El algoritmo de encaminamiento UD_ITB mejora la productividad de UD en términos medios en un factor de 1.42, 2.08 y 2.94 en redes de 16, 32 y 64 conmutadores, respectivamente. En una red particular de 64 conmutadores, la productividad alcanzada por UD es más que triplicada al utilizar UD_ITB (factor de incremento de

	UD_MITB vs UD			UD_ITB vs UD			DFS_ITB vs DFS		
Conm.	Mín	Máx	Media	Mín	Máx	Media	Mín	Máx	Media
8	0.94	1.10	1.01	0.94	1.22	1.02	0.97	1.08	1.03
16	1.05	1.40	1.27	1.27	1.56	1.42	0.93	1.28	1.13
32	1.24	1.86	1.60	1.71	2.47	2.08	1.32	1.62	1.47
64	1.71	2.41	2.13	2.34	3.52	2.94	1.33	1.92	1.63

Tabla 6: ITBs sobre UD y DFS. Factores de incremento de productividad al utilizar ITBs sobre UD y DFS. Tamaño de mensajes de 512 bytes. Distribución uniforme de destinos.

productividad de 3.52). Por otra parte, la productividad de DFS es también incrementada por DFS_ITB en factores de 1.13, 1.47 y 1.63. Por lo tanto, los beneficios en términos de productividad aportados por el mecanismo ITB a UD y DFS son significativos e independientes de la topología empleada.

No obstante, el estudio anterior ha sido realizado utilizando una distribución uniforme de destinos. Por lo tanto, ahora evaluamos el mecanismo para otros patrones de tráfico. Para ello, la tabla 7 muestra los factores de incremento de productividad del mecanismo para las distribuciones *hot-spot*, *bit-reversal*, local y combinada.

		UD_MITB vs UD			UD_ITB vs UD			DFS_ITB vs DFS		
Distribución	Con	Mín	Máx	Media	Mín	Máx	Media	Mín	Máx	Media
Hot-spot	16	0.99	1.17	1.04	0.99	1.21	1.10	1.00	1.17	1.05
Hot-spot	32	1.00	1.40	1.18	1.00	1.39	1.18	0.98	1.17	1.03
Hot-spot	64	1.60	2.08	1.71	1.66	2.57	2.03	1.21	1.49	1.35
Bit-reversal	16	0.94	1.44	1.16	0.87	1.81	1.17	0.79	1.27	1.03
Bit-reversal	32	1.12	2.00	1.59	1.56	2.57	1.87	1.20	1.99	1.51
Bit-reversal	64	1.74	2.99	2.05	2.21	3.50	2.76	1.46	2.20	1.78
Local	16	0.97	1.26	1.08	1.02	1.56	1.24	1.00	1.30	1.17
Local	32	1.00	1.40	1.16	1.12	1.60	1.44	1.15	1.45	1.29
Local	64	1.00	1.20	1.07	1.40	1.57	1.49	1.13	1.33	1.24
Combinada	16	1.00	1.45	1.15	1.00	1.56	1.26	0.98	1.28	1.14
Combinada	32	1.12	1.57	1.31	1.31	1.86	1.65	1.20	1.50	1.35
Combinada	64	1.48	2.00	1.74	1.82	2.65	2.31	1.43	1.80	1.56

Tabla 7: ITBs sobre UD y DFS. Factores de incremento de productividad al utilizar ITBs sobre UD y DFS para diferentes patrones de tráfico. Tamaño de mensajes de 512 bytes.

Podemos observar en la tabla como el mecanismo de buffers en tránsito siempre

incrementa, en términos medios, la productividad de los algoritmos de encaminamiento UD y DFS. Dependiendo del patrón de tráfico utilizado, estos incrementos son más o menos significativos. Por ejemplo, en el patrón de tráfico *hot-spot*, UD_ITB incrementa la productividad de UD, como media, en un factor de 2.03 en redes de 64 conmutadores. Sin embargo, en una distribución *bit-reversal*, el incremento es mayor, incrementándose la productividad de UD en un factor de 2.76. Los incrementos menores se obtienen para la distribución local, en la que UD_ITB incrementa la productividad de UD en un factor de 1.49. Este menor incremento es debido a que los algoritmos tradicionales como UD y DFS obtienen más rutas mínimas para los destinos más cercanos. Por eso, el número de ITBs utilizados en la distribución local es mucho menor que el utilizado en las otras distribuciones por lo que el incremento en productividad en la distribución local es menor.

Por último, cuando se utiliza el patrón de tráfico combinado, UD_ITB mejora la productividad de UD en factores de 1.26, 1.65 y 2.31 para redes de 16, 32 y 64 conmutadores, respectivamente. También podemos observar que los resultados para UD_MITB y DFS_ITB son los esperados, obteniéndose incrementos en productividad menores pero significativos.

Una vez evaluados los beneficios del mecanismo (rutas mínimas y equilibrado), nos centramos en analizar los inconvenientes que puede presentar el mecanismo. Por lo tanto, evaluamos el posible incremento en latencia media de los mensajes que supone el utilizar el mecanismo. Hay que hacer constar que el peor caso de incremento en latencia será en condiciones de muy poco tráfico, ya que en condiciones de tráfico medio o alto, los algoritmos tradicionales se saturan mientras que los algoritmos que utilizan ITBs no. Por lo tanto, la evaluación del incremento en latencia la realizamos siempre en condiciones de muy poco tráfico. Adicionalmente, se evalúan diferentes tamaños de mensajes ya que la sobrecarga relativa aumentará conforme los mensajes sean más pequeños. Para ello, la tabla 8 muestra el incremento en latencia al utilizar ITBs (UD_MITB, UD_ITB y DFS_ITB) con respecto a los algoritmos tradicionales (UD y DFS) en condiciones de muy poco tráfico, para tamaños de mensajes de 32 y 512 bytes, utilizando 10 topologías generadas aleatoriamente para cada tamaño de red. Adicionalmente, también se muestra el incremento en latencia ante tráfico bimodal.

Para mensajes de 512 bytes observamos como el mecanismo incrementa ligeramente la latencia media de los mensajes. No obstante, este incremento no supera nunca el 5%. Por otra parte, el incremento en latencia sí que es apreciable para

		UD_MITB vs UD			UD_ITB vs UD			DFS_ITB vs DFS		
Tam.msjs	Conm.	Mín	Máx	Med	Mín	Máx	Med	Mín	Máx	Med
32	8	0.07	0.19	0.10	0.07	8.35	2.85	0.04	0.21	0.09
32	16	1.55	4.39	2.62	7.71	17.46	10.84	1.96	6.11	4.00
32	32	4.18	7.13	5.39	15.29	19.13	16.78	6.50	7.76	7.12
32	64	6.02	7.73	6.70	16.72	21.63	19.90	6.85	9.83	8.63
512	8	0.56	0.79	0.67	0.63	2.34	1.18	0.57	0.74	0.63
512	16	0.90	1.41	1.11	2.06	3.85	2.61	0.40	1.02	0.71
512	32	1.25	1.88	1.53	3.23	4.10	3.70	1.93	2.64	2.43
512	64	1.26	1.84	1.52	3.80	4.72	4.39	1.76	2.19	2.02
bimodal	8	0.79	1.06	0.90	1.34	4.10	1.93	0.94	1.32	1.14
bimodal	16	-1.58	0.64	-0.62	1.60	5.75	3.48	-2.75	-0.69	-1.25
bimodal	32	0.76	2.06	1.34	6.74	8.18	7.37	2.83	3.48	3.22
bimodal	64	2.64	4.53	3.29	9.43	12.17	10.66	4.04	6.17	5.01

Tabla 8: ITBs sobre UD y DFS. Porcentajes de incremento en latencia en condiciones de poco tráfico al utilizar ITBs sobre UD y DFS. Distribución uniforme de destinos.

mensajes cortos de 32 bytes. El incremento en latencia media varía, para UD_ITB, desde un 7.71% hasta un 21.63% en función del tamaño de la red. Esto es debido a que en redes pequeñas se introducen pocos ITBs ya que existen pocos ciclos. Por lo tanto, la sobrecarga relativa debido a la utilización de ITBs es tanto mayor cuanto menor sea el tamaño de los mensajes y mayor sea el tamaño de red.

Efectivamente, el incremento de latencia depende del número de ITBs utilizados. Esto lo podemos observar en la tabla 8 donde el incremento en latencia media es menor cuando se utiliza el mecanismo sobre DFS (DFS_ITB) que sobre UD (UD_ITB). Como máximo, el incremento en latencia del mecanismo sobre DFS es del 9.83%. También, el incremento en latencia es menor cuando se utiliza el mínimo número de ITBs sobre UD (UD_MITB), resultando en este caso un incremento máximo en latencia del 7.73%.

Por lo tanto, el incremento en latencia es mayor en mensajes cortos, en redes grandes y en algoritmos que utilizan más ITBs. No obstante, como podemos apreciar, para un tráfico bimodal con una tasa del 30% de mensajes largos, los incrementos en latencia del mecanismo se reducen considerablemente, no superando el 12.17% como máximo.

Podemos concluir que, al utilizar buffers en tránsito sobre los algoritmos de encaminamiento UD y DFS, la productividad de la red se incrementa notablemente. Al aumentar el tamaño de la red, se obtienen mayores incrementos en productividad.

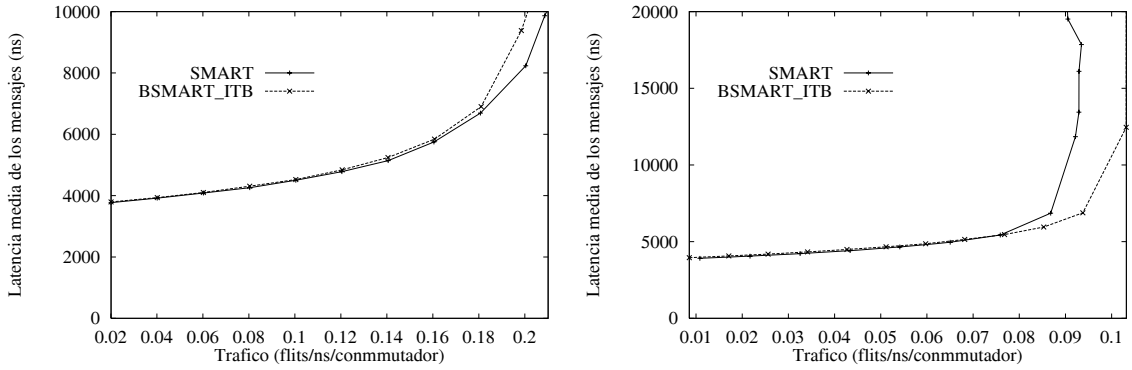
El mecanismo ITB evita la congestión en la zona cercana al nodo raíz, a la vez que facilita rutas mínimas libres de bloqueo y equilibra mejor el tráfico. Por otra parte, la latencia media de los mensajes se ve ligeramente incrementada. Este incremento es apreciable solamente en condiciones de poca carga y para mensajes cortos.

5.2.3 ITBs aplicados a *smart-routing*

En la sección anterior hemos visto como el mecanismo de buffers en tránsito incrementa notablemente las prestaciones de los algoritmos *up*/down** y *DFS*. El principal motivo para este incremento hemos visto que está en el mejor equilibrado que se obtiene. El desequilibrado de *up*/down** y *DFS* viene motivado por el uso de árboles para el cálculo de rutas. No obstante, el algoritmo de encaminamiento *smart-routing* no se basa en árboles. Es más, su principal objetivo es equilibrar el tráfico de la red, sacrificando por un lado el uso de rutas mínimas y por otro el tiempo de cálculo. De hecho, ya hemos visto el buen equilibrado de tráfico que obtiene (figura 71.c). Por lo tanto, las mejoras que el mecanismo ITB ha obtenido sobre *up*/down** y *DFS* no parece que se vayan a producir en la misma medida sobre *smart-routing*.

En esta sección evaluamos el mecanismo ITB cuando es aplicado sobre el algoritmo de encaminamiento *smart-routing*. Para ello, vamos a aplicar el mecanismo según se ha descrito en la sección 3.2.2. Esto es, las rutas se calculan de igual forma que las calcula el algoritmo *smart-routing* pero sin tener en cuenta la condición de que el conjunto final de rutas sea libre de bloqueo. Posteriormente, aplicamos un algoritmo (figura 49) que busca ciclos en el GDC resultante, rompiéndolos con ITBs. El algoritmo resultante lo vamos a denominar BSMART_ITB (*Balanced as SMART with ITBs*).

La figura 74 muestra los resultados de los algoritmos SMART y BSMART_ITB en redes de 8, 16 y 32 conmutadores para una distribución uniforme de destinos y mensajes de 512 bytes. Para redes de 64 conmutadores, las rutas del algoritmo *smart-routing* no se pudieron calcular debido a su elevado tiempo de cálculo. Podemos observar como, también para el algoritmo *smart-routing*, el mecanismo de buffers en tránsito incrementa la productividad de la red aunque de una forma más moderada. En concreto, el algoritmo BSMART_ITB incrementa la productividad de SMART en un factor de 1.12 y 1.33 en la red de 16 y 32 conmutadores, respectivamente.



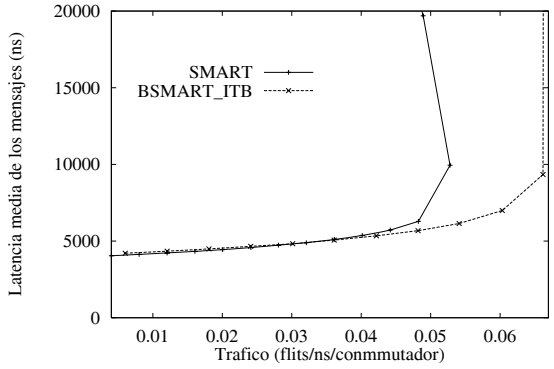
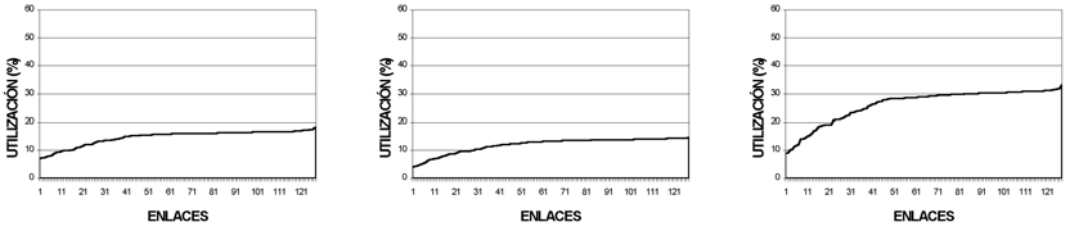


Figura 74: ITBs sobre SMART. Latencia media vs. tráfico. Red de (a) 8, (b) 16 y (c) 32 conmutadores. Tamaño de mensajes de 512 bytes. Distribución uniforme de destinos.

Hemos visto anteriormente que el tráfico está muy equilibrado entre todos los enlaces cuando se utiliza el algoritmo de encaminamiento SMART. La figura 75 muestra la utilización de los enlaces entre conmutadores cuando se utiliza SMART y BSMART_ITB en la red de 32 conmutadores para distintas tasas de tráfico por la red.

Como podemos observar, el equilibrado de tráfico es muy similar en ambos casos (figuras 75.a y 75.b). Esto es debido a que los dos algoritmos de encaminamiento utilizan un algoritmo de equilibrado de tráfico parecido (cálculo y selección de rutas). La única diferencia entre ambos es que el algoritmo BSMART_ITB es menos



(a) (b) (c)

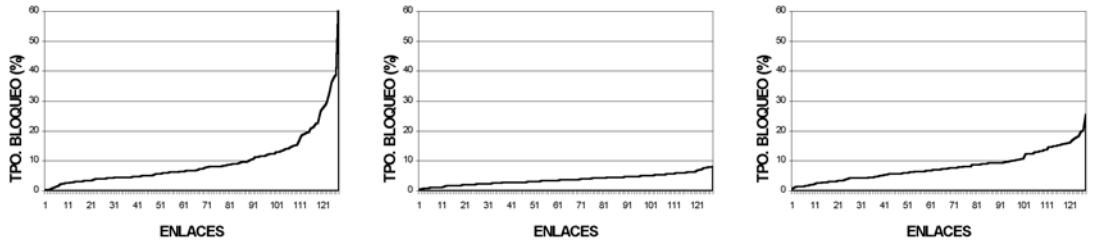
Figura 75: ITBs sobre SMART. Utilización de los enlaces. (a) SMART y (b,c) BSMART_ITB. Tráfico es (a, b) 0.03 y (c) 0.065 flits/ns/conmutador. Red de 32 conmutadores. Tamaño de mensajes de 512 bytes. Distribución uniforme de destinos.

restrictivo y permite la formación de ciclos en el GDC. Como podemos observar, el algoritmo BSMART_ITB mantiene un buen equilibrado en su punto de saturación (figura 75.c). En definitiva, el algoritmo de equilibrado de tráfico (cálculo y selección de rutas) del algoritmo *smart-routing* es muy eficiente en el equilibrado final, y por lo tanto, el mecanismo de buffers en tránsito no mejora las prestaciones por un mejor equilibrado.

Por lo tanto, el motivo de que BSMART_ITB obtenga una mayor productividad es debido a otra ventaja del uso del mecanismo. Por ello, debemos centrarnos en la reducción de contención de red que aporta de forma natural el mecanismo ITB. Para ello, vamos a analizar el porcentaje de tiempo de bloqueo[2] de los enlaces entre conmutadores. El porcentaje de tiempo de bloqueo es el porcentaje del tiempo que los enlaces están detenidos tras recibir un flit de control *STOP* hasta que se recibe el flit de control *GO*. Este parámetro es una medida directa de la contención de red. Las figuras 76.a y 76.b muestran el tiempo de bloqueo de los enlaces para la red de 32 conmutadores cuando se utilizan los algoritmos SMART y BSMART_ITB, respectivamente. El punto de tráfico es 0.05 flits/ns/conmutador. En este punto de tráfico, el algoritmo de encaminamiento SMART está entrando en saturación.

Como podemos observar, existe una gran diferencia en los porcentajes del tiempo de bloqueo entre los dos algoritmos de encaminamiento. El algoritmo SMART presenta unas diferencias importantes entre los enlaces. En concreto, algunos enlaces se encuentran bloqueados más del 10% del tiempo, estando algunos de ellos

[2]No confundir con el bloqueo (*deadlock*) de los mensajes en la red.



(a) (b) (c)

Figura 76: ITBs sobre SMART. Porcentaje de tiempo de bloqueo. Tráfico es (a, b) 0.05 flits/ns/conmutador y (c) 0.065 flits/ns/conmutador. (a) SMART y (b, c) BSMART_ITB. Tamaño de red de 32 conmutadores. Tamaño de mensajes de 512 bytes. Distribución uniforme de destinos.

bloqueados mas del 30% del tiempo, llegando alguno al 60%. Estos pocos enlaces que están tan bloqueados provocan que la red se sature por completo ya que este bloqueo se propaga muy rápidamente por toda la red (recuérdese que se utiliza conmutación *wormhole*). Por el contrario, al utilizar buffers en tránsito, para el mismo punto de tráfico, el tiempo de bloqueo es siempre menor del 8% para todos los enlaces. Por lo tanto, el mecanismo de buffers en tránsito mejora las prestaciones del algoritmo SMART no por un mejor equilibrado del tráfico, sino por una reducción significativa de la contención de red. La figura 76.c muestra el tiempo de bloqueo de los enlaces cuando el algoritmo BSMART_ITB está entrando en saturación (0.06 flits/ns/conmutador). Como podemos observar, el algoritmo BSMART_ITB se satura debido a la contención de red, más que al desequilibrado de tráfico.

En la tabla 9 podemos ver los factores de incremento de productividad mínimos, máximos y medios obtenidos para 10 topologías generadas aleatoriamente para cada tamaño de red evaluado. Como podemos observar, con el mecanismo de buffers en tránsito (BSMART_ITB) siempre se mejora, por término medio, la productividad de SMART.

Para otros patrones de tráfico, el algoritmo BSMART_ITB también incrementa, por término medio, la productividad de red del algoritmo SMART. La tabla 10 muestra los factores de incremento para las distribuciones de tráfico *hot-spot*, *bit-reversal*, local y combinada. Como ejemplo, cuando se utiliza el patrón de tráfico combinado, el algoritmo de encaminamiento BSMART_ITB incrementa para redes

	BSMART_ITB vs SMART		
Conm.	Mín	Máx	Media
8	0.90	1.11	1.00
16	0.98	1.12	1.06
32	1.15	1.37	1.26

Tabla 9: ITBs sobre SMART. Factores de incremento de productividad entre BSMART_ITB y SMART. Distribución uniforme de destinos. Tamaño de mensajes de 512 bytes.

con 32 conmutadores, por término medio, la productividad de SMART en un factor de 1.14.

		BSMART_ITB vs SMART		
Distribución	Conm.	Mín	Máx	Media
Hot-spot	16	0.85	1.17	0.96
Hot-spot	32	1.00	1.00	1.00
Bit-reversal	16	0.73	1.13	0.93
Bit-reversal	32	0.99	1.45	1.21
Local	16	1.00	1.17	1.10
Local	32	1.10	1.29	1.17
Combinada	16	1.00	1.17	1.06
Combinada	32	1.04	1.27	1.14

Tabla 10: ITBs sobre SMART. Factores de incremento de productividad entre BSMART_ITB y SMART en diferentes patrones de tráfico. Tamaño de mensajes de 512 bytes.

Por último, nos queda analizar el incremento en latencia del mecanismo cuando es aplicado sobre el algoritmo *smart-routing*. La tabla 11 muestra el porcentaje de incremento en latencia para una condición de poco tráfico. Para mensajes de 512 bytes, el incremento en latencia es menor del 5% para todas las topologías estudiadas. Sin embargo, para mensajes cortos de 32 bytes se observa un mayor incremento en latencia. Sin embargo, este incremento nunca es mayor del 14%. Por último, al utilizar tráfico bimodal, vemos como el incremento en latencia del mecanismo se reduce ostensiblemente, siendo siempre menor del 7% en todas las redes evaluadas.

Es interesante observar como, para redes pequeñas, el incremento en latencia es mucho menor que en redes más grandes (en cualquier tamaño de red). Esto es

debido a que en redes pequeñas se introducen pocos ITBs ya que existen pocos ciclos. También podemos observar como en determinadas redes, el uso del mecanismo decrementa incluso la latencia media de los mensajes. Esto es debido a que utilizar un buffer en tránsito en una ruta en el algoritmo BALANCED_ITB ayuda a que la ruta sea bastante más corta que la ruta equivalente en el algoritmo SMART, por lo que la latencia que nos ahorramos al proporcionar una ruta más corta es mayor que la latencia que añadimos por utilizar el mecanismo.

		BSMART_ITB vs SMART		
Tam.Msj.	Conm	Mín	Máx	Media
32	8	-1.59	4.22	1.65
32	16	0.27	10.76	8.30
32	32	9.42	13.99	11.45
512	8	-0.43	0.97	0.34
512	16	-1.21	1.42	0.84
512	32	2.92	4.06	3.38
bimodal	8	-0.77	1.44	0.37
bimodal	16	2.21	6.68	5.61
bimodal	32	4.81	6.84	5.74

Tabla 11: ITBs sobre SMART. Porcentaje de incremento de latencia en condiciones de baja carga al utilizar ITBs sobre SMART. Distribución uniforme de destinos.

Por lo tanto, y como conclusión al estudio de la aplicación del mecanismo al algoritmo de encaminamiento *smart-routing*, podemos indicar que el mecanismo de buffers en tránsito ayuda a mejorar la productividad de la red obtenida por el algoritmo *smart-routing* al disminuir la contención de red. Dicha mejora se observa para diferentes patrones de tráfico. Asimismo, al igual que ocurre con los algoritmos *up*/down** y *DFS*, el mecanismo de buffers en tránsito incrementa ligeramente la latencia de los mensajes. No obstante, este incremento en latencia es significativo sólo para mensajes cortos y en condiciones de poco tráfico.

5.3 ITBs en redes regulares

Como hemos visto, el mecanismo de buffers en tránsito incrementa las prestaciones de los algoritmos de encaminamiento tradicionales de forma significativa. Estas

conclusiones se han obtenido a partir de la evaluación del mecanismo en redes irregulares generadas aleatoriamente. En estas redes, hemos visto que el desequilibrado del tráfico y la utilización de rutas no mínimas por parte de los algoritmos *up*/down** y *DFS* son los principales motivos para que el mecanismo ITB obtenga un incremento significativo en la productividad. Asimismo, hemos visto cómo la reducción de la contención también ayuda a obtener una mayor productividad de red. Este es el caso de aplicar ITBs al algoritmo de encaminamiento *smart-routing*.

Por otro lado, algoritmos como *up*/down** y *DFS* en redes regulares van a tener un comportamiento distinto, ya que en dichas redes los algoritmos van a tener las restricciones de encaminamiento repartidas por toda la red debido a su regularidad. La regularidad en dichas redes también puede ocasionar que los algoritmos *up*/down** y *DFS* obtengan un elevado porcentaje de rutas mínimas. Por lo tanto, no sabemos si el mecanismo ITB va a incrementar significativamente las prestaciones en dichas redes regulares. Es por ello que, en esta sección, evaluamos el mecanismo sobre redes regulares. Para ello nos centraremos en el algoritmo de encaminamiento *up*/down** por lo que evaluaremos los algoritmos UD y UD_ITB[3]. Las topologías que se evalúan son el Toro 2D, el Toro 2D con canales *express* y la red CPLANT descritos en la sección 4.4.

La figura 77 muestra las prestaciones obtenidas por los algoritmos UD y UD_ITB bajo una distribución uniforme de destinos y mensajes de 512 bytes para las tres topologías.

Podemos ver como en la red Toro 2D (figura 77.a) el algoritmo UD_ITB dobla la productividad obtenida por el algoritmo UD. En particular, UD_ITB alcanza 0.029 flits/ns/conmutador, mientras que UD se satura en 0.015 flits/ns/conmutador. En esta topología, el 80% de las rutas calculadas por el algoritmo de encaminamiento UD son rutas mínimas. Por lo tanto, el mecanismo de buffers en tránsito añade 20% de rutas mínimas al algoritmo UD. Asimismo, la distancia media al destino (medido en el número de enlaces atravesados) para el algoritmo de encaminamiento UD es 4.57 mientras que con el mecanismo de buffers en tránsito es de 4.06. Sin embargo, esto no justifica completamente el incremento de productividad obtenido al utilizar buffers en tránsito.

Aunque la topología Toro 2D permite que UD obtenga un mejor equilibrado, el mecanismo de buffers en tránsito va a equilibrar el tráfico aún más. Esto es debido a

[3]El algoritmo de encaminamiento *smart-routing* no se evalúa debido a que el número de conmutadores utilizados provoca unos tiempos de cálculo de rutas inalcanzables.

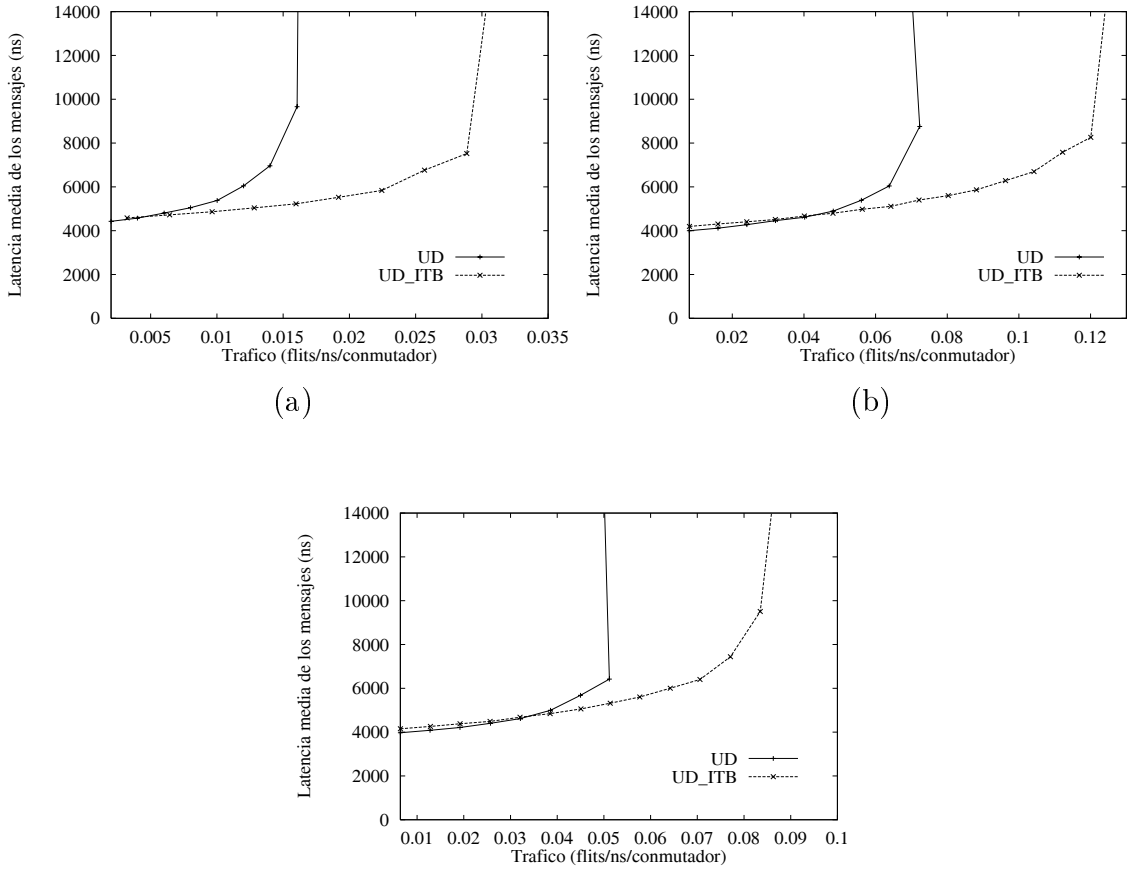


Figura 77: ITBs en redes regulares. Latencia media vs tráfico. (a) Toro 2D, (b) Toro 2D con canales *express* y (c) CPLANT. Distribución uniforme de destinos. Tamaño de mensajes de 512 bytes.

que UD fuerza mucho tráfico a cruzar el conmutador raíz. Sin embargo, el mecanismo de buffers en tránsito ayuda a distribuir mejor el tráfico de la red al permitir el uso de rutas alternativas a las rutas que con UD pasan por el conmutador raíz (rutas antes prohibidas por el algoritmo de encaminamiento UD). La figura 78 muestra la utilización de los enlaces entre conmutadores para UD y UD_ITB cuando el tráfico es 0.015 flits/ns/conmutador (UD alcanza su punto de saturación). Cuando se utiliza el algoritmo de encaminamiento UD (figura 78.a), los enlaces cerca del nodo raíz (arriba a la izquierda) están congestionados (la utilización en estos enlaces alcanza el 50%), mientras que la utilización del resto de los enlaces en la red es

baja (64% de los enlaces tienen una utilización menor del 10%). Por otra parte, el algoritmo de encaminamiento UD_ITB (figura 78.b) equilibra el tráfico entre todos los enlaces de la red. La utilización de todos los enlaces es menor del 12%. Para un mayor tráfico, cuando UD_ITB está alcanzando su punto de saturación, el algoritmo UD_ITB aún distribuye el tráfico uniformemente entre todos los enlaces. La figura 79 muestra la utilización de los enlaces entre conmutadores para el algoritmo UD_ITB en ratios de inyección de 0.03 flits/ns/conmutador. Como vemos, el tráfico está muy equilibrado entre todos los enlaces (la utilización de los enlaces varía desde un 14% a un 29%). Por lo tanto, cuando se utilizan buffers en tránsito, siempre se utilizan rutas mínimas y, más importante, el tráfico es equilibrado entre todos los enlaces de la red, doblándose la productividad de la red obtenida por UD.

Si nos fijamos en la figura 79 se puede observar que la red se satura con una utilización de los enlaces baja. El elevado tiempo de encaminamiento (150 ns) y la reducida capacidad de los *slack buffers* (80 bytes) conllevan a un bloqueo de mensajes en cascada. Así, en el punto de saturación del algoritmo UD_ITB (no mostrado gráficamente), el 20% de los enlaces están bloqueados más del 10% del tiempo total debido al mecanismo del control de flujo, alcanzándose un 30% del tiempo total en algunos enlaces determinados.

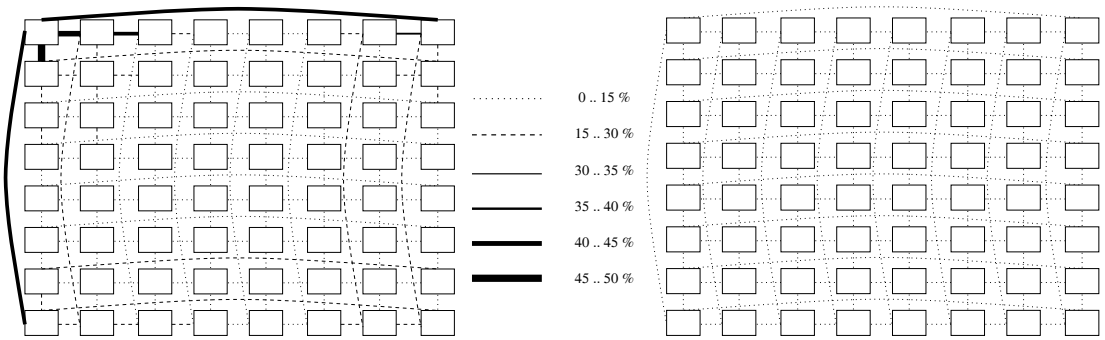


Figura 78: ITBs en redes regulares. Utilización de los enlaces en el Toro 2D en el punto de saturación de UD (0.015 flits/ns/conmutador). (a) UD y (b) UD_ITB. Tamaño de mensajes de 512 bytes. Distribución uniforme de destinos.

Con respecto a la red Toro 2D con canales *express*, la figura 77.b muestra las prestaciones obtenidas por los algoritmos de encaminamiento en dicha red. Como

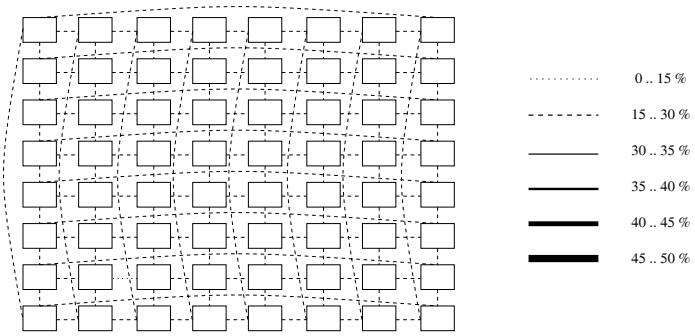


Figura 79: ITBs en redes regulares. Utilización de los enlaces en el Toro 2D con UD_ITB en su punto de saturación (0.03 flits/ns/conmutador). Tamaño de mensajes de 512 bytes. Distribución uniforme de destinos.

se aprecia, el algoritmo de encaminamiento UD_ITB prácticamente dobla la productividad alcanzada por el algoritmo de encaminamiento UD. Los beneficios de utilizar buffers en tránsito son ligeramente menores que en el Toro 2D ya que al utilizarse canales *express*, hay el doble de enlaces y hay más rutas alternativas hacia el conmutador raíz (donde el algoritmo de encaminamiento UD se satura). UD multiplica la productividad alcanzada en la red Toro 2D por 4.6, alcanzando 0.07 flits/ns/conmutador, mientras que UD_ITB multiplica la productividad de la red por 4, alcanzando 0.12 flits/ns/conmutador. El incremento en productividad (con respecto a la red Toro 2D) es debido a los canales *express* adicionales. El número de enlaces en la red se multiplica por 2, por lo tanto, más mensajes pueden estar cruzando la red al mismo tiempo. También, la distancia media a los destinos es reducida a la mitad. Así, cada paquete utiliza la mitad de recursos (*slack buffers*) y la componente de la latencia dependiente de la distancia se reduce.

El uso de canales *express* incrementa el número de rutas mínimas obtenidas por UD. En este caso, el porcentaje de rutas mínimas es 94%. Por lo tanto, de cara al incremento de productividad, proveer más rutas mínimas ya no es tan importante en esta red como lo era en el Toro 2D. Por otra parte, la distribución del tráfico juega un papel fundamental en esta red. La figura 80 muestra la utilización de los enlaces entre conmutadores para UD y UD_ITB en el punto de saturación de UD (0.066 flits/ns/conmutador). Podemos observar como, al utilizarse UD (figura 80.a), los

enlaces cerca del conmutador raíz tienen una utilización cercana al 50%, mientras que el resto de los enlaces en la red tienen una utilización baja (como en el Toro 2D). Por otra parte, al utilizar buffers en tránsito en el algoritmo UD_ITB (figura 80.b), la utilización de los enlaces es mucho más equilibrada (como en el Toro 2D). Todos los enlaces tienen una utilización inferior al 30%.

Si observamos con mayor detenimiento la figura 80.b podemos observar que hay enlaces más utilizados que otros. En concreto, los canales *express* tienen una utilización del 25%, mientras que el resto de los enlaces tienen una utilización del 10%. Los canales *express* son utilizados más frecuentemente ya que proveen rutas más cortas hacia los destinos, mientras que los otros enlaces son utilizados solamente para entregar los paquetes al conmutador final (cuando el paquete está a una distancia de un salto del destino). Por lo tanto, el tráfico no se distribuye equitativamente entre todos los enlaces debido al diferente uso de los enlaces, y por lo tanto, UD_ITB no distribuye tan uniformemente el tráfico entre todos los enlaces. Por lo tanto, el incremento en productividad al utilizar buffers en tránsito sobre *up*/down** es ligeramente menor que en el Toro 2D. Sin embargo, no hay que olvidar que el mecanismo de buffers en tránsito aún permite incrementar la productividad de UD en un factor de 1.71.

Por último, respecto a la red CPLANT (figura 77.c), UD_ITB prácticamente dobla la productividad de UD. UD se satura en torno a 0.05 flits/ns/conmutador mientras que UD_ITB se satura en 0.085 flits/ns/conmutador. La red CPLANT tiene una topología compleja formada por grupos de conmutadores. Cuando se utiliza el algoritmo de encaminamiento UD, gran cantidad del tráfico debe atravesar el conmutador raíz (que está ubicado en un determinado grupo de conmutadores), desequilibrando el tráfico entre todos los grupos. Por otra parte, al utilizarse buffers en tránsito, atravesar el conmutador raíz ya no es necesario, por lo que se equilibra mejor el tráfico entre todos los grupos de conmutadores. Con respecto a las rutas mínimas, todas las rutas obtenidas al utilizar UD en esta red son rutas mínimas, por lo que las mejoras obtenidas por el mecanismo son principalmente debidas al mejor equilibrado de tráfico obtenido.

Para el patrón de tráfico *bit-reversal* se obtienen resultados similares. La figura 81 muestra las prestaciones de los algoritmos UD y UD_ITB cuando hay en la red un patrón de tráfico *bit-reversal*. La figura muestra los resultados obtenidos en el

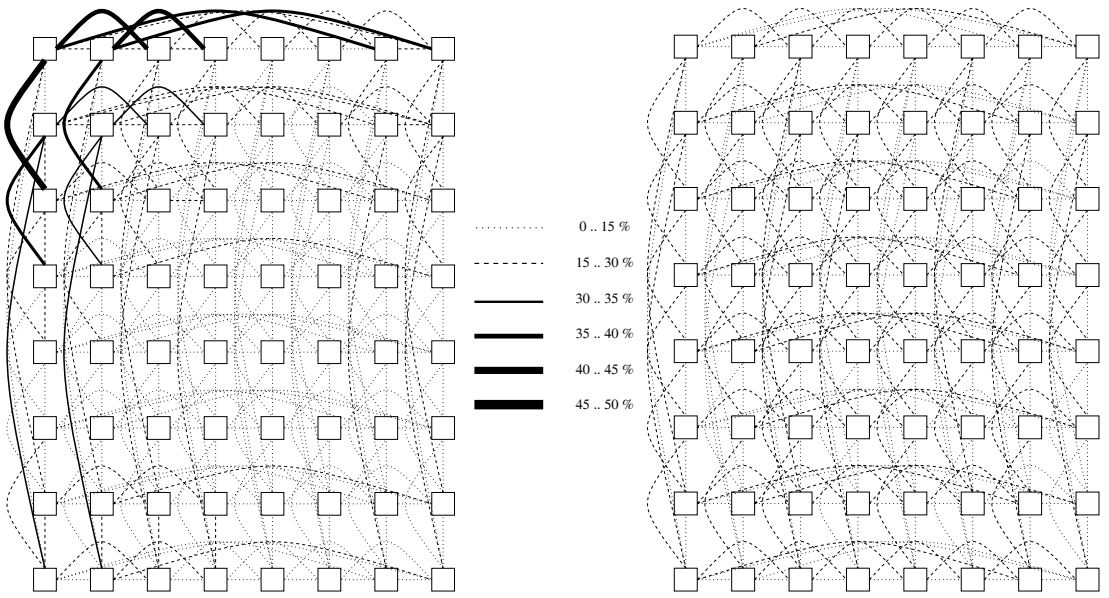


Figura 80: ITBs en redes regulares. Utilización de los enlaces en el Toro 2D con canales *express* en el punto de saturación de UD (0.066 flits/ns/conmutador). (a) UD y (b) UD_ITB. Tamaño de mensajes de 512 bytes. Distribución uniforme de destinos.

Toro 2D y Toro 2D con canales *express*, respectivamente[4].

Como vemos, en el Toro 2D (figura 81.a), la productividad de UD es prácticamente doblada al utilizar buffers en tránsito (0.017 flits/ns/conmutador para UD y 0.029 flits/ns/conmutador para UD_ITB). Por otra parte, para el Toro 2D con canales *express* (figura 81.b), los beneficios de utilizar el mecanismo de buffers en tránsito son ligeramente menores (al igual que en la distribución de destinos uniforme). UD se satura en 0.07 flits/ns/conmutador mientras que UD_ITB lo hace en 0.11 flits/ns/conmutador.

Ahora vamos a evaluar el mecanismo bajo un tráfico *hot-spot*. Para ello, evaluamos 10 ubicaciones distintas del *hot-spot* para cada topología regular. También evaluamos diferentes tasas de mensajes enviados al *hot-spot* para modelar el tráfico *hot-spot*.

[4]Para la red CPLANT no se han obtenido resultados ya que no posee un número de *hosts* múltiplo de 2.

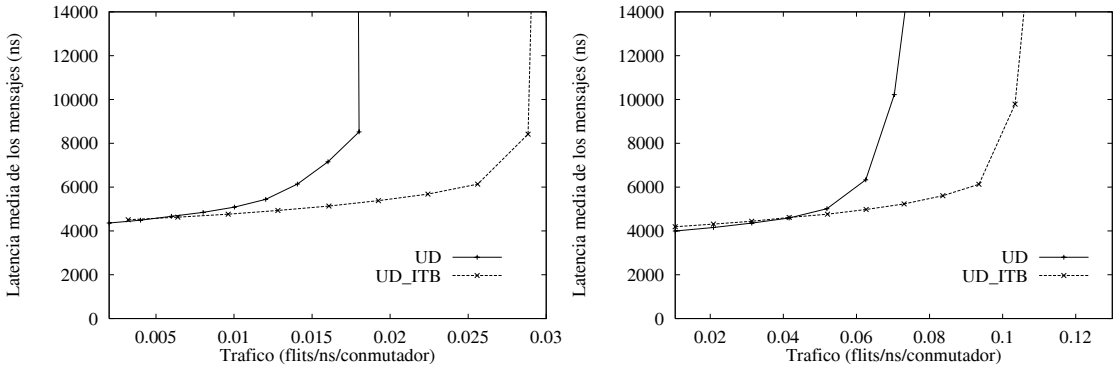


Figura 81: ITBs en redes regulares. Latencia media vs tráfico. (a) Toro 2D y (b) Toro 2D con canales *express*. Patrón de tráfico *bit-reversal*. Tamaño de mensajes de 512 bytes.

La tabla 12 muestra la productividad alcanzada por cada algoritmo de encaminamiento en el Toro 2D. En particular, se han utilizado dos cargas de *hot-spot*: 5% de tráfico *hot-spot* y 10% de tráfico *hot-spot*. Como media, UD_ITB incrementa la productividad de UD en un factor de 2.13 al utilizar el *hot-spot* del 5%. Con un *hot-spot* mayor (10%) UD_ITB obtiene un factor de incremento de 1.4.

En la tabla 12 podemos observar como el algoritmo de encaminamiento UD se ve muy poco afectado por el *hot-spot*. Para una distribución de destinos uniforme, UD alcanza la productividad de 0.015 flits/ns/conmutador. Con un *hot-spot* del 5%, la productividad de la red se decrementa solamente en un 16% (0.0125 flits/ns/conmutador como media) y para un *hot-spot* del 10%, la productividad es prácticamente la misma (como media, la productividad de la red es 0.0123 flits/ns/conmutador). Esto es debido a que el conmutador raíz se comporta como un *hot-spot* más severo y UD se satura cuando el conmutador raíz se congestiona antes de que el *hot-spot* sature la red.

La figura 82 muestra la utilización de los enlaces para UD y UD_ITB para un *hot-spot* del 10% en el punto de tráfico 0.0123 flits/ns/conmutador (cuando UD está entrando en saturación). Para el algoritmo UD (figura 82.a) los enlaces cercanos al conmutador raíz tienen una utilización mucho más elevada que los enlaces cercanos

	5%		10%	
Hot-spot	UD	UD_ITB	UD	UD_ITB
1	0.0120	0.0264	0.0120	0.0168
2	0.0144	0.0240	0.0121	0.0191
3	0.0120	0.0287	0.0145	0.0168
4	0.0120	0.0289	0.0120	0.0168
5	0.0120	0.0241	0.0120	0.0168
6	0.0120	0.0312	0.0120	0.0168
7	0.0144	0.0241	0.0120	0.0191
8	0.0120	0.0241	0.0120	0.0168
9	0.0120	0.0290	0.0120	0.0169
10	0.0120	0.0265	0.0120	0.0168
Media	0.0125	0.0267	0.0123	0.0173

Tabla 12: ITBs en redes regulares. Productividad para diferentes ubicaciones del *hot-spot* y diferentes tasas de *hot-spot*. Toro 2D. Tamaño de mensajes de 512 bytes.

al *hot-spot* (marcado como H). Por otra parte, cuando se utiliza el algoritmo de encaminamiento UD_ITB (figura 82.b), solamente los enlaces cercanos al conmutador *hot-spot* empiezan a saturarse. Por lo tanto, para el algoritmo de encaminamiento UD, el conmutador raíz se comporta como un gran *hot-spot* y satura la red, mientras que el algoritmo UD_ITB satura la red más tarde debido al *hot-spot*.

La tabla 13 muestra la productividad alcanzada cuando se utilizan diferentes ubicaciones del *hot-spot* y diferentes cargas de *hot-spot* para el toro 2D con canales *express*. En particular, se utilizan cargas *hot-spot* del 3% y del 5%. UD_ITB incrementa la productividad de UD en un factor de 1.13 y 1.08 para cargas de *hot-spot* del 3% y 5%, respectivamente.

Teniendo en cuenta los resultados obtenidos para la distribución uniforme, la productividad alcanzada por UD con una distribución uniforme se reduce en un 26% para un *hot-spot* del 3%, y por un 49% para un *hot-spot* del 5%. Para UD_ITB, la reducción en productividad, con respecto a la distribución uniforme, es de un 50% y un 67%, respectivamente. Por lo tanto, UD_ITB está más afectado por el *hot-spot* que UD. La figura 83 muestra la utilización de los enlaces para UD y UD_ITB en sus puntos de saturación (0.0483 flits/ns/conmutador para UD y 0.0542 flits/ns/conmutador para UD_ITB) utilizando tráfico *hot-spot* del 3%. Se puede observar que, debido al *hot-spot*, en UD (figura 83.a) el conmutador raíz está menos congestionado, pero los enlaces entre el conmutador raíz y el *hot-spot* están mucho

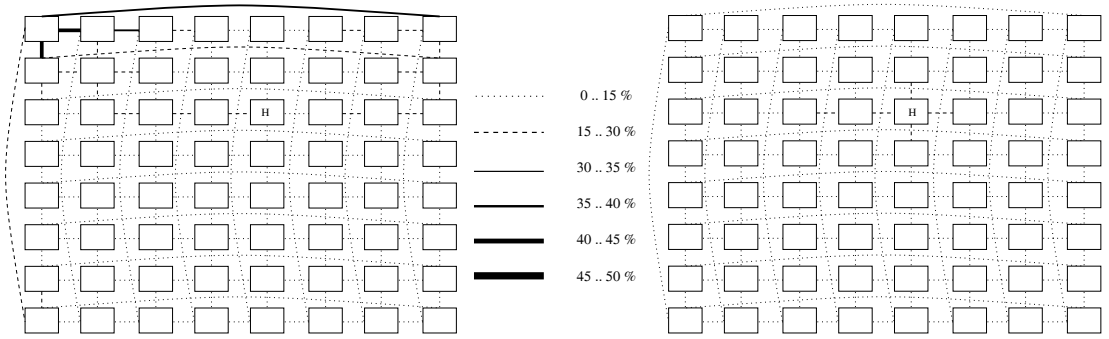


Figura 82: ITBs en redes regulares. Utilización de los enlaces en el Toro 2D con tráfico *hot-spot* en el punto de saturación de UD. (a) UD y (b) UD_ITB. Tamaño de mensajes de 512 bytes.

más utilizados que los otros enlaces en la red. Para UD_ITB (figura 83.b) solamente los enlaces cercanos al *hot-spot* están más utilizados. Se puede observar que todos los enlaces saturados son canales *express*, igual que ocurría con la distribución uniforme.

Finalmente, para la red CPLANT, la tabla 14 muestra la productividad obtenida por cada algoritmo de encaminamiento para diferentes ubicaciones de *hot-spot* y un *hot-spot* del 5%. Como media, UD_ITB incrementa la productividad de la red de UD en un factor de 1.24. Asimismo, la figura 84 muestra las prestaciones obtenidas por los algoritmos UD y UD_ITB para todas las topologías regulares evaluadas bajo diferentes condiciones de *hot-spot*.

Por último, vamos a evaluar el mecanismo en las redes regulares consideradas con un tráfico local. La figura 85 muestra las prestaciones obtenidas por los algoritmos cuando los mensajes cruzan como máximo 3 conmutadores para llegar a sus destinos. Para el Toro 2D (figura 85.a), UD_ITB obtiene una mayor productividad que UD. UD se satura cerca de 0.1 flits/ns/conmutador, mientras que UD_ITB se satura cerca de 0.13 flits/ns/conmutador. Para el Toro 2D con canales *express* (figura 85.b) UD_ITB obtiene una productividad ligeramente mayor. Para la red CPLANT (figura 85.c) también se obtienen pocas mejoras. En general, UD obtiene buenos resultados ya que el tráfico es distribuido homogéneamente entre todos los enlaces de la red (debido al patrón de tráfico utilizado). Hay que hacer constar que el encaminamiento UD siempre utiliza rutas mínimas cuando el destino está a distancia de un salto (los mensajes cruzan dos conmutadores) o está conectado directamente

	3%		5%	
Hot-spot	UD	UD_ITB	UD	UD_ITB
1	0.0481	0.0528	0.0312	0.0361
2	0.0512	0.0529	0.0337	0.0362
3	0.0482	0.0573	0.0312	0.0361
4	0.0480	0.0567	0.0362	0.0359
5	0.0449	0.0529	0.0337	0.0384
6	0.0513	0.0575	0.0355	0.0359
7	0.0470	0.0528	0.0338	0.0359
8	0.0482	0.0528	0.0338	0.0384
9	0.0481	0.0574	0.0336	0.0338
10	0.0481	0.0528	0.0312	0.0361
Media	0.0483	0.0546	0.0334	0.0363

Tabla 13: ITBs en redes regulares. Productividad para diferentes ubicaciones del *hot-spot* y diferentes tasas de *hot-spot*. Toro 2D con canales *express*. Tamaño de mensajes de 512 bytes.

al conmutador. Por lo tanto, los beneficios de utilizar ITBs son reducidos. Sin embargo, hay que hacer constar que el uso de ITBs no repercute negativamente en las prestaciones.

Si la localidad de los mensajes es menor (ahora los mensajes cruzan como máximo 4 conmutadores), el mecanismo de buffers en tránsito incrementa la productividad de la red. La figura 86 muestra las prestaciones obtenidas por los algoritmos de encaminamiento cuando los destinos de los mensajes están como máximo a 4 conmutadores de sus fuentes. La productividad de UD en la red Toro 2D es incrementada en un factor de 1.33 por UD_ITB. Para el Toro 2D con canales *express*, UD_ITB incrementa la productividad de UD en un factor de 1.50. Finalmente, en la red CPLANT, UD_ITB incrementa la productividad de UD en un factor de 1.81.

Como conclusión, en las redes regulares evaluadas, al igual que ocurrió en las redes irregulares, el mecanismo de buffers en tránsito mejora las prestaciones del algoritmo de encaminamiento UD al permitir un mejor equilibrado del tráfico entre todos los enlaces de la red, siendo la aportación de rutas mínimas por el mecanismo menos significativa.

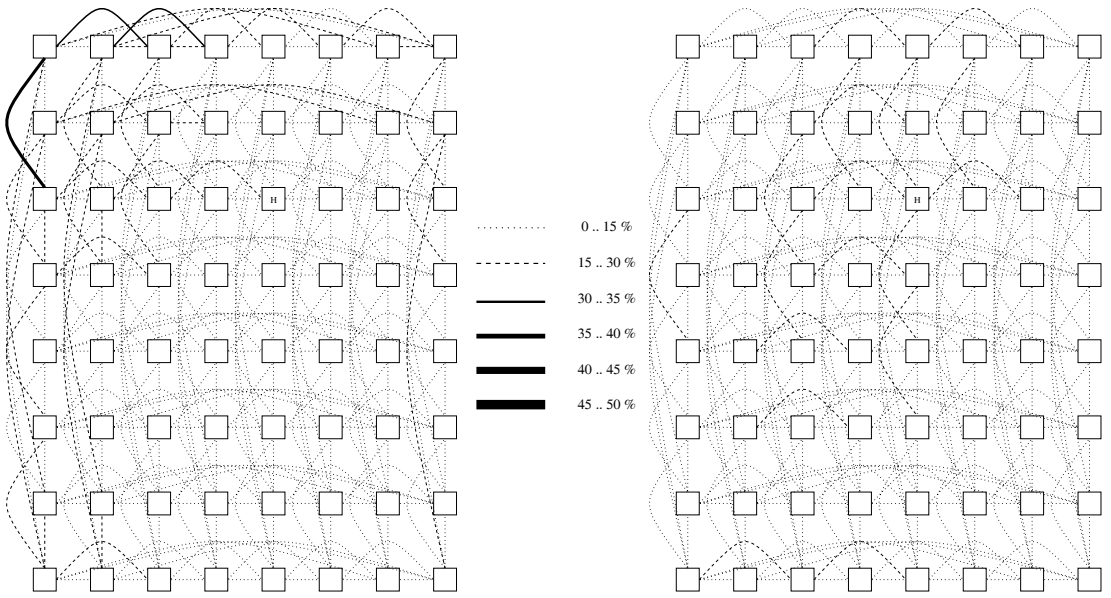


Figura 83: ITBs en redes regulares. Utilización de los enlaces en el Toro 2D con canales *express* con tráfico *hot-spot* en los puntos de saturación de (a) UD y (b) UD_ITB. Tamaño de mensajes de 512 bytes.

5.4 Nuevo algoritmo de encaminamiento

En las secciones anteriores se ha evaluado la aplicación del mecanismo ITB sobre los algoritmos de encaminamiento *up*/down**, *DFS* y *smart-routing* bajo diferentes tipos de redes regulares e irregulares, distintos patrones de tráfico y diferentes tamaños de mensajes. Se ha comprobado que el mecanismo incrementa las prestaciones de estos algoritmos de encaminamiento en términos de productividad al aportar una serie de ventajas como son: la utilización de rutas mínimas, la obtención de un mejor equilibrado de tráfico y la reducción de la contención de red.

No obstante, al aplicar el mecanismo sobre dichos algoritmos de encaminamiento aún podemos estar sufriendo algunas limitaciones implícitas de los algoritmos, puesto que la utilización de ITBs se reduce a eliminar las restricciones impuestas por la técnica de encaminamiento correspondiente, garantizando que el algoritmo resultante sea libre de bloqueo. Pero esto no supone, en modo alguno, que las tablas de encaminamiento obtenidas estén optimizadas para, por ejemplo, obtener un buen equilibrado de tráfico.

	5%	
Hot-spot	UD	UD_ITB
1	0.0336	0.0432
2	0.0336	0.0431
3	0.0336	0.0384
4	0.0289	0.0386
5	0.0338	0.0432
6	0.0338	0.0430
7	0.0337	0.0434
8	0.0389	0.0431
9	0.0337	0.0431
10	0.0366	0.0434
Media	0.0340	0.0423

Tabla 14: ITBs en redes regulares. Productividad para diferentes ubicaciones del *hot-spot*. CPLANT. Tamaño de mensajes de 512 bytes.

Por otra parte, la utilización del mecanismo ITB permite utilizar cualquier conjunto de rutas ya que los ITBs eliminarán los ciclos que aparezcan en el GDC. Por ello, la utilización de ITBs puede permitir el diseño de un algoritmo de encaminamiento que pretenda optimizar las tablas obtenidas con un criterio dado, olvidándose de la ausencia de bloqueo, lo cual será resuelto posteriormente mediante la inserción de ITBs. De hecho, el algoritmo BSMART_ITB utilizado en la sección 5.2.3 ya procedía de esta manera tratando de equilibrar el tráfico, ignorando la ausencia de bloqueo. Sin embargo, su principal inconveniente es el elevado tiempo requerido para calcular las rutas. Es por esto que, en esta sección, evaluamos el nuevo algoritmo de encaminamiento propuesto en 3.2.3 diseñado según esta filosofía. Para ello, en una primera sección evaluamos el algoritmo de equilibrado utilizado en el cálculo de rutas (recordar que dicho algoritmo realiza una reducción de la desviación típica de las rutas que pasan por los canales). Posteriormente, en una segunda sección, nos centramos en las prestaciones obtenidas por las distintas alternativas propuestas para insertar ITBs en las rutas generadas, las cuales suponen diferentes grados de reducción de contención en la red.

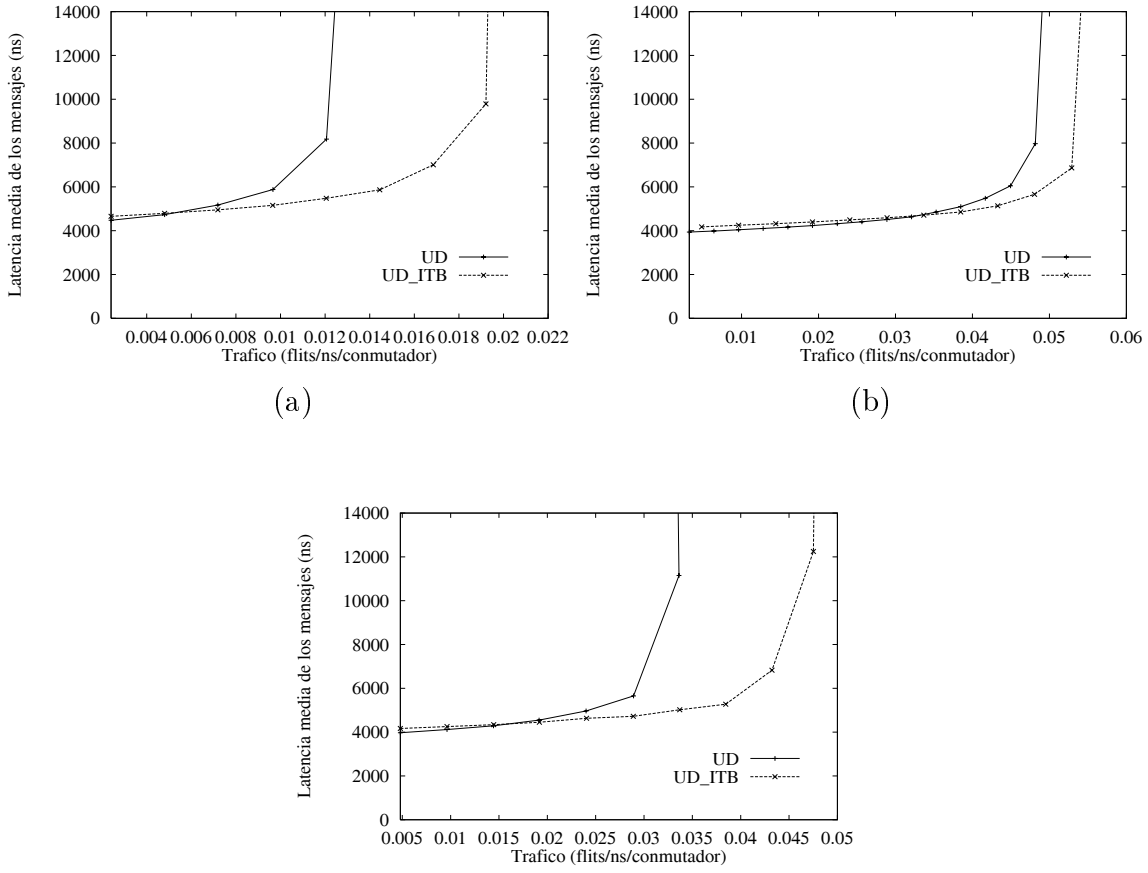


Figura 84: ITBs en redes regulares. Latencia media vs tráfico. Patrón de tráfico *hot-spot*. (a) Toro 2D (10% de *hot-spot*), (b) Toro 2D con canales *express* (3% de *hot-spot*) y (c) CPLANT (5% de *hotspot*). Tamaño de mensajes de 512 bytes.

5.4.1 Nuevo algoritmo: equilibrado del tráfico

Para poder aislar el efecto de la contención de red, y así poder evaluar solamente el algoritmo de equilibrado, vamos a utilizar en esta sección el mínimo número de ITBs que garantiza que el conjunto de rutas final sea libre de bloqueo. Para ello, se buscan ciclos en el GDC y se rompen poniendo ITBs según el procedimiento de la figura 49. El algoritmo de encaminamiento resultante lo denominamos BMIN_ITB (*Balanced with MINimum ITBs*). Este algoritmo de encaminamiento lo compararemos con los algoritmos *up*/down** y *smart-routing* (denominados UD y SMART). También lo compararemos con el mejor algoritmo de encaminamiento obtenido en las secciones

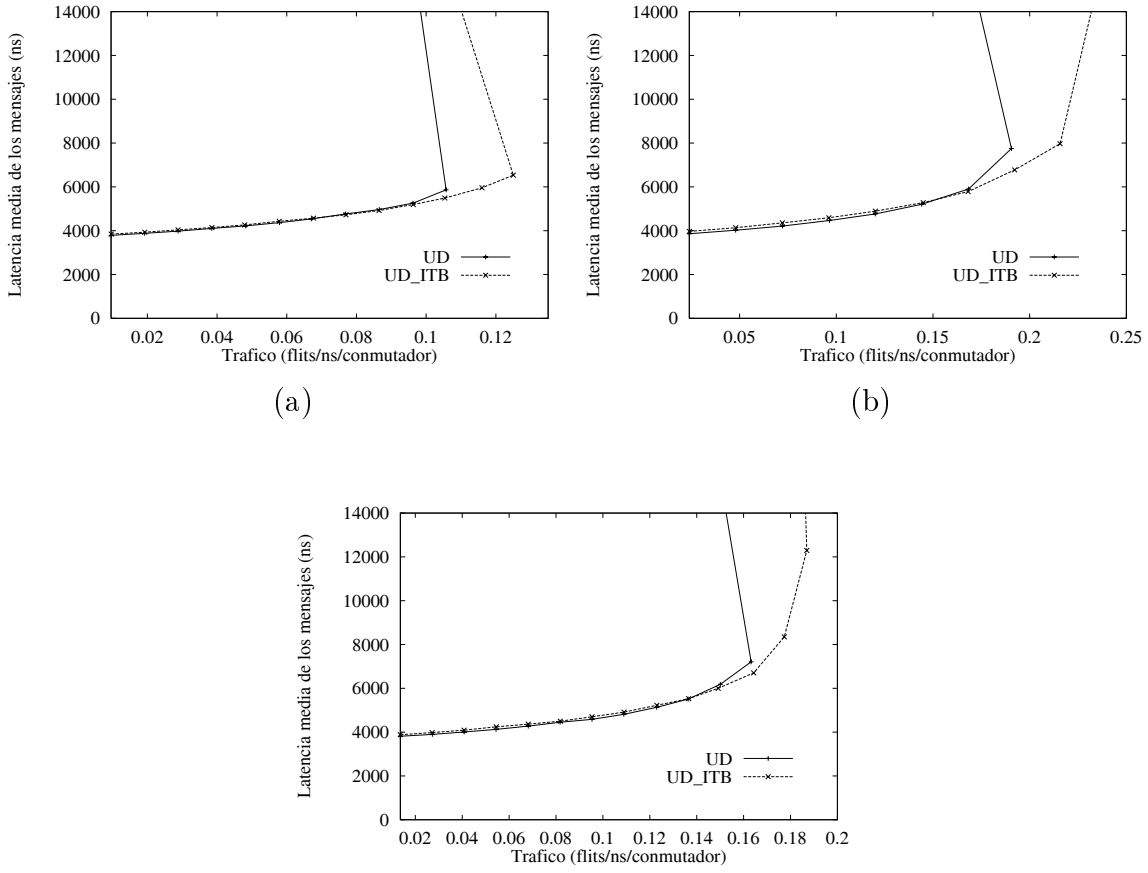


Figura 85: ITBs en redes regulares. Latencia media vs tráfico. (a) Toro 2D, (b) Toro 2D con canales *express* y (c) CPLANT. Distribución local (los destinos están como máximo a 3 conmutadores de distancia). Tamaño de mensajes de 512 bytes.

anteriores que utiliza ITBs: BSMART_ITB. Este algoritmo de encaminamiento, al igual que el algoritmo BMIN_ITB, también utiliza el mínimo número de ITBs.

Finalmente, como referencia, vamos a evaluar también un algoritmo de equilibrado de carga aleatorio con el fin de poder evaluar la aportación real de nuestro algoritmo sobre el equilibrado del tráfico. Este nuevo algoritmo calculará todas las posibles rutas mínimas, seleccionando el conjunto final de rutas de una forma aleatoria y utilizando también el mínimo número de ITBs para romper ciclos. Este algoritmo de encaminamiento lo denominaremos RANDOM_ITB (*RANDOM paths with ITBs*).

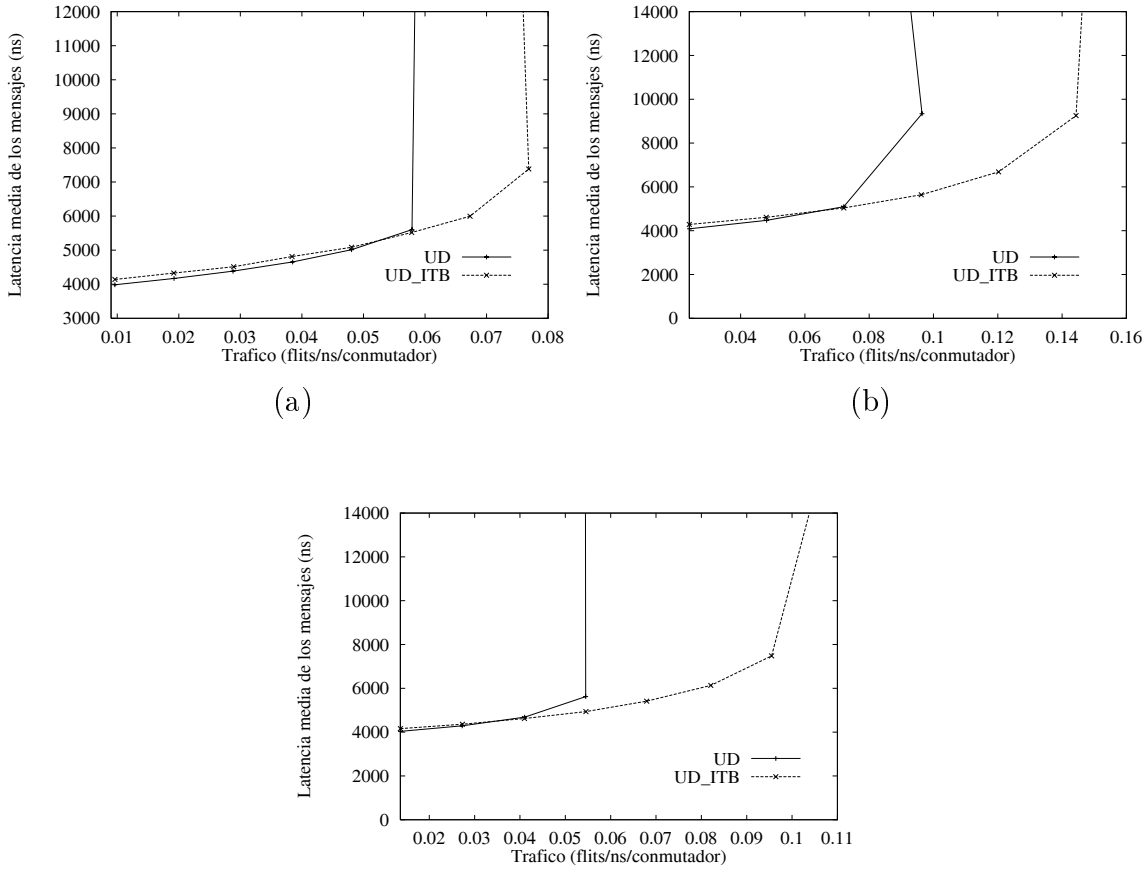


Figura 86: ITBs en redes regulares. Latencia media vs tráfico. (a) Toro 2D, (b) Toro 2D con canales *express* y (c) CPLANT. Distribución local (los destinos están como máximo a 4 conmutadores de distancia). Tamaño de mensajes de 512 bytes.

La figura 87 muestra los resultados obtenidos para los diferentes algoritmos de encaminamiento en redes irregulares de 8, 16, 32 y 64 conmutadores. El tamaño de los mensajes es de 512 bytes y la distribución de destinos es uniforme. Nuevamente, los algoritmos SMART y BSMART_ITB no estaban disponibles para redes de 64 conmutadores.

Como podemos observar, los algoritmos de encaminamiento que no utilizan buffers en tránsito (UD y SMART) obtienen productividades menores, excepto para el algoritmo SMART en la red de 16 conmutadores que obtiene mayor productividad

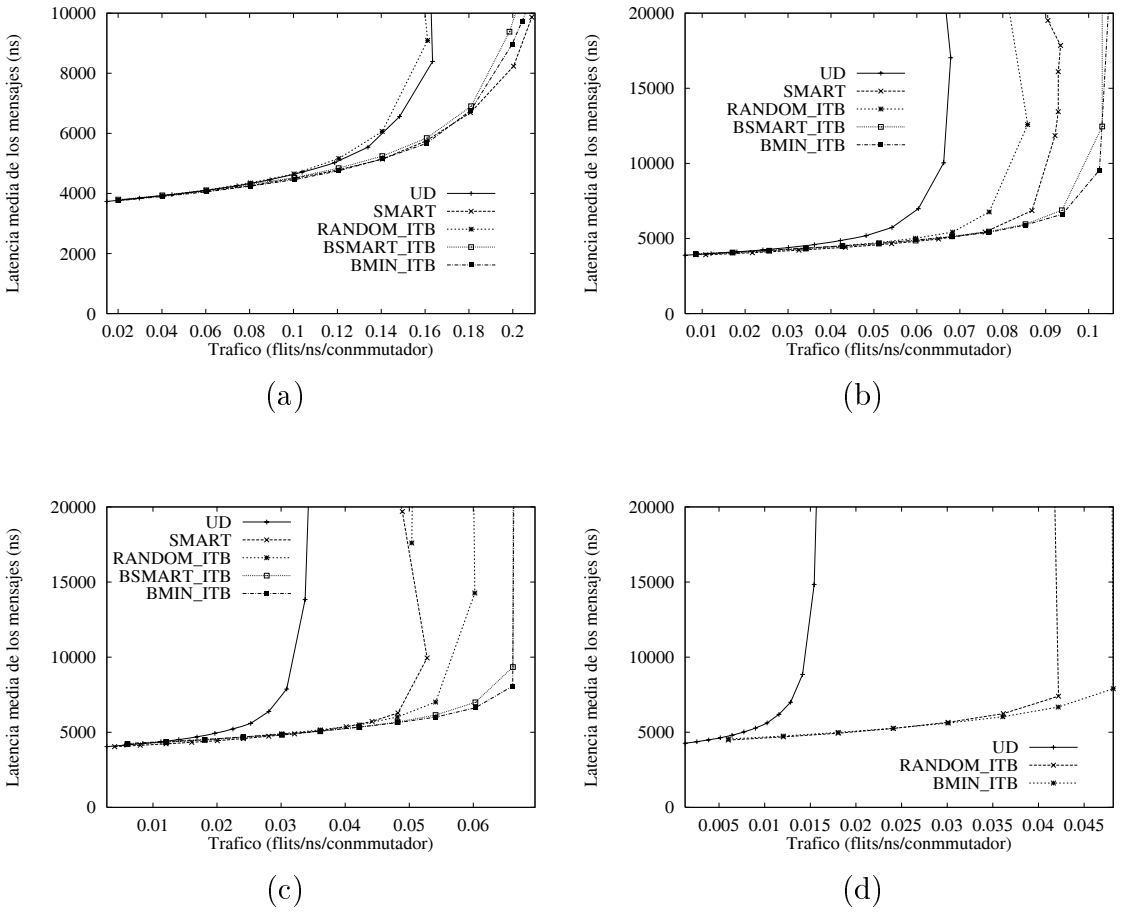


Figura 87: Nuevo algoritmo de encaminamiento (equilibrado). Latencia media vs. tráfico. Red de (a) 8, (b) 16, (c) 32 y (d) 64 conmutadores. Tamaño de mensajes de 512 bytes. Distribución uniforme de destinos.

que el algoritmo RANDOM_ITB. El comportamiento de dichos algoritmos de encaminamiento ya ha sido descrito en secciones anteriores por lo que nos centramos exclusivamente en los nuevos algoritmos que utilizan ITBs (dichos algoritmos se han representado solamente como referencia).

Respecto a los algoritmos de encaminamiento que utilizan ITBs (BMIN_ITB, BSMART_ITB y RANDOM_ITB), los algoritmos BMIN_ITB y BSMART_ITB obtienen en todas las redes mayores productividades que las alcanzadas por el algoritmo RANDOM_ITB. Este incremento en productividad con respecto a RANDOM_ITB

no es debido a una mayor utilización de los ITBs, ya que los tres algoritmos de encaminamiento utilizan la misma cantidad de ITBs (aproximadamente). Este incremento en productividad es debido al mejor equilibrado obtenido por BSMART_ITB y BMIN_ITB. Las figuras 88.a y 88.b muestran la utilización de los enlaces entre conmutadores cuando el tráfico es 0.066 flits/ns/conmutador en la red de 32 conmutadores para los algoritmos BSMART_ITB y BMIN_ITB, respectivamente. En este punto de tráfico, los dos algoritmos están entrando en saturación.

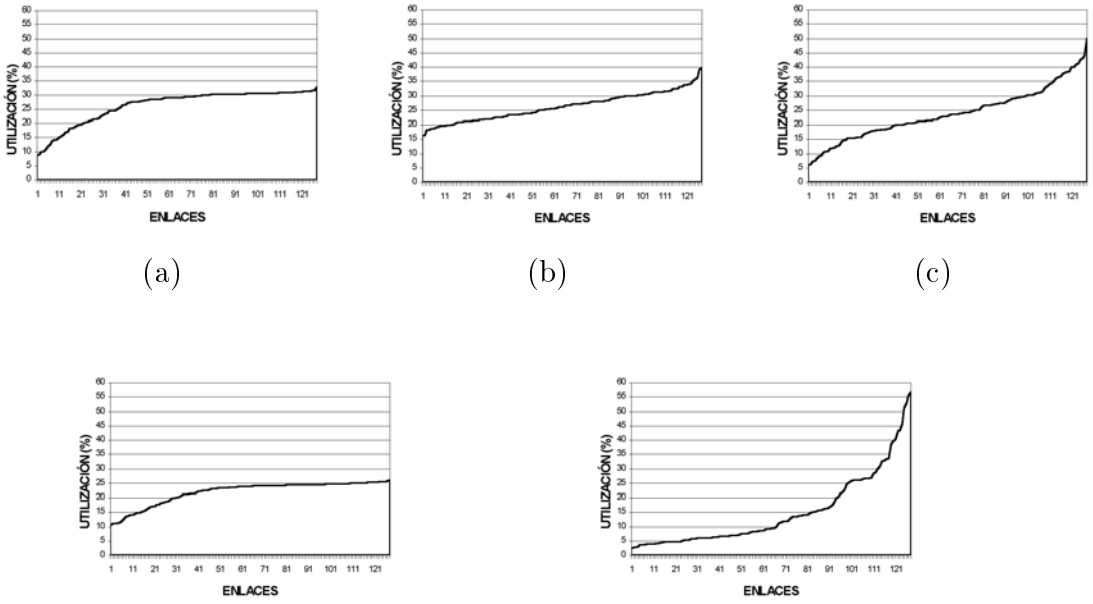


Figura 88: Nuevo algoritmo de encaminamiento (equilibrado). Utilización de los enlaces. (a) BSMART_ITB, (b) BMIN_ITB, (c) RANDOM_ITB, (d) SMART y (e) UD. El tráfico es (a, b) 0.066, (c) 0.053, (d) 0.05 y (e) 0.03 flits/ns/conmutador. Red de 32 conmutadores. Tamaño de mensajes de 512 bytes.

Podemos observar como el equilibrado del tráfico obtenido por el algoritmo BSMART_ITB (figura 88.a) es ligeramente mejor que el obtenido por el algoritmo BMIN_ITB (figura 88.b). Sin embargo, este mejor equilibrado del tráfico no tiene un impacto muy importante en las prestaciones de la red, tal y como se puede observar en la figura 87.c.

Sin embargo, la principal diferencia entre ambos algoritmos de encaminamiento está en que BSMART_ITB utiliza un algoritmo de programación lineal para calcular

las rutas, mientras que el nuevo algoritmo de encaminamiento BMIN_ITB utiliza un proceso iterativo, por lo que es mucho más rápido. Incluso en redes grandes de 64 conmutadores, el nuevo algoritmo de equilibrado está disponible en un tiempo de cálculo reducido mientras que el algoritmo BSMART_ITB no está disponible.

La figura 88.c muestra la utilización de los enlaces para el algoritmo RANDOM_ITB cuando está alcanzando la saturación (0.053 flits/ns/conmutador) en la misma red de 32 conmutadores. Podemos observar como el algoritmo RANDOM_ITB obtiene peor equilibrado que los algoritmos BMIN_ITB y BSMART_ITB. Esta diferencia afecta a las prestaciones de la red tal y como se observa en la figura 87.c. En particular, BMIN_ITB y BSMART_ITB incrementan la productividad del algoritmo RANDOM_ITB en un factor de 1.24. Este resultado indica que es importante llevar a cabo un esfuerzo de equilibrado del tráfico al calcular las rutas definitivas. En concreto, con un algoritmo simple y rápido como el implementado en el algoritmo BMIN_ITB, se obtienen buenos resultados. Por otra parte, con esfuerzos adicionales en el equilibrado (BSMART_ITB) se pueden obtener ligeras mejoras, pero no sustanciales.

La tabla 15 muestra los resultados medios obtenidos para 10 topologías de cada tamaño de red. La tabla muestra el incremento en productividad mínimo, máximo y medio cuando se utiliza el algoritmo de encaminamiento BMIN_ITB con respecto a los algoritmos de encaminamiento UD, SMART, BSMART_ITB y RANDOM_ITB, respectivamente.

	UD			SMART			BSMART_ITB			RANDOM_ITB		
Con	Mín	Máx	Med	Mín	Máx	Med	Mín	Máx	Med	Mín	Máx	Med
16	1.21	1.63	1.44	1.06	1.21	1.12	0.95	1.13	1.03	1.07	1.39	1.21
32	1.65	2.30	1.99	1.22	1.49	1.33	0.99	1.12	1.03	1.08	1.27	1.18
64	2.34	3.25	2.80	N/D	N/D	N/D	N/D	N/D	N/D	1.00	1.27	1.16

Tabla 15: Nuevo algoritmo de encaminamiento (equilibrado). Factores de incremento de productividad al utilizar BMIN_ITB con respecto a UD, SMART, BSMART_ITB y RANDOM_ITB. Distribución uniforme de destinos. Tamaño de mensajes de 512 bytes.

Podemos observar como el algoritmo BMIN_ITB obtiene, conforme aumenta el tamaño de la red, en términos medios mayor productividad que los algoritmos UD y SMART . La productividad de UD es doblada, como media, para una red de 32 conmutadores mientras que la productividad de SMART es incrementada, como

media, en un factor de 1.33.

Con respecto al algoritmo de encaminamiento BSMART_ITB, prácticamente se alcanza la misma productividad. Por lo tanto, con un algoritmo de equilibrado sencillo (BMIN_ITB) podemos obtener prestaciones similares a las de un algoritmo sofisticado (y con demandas de tiempo de cálculo elevadas) como BSMART_ITB. Por contra, el algoritmo BMIN_ITB requiere de un tiempo de cálculo de rutas mucho menor que BSMART_ITB.

Finalmente, con respecto al algoritmo de encaminamiento RANDOM_ITB, el algoritmo BMIN_ITB también incrementa su productividad (como media, en un factor de 1.21 para redes de 16 conmutadores). Sin embargo, conforme aumenta el tamaño de la red, las mejoras son, en términos medios, prácticamente las mismas. Esto es debido a que el algoritmo RANDOM_ITB es completamente escalable (siempre utiliza rutas mínimas) y la única diferencia con respecto al algoritmo BMIN_ITB es el algoritmo de equilibrado de tráfico.

La tabla 16 muestra el porcentaje de incremento en latencia al utilizar el algoritmo de encaminamiento BMIN_ITB en muy baja carga con respecto a los algoritmos UD, SMART, BSMART_ITB y RANDOM_ITB, respectivamente. La tabla muestra resultados para mensajes de 512 bytes y 32 bytes por separado así como combinados (bimodal).

En la tabla podemos observar como, el incremento en latencia es superior al 10% solamente al utilizar mensajes cortos y en redes medianas y grandes (32 y 64 conmutadores) y sólo con respecto a los algoritmos de encaminamiento UD y SMART. Sin embargo, cuando el tráfico tiene diferentes tamaños de mensajes (bimodal), el incremento de latencia sobre UD y SMART es reducido (menor del 10% en todas las redes).

Con respecto al resto de algoritmos de encaminamiento, el incremento en latencia de BMIN_ITB es inferior al 10% en todas las redes y con todos los tamaños de mensajes. Todos estos algoritmos de encaminamiento utilizan ITBs, y por lo tanto, todos ellos están expuestos a los incrementos en latencia del propio mecanismo.

Tan sólo nos queda analizar los algoritmos para otros patrones de tráfico. La tabla 17 muestra el factor de incremento en productividad al utilizar el algoritmo BMIN_ITB bajo otros patrones de tráfico con respecto a UD, SMART, BSMART_ITB y RANDOM_ITB.

Con los patrones de tráfico *hot-spot* y *bit-reversal*, el algoritmo BMIN_ITB obtiene menores incrementos en productividad que con la distribución uniforme. No

		32 bytes			512 bytes			bimodal		
Con	Algoritmos	Mín	Máx	Med	Mín	Máx	Med	Mín	Máx	Med
16	BMIN_ITB	2.27	10.92	6.83	0.06	1.65	0.81	0.70	3.92	2.00
32	vs	12.45	19.15	15.22	2.32	4.56	3.22	3.89	8.90	5.91
64	UD	13.95	20.09	17.18	-0.14	2.83	1.76	-1.42	4.69	2.95
16	BMIN_ITB	9.95	12.77	8.04	0.07	2.44	1.63	0.98	5.61	3.69
32	vs	12.43	18.58	15.57	2.35	3.97	3.38	6.35	9.86	8.23
64	SMART	N/D	N/D	N/D	N/D	N/D	N/D	N/D	N/D	N/D
16	BMIN_ITB	-2.91	3.92	0.25	-0.96	0.70	0.00	-2.40	1.84	-0.08
32	vs	0.40	6.56	4.12	0.30	1.61	1.02	0.14	3.97	2.27
64	BSMART_ITB	N/D	N/D	N/D	N/D	N/D	N/D	N/D	N/D	N/D
16	BMIN_ITB	0.27	7.05	3.13	-0.17	1.15	0.48	-0.70	3.23	1.11
32	vs	-0.83	6.96	4.23	-0.17	1.59	0.94	0.93	4.02	1.93
64	RANDOM_ITB	1.16	6.37	4.31	0.43	1.68	1.12	0.90	5.28	2.79

Tabla 16: Nuevo algoritmo de encaminamiento (equilibrado). Porcentaje de incremento en latencia al utilizar BMIN_ITB con respecto a UD, SMART, BSMART_ITB y RANDOM_ITB. Distribución uniforme de destinos. Tamaño de mensajes de 512 bytes.

obstante, aún son significativos. La productividad de UD y SMART es incrementada por el algoritmo BMIN_ITB. En particular, el algoritmo BMIN_ITB dobla la productividad de UD, como media, en redes de 32 y 64 conmutadores, e incrementa la productividad de SMART en factores de 1.09 y 1.29 para redes de 32 conmutadores.

Respecto al algoritmo BSMART_ITB, el algoritmo BMIN_ITB prácticamente obtiene la misma productividad. Por último, comparando con RANDOM_ITB, el algoritmo BMIN_ITB obtiene menores beneficios en las distribuciones *hot-spot* y *bit-reversal* (1.07 y 1.12 como media, para redes de 32 conmutadores) que en la distribución uniforme. Para el patrón de tráfico local, se obtiene prácticamente la misma productividad en todos los algoritmos. Sin embargo, nótese que para este patrón de tráfico el mecanismo ITB no disminuye la productividad.

Por último, con el patrón de tráfico combinado el algoritmo de encaminamiento BMIN_ITB obtiene prácticamente los mismos beneficios que los descritos para la distribución uniforme de destinos. En resumen, al cambiar el patrón de tráfico, la longitud de las rutas también cambia. Con ITBs, cuanto más largas sean las rutas originales, mayores beneficios se obtendrán. Al reducirse la longitud de las rutas (tráfico local) menos ITBs se necesitarán para realizar el equilibrado y, por consiguiente, la productividad de los diferentes algoritmos de encaminamiento será

		UD			SMART			BSMART_ITB			RANDOM_ITB		
Con	Tráfico	Mín	Máx	Med	Mín	Máx	Med	Mín	Máx	Med	Mín	Máx	Med
16	*Hspot*	0.99	1.18	1.06	0.99	1.03	1.01	0.99	1.02	1.00	0.99	1.17	1.02
32	*Hspot*	1.11	1.43	1.25	1.01	1.15	1.09	0.99	1.15	1.05	0.99	1.15	1.07
64	*Hspot*	1.66	2.46	2.00	N/D	N/D	N/D	N/D	N/D	N/D	0.87	1.15	1.02
16	*Brev*	1.11	2.16	1.45	0.74	1.02	0.93	0.94	1.50	1.09	0.76	1.59	1.18
32	*Brev*	1.37	2.58	2.13	1.12	1.59	1.29	0.80	1.24	1.05	0.88	1.51	1.12
64	*Brev*	2.21	3.75	2.94	N/D	N/D	N/D	N/D	N/D	N/D	0.92	1.20	1.07
16	Local	0.97	1.11	1.04	0.95	1.21	1.07	0.96	1.10	1.04	0.99	1.15	1.06
32	Local	0.96	1.08	1.04	1.00	1.22	1.10	1.00	1.11	1.07	0.95	1.10	1.04
64	Local	0.98	1.20	1.06	N/D	N/D	N/D	N/D	N/D	N/D	0.96	1.09	1.03
16	Comb	1.19	1.68	1.40	0.99	1.27	1.10	0.99	1.09	1.02	1.08	1.32	1.16
32	Comb	1.57	2.25	1.84	1.14	1.34	1.25	1.00	1.13	1.06	1.03	1.37	1.17
64	Comb	1.98	2.79	2.45	N/D	N/D	N/D	N/D	N/D	N/D	0.96	1.26	1.11

Tabla 17: Nuevo algoritmo de encaminamiento (equilibrado). Factores de incremento de productividad al utilizar BMIN_ITB con respecto a UD, SMART, BSMART_ITB y RANDOM_ITB con diferentes patrones de tráfico. Tamaño de mensajes de 512 bytes.

muy parecida.

Como conclusión, hemos visto que el algoritmo de equilibrado de tráfico basado en el método de la desviación típica proporciona un equilibrado parecido al aportado por el algoritmo SMART pero a diferencia de éste, las rutas se calculan en un tiempo mucho menor.

5.4.2 Nuevo algoritmo: reducción de la contención

En esta sección nos centramos en la segunda parte del algoritmo de encaminamiento (rotura de ciclos en el GDC y control de la contención de la red). Para ello, hemos propuesto varios métodos para asignar ITBs en la red mientras se rompen ciclos (ver sección 3.2.3). Cada método utiliza un número diferente de ITBs en la red, asegurando todos ellos que el GDC resultante sea acíclico. Los algoritmos resultantes de aplicar estos métodos son: BMIN_ITB que utiliza el mínimo número de ITBs posibles (ya evaluado en la sección anterior), B33_ITB que utiliza como media un 33% de ITBs en toda la red, B50_ITB que utiliza un 50% de ITBs, B66_ITB que utiliza un 66% de ITBs y finalmente BMAX_ITB que utiliza ITBs en todos los conmutadores en todas las rutas. Todos estos algoritmos los vamos a comparar con los algoritmos de encaminamiento sin ITBs: UD y SMART. Hay que hacer constar

que BMIN_ITB, B33_ITB, B50_ITB, B66_ITB y BMAX_ITB utilizan las mismas rutas calculadas con el algoritmo de equilibrado de tráfico propuesto en la sección 3.2.3 y evaluado en la sección anterior. La única diferencia entre ellos es el número de ITBs utilizados.

La figura 89 muestra los resultados obtenidos para los diferentes algoritmos de encaminamiento utilizando mensajes de 512 bytes y una distribución uniforme de destinos.

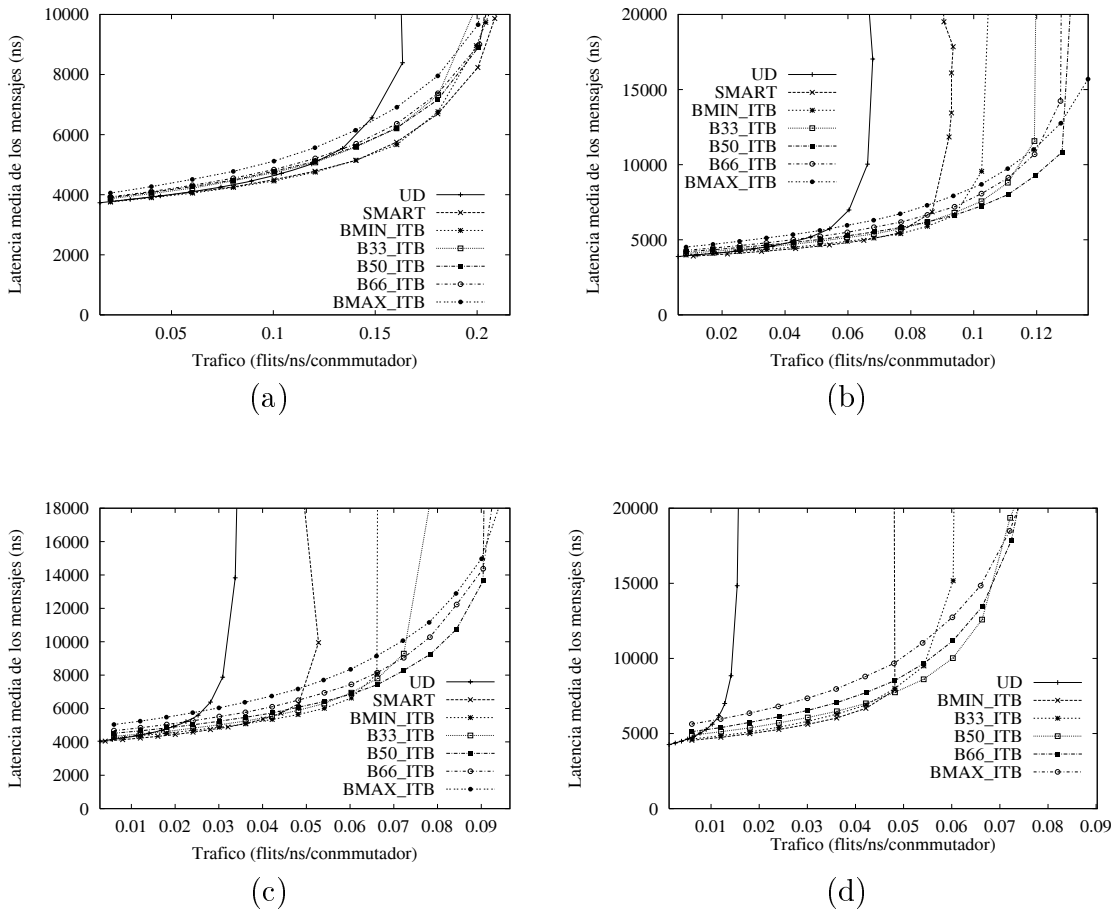


Figura 89: Nuevo algoritmo de encaminamiento (contención). Latencia media vs. tráfico. Red de (a) 8, (b) 16, (c) 32 y (d) 64 conmutadores. Tamaño de mensajes de 512 bytes. Distribución uniforme de destinos.

Como podemos observar, en la red de 8 conmutadores (figura 89.a), todos los algoritmos de encaminamiento (a excepción del algoritmo UD) obtienen prácticamente

las mismas prestaciones en términos de productividad. Sin embargo, conforme aumenta el tamaño de la red podemos ver como existen diferencias apreciables entre los diferentes algoritmos de encaminamiento.

En particular, en la red de 16 conmutadores podemos ver como los algoritmos que menor productividad alcanzan son UD y SMART. En términos absolutos vemos como el algoritmo BMAX_ITB alcanza la mayor productividad, siendo después B66_ITB y B50_ITB los que obtienen una mayor productividad. Por detrás de estos, B33_ITB alcanza mayor productividad que BMIN_ITB. Por lo tanto, podemos ver como los algoritmos alcanzan mayor o menor productividad en función del número de ITBs que utilizan, siendo los que utilizan más ITBs los que alcanzan una productividad mayor. En particular, BMAX_ITB incrementa la productividad de BMIN_ITB en factores de incremento de 1.27. Sin embargo, cabe resaltar que las diferencias en productividad entre los algoritmos que emplean mayor número de ITBs (B50_ITB, B66_ITB y BMAX_ITB) son pequeñas.

En términos de latencia, sin embargo, vemos que el uso de más ITBs incrementa la latencia media de los mensajes como cabía esperar. No obstante, como vemos, este incremento no es muy significativo (especialmente en este caso en que los mensajes son de 512 bytes). Más adelante evaluaremos el incremento en latencia para mensajes cortos.

Para las redes de 32 y 64 conmutadores (figuras 89.c y 89.d) se obtienen conclusiones similares. Como se puede apreciar, vemos que la productividad alcanzada por los algoritmos B50_ITB, B66_ITB y BMAX_ITB es prácticamente la misma, por lo que el incremento en productividad no está en función directa con el número de ITBs. Por lo tanto, con un número moderado de ITBs (B50_ITB) podemos alcanzar prácticamente la productividad de BMAX_ITB y con una penalización en latencia menor. Por último, con respecto a BMIN_ITB vemos que BMAX_ITB incrementa su productividad en unos factores de 1.36 y 1.5 en las redes de 32 y 64 conmutadores, respectivamente.

Para más topologías de red se obtienen resultados muy similares. La tabla 18 muestra los factores de incremento de productividad mínimo, máximo y medio para diferentes topologías de diferentes tamaños de red. Como media, el algoritmo de encaminamiento BMAX_ITB incrementa la productividad de BMIN_ITB en un factor de incremento de 1.36 para redes de 64 conmutadores. Con respecto a los demás algoritmos, los incrementos en productividad de BMAX_ITB frente a los algoritmos

B33_ITB, B50_ITB y B66_ITB son menores. Cabe destacar los significativos incrementos en productividad que alcanzan en general los algoritmos con respecto a UD y SMART.

Conm.	Algoritmos	Mín	Máx	Media
8		1.30	1.69	1.42
16	BMAX_ITB	1.42	2.12	1.85
32	vs	2.44	3.42	2.98
64	UD	3.42	4.67	4.11
8		0.97	1.09	1.03
16	BMAX_ITB	0.91	1.46	1.32
32	vs	1.68	1.90	1.78
64	SMART	N/D	N/D	N/D
8		0.99	1.15	1.07
16	BMAX_ITB	0.89	1.26	1.18
32	vs	1.27	1.41	1.34
64	BMIN_ITB	1.24	1.42	1.36
8		0.98	1.11	1.04
16	BMAX_ITB	0.89	1.18	1.09
32	vs	1.14	1.33	1.22
64	B33_ITB	1.09	1.22	1.13
8		0.95	1.03	1.00
16	BMAX_ITB	1.00	1.15	1.03
32	vs	1.06	1.15	1.08
64	B50_ITB	1.00	1.09	1.01
8		1.00	1.07	1.02
16	BMAX_ITB	1.00	1.08	1.03
32	vs	1.07	1.15	1.12
64	B66_ITB	1.00	1.10	1.05

Tabla 18: Nuevo algoritmo de encaminamiento (contención). Factor de incremento de productividad al utilizar BMAX_ITB con respecto a UD, SMART, BMIN_ITB, B33_ITB, B50_ITB y B66_ITB. Distribución uniforme de destinos. Tamaño de mensajes de 512 bytes.

Para comprender mejor los resultados obtenidos, vamos a analizar el porcentaje de tiempo de bloqueo de los enlaces entre conmutadores obtenido por cada algoritmo de encaminamiento. La figura 90 muestra los porcentajes para los algoritmos BMIN_ITB, B33_ITB, B50_ITB, B66_ITB y BMAX_ITB para la red de 32 conmutadores y tráfico de 0.066 flits/ns/conmutador (BMIN_ITB está entrando en

saturación).

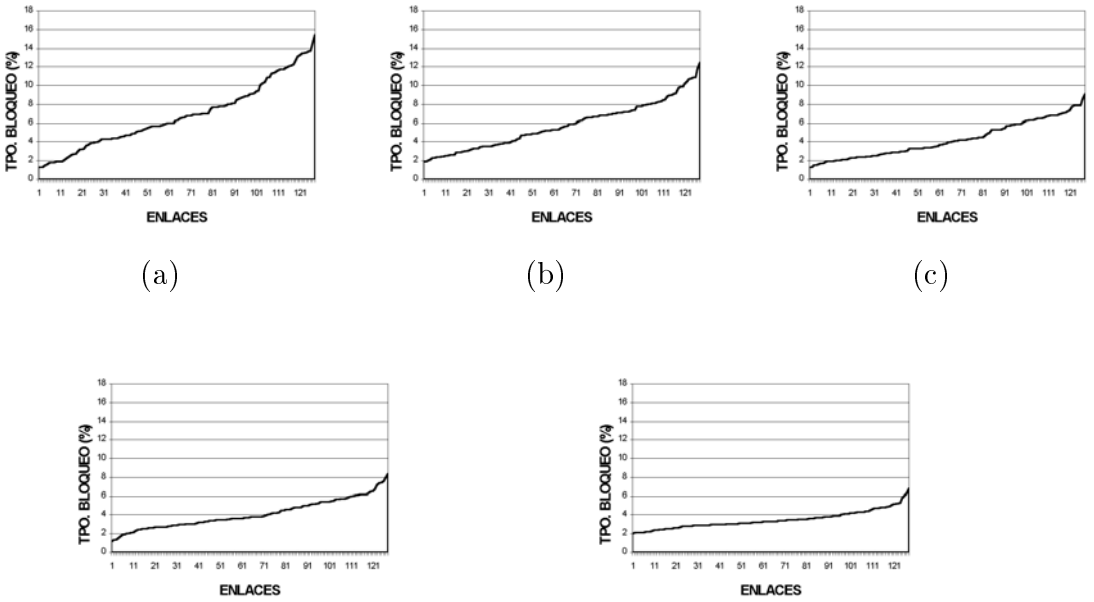


Figura 90: Nuevo algoritmo de encaminamiento (contención). Porcentaje de tiempo de bloqueo. (a) BMIN_ITB, (b) B33_ITB, (c) B50_ITB, (d) B66_ITB y (e) BMAX_ITB. Tráfico es 0.066 flits/ns/conmutador. Red de 32 conmutadores. Tamaño de mensajes de 512 bytes.

En las figuras podemos observar como, al aumentar el uso de ITBs por los algoritmos, los enlaces están menos tiempo bloqueados. En concreto, BMIN_ITB es el algoritmo en el que los enlaces están más tiempo bloqueados. Conforme se van utilizando más ITBs por parte de B33_ITB, B50_ITB, B66_ITB y BMAX_ITB, los enlaces están menos tiempo bloqueados. En concreto, para BMIN_ITB hay enlaces con un tiempo de bloqueo del 14%, mientras que para BMAX_ITB el tiempo de bloqueo máximo es del 6%. Por lo tanto, el mecanismo ITB reduce la contención de red, lo que repercute en un incremento significativo de la productividad.

Sin embargo, aunque en términos de productividad, la mejor opción de encaminamiento la ofrece BMAX_ITB, por otra parte, el incremento en latencia (que estudiamos a continuación) debe tomarse también en cuenta. Por el momento, podemos observar como los algoritmos de encaminamiento B50_ITB y B66_ITB alcanzan prácticamente la misma productividad que el algoritmo de encaminamiento

BMAX_ITB utilizando bastantes menos ITBs (la mitad y un 33% de ITBs menos, respectivamente).

Como ya hemos indicado, cuando se emplean ITBs en los algoritmos de encaminamiento, el posible incremento en latencia es un factor a tener en cuenta. Para ello, la tabla 19 muestra el porcentaje de incremento en latencia al utilizar el algoritmo BMAX_ITB con respecto a los demás algoritmos de encaminamiento en condiciones de poco tráfico.

		32 bytes			512 bytes		
Conm.	Algoritmos	Mín	Máx	Media	Mín	Máx	Media
8		36.52	41.46	39.97	7.83	8.73	8.54
16	BMAX_ITB	74.75	102.05	81.52	14.92	19.67	16.22
32	vs	101.96	115.80	109.14	21.56	24.75	23.82
64	UD	137.75	150.63	141.56	31.94	34.21	32.92
8		35.90	40.41	38.09	7.30	7.95	7.64
16	BMAX_ITB	75.68	102.52	83.01	14.03	18.03	15.18
32	vs	104.61	118.71	110.45	22.63	25.50	23.94
64	SMART	N/D	N/D	N/D	N/D	N/D	N/D
8		37.87	41.55	39.88	7.64	8.08	7.97
16	BMAX_ITB	62.80	98.82	68.86	13.47	19.25	14.29
32	vs	74.26	87.86	81.42	17.32	19.95	18.62
64	BMIN_ITB	92.16	106.28	98.43	22.88	25.63	23.91
8		16.32	25.13	22.61	3.33	5.60	4.97
16	BMAX_ITB	44.45	65.57	49.11	10.25	14.54	11.10
32	vs	59.10	67.82	64.33	14.56	16.55	16.02
64	B33_ITB	73.39	83.54	78.01	19.46	21.79	20.81
8		10.88	20.30	15.88	2.62	4.54	3.62
16	BMAX_ITB	26.75	36.72	31.94	6.77	8.82	7.58
32	vs	40.17	48.35	44.05	10.40	12.45	11.34
64	B50_ITB	53.23	61.12	57.16	14.38	16.42	15.16
8		9.09	15.00	12.05	2.21	3.59	2.82
16	BMAX_ITB	13.03	24.46	19.87	3.05	6.51	5.12
32	vs	23.01	30.91	27.21	6.45	8.44	7.61
64	B66_ITB	31.04	40.17	35.05	8.81	10.17	9.71

Tabla 19: Nuevo algoritmo de encaminamiento (contención). Porcentaje de incremento de latencia al utilizar BMAX_ITB. Distribución uniforme de destinos.

Podemos observar como, al utilizar muchos ITBs (tal y como utiliza BMAX_ITB), la latencia sufrida por los mensajes cortos se dobla en algunos casos, con respecto a

UD y SMART. Pero esto ocurre sólo para mensajes cortos y en condiciones de poco tráfico. Por otra parte, el incremento en latencia con respecto a B50_ITB llega hasta el 60%. Por lo tanto, el algoritmo B50_ITB que obtiene prácticamente la misma productividad ofrece una penalización en latencia mucho menor.

En resumen, el equilibrado de tráfico no es lo único que contribuye a obtener elevadas prestaciones en productividad. Reducir la contención de red con ITBs puede llevar a mejoras incluso mayores en productividad pagando como precio un moderado aumento en la latencia sufrida por los mensajes cortos con bajo nivel de tráfico. De entre los nuevos algoritmos propuestos para reducir la contención, el algoritmo de encaminamiento B50_ITB explota el equilibrado y la contención de red, mejorando ostensiblemente anteriores propuestas y manteniendo el incremento en latencia en un límite razonable.

5.5 Actuación sobre la latencia

Como se ha visto en las secciones anteriores, el mecanismo de buffers en tránsito incrementa de forma notable las prestaciones obtenidas por los algoritmos de encaminamiento tradicionales. Por otra parte, este mecanismo tiene como principal inconveniente el incremento en latencia de los mensajes. No obstante, se ha visto como esta penalización en latencia es significativa solamente en condiciones de tráfico reducido y en mensajes cortos. Cuando los mensajes son más largos, el incremento relativo de latencia es menor debido a que el tiempo de transmisión de los mensajes es mucho mayor. Incluso se ha comprobado que en un entorno con tráfico bimodal con un porcentaje reducido de mensajes largos (30% de 512 bytes) la penalización en latencia se reduce ostensiblemente, llegándose a incrementos en latencias menores del 10%. Por último, también se ha concluido que el utilizar menos ITBs también ayuda a reducir la penalización en latencia, a costa de un menor aumento en la productividad alcanzada por la red.

Sin embargo, con el afán de disminuir aún más dicho efecto negativo del mecanismo repercutiendo al mínimo en las prestaciones obtenidas en términos de productividad, en esta sección evaluamos posibles modificaciones del mecanismo con el fin de reducir la latencia añadida a los mensajes. Brevemente, dichas modificaciones son:

- Utilizar el mecanismo ITB de una forma más conservadora para los mensajes

cortos, mientras que para los mensajes largos se utilizará el mecanismo sin restricciones (donde el efecto de la latencia es menor). Con esta técnica, los mensajes cortos utilizarán pocos ITBs por lo que se verán menos afectados por la latencia introducida por el mecanismo.

- Limitar la utilización del mecanismo ITB dependiendo del porcentaje de incremento de latencia máximo permitido. En función de este porcentaje se definirá la probabilidad de utilizar una ruta con o sin ITBs para un mensaje determinado. Con esto, limitaremos el uso del mecanismo ITB y, por consiguiente la latencia adicional que el mecanismo introduce sobre los mensajes.

- Utilizar el mecanismo ITB a partir de cierto nivel de tráfico. Este nivel de tráfico será detectado con el mecanismo de detección de contención descrito en la sección 2.4. Por lo tanto, los mensajes utilizarán rutas sin ITBs cuando el nivel de tráfico sea reducido y rutas con ITBs para niveles de tráfico medios o elevados, donde la sobrecarga introducida por el mecanismo ya no afecta.

En las siguientes secciones se evalúan dichas mejoras. Cabe recordar que el objetivo final de todas ellas es reducir la latencia extra introducida por el mecanismo perdiendo el mínimo posible de productividad obtenido.

5.5.1 Selección de rutas en función del tamaño de los mensajes

Para utilizar ITBs en función del tamaño del mensaje necesitamos un mecanismo de selección de rutas que, en función del tamaño del mensaje, seleccione una ruta u otra. Para ello, vamos a utilizar un tráfico bimodal con dos tamaños de mensajes:

- Mensajes cortos. Estos mensajes serán de 32 bytes y representarán el 70% de la carga total.

- Mensajes largos. Estos mensajes serán de 512 bytes y representarán el resto de la carga (30%).

Para la evaluación del mecanismo de selección vamos a calcular dos rutas para cada par origen-destino. La primera ruta será calculada de la misma forma que la calcula el algoritmo UD_MITB. Esta ruta será la que contiene un menor número

de ITBs. Por otra parte, la segunda ruta será calculada de la misma forma que la calcula el algoritmo UD_ITB. Esta es la ruta con más ITBs. El algoritmo de selección escogerá la ruta UD_MITB cuando el tamaño del mensaje sea de 32 bytes, escogiendo la ruta UD_ITB en caso contrario. Con este funcionamiento aseguramos que los mensajes cortos, que se ven altamente afectados por la latencia, utilizarán el mínimo número de ITBs, mientras que los mensajes largos utilizarán más ITBs. Nótese que el hecho de utilizar ambos algoritmos de encaminamiento al mismo tiempo no puede producir bloqueos en la red de interconexión ya que los dos algoritmos de encaminamiento están calculados sobre el mismo árbol y ambos insertan ITBs en las dependencias de canales *down-up*. La única diferencia es que UD_ITB inserta algunos ITBs adicionales.

Este nuevo algoritmo lo vamos a denominar UD_ITB_ML (*Up*/Down* with ITBs considering Message Length*). A su vez, también vamos a evaluar los algoritmos UD, UD_MITB y UD_ITB con tráfico bimodal para poder observar los beneficios que obtiene el algoritmo UD_ITB_ML. En la figura 91 podemos observar las prestaciones obtenidas por los algoritmos de encaminamiento UD, UD_MITB, UD_ITB y UD_ITB_ML al utilizarse tráfico bimodal en redes de 8, 16, 32 y 64 conmutadores. La distribución de destinos es uniforme.

Se puede observar como el algoritmo UD_ITB_ML obtiene prácticamente la misma productividad que UD_ITB, con la excepción en la red de 64 conmutadores donde obtiene una productividad ligeramente menor. El algoritmo UD_ITB_ML utiliza tanto rutas UD_MITB como UD_ITB, por lo que su productividad cae entre las dos productividades de UD_MITB y UD_ITB. En términos medios, la tabla 20 muestra los factores de incremento de productividad obtenidos por los algoritmos UD_MITB, UD_ITB y UD_ITB_ML sobre el algoritmo de encaminamiento UD. Podemos observar como, en términos medios, el algoritmo UD_ITB_ML obtiene una productividad intermedia entre UD_MITB y UD_ITB, siendo más cercana a UD_ITB.

Centrándonos en la reducción de latencia, en la tabla 21 podemos ver los porcentajes de incremento en latencia de los algoritmos de encaminamiento UD_MITB, UD_ITB y UD_ITB_ML con respecto a UD en condiciones de poco tráfico. Podemos observar como, el algoritmo UD_MITB obtiene una incremento en latencia menor del 3.29%. Esto es debido a que utiliza menos ITBs que el algoritmo UD_ITB, el cual obtiene unos incrementos en latencia mayores (incremento máximo del 12.17% en una red de 64 conmutadores). Con respecto al algoritmo UD_ITB_ML vemos que la sobrecarga en la latencia media de los mensajes es menor que la obtenida por el

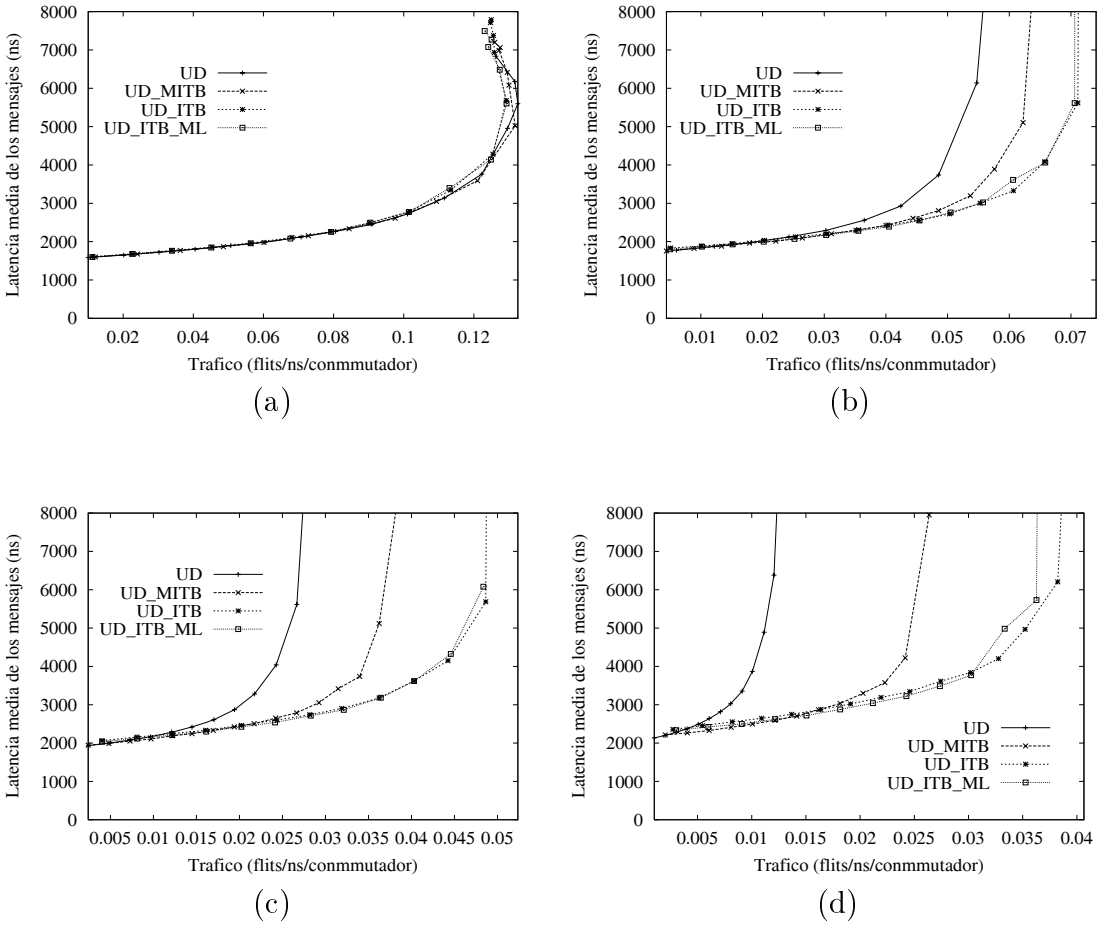


Figura 91: Selección de rutas en función del tamaño del mensaje. Latencia media vs. tráfico. Red de (a) 8, (b) 16, (c) 32 y (d) 64 conmutadores. Tráfico bimodal. Distribución uniforme de destinos.

algoritmo UD_ITB, aunque dicha disminución no es muy significativa.

Por consiguiente, podemos concluir que el mecanismo reduce ligeramente la latencia del mecanismo con respecto a UD_ITB, manteniendo prácticamente las productividades obtenidas por UD_ITB. Sin embargo, la productividad obtenida está muy condicionada por la tasa de mensajes cortos utilizados, ya que si el 100% del tráfico estuviera formado por mensajes cortos, entonces las prestaciones obtenidas en términos de productividad serían las mismas que las obtenidas por UD_MITB. Por lo tanto, debemos buscar otra solución que no dependa del tráfico que haya en la red.

	UD_MITB			UD_ITB			UD_ITB_ML		
Conm.	Mín	Máx	Media	Mín	Máx	Media	Mín	Máx	Media
8	0.96	1.08	1.01	0.93	1.21	1.01	0.95	1.21	1.02
16	1.05	1.29	1.17	1.20	1.47	1.33	1.11	1.46	1.31
32	1.36	2.03	1.59	1.66	2.70	2.10	1.65	2.72	2.07
64	1.99	2.40	2.11	2.70	3.50	2.94	2.59	3.31	2.87

Tabla 20: Selección de rutas en función del tamaño del mensaje. Factor de incremento de productividad al utilizar UD_MITB, UD_ITB y UD_ITB_ML respecto a UD. Tráfico bimodal. Distribución uniforme de destinos.

	UD_MITB			UD_ITB			UD_ITB_ML		
Conm.	Mín	Máx	Media	Mín	Máx	Media	Mín	Máx	Media
8	0.74	1.05	0.90	1.17	4.10	1.93	0.74	1.68	1.30
16	-1.58	0.64	-0.62	1.60	5.75	3.48	0.29	2.95	1.61
32	0.76	2.06	1.34	6.74	8.18	7.37	4.55	6.30	5.25
64	2.64	4.53	3.29	9.43	12.17	10.66	7.56	9.52	8.22

Tabla 21: Selección de rutas en función del tamaño del mensaje. Porcentaje de incremento de latencia al utilizar UD_MITB, UD_ITB y UD_ITB_ML respecto a UD. Tráfico bimodal. Distribución uniforme de destinos.

5.5.2 Selección de rutas en función de la latencia máxima permitida

En esta sección utilizamos un mecanismo de selección de rutas que seleccionará rutas con más o menos ITBs en función de la tasa de mensajes que queramos que a priori utilicen ITBs. En este mecanismo, un cierto número de mensajes se enviará en cada nodo por rutas UD (sin ITBs), mientras que el resto de los mensajes se enviará por rutas UD_ITB. Por ejemplo, fijaremos la tasa de mensajes a enviar por la ruta UD en un 50%. Con esta tasa debemos reducir el incremento en latencia de UD_ITB con respecto a UD en un 50%. El algoritmo resultante de utilizar ambas rutas lo denominamos UD_ITB_50. En la figura 92 podemos observar los resultados obtenidos para los algoritmos UD, UD_ITB y UD_ITB_50 en redes de 8, 16, 32 y 64 conmutadores para una distribución de destinos uniforme y tamaños de mensajes cortos de 32 bytes.

En la figura podemos observar como la latencia adicional de UD_ITB_50 respecto de UD es la mitad que la latencia adicional de UD_ITB respecto de UD. Sin embargo,

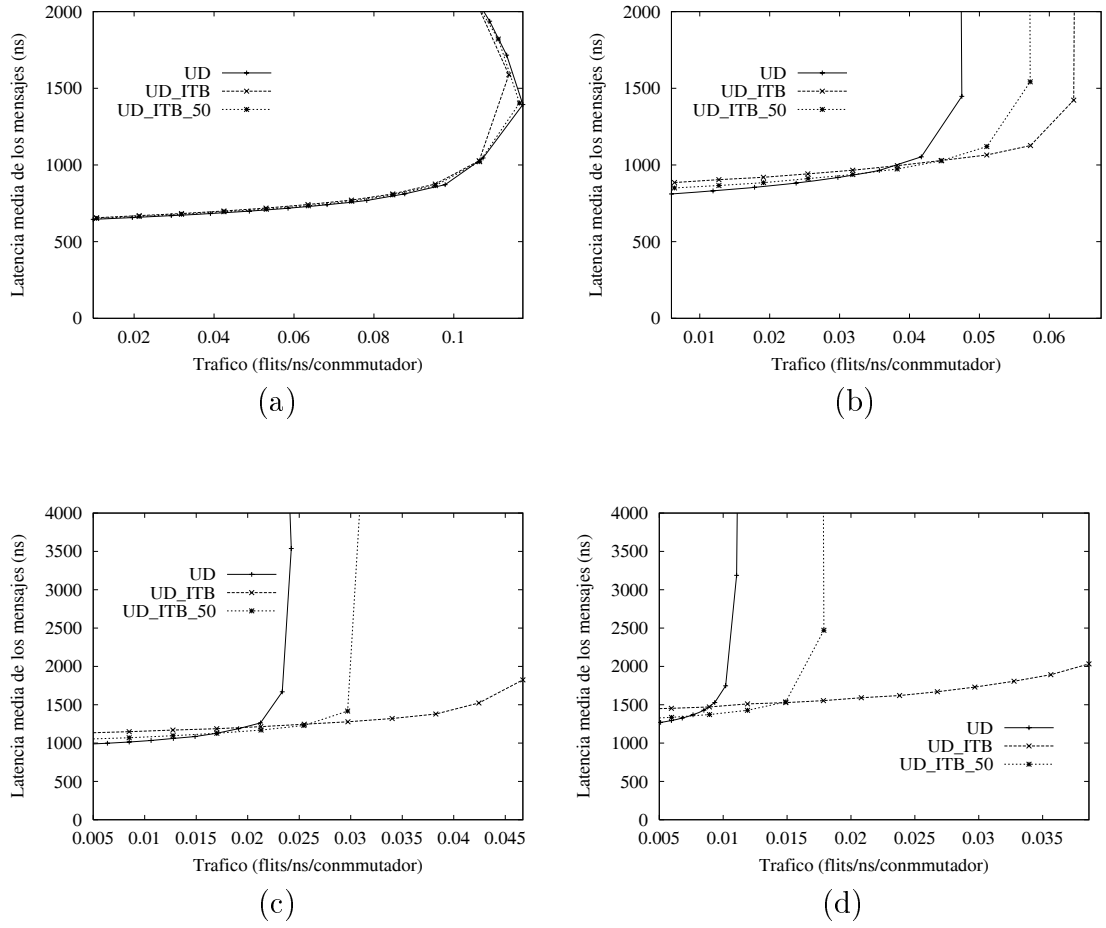


Figura 92: Selección de rutas en función del incremento máximo de latencia. Latencia media vs. tráfico. Red de (a) 8, (b) 16, (c) 32 y (d) 64 conmutadores. Distribución uniforme de destinos. Tamaño de mensajes de 32 bytes.

como podemos observar, la productividad se ve seriamente limitada por el algoritmo UD_ITB_50. Aunque aún mejora la productividad obtenida por UD, este incremento en productividad es sensiblemente menor que el obtenido por el algoritmo UD_ITB. Al utilizarse rutas exclusivamente UD, los efectos de reducción de contención y de mejor equilibrado de los ITBs se ve notablemente reducido por lo que la red se satura mucho antes.

En las tablas 22 y 23 podemos ver, respectivamente, los factores de incremento de productividad y los porcentajes de incremento de latencia al utilizar los algoritmos UD_ITB y UD_ITB_50 sobre UD. Como era de esperar, en términos medios, el

algoritmo UD_ITB_50 decrementa el incremento de latencia del algoritmo UD_ITB en un 50% ofreciendo un máximo de incremento de latencia del 10.91% para redes de 64 conmutadores. No obstante, en la tabla de factores de incremento de productividad podemos ver como la productividad media obtenida por UD_ITB_50 (factor de incremento de 1.66) es sensiblemente menor a la obtenida por UD_ITB (factor de incremento de 3.22).

	UD_ITB			UD_ITB_50		
Conm	Mín	Máx	Media	Mín	Máx	Media
8	0.96	1.36	1.03	0.96	1.22	1.02
16	1.20	1.65	1.39	1.08	1.31	1.19
32	1.69	2.88	2.30	1.33	1.60	1.49
64	2.84	3.76	3.22	1.53	1.75	1.66

Tabla 22: Selección de rutas en función del incremento máximo de latencia. Factor de incremento de productividad al utilizar UD_ITB y UD_ITB_50 respecto a UD. Distribución uniforme de destinos. Tamaño de mensajes de 32 bytes.

	UD_ITB			UD_ITB_50		
Conm	Mín	Máx	Media	Mín	Máx	Media
8	0.07	8.35	2.85	0.07	4.14	1.46
16	7.71	17.46	10.84	3.84	8.19	5.24
32	15.29	19.13	16.78	7.39	8.66	7.97
64	16.72	21.64	19.90	8.23	10.91	9.63

Tabla 23: Selección de rutas en función del incremento máximo de latencia. Porcentaje de incremento de latencia al utilizar UD_ITB y UD_ITB_50 respecto a UD. Distribución uniforme de destinos. Tamaño de mensajes de 32 bytes.

Por lo tanto, con este mecanismo de selección de rutas (UD_ITB_50) obtenemos efectivamente una reducción de la latencia del mecanismo, pero por contra limitamos notablemente la productividad del algoritmo UD_ITB. En definitiva, cuantos más ITBs se utilicen más productividad alcanzaremos a coste de una mayor penalización en la latencia. Por lo tanto, tampoco este mecanismo nos permite obtener unas cotas elevadas de productividad sin una excesiva penalización en latencia en baja carga.

5.5.3 Utilización de ITBs solamente con tráfico medio y alto

Como hemos visto en las secciones anteriores se ha conseguido reducir la latencia introducida por el mecanismo a costa de una perdida de prestaciones en términos de productividad. Estas soluciones trabajan con dos conjuntos de rutas: UD (o UD_MITB) y UD_ITB. Las rutas UD son buenas rutas para obtener reducidas latencias (ya que no utilizan el mecanismo ITB) pero, por contra, no obtienen elevadas productividades. Por otra parte, las rutas UD_ITB incrementan sustancialmente la productividad de la red pero, por contra, obtienen unas latencias mayores debido a la utilización del mecanismo ITB. El algoritmo ideal debería ser aquel que utilizara rutas UD sólo en condiciones de un tráfico reducido y que utilizara las rutas UD_ITB en condiciones de media y alta carga. En esta sección evaluamos la tercera y última propuesta para reducir la latencia extra introducida por utilizar los ITBs. También se utilizarán dos conjuntos de rutas: UD y UD_ITB. Las rutas UD se utilizarán en baja carga, y las rutas UD_ITB en media y alta carga. Para poder detectar la transición entre baja carga y media/alta carga utilizaremos el mecanismo de detección de contención utilizado anteriormente en la selección de rutas (ver sección 2.4).

Cada vez que se envía un mensaje por una ruta UD, se le asocia el coeficiente de contención C_c calculado de la misma forma que la presentada en la sección 2.4. En ausencia de contención, la ruta tendrá un coeficiente $C_c = 1$. Para las rutas UD_ITB vamos a utilizar una aproximación parecida. Cada vez que enviamos un mensaje por una ruta UD_ITB se asocia a la ruta el mismo coeficiente C_c pero sumándole una constante K. Esta constante permite forzar el uso de estas rutas sólo cuando las rutas UD empiecen a saturarse (tengan un ratio mayor de $1 + K$). Vamos a evaluar dos constantes. Una constante de $K = 1$ y una constante de $K = 2$. Con una constante $K = 1$ se utilizarán rutas UD_ITB cuando las rutas UD se encuentren una contención $C_c > 2$, o lo que es lo mismo, cuando el tiempo de inyección de un mensaje sea el doble que el tiempo de inyección sin contención. Por otra parte, con $K = 2$ se utilizarán las rutas UD_ITB cuando el tiempo de inyección por una ruta UD sea el triple del tiempo sin contención (cuando $C_c > 3$). Los algoritmos resultantes los denominamos UD_ITB_DET1 ($K = 1$) y UD_ITB_DET2 ($K = 2$), respectivamente.

En la figura 93 podemos ver los resultados obtenidos por los algoritmos de encaminamiento UD, UD_ITB, UD_ITB_DET1 y UD_ITB_DET2 en redes de 8, 16, 32 y 64 conmutadores para una distribución de destinos uniforme y con tamaños de

mensajes cortos (32 bytes).

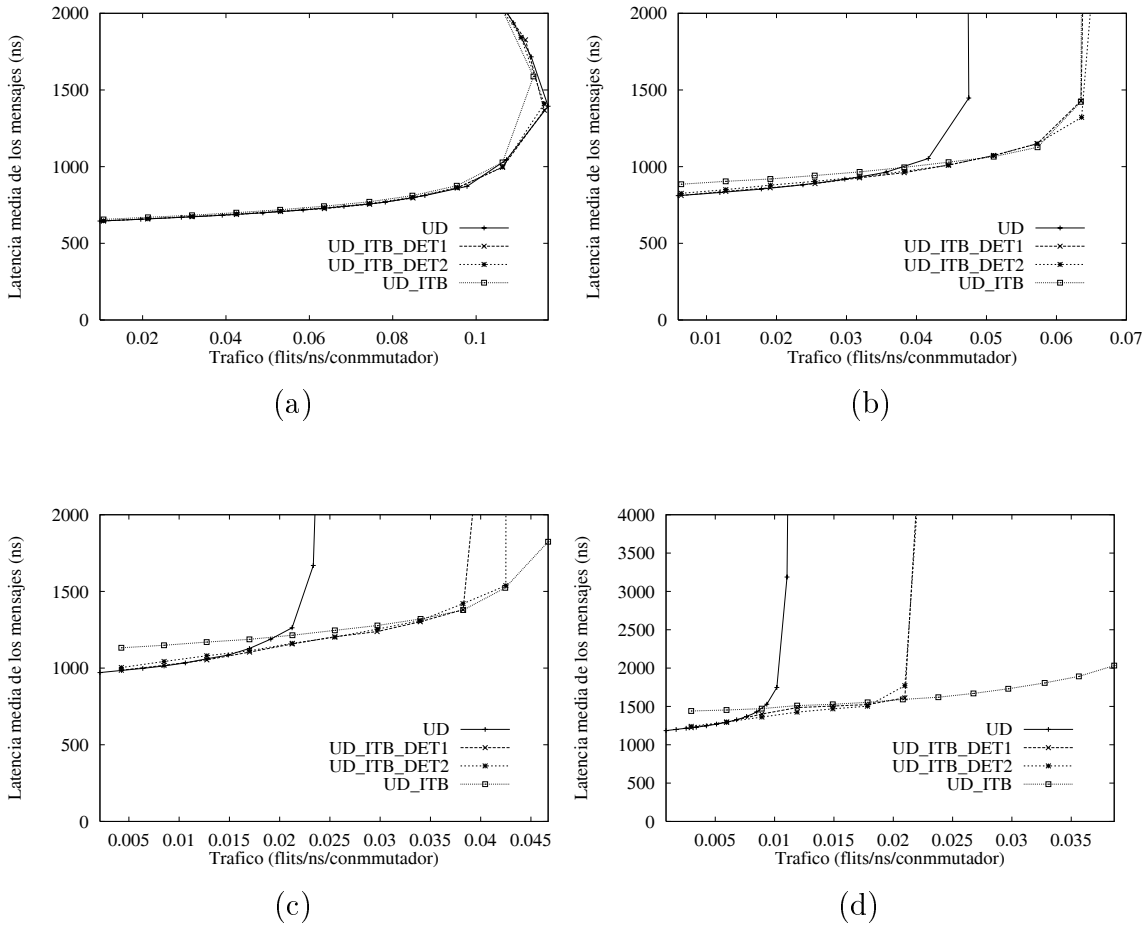


Figura 93: Utilización de ITBs solamente en tráfico medio y alto. Latencia media vs. tráfico. Red de (a) 8, (b) 16, (c) 32 y (d) 64 conmutadores. Distribución uniforme de destinos. Tamaño de mensajes de 32 bytes.

Como podemos apreciar, las latencias de los algoritmos de encaminamiento UD_ITB_DET1 y UD_ITB_DET2 en baja carga son las mismas que la obtenida por el algoritmo de encaminamiento UD. En este rango de funcionamiento, el mecanismo de detección de contención no detecta ninguna contención por lo que siempre se están utilizando rutas UD (sin ITBs). Podemos apreciar también, que la latencia del algoritmo UD_ITB_DET1 (y UD_ITB_DET2) evoluciona hacia la latencia del algoritmo UD_ITB al aumentar el nivel de tráfico. Esto es debido a que el mecanismo de detección de contención empieza a detectar contención en las rutas

UD por lo que las rutas UD_ITB del algoritmo UD_ITB_DET1 (y UD_ITB_DET2) empiezan a utilizarse. Por lo tanto, como vemos, el algoritmo UD_ITB_DET1 (y UD_ITB_DET2) no incrementan la latencia de los mensajes en baja carga. Por otra parte, como podemos observar, la productividad del algoritmo UD_ITB_DET1 (y UD_ITB_DET2) es prácticamente la misma que la productividad del algoritmo UD_ITB excepto en la red de 64 conmutadores donde la productividad alcanzada por el algoritmo UD_ITB_DET1 (y UD_ITB_DET2) es ostensiblemente menor (no obstante aún permite doblar la productividad de UD). Esto es debido a que en el rango elevado de tráfico al permitir que las rutas UD puedan volver a utilizarse por la constante de olvido, las rutas UD se vuelven a utilizar provocando la inmediata saturación instantánea de la red. Por último, cabe destacar que las diferencias entre UD_ITB_DET1 y UD_ITB_DET2 son mínimas en términos de productividad. En términos de latencia el algoritmo UD_ITB_DET1 obtiene un pequeño incremento en latencia en media carga con respecto a la latencia obtenida por el algoritmo UD_ITB_DET2. Esto es debido a que las rutas UD_ITB en UD_ITB_DET1 se utilizan antes debido a que tienen una constante K menor ($K = 1$).

Por lo tanto, con el mecanismo de detección de contención podemos implementar el mecanismo ITB de tal forma que no se utilice en condiciones de tráfico reducido donde sus efectos en latencia son significativos y que obtenga productividades elevadas.

5.5.4 Resumen

Por lo tanto, y como resumen, hemos propuesto diferentes modificaciones del mecanismo de buffers en tránsito para minimizar el efecto del incremento en latencia. En una primera aproximación hemos utilizado ITBs sin restricción sólo para mensajes largos. Con esta alternativa se reduce la latencia del mecanismo. Sin embargo, la efectividad de esta solución depende de la tasa de mensajes cortos que se utilicen. Como segunda alternativa se ha limitado el número de mensajes que utilizan ITBs, independientemente del tamaño de los mensajes. Con esta alternativa se reduce significativamente la latencia del mecanismo pero, por contra, se limitan los beneficios seriamente en términos de productividad. Como última alternativa, se utiliza el mecanismo de detección de contención con el fin de utilizar los ITBs solamente en rangos de tráfico medios o altos. Con esta alternativa se ha obtenido un incremento nulo de la latencia introducida por el mecanismo en baja carga, obteniéndose

resultados de productividad muy próximos a los obtenidos por UD_ITB (con la excepción de la red de 64 conmutadores). Por lo tanto, el inconveniente del incremento en latencia ocasionado por el mecanismo ITB puede ser tratado eficientemente al combinar el mecanismo ITB con el mecanismo de detección de contención.

5.6 Estudio de sensibilidad

En esta sección evaluamos el efecto de diferentes parámetros de diseño de la red sobre las prestaciones del mecanismo de buffers en tránsito. En primer lugar evaluamos la influencia de la rapidez del mecanismo sobre las prestaciones globales de la red. A continuación, evaluamos el efecto del mecanismo ITB sobre los mensajes locales y por último evaluamos la influencia del número de *hosts* conectados a cada conmutador.

5.6.1 Tiempo de detección y reprogramación

Como se ha comprobado, el mecanismo necesita de una implementación eficiente con el fin de minimizar el efecto del incremento en latencia. Los parámetros utilizados para modelar el mecanismo han sido extraídos de la red Myrinet por medio de pruebas reales. No obstante, puede ocurrir que en otros entornos (como por ejemplo otras redes que también puedan soportar el mecanismo de buffers en tránsito) estos parámetros no sean realistas y sean incluso mayores. Por este motivo, en esta sección evaluamos el impacto de los tiempos de detección y reprogramación del mecanismo ITB sobre las prestaciones. En concreto, vamos a evaluar el impacto que tiene la utilización de un mecanismo ITB que fuera 5 veces más lento que el evaluado hasta el momento sobre las prestaciones del algoritmo de encaminamiento UD_ITB . Para ello, ahora el tiempo de detección del mecanismo será de 1375 ns (T_d = 1375 ns) y el tiempo de reprogramación será de 1000 ns (T_r = 1000 ns).

En la figura 94 podemos ver los resultados obtenidos en una red de 32 conmutadores por el algoritmo UD_ITB utilizando un mecanismo ITB con los parámetros T_d = 275 ns y T_r = 200 ns (denominado en la gráfica UD_ITB) y utilizando un mecanismo ITB con parámetros T_d = 1375 ns y T_r = 1000 ns (denominado en la gráfica UD_ITB_5x). Con el fin de comparar resultados, se muestra también las prestaciones del algoritmo de encaminamiento UD. La distribución de destinos es uniforme y el tamaño de mensajes es de 32 bytes (figura 94.a) y 512 bytes (figura 94.b).

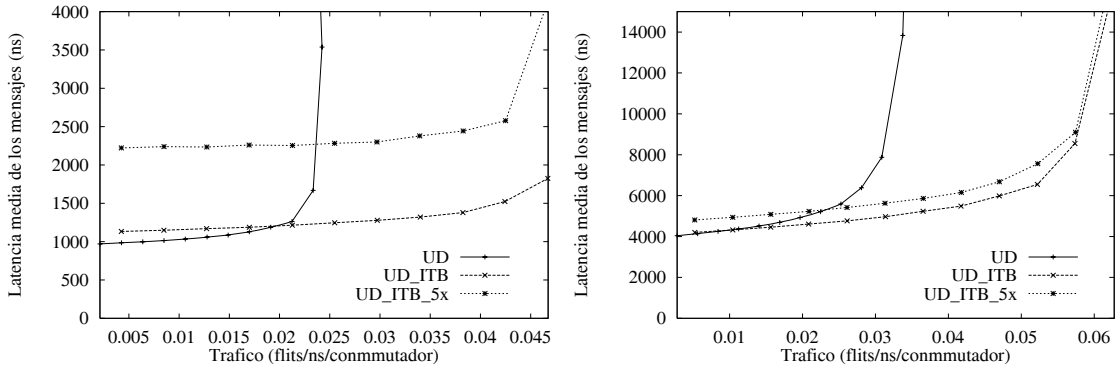


Figura 94: Tiempo de detección y reprogramación. Latencia media vs. tráfico. Red de 32 conmutadores. Tamaño de mensajes de (a) 32 bytes y (b) 512 bytes. Distribución uniforme de destinos.

Como podemos observar, el incremento en latencia provocado por un mecanismo 5 veces más lento es importante, especialmente significativo al utilizar mensajes cortos para los cuales la latencia es prácticamente duplicada. Por otra parte, la productividad, no se ve influenciada por la rapidez del mecanismo, como podemos observar. El algoritmo UD_ITB_5x obtiene prácticamente la misma productividad que UD_ITB. Esto es debido a que el mecanismo, aunque sea 5 veces más lento, sigue teniendo las mismas propiedades, permitiendo el equilibrado del tráfico, la reducción de la contención y la utilización de rutas mínimas. Por lo tanto, la eficiencia (rapidez) del mecanismo influye solamente en la componente de latencia añadida a los mensajes, mientras que la componente de productividad no se ve afectada.

De todas formas, este incremento tan importante en la latencia podría ser suavizado aplicando algunos de los métodos evaluados anteriormente de reducción de la latencia.

5.6.2 Efecto del mecanismo ITB sobre los mensajes locales

El mecanismo ITB se ejecuta en el interfaz de red, donde hay mensajes locales (del propio *host*) que compiten con los mensajes en tránsito por el canal de salida. Por lo tanto, un uso indiscriminado del mecanismo en un determinado nodo podría conllevar cierta penalización para los mensajes locales. No obstante, se ha utilizado

un mecanismo de prioridades con el fin de minimizar dicha penalización (ver sección 3.1.4) así como una selección del *nodo en tránsito* a cada ruta con necesidad de ITB en un conmutador (ver sección 3.1.3).

Sin embargo, vamos a evaluar en esta sección cómo afecta el mecanismo a los mensajes locales. En concreto, vamos a evaluar el número de mensajes locales pendientes de ser inyectados en cada nodo de la red para diferentes algoritmos de encaminamiento en diferentes puntos de tráfico.

Cabe observar que la posible penalización del mecanismo sobre los mensajes locales será tanto mayor conforme aumente el tráfico de la red. Al existir un mayor tráfico de red, fluirán más mensajes en tránsito a un determinado nodo, y por lo tanto, los mensajes locales tendrán una mayor dificultad en salir a la red. Es por ello que evaluamos el efecto del mecanismo ITB sobre los mensajes locales solamente en puntos cercanos a la saturación.

En concreto evaluamos los algoritmos UD, UD_ITB, BMIN_ITB y BMAX_ITB en una red de 32 conmutadores. En la figura 95[5] podemos observar las prestaciones obtenidas por los algoritmos de encaminamiento en dicha red con una distribución uniforme de destinos y con mensajes de 32 bytes[6]. Podemos observar como cada algoritmo se satura en puntos de tráfico diferente, siendo UD el algoritmo de encaminamiento que se satura con una menor tasa de inyección de tráfico.

En la figura 96 podemos observar el número de mensajes locales pendientes de ser inyectados a la red en cada nodo (o *nodo en tránsito*) de la red para diferentes puntos de tráfico (cercanos a saturación). En concreto, en la figura 96.a podemos ver como el algoritmo UD, cuando entra en saturación (tasa de inyección de 0.024 flits/ns/conmutador), empieza a tener mensajes locales encolados en determinados nodos. Ya en plena saturación (inyección de tráfico de 0.026 y 0.027 flits/ns/conmutador), el número de mensajes locales crece hasta llegar a 2500 mensajes encolados en determinados nodos. Esta penalización sufrida por los mensajes locales en UD no es debida al mecanismo, ya que UD no lo utiliza. La penalización viene ocasionada por la elevada contención que existe en la red para los puntos de

[5]Hay que hacer constar que en esta configuración de red y tráfico, el algoritmo de encaminamiento UD_ITB presenta un comportamiento distinto al de los demás algoritmos de encaminamiento al no alcanzar la saturación. Esto es debido a que en tasas de inyección de tráfico superiores a 0.05 flits/ns/conmutador se produce un desbordamiento de los buffers en tránsito en un determinado nodo. Es por ello que en la gráfica no se muestran resultados del algoritmo UD_ITB con tasas de inyección de tráfico superiores a 0.05 flits/ns/conmutador.

[6]Similares resultados se han obtenido para mensajes de 512 bytes.

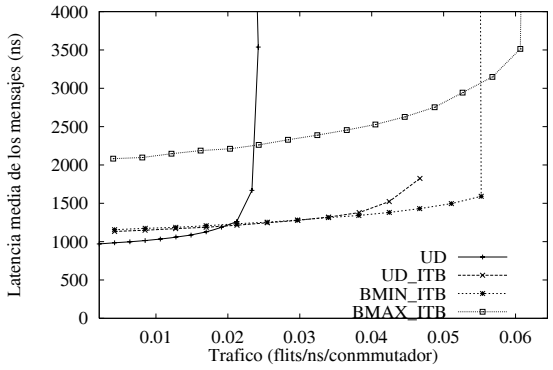


Figura 95: Efecto sobre los mensajes locales. Tráfico vs latencia. Red de 32 conmutadores. Tamaño de mensajes de 32 bytes. Distribución uniforme de destinos.

tráfico evaluados.

En la figura 96.b podemos observar los mensajes locales pendientes de envío para diferentes puntos de tráfico al utilizar el mecanismo ITB sobre *up*/down** (UD_ITB). Podemos observar como, en puntos muy cercanos a la saturación de UD_ITB (tasa de inyección de 0.047 flits/ns/conmutador), algunos nodos empiezan a tener problemas de inyección de mensajes locales. No obstante, el número máximo de mensajes locales encolados es 11 y solamente en 2 nodos, por lo que en este punto de tráfico, los mensajes locales no tienen prácticamente ningún problema de inyección. Por lo tanto, en esta red el uso del mecanismo de buffers en tránsito en el algoritmo UD_ITB no presenta ningún problema a los mensajes locales en cualquier punto de tráfico.

Por último, en las figuras 96.c y 96.d podemos observar los mensajes locales pendientes de envío para los algoritmos BMIN_ITB y BMAX_ITB, respectivamente, en tasas de inyección de tráfico superiores a los anteriores y cercanos a la saturación de estos algoritmos. Podemos observar como, para el algoritmo BMIN_ITB, los mensajes sufren penalización solamente en puntos de tráfico dentro de saturación (inyección de tráfico superior a 0.055 flits/ns/conmutador). Asimismo, al utilizar BMAX_ITB, los mensajes locales sufren una ligera penalización pero solamente en el punto de saturación de BMAX_ITB (inyección de tráfico de 0.064 flits/ns/conmutador). Como podemos observar, dicha penalización supone la acumulación de unos 50 mensajes en 4 nodos.

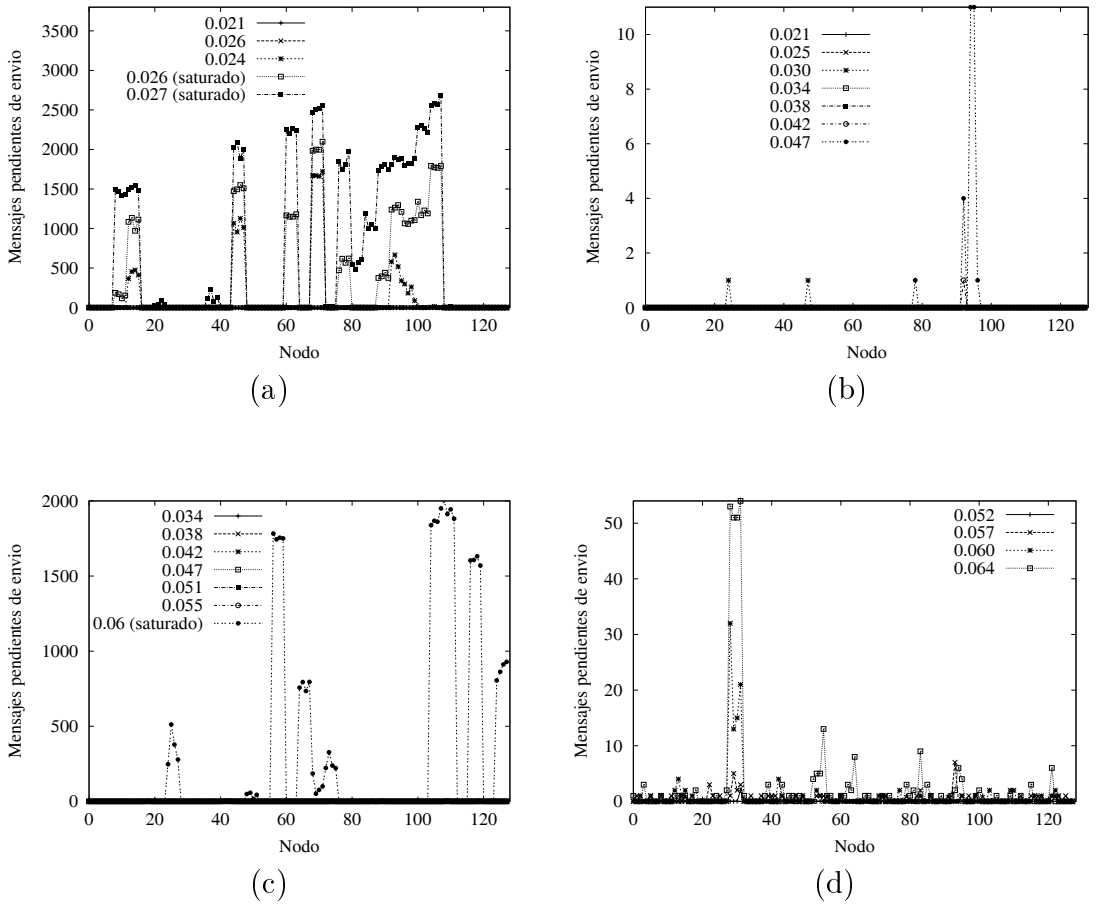


Figura 96: Efecto sobre los mensajes locales. Mensajes locales pendientes de envío. (a) UD, (b) UD_ITB, (c) BMIN_ITB y (d) BMAX_ITB. Red de 32 conmutadores. Tamaño de mensajes de 32 bytes. Distribución uniforme de destinos.

Por lo tanto, y como conclusión, cabe decir que el mecanismo de buffers en tránsito no afecta a los mensajes locales en tasas de inyección de tráfico por debajo de la saturación de la red y tan sólo repercute negativamente en tasas de inyección de tráfico superiores a la productividad máxima alcanzable por los nuevos algoritmos de encaminamiento que utilizan ITBs. Esto nos lleva a la conclusión de que las políticas de selección de *nodo en tránsito* y de prioridades entre mensajes locales y mensajes en tránsito comentadas en las secciones 3.1.3 y 3.1.4, respectivamente, son suficientes para garantizar a los mensajes locales de un uso equitativo del canal de salida junto a los mensajes en tránsito.

Cabe comentar que en todas las evaluaciones realizadas en el presente trabajo, se ha limitado la memoria destinada a mensajes locales y a mensajes en tránsito, por lo que en ningún caso de los mostrados en el presente trabajo se ha producido ningún desbordamiento.

5.6.3 Efecto del número de *hosts* conectados a cada conmutador

En las evaluaciones realizadas del mecanismo ITB se han utilizado topologías irregulares generadas de forma aleatoria. No obstante, en todas las topologías evaluadas se ha mantenido constante el número de *hosts* conectados a cada conmutador y el número de enlaces entre conmutadores. Ambos parámetros han sido fijados a 4. De esta asignación se deduce que el mecanismo ITB no supone un cuello de botella en la red ya que cada conmutador puede recibir hasta 4 flujos de mensajes con necesidad de ITBs por sus cuatro canales de entrada de otros conmutadores y va a disponer de 4 *hosts* para poder hacer uso del mecanismo ITB (*nodos en tránsito*). Por lo tanto, con una asignación de *hosts* a rutas equitativa, el mecanismo ITB no debería suponer un cuello de botella. No obstante, esta configuración con el mismo número de *hosts* y enlaces a conmutadores en cada conmutador puede que no se de en determinadas redes.

Es por ello que, en este apartado, evaluamos el mecanismo ITB en redes con un número de *hosts* menor al número de enlaces entre conmutadores. Con ello obtenemos las prestaciones del mecanismo en una situación más desfavorable, en la cual el número de flujos de mensajes es superior al número de *hosts* que facilitan la utilización de ITBs. Para evaluar este aspecto, vamos a asociar sólo dos *hosts* a cada conmutador, manteniendo los cuatro enlaces entre conmutadores. El estudio lo realizamos sobre una red de 32 conmutadores, con lo que habrán 64 *hosts* en el sistema. Evaluamos los algoritmos de encaminamiento BMIN_ITB, B33_ITB, B50_ITB, B66_ITB y BMAX_ITB, así como UD.

En la figura 97 podemos observar los resultados obtenidos por los algoritmos de encaminamiento en la red de 32 conmutadores en los dos casos, cuando el número de *hosts* por conmutador es igual al número de enlaces entre conmutadores (figuras 97.a y 97.c) y cuando el número de *hosts* es la mitad del número de enlaces, esto es, 2 *hosts* y 4 enlaces por conmutador (figuras 97.b y 97.d). Se muestran resultados para tamaños de mensajes de 32 bytes (figuras 97.a y 97.b) y para tamaños de mensajes

de 512 bytes (figuras 97.c y 97.d).

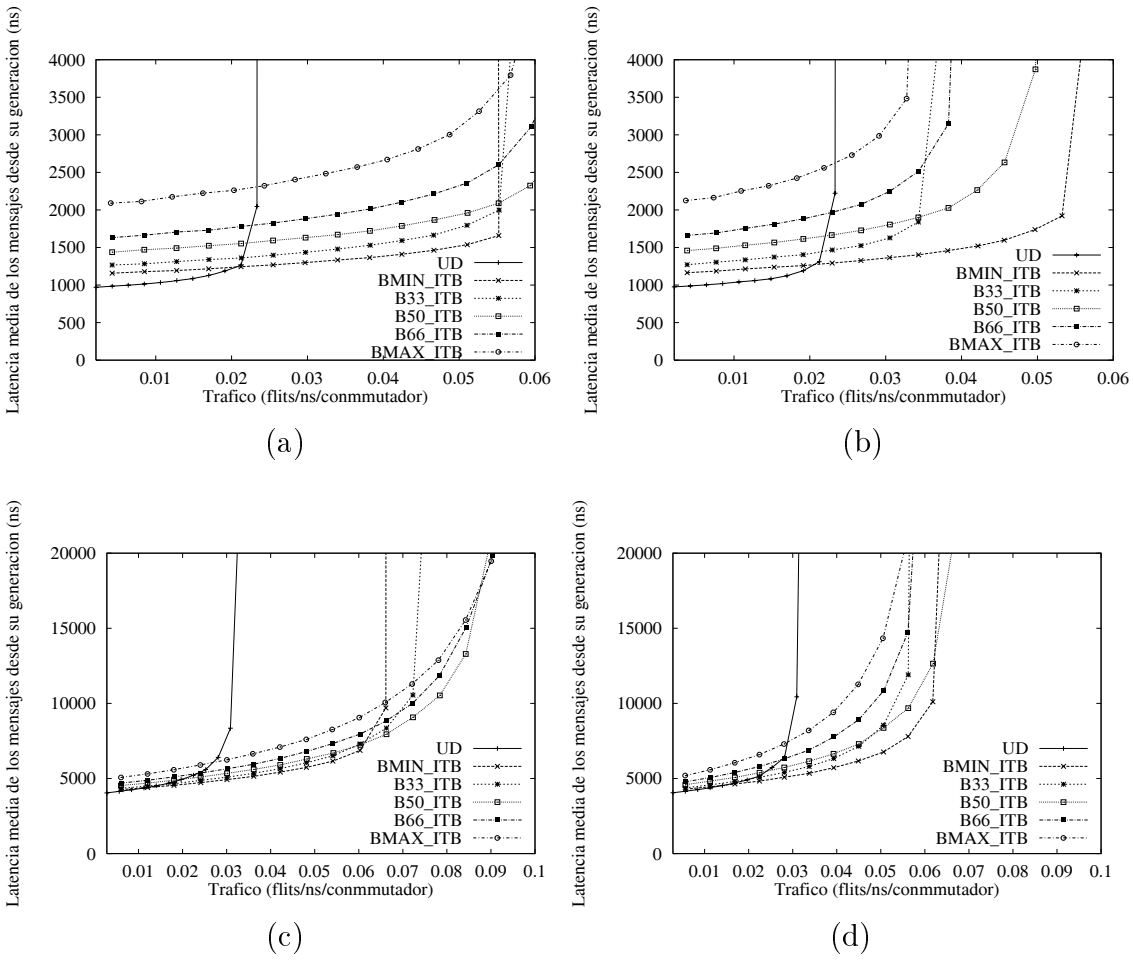


Figura 97: Efecto del número de *hosts* conectados a cada conmutador. (a, c) 4 *hosts* por conmutador y (b, d) 2 *hosts* por conmutador. Distribución uniforme de destinos. Tamaño de mensaje es (a, b) 32 bytes y (c, d) 512 bytes.

Como podemos observar, cuando hay solamente dos *hosts* por conmutador, las prestaciones de los algoritmos de encaminamiento que utilizan ITBs decrecen con respecto a la situación donde el número de *hosts* es igual al número de enlaces entre conmutadores. Podemos observar como en los algoritmos que utilizan más ITBs (BMAX_ITB, B66_ITB y B50_ITB) la disminución de productividad es más acusada. En concreto, se observa como las prestaciones son peores conforme utilizamos más ITBs (al contrario que los resultados obtenidos cuando el número de *hosts* es igual al número de enlaces). Esto es lógico, ya que la red se satura en estos casos

debido a la formación de cuellos de botella en los interfaces de red de los *hosts* que alojan ITBs (cuantos más ITBs tengamos, más cuello de botella será el mecanismo). Sin embargo, podemos apreciar también como el algoritmo de encaminamiento BMIN_ITB se comporta prácticamente igual en los dos casos (y con los dos tamaños de mensajes). Este algoritmo utiliza el mínimo número de ITBs por lo que el efecto de cuello de botella es menor y realmente la red se satura por congestión.

Por lo tanto, el porcentaje de ITBs a utilizar debe estar en proporción con el número de *hosts* disponibles en cada conmutador y con el número de enlaces entre conmutadores.

Capítulo 6

Conclusiones, aportaciones y desarrollos futuros

En esta tesis se han propuesto diversos mecanismos con el fin de mejorar las prestaciones de las redes con encaminamiento fuente. Para ello, se han identificado los diferentes inconvenientes que presentan las redes con encaminamiento fuente y conmutación *wormhole*. Los inconvenientes detectados son:

- **Elevada utilización de rutas no mínimas**. Debido a las restricciones impuestas en el encaminamiento, los algoritmos de encaminamiento aplicables a las redes irregulares con encaminamiento fuente, tales como *up*/down** y *DFS* tienden a la utilización de rutas no mínimas, obteniendo valores de productividad de red relativamente bajos.

- **Desequilibrado del tráfico**. Los algoritmos de encaminamiento *up*/down** y *DFS* tienden a desequilibrar el tráfico debido al algoritmo utilizado de construcción de rutas basado en árbol. La mayor parte del tráfico atraviesa las proximidades del nodo raíz.

- **Elevada contención de red**. Al utilizarse conmutación *wormhole* sin canales virtuales, la red se satura por completo con un nivel de tráfico medio, ya que el bloqueo de algunos mensajes propaga dicho bloqueo a otros mensajes.

6.1 Aportaciones

Una vez identificados los inconvenientes, en la presente tesis se han propuesto diferentes mecanismos con el fin de eliminar o reducir dichos inconvenientes. Estos mecanismos se han propuesto considerando una implementación sencilla sobre las redes con encaminamiento fuente actuales (en especial sobre Myrinet), sin modificación alguna del *hardware*. En concreto, dichos mecanismos se han propuesto como implementaciones *software* en los interfaces de red de Myrinet. Los mecanismos propuestos en la presente tesis son:

- **Utilización de diferentes rutas alternativas para un mismo par origen-destino**. Típicamente, las redes utilizan solamente una ruta para cada par origen-destino. Basándose en el hecho de que pueden existir diferentes rutas válidas para cada par origen-destino, se presentan diferentes algoritmos de selección de rutas con el fin de obtener un mejor equilibrado del tráfico.

- **Utilización de un mecanismo de detección de contención para obtener un mejor equilibrado**. Se propone un mecanismo que, en función únicamente de información local al nodo (el tiempo de inyección de un mensaje por una ruta), obtiene una estimación de la contención a lo largo de dicha ruta. Por lo tanto, se utiliza el mecanismo de detección de contención para seleccionar la ruta a utilizar de entre las diferentes rutas alternativas a un mismo destino con el fin de evitar ciertas zonas congestionadas en la red y así poder equilibrar mejor el tráfico.

- **Utilización de un mecanismo que elimina las restricciones de encaminamiento de los algoritmos tradicionales (*buffers en tránsito*, ITB)**. Con este mecanismo, un mensaje es extraído de la red en un nodo determinado para posteriormente ser reinyectado hacia el destino final (u otro *nodo en tránsito*). Dicho mecanismo será utilizado para eliminar las restricciones de encaminamiento de los algoritmos de encaminamiento *up*/down**, *DFS* y *smart-routing*. Al eliminar dichas restricciones de encaminamiento, se permite la utilización de rutas mínimas para todo par origen-destino. Asimismo, se permite un mejor equilibrado del tráfico ya que, al eliminar las restricciones de encaminamiento de los algoritmos, menos rutas van a pasar por las proximidades del nodo raíz (*up*/down** y *DFS*), por lo que se equilibrará mejor el

tráfico. Al aplicar el mecanismo a *smart-routing* se permite la obtención de rutas *smart* equilibradas, pero calculadas en un tiempo menor que las rutas originales de *smart-routing*. Finalmente, el mecanismo ITB ayuda a reducir notablemente la contención de red. Por otra parte, se ha propuesto un nuevo algoritmo de encaminamiento que explota todas estas posibilidades del mecanismo ITB al tiempo que permite calcular las rutas en tiempos razonables.

- **Utilización del mecanismo ITB junto al mecanismo de detección de contención, para minimizar el efecto del incremento en la latencia de los mensajes por parte del mecanismo ITB**. Al extraerse temporalmente los mensajes de la red, se incurre en un incremento de la latencia de dichos mensajes. Dado que este efecto es especialmente importante con cargas bajas, se han propuesto varios métodos para reducir esta penalización, siendo el más apropiado aquel que combina el mecanismo de detección de contención junto con el de buffers en tránsito, de manera que sólo se utilice el mecanismo ITB con cargas de red medias y altas.

6.2 Conclusiones

Dichos mecanismos, y en especial el mecanismo ITB, han sido evaluados sobre diferentes topologías, tamaños de mensajes y patrones de tráfico. De dichas evaluaciones se obtienen las siguientes conclusiones:

- **La utilización de rutas alternativas no incrementa significativamente las prestaciones de las redes irregulares**. Esto es debido a que dichas rutas alternativas tienden a utilizar los mismos canales, comportándose, por tanto, como una única ruta. Este efecto es más notable conforme aumenta el tamaño de la red. En las evaluaciones se ha comprobado que la mejora en productividad es menor conforme aumenta el tamaño de la red.

  La utilización de un mecanismo de detección de contención para seleccionar la ruta a utilizar tampoco mejora significativamente las prestaciones en redes irregulares.

- **La utilización del mecanismo ITB sobre los algoritmos de encaminamiento *up*/down** y *DFS* permite incrementar significativamente la productividad de las redes**. Más aún, estas mejoras en productividad

son mayores conforme aumenta el tamaño de la red. Los algoritmos de encaminamiento *up*/down** y *DFS* son muy restrictivos, tendiendo a utilizar más rutas no mínimas y desequilibrar más el tráfico conforme aumenta el tamaño de la red. Por otra parte, al eliminarse las restricciones de encaminamiento con el mecanismo ITB, se permite utilizar siempre rutas mínimas y, sobre todo, mantener un buen equilibrado de tráfico. En particular, el incremento medio de productividad al utilizar el mecanismo ITB frente a *up*/down**, varía en redes irregulares desde un factor de 1.4 para redes medianas de 16 conmutadores hasta triplicar las prestaciones en algunas redes de 64 conmutadores. Con respecto a *DFS*, el incremento en productividad varía, en términos medios, desde un factor de 1.05 en redes de 16 conmutadores hasta un factor de 1.58 en redes de 64 conmutadores. En redes regulares, el mecanismo ITB dobla la productividad obtenida por *up*/down**.

- **La utilización del mecanismo ITB sobre el algoritmo de encaminamiento *smart-routing* también permite aumentar la productividad de la red**. Aunque el algoritmo *smart-routing* obtiene un buen equilibrado de tráfico, la utilización de ITBs reduce la contención de red obtenida por *smart-routing*, por lo que se obtienen unas mayores productividades. En particular, la productividad de *smart-routing* es incrementada por el mecanismo ITB en un factor de 1.26 en redes de 32 conmutadores.

- **El nuevo algoritmo de encaminamiento basado exclusivamente en la utilización de ITBs mejora significativamente cualquier propuesta anterior (con o sin ITBs)**. El algoritmo obtiene un equilibrado de carga bastante bueno, comparable al de *smart-routing*, con la diferencia de que el tiempo de cálculo de las rutas del nuevo algoritmo es alcanzable, a diferencia del tiempo de cálculo empleado por *smart-routing*. Con respecto a la contención, el nuevo algoritmo de encaminamiento la reduce significativamente, pudiendo emplearse diferentes cantidades de ITBs. En particular, la productividad de *smart-routing* es incrementada por dicho algoritmo en un factor de 1.78 para redes de 32 conmutadores.

- **La penalización sobre la latencia en la que incurre el mecanismo ITB es significativa solamente para mensajes cortos y en rangos de tráfico reducidos**. Incluso, al evaluarse un tráfico bimodal con un pequeño porcentaje

de mensajes largos, dicha penalización es reducida significativamente.

- **La utilización del mecanismo de detección de contención propuesto junto con el mecanismo ITB elimina la penalización de la latencia del mecanismo, al permitir la utilización de ITBs a partir de cierto nivel de tráfico**. Sin embargo, la productividad alcanzada por el algoritmo resultante se ve ligeramente reducida en redes grandes, pero sigue siendo mucho mayor que la productividad obtenida por cualquier algoritmo de encaminamiento sin ITBs.

- **Se han propuesto las líneas maestras para obtener una implementación eficiente del mecanismo de buffers en tránsito sobre Myrinet**, siendo el objetivo principal el minimizar la sobrecarga del mecanismo sobre los mensajes que lo utilicen. De hecho, se ha implementado satisfactoriamente el mecanismo de buffers en tránsito sobre el *software* de comunicaciones GM de Myricom.

6.3 Trabajos publicados relacionados directamente con la tesis

Versiones preliminares de este trabajo han sido publicadas (o están pendientes de publicación) en los *Proceedings* de varios congresos:

- J. Flich, M.P. Malumbres, P. López, J. Duato, and R. Felderman, "Using Minimal Routing in Myrinet Networks," *X Jornadas de Paralelismo*, Septiembre 1999.

- J. Flich, M.P. Malumbres, P. López, and J. Duato, "Improving Routing Performance in Myrinet Networks," *International Parallel and Distributed Processing Symposium*, IEEE Computer Society Press, May 2000.

- J. Flich, M.P. Malumbres, P. López, and J. Duato, "In-Transit Buffers: A Mechanism to Support Minimal Routing in Myrinet," *IEEE Technical Committe on Computer Architecture Newsletter*, IEEE Computer Society Press, June 2000.

En estos tres artículos se propone la utilización de rutas alternativas junto con diferentes políticas de selección de rutas, así como la propuesta del mecanismo de buffers en tránsito para garantizar ruta mínima en el algoritmo de encaminamiento *up*/down** así como una primera evaluación del mismo.

- J. Flich, M.P. Malumbres, P. López, and J. Duato, "Performance Evaluation of a New Routing Strategy for Irregular Networks with Source Routing," *International Conference on Supercomputing*, IEEE Computer Society Press, May 2000.

- J. Flich, P. López, M.P. Malumbres, and J. Duato, "Improving the Performance of Regular Networks with Source Routing," *International Conference on Parallel Processing*, IEEE Computer Society Press, August 2000.

En estos dos artículos se realiza un profundo análisis del mecanismo de buffers en tránsito aplicado a *up*/down** en redes irregulares y redes regulares, bajo diferentes patrones de tráfico y diferentes tamaños de mensajes, observando con detalle el efecto beneficioso de los ITBs sobre el equilibrado del tráfico en la red.

- J. Flich, P. López, M.P. Malumbres, J. Duato, and T. Rokicki, "Combining In-Transit Buffers with Optimized Routing Schemes to Boost the Performance of Networks with Source Routing," *International Symposium on High Performance Computing*, Lecture Notes in Computer Science, Springer Verlag, October 2000.

- J. Flich, P. López, M.P. Malumbres, and J. Duato, "In-Transit Buffers: A Mechanism to Improve Performance in Networks of Workstations", *XI Jornadas de Paralelismo*, Septiembre 2000.

En estos dos artículos se presenta una evaluación del mecanismo ITB sobre los algoritmos *DFS* y *smart-routing*, analizando con detalle las tres ventajas del mecanismo: obtención de rutas mínimas, equilibrado del tráfico y reducción en la contención.

- J. Flich, P. López, M.P. Malumbres, J. Duato, and T. Rokicki, "Improving Network Performance by Reducing Network Contention in Source-Based COWs with a Low Path-Computation Overhead," aceptado para publicar en *International Parallel and Distributed Processing Symposium*, IEEE Computer Society Press, April 2001.

En este artículo se presenta el nuevo algoritmo de encaminamiento que explota todas las posibilidades del mecanismo de buffers en tránsito, equilibrando la carga, reduciendo la contención y manteniendo el tiempo de cálculo de rutas bajo límites razonables.

- S. Coll, J. Flich, M.P. Malumbres, P. López, J. Duato, and F.J. Mora, "A First Implementation of In-Transit Buffers on Myrinet GM Software," aceptado para publicar en *Communication Architecture for Clusters*, April 2001.

En este artículo se presenta una primera implementación (y su evaluación) del mecanismo de buffers en tránsito en Myrinet sobre el software de comunicaciones GM de Myricom.

6.4 Trabajo futuro

Como trabajo futuro, se plantean las siguientes actividades:

- *Implementación del mecanismo de buffers en tránsito sobre una red real.* Una primera línea de trabajo inmediata es la implementación definitiva del mecanismo de buffers en tránsito sobre Myrinet. De hecho, ya disponemos de una primera implementación. Dicha implementación final permitirá evaluar las prestaciones obtenidas por los ITBs en un entorno real.

- *Ubicación estratégica de los ITBs.* Los buffers en tránsito reducen la contención de la red a la vez que pueden incrementar la latencia media de los mensajes. Para minimizar este impacto en la latencia, cabe pensar que una ubicación estratégica de los ITBs ayudaría a incrementar las prestaciones pero con un incremento mínimo en la latencia media de los mensajes. Por ejemplo, la ubicación de ITBs en aquellos conmutadores en los que diferentes rutas entran por diferentes canales de entrada y salen por el mismo canal de salida (Y's) supondría aliviar la contención en dicho punto.

- *Nuevas modificaciones al mecanismo ITB para minimizar aún más el incremento en latencia.* Al utilizar dos conjuntos de rutas (unas sin ITBs y otras con ITBs) junto con el mecanismo de detección de contención, hemos eliminado la latencia en condiciones de poco tráfico. Sin embargo, la productividad se ve

disminuida en redes grandes. No obstante pueden existir otras aproximaciones que mejoren este comportamiento. Por ejemplo:

- *Utilizar tres conjuntos de rutas.* Cada conjunto de rutas utilizará un cantidad diferente de ITBs. Para cambiar de conjunto de rutas se utilizarán dos umbrales (un primer umbral para pasar del primer conjunto de rutas al segundo y un segundo umbral superior al primero para pasar del segundo conjunto de rutas al tercero). Con esta solución, las rutas sin ITBs serán las del primer conjunto. Rutas con pocos ITBs se ubicarán en el segundo conjunto y rutas con muchos ITBs se ubicarán en el tercer conjunto de rutas. Por lo tanto, con esta solución, en condiciones, de elevado tráfico, se utilizarán rutas con muchos ITBs y volverán a entrar en juego (debido a la constante de olvido) las rutas del segundo conjunto (que tienen unos pocos ITBs) por lo que las rutas del primer conjunto (que antes saturaban la red) no se utilizarán en condiciones de elevado tráfico.
- *Utilizar ITBs con mensajes largos y en función del mecanismo de detección de contención con mensajes cortos.* Se ha visto que la latencia media de los mensajes largos al utilizar ITBs no es significativa. Por otra parte, para mensajes cortos el incremento es mayor. Con el mecanismo de selección de rutas con ITBs en función del tamaño del mensaje se obtienen buenas prestaciones (tanto en latencia como en productividad) cuando existe un elevado porcentaje de mensajes largos. Sin embargo, esta solución es dependiente del porcentaje de mensajes largos. Por lo tanto, la idea propuesta como trabajo futuro es utilizar rutas con ITBs para mensajes largos y utilizar un mecanismo ya evaluado para el caso de que los mensajes sean cortos. En este caso se puede seleccionar el mecanismo de detección de contención para decidir ante un mensaje corto si se utilizan o no rutas con ITBs.
- *Inserción y eliminación dinámica de ITBs.* Junto con el mecanismo de detección de contención se puede rediseñar el mecanismo ITB para que conforme aumente la contención a lo largo de la ruta, se vayan insertando más ITBs en esta ruta con el fin de que la contención se reduzca. Con esta solución se puede evolucionar dinámicamente desde un algoritmo con pocos ITBs hasta un algoritmo con muchos ITBs (al aumentar el tráfico

en la red) y viceversa (al disminuir el tráfico por la red).

- *Utilización del mecanismo de buffers en tránsito en otros tipos de redes.* Por ejemplo, analizar la viabilidad del mecanismo en la recientemente propuesta red InfiniBand [IBA] y, en su caso, realizar la evaluación de prestaciones.

- *Utilización de los ITBs en procesos de reconfiguración.* El mecanismo de buffers en tránsito puede ser una buena solución para simplificar la sofisticación en los algoritmos de reconfiguración. De hecho, su utilización puede ser una buena opción para implementar algoritmos sencillos de reconfiguración dinámica en redes con encaminamiento fuente (como Myrinet).

Apéndice A

Implementación de ITBs en GM

En este apéndice se describen los cambios realizados[1] sobre el *software* de comunicaciones estándar GM de Myricom para el soporte del mecanismo de buffers en tránsito (ITB). Dicha implementación ha sido evaluada habiéndose publicado en [Col01]. El propósito último de este apéndice no es el describir con detalle el funcionamiento de GM sino el de indicar los cambios necesarios para una implementación eficiente del mecanismo. Por lo tanto, para un mejor entendimiento del código indicado en este apéndice se recomienda un análisis previo del código GM.

Todos los cambios realizados se encuentran identificados por una macro de compilación condicional denominada *GM_ENABLE_ITB*. A continuación se muestran los listados de los ficheros modificados en la versión 1.2pre16 de GM.

A.1 Fichero ./mcp/gm_types.h

Para soportar el mecanismo sobre GM se definen ciertas estructuras y tipos de datos. En concreto se define la longitud máxima de la ruta y el tipo de mensaje ITB[2]. También se define la estructura que albergará un mensaje de tipo ITB (*gm_itb_packet*). Por último, se crean tres eventos adicionales para el tratamiento de los mensajes ITB. En concreto, se crea el evento *EARLY_RECV_PACKET_EVENT* que indica la recepción de un paquete ITB que se intentará reenviar inmediatamente, el evento *SEND_ITB_EVENT* para indicar que existe un mensaje ITB que ha tenido que esperar porque el DMA de envío estaba ocupado y que tiene que reenviarse lo más

[1] Esta implementación ha sido realizada principalmente por Salvador Coll.

[2] Este tipo de mensaje ha sido reservado por Myricom.

pronto posible y un evento *FINISH_SEND_ITB_EVENT* para indicar la finalización del envío de un mensaje ITB.

```
/**********************************************************************
 * Macros
 **********************************************************************/

/* Misc. nonconfigurable constants. */

#define GM_NULL  0
#define GM_TRUE  1
#define GM_FALSE  0
#define GM_MTU 4096
#define GM_MIN_MESSAGE_SIZE 3

#if GM_ENABLE_ITB
#define MAX_HOP_LEN 32
#endif

.... (code not shown) ...

#if GM_ENABLE_ITB
    GM_ITB_PACKET_TYPE = 0x0320,
#endif

.... (code not shown) ...

#if GM_ENABLE_ITB
gm_itb_packet_t as_itb;
#endif

.... (code not shown) ...


/**********************************************************************
 * ITB types
 **********************************************************************/
#if GM_ENABLE_ITB
```

```
typedef struct gm_itb_packet
  {
    GM_PACKET_TYPE_16 (type);
    gm_u16_t length;
    /* Reserve room for ITB hops, payload, gm_header (16) */
    char as_bytes[MAX_HOP_LEN+GM_MTU+16];
  }
gm_itb_packet_t;
#endif


.... (code not shown) ...


/****************
 * event indices.
 ****************/

/* This definition is here only to define NUM_EVENT_TYPES, which is used
   to define the gm_lanai_globals structure below. */

enum gm_event_index
{
/* POLLing events */
  POLL_EVENT = 0,
/* SDMA events */
  START_SDMA_EVENT,
  FINISH_SDMA_EVENT,
/* SEND events */
  START_SEND_EVENT,
  FINISH_SEND_EVENT,
  SEND_ACK_EVENT,
/* RECV events */
  START_RECV_PACKET_EVENT,
  FINISH_RECV_PACKET_EVENT,
  RECV_BUFFER_OVERFLOW_EVENT,
/* RDMA events */
```

```
  START_RDMA_EVENT,
  FINISH_RDMA_EVENT,
/* TIMER events */
  TIMER_EVENT,
/* FAIRness events */
  FAIR_RDMA_EVENT,
  FAIR_SDMA_EVENT,
  FAIR_SDMA_RDMA_EVENT,

#if GM_ENABLE_ITB
  SEND_ITB_EVENT,   /* ITB packet events */
  FINISH_SEND_ITB_EVENT,
  EARLY_RECV_PACKET_EVENT,<cr>
#endif

/* number of events */
  NUM_EVENT_TYPES
};
```

A.2 Fichero ./mcp/gm_recv.h

En este fichero se encuentra el código que se ejecuta ante la recepción de un nuevo mensaje. La parte más importante se encuentra definida en la macro *MARK_LABEL (L_recv__early_recv_packet_, LZERO)*. Allí se ubica el código para la detección del mensaje ITB así como su reinyección (programando el dispositivo DMA) si procede.

```
/* Handle receiving a message that fits */
/* entirely within a chunk. */
MARK_LABEL (L_recv__got_chunk_, LZERO);
{
  const gm_packet_header_t *p;
  char *rmp;
  int bytes_received;
  GM_CRC_TYPE crc;
#if GM_ENABLE_CRC32
```

```
  gm_u8_t *crc32;
#endif
  void *_RMP;
  gm_u32_t isr;
  enum gm_packet_type p__type;
  enum gm_packet_subtype p__subtype;
  unsigned int p__sender_node_id;
  unsigned int gm_max_node_id;

  GM_LOG_EVT(GM_GOT_CHUNK);
  gm_assert_p (NOTICED (FREE_RECV_CHUNK));
  gm_assert_p (NOTICED (RECEIVING));
  gm_assert_p (get_ISR() & RECV_INT_BIT);
  ASSERT_HANDLER (FINISH_RECV_PACKET_EVENT, L_recv__got_chunk_, LZERO);
  ASSERT_HANDLER (RECV_BUFFER_OVERFLOW_EVENT, L_recv__discard_overflow_,
  LZERO);

  gm_puts ("gm_recv: received a packet.\n");
  GM_INCR_PACKET_CNT (gm.netrecv_cnt);


  isr = get_ISR ();
  /* pre */ _RMP = RMP;
  p = &gm.recv_chunk[LZERO].packet.as_gm.header;
  p__type = p->type;
  rmp = (char *)_RMP - OVERRUN(isr) - sizeof (GM_CRC_TYPE);


  /* ITB */
#if GM_ENABLE_ITB

  if (p__type==GM_ITB_PACKET_TYPE)
  { /* It has to be resent asap*/
  void *smp;
  void *sml;
  gm_itb_packet_t *itbp;
  gm_packet_t    *gmp;
```

```
  enum gm_packet_subtype subtipo;
  enum gm_packet_type tipo;


  itbp = (gm_itb_packet_t *) p;

#if DEBUG_ITB
gmp= (gm_packet_t *)((itbp->as_bytes) + (itbp->length));
        subtipo = gmp->header.subtype;
tipo = gmp->header.type;
printf("RECV paquete ITB entero (tipo=\%x,subtipo=\%x).LZERO=\%d\n",
  gm_htons(tipo),gm_htons(subtipo),LZERO);
fflush(stdout);
#endif

         /* prepare next receive */
        if (--gm.free_recv_chunk_cnt)
          {
            RMW = gm.recv_chunk[LONE].packet.as_bytes ;
            RMP = gm.recv_chunk[LONE].packet.as_bytes;
            set_RML (&gm.recv_chunk[LONE].end);

            SET_HANDLER (FINISH_RECV_PACKET_EVENT,
          L_recv__got_chunk_,LONE);
            SET_HANDLER (RECV_BUFFER_OVERFLOW_EVENT,
          L_recv__discard_overflow_,LONE);
            SET_HANDLER (EARLY_RECV_PACKET_EVENT,
          L_recv__early_recv_packet_,LONE );

          }
        else
          {
            NOTICE_NOT (FREE_RECV_CHUNK | RECEIVING);
            SET_HANDLER (START_RECV_PACKET_EVENT,
          L_recv__start_receiving_chunk_, LONE);
          }
```

```
  DISPATCH (905 , "ITB packet received, prepared next receive");


  } /* from (p__type==GM_ITB_PACKET_TYPE)*/
#endif


...(code not shown)...


MARK_LABEL (L_recv__discard_overflow_, LZERO);
{
  gm_assert_p (NOTICED (FREE_RECV_CHUNK));
  gm_assert_p (NOTICED (RECEIVING));
  SET_HANDLER (FINISH_RECV_PACKET_EVENT, L_recv__done_discarding_overflow_,
       LZERO);
  ASSERT_HANDLER (RECV_BUFFER_OVERFLOW_EVENT,L_recv__discard_overflow_, LZERO);

#if GM_ENABLE_ITB
  RMW = gm.recv_chunk[LZERO].packet.as_bytes ;
  SET_HANDLER (EARLY_RECV_PACKET_EVENT,L_recv__early_recv_packet_,LZERO );
#endif
  RMP = gm.recv_chunk[LZERO].packet.as_bytes;
  set_RML (&gm.recv_chunk[LZERO].end);
  gm_printf_p ("Discarded a chunk.\n");
  DISPATCH (66, "starting to drop overlarge recv (buff_int)");
}

MARK_LABEL (L_recv__done_discarding_overflow_, LZERO);
{
  gm_assert_p (NOTICED (FREE_RECV_CHUNK));
  gm_assert_p (NOTICED (RECEIVING));
  ASSERT_HANDLER (FINISH_RECV_PACKET_EVENT, L_recv__done_discarding_overflow_,
  LZERO);
  ASSERT_HANDLER (RECV_BUFFER_OVERFLOW_EVENT,L_recv__discard_overflow_, LZERO);
```

```
#if GM_ENABLE_ITB
  RMW = gm.recv_chunk[LZERO].packet.as_bytes ;
  SET_HANDLER (EARLY_RECV_PACKET_EVENT,L_recv__early_recv_packet_,LZERO );
#endif
  RMP = gm.recv_chunk[LZERO].packet.as_bytes;
  set_RML (&gm.recv_chunk[LZERO].end);

  SET_HANDLER (FINISH_RECV_PACKET_EVENT, L_recv__got_chunk_, LZERO);
  gm_printf_p ("Discarded last chunk.\n");
  DISPATCH (67, "done dropping overlarge recv");
}

MARK_LABEL (L_recv__start_receiving_chunk_, LZERO);
{
  gm_assert_p (NOTICED (FREE_RECV_CHUNK));
  gm_assert_p (NOTICED_NOT (RECEIVING));

#if GM_ENABLE_ITB
  RMW = gm.recv_chunk[LZERO].packet.as_bytes ;
  SET_HANDLER (EARLY_RECV_PACKET_EVENT,L_recv__early_recv_packet_,LZERO );
#endif

  RMP = gm.recv_chunk[LZERO].packet.as_bytes;
  set_RML (&gm.recv_chunk[LZERO].end);

  SET_HANDLER (FINISH_RECV_PACKET_EVENT, L_recv__got_chunk_, LZERO);
  SET_HANDLER (RECV_BUFFER_OVERFLOW_EVENT, L_recv__discard_overflow_, LZERO);

  NOTICE (RECEIVING);
  gm_printf_p ("Set up next recv.\n");
  DISPATCH (68, "set up recv (no longer out of recv chunks)");
}


#if GM_ENABLE_ITB
MARK_LABEL (L_recv__early_recv_packet_, LZERO);
{
```

```
  register const gm_packet_header_t *p;
  register const gm_itb_packet_t *itbp asm ("r6");
  enum gm_packet_type itbp__type;
  gm_u32_t isr;


  itbp = &gm.recv_chunk[LZERO].packet.as_itb;
  isr = get_ISR();
  itbp__type = itbp->type;


  if (itbp__type==GM_ITB_PACKET_TYPE)
  { /* It has to be resent asap*/
  register void *smp asm ("r9");
  register void *sml asm ("r16");
  register gm_packet_t *gmp asm ("r10");
  register gm_u16_t longitud;

#if DEBUG_ITB
  enum gm_packet_subtype subtipo;
  enum gm_packet_type tipo;
#endif


  if ( (isr & SEND_INT_BIT)
   && NOTICED_NOT (ITB_PACKET_SENDING)
   && NOTICED_NOT (ITB_PACKET_PENDING)
   && NOTICED_NOT (SENDING) ) /* send machine is free */
   {


#if DEBUG_ITB
   //printf("as_bytes: \%d\n",itbp->as_bytes);
gmp= (gm_packet_t *)((itbp->as_bytes) + (itbp->length));
        //printf("gmp: \%d\n",gmp);
subtipo = gmp->header.subtype;
```

```
tipo = gmp->header.type;
printf("+++++ Early RECV (tipo=\%x,subtipo=\%x) :envio inmediato.LZERO=\%d\n",
       gm_htons(tipo),gm_htons(subtipo),LZERO);
        //printf("Longitud ruta ITB = \%d\n",itbp->length);
//print_state();
   fflush(stdout);
#endif

longitud = itbp->length;
   /* tres soluciones equivalentes */
   //smp = gm.recv_chunk[LZERO].packet.as_itb.as_bytes;
   //smp = ((void *)(itbp) + 4);
   smp = (void *) (itbp->as_bytes);

/* points to the beginning of the gm packet header */

/* dos soluciones equivalentes */
   //gmp = (gm_packet_t *)(itbp->as_bytes+itbp->length);
   gmp = (gm_packet_t *)(smp+longitud);

   SA = (int) smp;

   SMP = smp;

   /* sml is loaded this way in sdma.h */
   sml = gmp->payload + gmp->header.length;

   set_SMLT (sml);


   NOTICE (ITB_PACKET_SENDING);
   /* Evita que vuelva a entrar */
   /* Writing "1" in HEAD_INT_BIT resets it */
   set_ISR(HEAD_INT_BIT);

SET_HANDLER (FINISH_SEND_ITB_EVENT,L_send__finish_sending_itb_packet_,LZERO);
DISPATCH (901 , "ITB packet inmediately sent");
```

```
}
  else /* send machine is busy */
  /* we must activate an event for resending the packet asap */
   {

#if DEBUG_ITB
  itbp = (gm_itb_packet_t *) p;
     gmp = (gm_packet_t *)((itbp->as_bytes) + (itbp->length));
          subtipo = gmp->header.subtype;
  tipo = gmp->header.type;
  printf("+++++Early RECV (tipo=\%x,subtipo=\%x) :envio pendiente.LZERO=\%d\n",
         gm_htons(tipo),gm_htons(subtipo),LZERO);
          //print_state();
     fflush(stdout);
#endif
     gm.itb_chunk_cnt++;
  NOTICE (ITB_PACKET_PENDING);
  /* Evita que vuelva a entrar */
     /* Writing "1" in HEAD_INT_BIT resets it */
  set_ISR(HEAD_INT_BIT);
  SET_HANDLER (SEND_ITB_EVENT,L_send__send_itb_packet_,LZERO);
  DISPATCH (902 , "ITB packet will be sent afterwards");

   }

  } /* from (p__type==GM_ITB_PACKET_TYPE)*/
  else
{  set_ISR(HEAD_INT_BIT);

#if DEBUG_ITB
 p__subtype = p->subtype;
    printf ("No es ITB, tratamiento normal. Tipo: \%d, Subtipo: \%d\n",
         p__type,p__subtype);
    print_state();
    fflush(stdout);
#endif
```

```
 /* Evita que vuelva a entrar */
    /* Writing "1" in HEAD_INT_BIT resets it */

 DISPATCH (906, "It wasn't an ITB packet");
}

}
#endif
```

A.3 Fichero ./mcp/gm_sdma.h

En este fichero se definen macros que intentan enviar un mensaje lo antes posible ante la liberación del canal de salida. Estas macros han sido instrumentadas para incluir los eventos *ITB_PACKET_PENDING* y *ITB_PACKET_SENDING* así como código adicional necesario.

```
#if GM_ENABLE_ITB

#if GM_ENABLE_SHORTCUT

#define SHORTCUT_IF_POSSIBLE(dispatch) do { \
  gm_connection_t *c; \
  gm_send_record_t *sr, *sr__next; \
  gm_packet_header_t *p; \
  unsigned int c__route_len, this_node_id; \
  gm_s32_t rtc, sexno; \
  gm_s16_t seqno; \
  gm_subport_t *sp, *sp__next; \
  void *start_sdma_handler; \
  gm_u32_t isr; \
\
  if (NOTICED (SDMAING \
       | ITB_PACKET_SENDING \
       | ITB_PACKET_PENDING \
```

```
       | RDMAING \
       | SDMA_PENDING \
       | SENDING \
       | SEND_PENDING)) \
    { \
      break; \
    } \
\
  /******** \
   * Shortcut for the idle case: (HACK) \
   ********/ \
\
  log_it (254, __FILE__, __LINE__, 0);                              \
  start_sdma_handler = GET_HANDLER (START_SDMA_EVENT); \
  c = &gm.connection[target_node_id]; \
  if (start_sdma_handler == && L_sdma__start_sdma_0) \
    { \
      p = &gm.send_chunk[0].packet.as_gm.header; \
      SET_HANDLER (START_SDMA_EVENT, L_sdma__start_sdma_, 1); \
    } \
  else \
    { \
      p = &gm.send_chunk[1].packet.as_gm.header; \
      SET_HANDLER (START_SDMA_EVENT, L_sdma__start_sdma_, 0); \
    } \
\
  /**** \
   * Build the packet \
   ****/ \
\
  /* start DMA */ \
\
  gm_assert_p (send_len); \
  USER_SDMA_NO_CONTINUATION (message, p+1,\
     send_len + GM_DMA_GRANULARITY-1,\
     GM_SUBPORT_PORT (sp_id)); \
  sexno = c->send_sexno.whole; \
```

```
  /* pre */ this_node_id = gm.this_node_id; \
\
  /* build packet header */ \
\
  copy_route (c->route, ((char *) p - GM_MAX_NETWORK_DIAMETER)); \
  p->type = GM_PACKET_TYPE; \
  p->subtype = (GM_RELIABLE_DATA_SUBTYPE_0 + size); \
  p->target_node_id = target_node_id; \
  p->sender_node_id = this_node_id; \
  p->sexno.whole = sexno; \
  p->length = send_len; \
  /* pre */ c__route_len = c->route_len; \
  p->target_subport_id = target_subport_id; \
  p->sender_subport_id = sp_id; \
  gm_galvantech_set_header_checksum (p); \
  GM_GALVANTECH_SET_IP_CHECKSUM (p, st->reliable.ip_checksum); \
\
  /**** \
   * enqueue the send token and log the send \
   ****/ \
\
  /* commit to using the send token. */ \
\
  /* pre */ sp = gm.free_subports; \
  gm_send_token_commit (port, st); \
  gm_assert_p (sp); \
  st->reliable.subport = sp; \
  st->reliable.send_len = 0; \
\
  /* build send record */ \
\
  sr = gm.free_send_records; \
  gm_assert_p (sr); /* since idle */ \
  /* pre */ rtc = RTC; \
  gm_assert_p (port->active_subport_cnt == 0); \
  port->active_subport_cnt = 1; \
  sr__next = sr->next; \
```

```
  sr->next = 0; \
  sr->send_token = st; \
  sr->before_ptr = message; \
  sr->sexno.whole = sexno; \
  sr->before_len = send_len; \
  sr->resend_time = rtc; \
  gm.free_send_records = sr__next; \
\
  /* append the send token to the send queue */ \
\
  GM_INCR_DEBUG_CNT (gm.sends_in_send_queue_cnt); \
  gm_assert_p (!gm.first_active_connection); \
  gm.first_active_connection = c; \
\
  seqno = c->send_sexno.parts.seqno; \
  gm_assert_p (c->first_send_record == 0); \
  c->next_active = c; \
  c->prev_active = c; \
  c->send_sexno.parts.seqno = seqno + 1; \
  c->active_subport_bitmask = 1 << sp_id; \
  c->first_send_record = sr; \
  c->last_send_record = sr; \
  c->first_active_send_port = sp; \
  c->known_alive_time = rtc; \
\
  /* init the subport */ \
\
  sp__next = sp->next; \
  sp->next = sp; \
  sp->prev = sp; \
  /* post */ gm.free_subports = sp__next; \
  sp->connection = c; \
  sp->first_send_token = st; \
  sp->last_send_token = st; \
  gm_assert_p (st->common.next == 0); \
  sp->delay_until = rtc; \
  sp->id = sp_id; \
```

```
  sp->disabled = 0; \
  sp->progress_time = rtc; \
  incr_subport_cnt (); \
\
  /* wait for the small packet DMA to complete */ \
\
  await_free_DMA_engine (); \
  isr = get_ISR (); \
  NOTICE (SEND_PENDING + SDMA_PENDING); \
  gm_assert_p (gm.free_send_chunk_cnt == 2); \
  gm.free_send_chunk_cnt = 1; \
\
  /**** \
   * send the packet, or arrange for send \
   ****/ \
\
  NOTICE (SEND_PENDING); \
  if (isr & SEND_INT_BIT) \
    { \
      void *start_send_handler, *smp, *smlt; \
\
      /* ITB scoll */    \
      /*printf("Enviado desde sdma\\n");*/  \
      /*fflush(stdout);*/  \
      smp = (char *) p - c__route_len; \
      smlt = (char *) (p+1) + send_len; \
      SA = (gm_u32_t) smp; \
      SMP = smp; \
      set_SMH (((char *) p) - 1); \
      set_SMLT (smlt); \
      start_send_handler = GET_HANDLER (START_SEND_EVENT); \
      NOTICE (SENDING); \
      if (start_send_handler == && L_send__start_sending_chunk_0) \
{ \
  SET_HANDLER (START_SEND_EVENT,\
       L_send__start_sending_chunk_, 1); \
} \
```

```
      else \
{ \
  gm_assert_p (start_send_handler \
       == &&L_send__start_sending_chunk_1); \
  SET_HANDLER (START_SEND_EVENT,\
       L_send__start_sending_chunk_, 0); \
} \
    } \
  dispatch; \
} while (0)

#else /* not GM_ENABLE_SHORTCUT */

#define SHORTCUT_IF_POSSIBLE(dispatch)

#endif /* not GM_ENABLE_SHORTCUT */

#else /* not GM_ENABLE_ITB */

... (code not shown) ...


#if GM_ENABLE_ITB

#if GM_ENABLE_SHORTCUT && GM_ENABLE_DATAGRAMS

#define DATAGRAM_SHORTCUT_IF_POSSIBLE(dispatch) do { \
  gm_connection_t *c; \
  gm_packet_header_t *p; \
  unsigned int c__route_len, this_node_id; \
  gm_s32_t rtc; \
  gm_subport_t *sp, *sp__next; \
  void *start_sdma_handler; \
  gm_u32_t isr; \
\
  if (NOTICED (SDMAING \
       | ITB_PACKET_PENDING \
```

```
       | ITB_PACKET_SENDING \
       | RDMAING \
       | SDMA_PENDING \
       | SENDING \
       | SEND_PENDING)) \
    { \
      break; \
    } \
\
  /******** \
   * Shortcut for the idle case: (HACK) \
   ********/ \
\
  start_sdma_handler = GET_HANDLER (START_SDMA_EVENT); \
  c = &gm.connection[target_node_id]; \
  if (start_sdma_handler == && L_sdma__start_sdma_0) \
    { \
      p = &gm.send_chunk[0].packet.as_gm.header; \
      SET_HANDLER (START_SDMA_EVENT, L_sdma__start_sdma_, 1); \
    } \
  else \
    { \
      p = &gm.send_chunk[1].packet.as_gm.header; \
      SET_HANDLER (START_SDMA_EVENT, L_sdma__start_sdma_, 0); \
    } \
\
  /**** \
   * Build the packet \
   ****/ \
\
  /* start DMA */ \
\
  gm_assert_p (send_len); \
  USER_SDMA_NO_CONTINUATION (message, p+1,\
     send_len + GM_DMA_GRANULARITY-1,\
     GM_SUBPORT_PORT (sp_id)); \
  /* pre */ this_node_id = gm.this_node_id; \
```

```
\
  /* build packet header */ \
\
  copy_route (c->route, ((char *) p - GM_MAX_NETWORK_DIAMETER)); \
  p->type = GM_PACKET_TYPE; \
  p->subtype = (GM_DATAGRAM_SUBTYPE_0 + size); \
  p->target_node_id = target_node_id; \
  p->sender_node_id = this_node_id; \
  p->length = send_len; \
  /* pre */ c__route_len = c->route_len; \
  p->target_subport_id = target_subport_id; \
  p->sender_subport_id = sp_id; \
\
  /* wait for the small packet DMA to complete */ \
\
  await_free_DMA_engine (); \
\
  /**** \
   * send the packet, or arrange for send \
   ****/ \
\
  gm_assert_p (gm.free_send_chunk_cnt == 2); \
  gm.free_send_chunk_cnt = 1; \
  isr = get_ISR (); \
  NOTICE (SEND_PENDING); \
  if (isr & SEND_INT_BIT) \
    { \
      void *start_send_handler, *smp, *smlt; \
\
      /* ITB */  \
      /*printf("Enviado desde sdma\\n");*/  \
      /*fflush(stdout);*/  \
      smp = (char *) p - c__route_len; \
      smlt = (char *) (p+1) + send_len; \
      SA = (gm_u32_t) smp; \
      SMP = smp; \
      set_SMH (((char *) p) - 1);                                          \
```

```
      set_SMLT (smlt); \
      start_send_handler = GET_HANDLER (START_SEND_EVENT); \
      NOTICE (SENDING); \
      if (start_send_handler == && L_send__start_sending_chunk_0) \
{ \
  SET_HANDLER (START_SEND_EVENT,\
       L_send__start_sending_chunk_, 1); \
} \
      else \
{ \
  gm_assert_p (start_send_handler \
       == &&L_send__start_sending_chunk_1); \
  SET_HANDLER (START_SEND_EVENT,\
       L_send__start_sending_chunk_, 0); \
} \
    } \
\
  /* report successful send completion */ \
   \
  gm_recycle_first_send_token (port, st); \
   \
  dispatch; \
} while (0)

#if GM_ENABLE_PIO_DATAGRAMS

#define PIO_DATAGRAM_SHORTCUT_IF_POSSIBLE(dispatch) do { \
  gm_connection_t *c; \
  gm_packet_header_t *p; \
  unsigned int c__route_len, this_node_id; \
  gm_s32_t rtc; \
  gm_subport_t *sp, *sp__next; \
  void *start_sdma_handler; \
  gm_u32_t isr; \
\
  if (NOTICED (SDMAING \
          | ITB_PACKET_PENDING \
```

```
          | ITB_PACKET_SENDING \
       | RDMAING \
       | SDMA_PENDING \
       | SENDING \
       | SEND_PENDING)) \
    { \
      break; \
    } \
\
  /******** \
   * Shortcut for the idle case: (HACK) \
   ********/ \
\
  start_sdma_handler = GET_HANDLER (START_SDMA_EVENT); \
  c = &gm.connection[target_node_id]; \
  if (start_sdma_handler == && L_sdma__start_sdma_0) \
    { \
      p = &gm.send_chunk[0].packet.as_gm.header; \
      SET_HANDLER (START_SDMA_EVENT, L_sdma__start_sdma_, 1); \
    } \
  else \
    { \
      p = &gm.send_chunk[1].packet.as_gm.header; \
      SET_HANDLER (START_SDMA_EVENT, L_sdma__start_sdma_, 0); \
    } \
\
  /**** \
   * Build the packet \
   ****/ \
\
  /* pre */ this_node_id = gm.this_node_id; \
\
  /* build packet header */ \
\
  copy_route (c->route, ((char *) p - GM_MAX_NETWORK_DIAMETER)); \
  p->type = GM_PACKET_TYPE; \
  p->subtype = (GM_DATAGRAM_SUBTYPE_0 + size); \
```

```
  p->target_node_id = target_node_id; \
  p->sender_node_id = this_node_id; \
  p->length = send_len; \
  /* pre */ c__route_len = c->route_len; \
  p->target_subport_id = target_subport_id; \
  p->sender_subport_id = sp_id; \
  *(gm_u32_t *)(p+1) = data; \
\
  /**** \
   * send the packet, or arrange for send \
   ****/ \
\
  gm_assert_p (gm.free_send_chunk_cnt == 2); \
  gm.free_send_chunk_cnt = 1; \
  isr = get_ISR (); \
  NOTICE (SEND_PENDING); \
  if (isr & SEND_INT_BIT) \
    { \
      void *start_send_handler, *smp, *smlt; \
      /* ITB */   \
      /*printf("Enviado chunk desde sdma\\n");*/  \
      /*fflush(stdout);*/                          \
      smp = (char *) p - c__route_len; \
      smlt = (char *) (p+1) + send_len; \
      SA = (gm_u32_t) smp; \
      SMP = smp; \
      set_SMLT (smlt); \
      start_send_handler = GET_HANDLER (START_SEND_EVENT); \
      NOTICE (SENDING); \
      if (start_send_handler == && L_send__start_sending_chunk_0) \
{ \
  SET_HANDLER (START_SEND_EVENT,\
       L_send__start_sending_chunk_, 1); \
} \
      else \
{ \
  gm_assert_p (start_send_handler \
```

```
       == &&L_send__start_sending_chunk_1); \
  SET_HANDLER (START_SEND_EVENT,\
       L_send__start_sending_chunk_, 0); \
} \
    } \
\
  /* report successful send completion */ \
   \
  gm_recycle_first_send_token (port, st); \
   \
  dispatch; \
} while (0)

#endif /* GM_ENABLE_PIO_DATAGRAMS*/
#else /* not (GM_ENABLE_SHORTCUT && GM_ENABLE_DATAGRAMS) */

#define DATAGRAM_SHORTCUT_IF_POSSIBLE(dispatch)
#define PIO_DATAGRAM_SHORTCUT_IF_POSSIBLE(dispatch)

#endif /* not (GM_ENABLE_SHORTCUT && GM_ENABLE_DATAGRAMS) */

#else  /* not GM_ENABLE_ITB */
```

A.4 Fichero ./mcp/gmcp.c

En este fichero se realizan diferentes tareas: formateado de las cadenas de depuración, inicialización del estado del autómata, asignación de los índices de eventos a los estados posibles del autómata, etc.

```
static void
print_state (void)
{
  gm_u32_t state;

  state = (get_ISR() & IMR) | GM_STATE;
```

```
  printf ("ISR =\\n"
#if GM_ENABLE_ITB
  "\\t%s %s %s\\n"
#endif
  "\\t%s %s %s %s\\n"
  "\\t%s %s %s \%s\\n"
  "\\t%s %s %s %s\\n"
  "\\t%s %s\\n",
#if GM_ENABLE_ITB
state & ITB_PACKET_SENDING ? "ITB_PACKET_SENDING" : "itb_packet_sending",
state & ITB_PACKET_PENDING ? "ITB_PACKET_PENDING" : "itb_packet_pending",
state & HEAD_INT_BIT ? "HEAD_INT_BIT" : "head_int_bit",
#endif
  state & FREE_RECV_CHUNK ? "FREE_RECV_CHUNK" : "free_recv_chunk",
  state & RECV_INT_BIT ? "RECV_INT_BIT" : "recv_int_bit",
  state & BUFF_INT_BIT ? "BUFF_INT_BIT" : "buff_int_bit",
  state & SEND_INT_BIT ? "SEND_INT_BIT" : "send_int_bit",
  state & SDMA_PENDING ? "SDMA_PENDING" : "sdma_pending",
  state & TIME_INT_BIT ? "TIME_INT_BIT" : "time_int_bit",
  state & FREE_SEND_CHUNK ? "FREE_SEND_CHUNK" : "free_send_chunk",
  state & ACK_PENDING ? "ACK_PENDING" : "ack_pending",
  state & RDMA_PENDING ? "RDMA_PENDING" : "rdma_pending",
  state & RDMAING ? "RDMAING" : "rdmaing",
  state & RECEIVING ? "RECEIVING" : "receiving",
  state & SENDING ? "SENDING" : "sending",
  state & SDMAING ? "SDMAING" : "sdmaing",
  state & SEND_PENDING ? "SEND_PENDING" : "send_pending");
}


...(code not shown)...



  /* Initialize State */
```

```
  NOTICE (FREE_RECV_CHUNK
  + FREE_SEND_CHUNK);

#if GM_ENABLE_ITB
  NOTICE_NO (RDMA_PENDING
     + RDMAING
     + RECEIVING
     + SDMAING
     + SEND_PENDING
     + ACK_PENDING
     + SENDING
     + SDMA_PENDING

     + ITB_PACKET_PENDING
     + ITB_PACKET_SENDING);
#else
  NOTICE_NO (RDMA_PENDING
     + RDMAING
     + RECEIVING
     + SDMAING
     + SEND_PENDING
     + ACK_PENDING
     + SENDING
     + SDMA_PENDING);
#endif

  /* Set up gm_event_index table to convert between state and event
     indices.  I.e.: the_index=gm_event_index[state]. */

  for (i = 0; i < (BIGGEST_STATE_BIT << 1); i++)
    {
      /* Highest priority so queues will be rewound when needed.
         Timer events happen only when no acks are pending. */

      if (i & TIME_INT_BIT && i & SEND_INT_BIT)
{
  GM_SET_EVENT_INDEX_FOR_STATE (TIMER_EVENT, i);
```

```
  continue;
}


#if GM_ENABLE_ITB
/* Link state with event */

      if (i & HEAD_INT_BIT
          && i & RECEIVING)
{
  GM_SET_EVENT_INDEX_FOR_STATE (EARLY_RECV_PACKET_EVENT, i);
  continue;
}


#endif


      if (1
#ifndef GM_CPU_lanai /* Support embedded DMA emulation */
  && i & DMA_INT_BIT
#endif
  && i & RDMAING
  )
{
  GM_SET_EVENT_INDEX_FOR_STATE (FINISH_RDMA_EVENT, i);
  continue;
}

      /* receive events */

      if (i & RECV_INT_BIT
  && i & RECEIVING)
{
  GM_SET_EVENT_INDEX_FOR_STATE (FINISH_RECV_PACKET_EVENT, i);
  continue;
}
```

```
      if (i & BUFF_INT_BIT
  && i & RECEIVING)
{
  GM_SET_EVENT_INDEX_FOR_STATE (RECV_BUFFER_OVERFLOW_EVENT, i);
  continue;
}
      if (~i & RECEIVING
  && i & FREE_RECV_CHUNK)
{
  GM_SET_EVENT_INDEX_FOR_STATE (START_RECV_PACKET_EVENT, i);
  continue;
}

      /* send events */

#if GM_ENABLE_ITB
/* Link state with event */
      if (i & SEND_INT_BIT
          && i & ITB_PACKET_SENDING)
{
  GM_SET_EVENT_INDEX_FOR_STATE (FINISH_SEND_ITB_EVENT, i);
  continue;
}
#endif

      if (i & SEND_INT_BIT
  && i & SENDING)
{
  GM_SET_EVENT_INDEX_FOR_STATE (FINISH_SEND_EVENT, i);
  continue;
}

#if GM_ENABLE_ITB
/* Link state with event */
      if (i & SEND_INT_BIT
   && ~i & SENDING
   && ~i & ITB_PACKET_SENDING
```

```
   && i & ITB_PACKET_PENDING)
{
  GM_SET_EVENT_INDEX_FOR_STATE (SEND_ITB_EVENT, i);
  continue;
}
#endif

      if (i & SEND_INT_BIT
   && ~i & SENDING
#if GM_ENABLE_ITB
   && ~i & ITB_PACKET_SENDING
   && ~i & ITB_PACKET_PENDING
#endif
   && i & ACK_PENDING)
{
  GM_SET_EVENT_INDEX_FOR_STATE (SEND_ACK_EVENT, i);
  continue;
}
      if (i & SEND_INT_BIT
  && ~i & SENDING
#if GM_ENABLE_ITB
  && ~i & ITB_PACKET_SENDING
  && ~i & ITB_PACKET_PENDING
#endif
  && i & SEND_PENDING)
{
  GM_SET_EVENT_INDEX_FOR_STATE (START_SEND_EVENT, i);
  continue;
}

      /* SDMA completion events */

      if (1
#ifndef GM_CPU_lanai /* support embedded DMA emulation */
  && i & DMA_INT_BIT
#endif
  && i & SDMAING
```

```
  )
{
  GM_SET_EVENT_INDEX_FOR_STATE (FINISH_SDMA_EVENT, i);
  continue;
}

      /* Interlocked SDMA/RDMA state machine fairness events */

#ifdef POLL_PENDING
      if (i & POLL_PENDING)
{
  GM_SET_EVENT_INDEX_FOR_STATE (POLL_EVENT, i);
}
#endif

      {
int bits;

bits = 0;

if (1
#ifdef GM_CPU_lanai
    && i & DMA_INT_BIT
#endif
    && ~i & SDMAING
    && ~i & RDMAING
    && i & FREE_SEND_CHUNK
    && i & SDMA_PENDING)
  bits |= SDMA_PENDING;

if (1
#ifdef GM_CPU_lanai
    && i & DMA_INT_BIT
#endif
    && ~i & SDMAING
    && ~i & RDMAING
    && i & RDMA_PENDING)
```

```
  bits |= RDMA_PENDING;

switch (bits)
  {
#ifndef POLL_PENDING
  case (SDMA_PENDING + RDMA_PENDING):
    GM_SET_EVENT_INDEX_FOR_STATE (FAIR_SDMA_RDMA_EVENT, i);
    continue;
  case (SDMA_PENDING):
    GM_SET_EVENT_INDEX_FOR_STATE (FAIR_SDMA_EVENT, i);
    continue;
  case (RDMA_PENDING):
    GM_SET_EVENT_INDEX_FOR_STATE (FAIR_RDMA_EVENT, i);
    continue;
  case (0):
    GM_SET_EVENT_INDEX_FOR_STATE (POLL_EVENT, i);
    continue;
#else  /* POLL_PENDING defined */
  case (SDMA_PENDING + RDMA_PENDING):
    GM_SET_EVENT_INDEX_FOR_STATE (FAIR_SDMA_RDMA_EVENT, i);
    continue;
  case (SDMA_PENDING):
    GM_SET_EVENT_INDEX_FOR_STATE (START_SDMA_EVENT, i);
    continue;
  case (RDMA_PENDING):
    GM_SET_EVENT_INDEX_FOR_STATE (START_RDMA_EVENT, i);
    continue;
  case (0):
    GM_SET_EVENT_INDEX_FOR_STATE (POLL_EVENT, i);
    continue;
#endif /* POLL_PENDING defined */
  default:
    gm_assert_p (0);
  }
      }
    }
```

```
...(code not shown)...



void
gm_dispatch (void)
{

  SET_HANDLER (POLL_EVENT, L_idle, );
  SET_HANDLER (START_SDMA_EVENT, L_sdma__start_sdma_0, );
  SET_HANDLER (FINISH_SDMA_EVENT, L_sdma__finish_sdma, );
  SET_HANDLER (SEND_ACK_EVENT, L_send__start_sending_ack, );
  SET_HANDLER (START_SEND_EVENT, L_send__start_sending_chunk_0, );
  SET_HANDLER (FINISH_SEND_EVENT, L_send__finish_sending_chunk, );
  SET_HANDLER (FINISH_RECV_PACKET_EVENT, L_recv__got_chunk_0, );
  SET_HANDLER (START_RECV_PACKET_EVENT, L_recv__start_receiving_chunk_0, );
  SET_HANDLER (RECV_BUFFER_OVERFLOW_EVENT, L_recv__discard_overflow_0, );
  SET_HANDLER (START_RDMA_EVENT, L_rdma__rdma_chunk_0, );
  SET_HANDLER (FINISH_RDMA_EVENT, L_rdma__token_done, );
  SET_HANDLER (TIMER_EVENT, L_timer, );

#if GM_ENABLE_ITB
  SET_HANDLER (SEND_ITB_EVENT, L_send__send_itb_packet_0, );
  SET_HANDLER (FINISH_SEND_ITB_EVENT,L_send__finish_sending_itb_packet_0, );
  SET_HANDLER (EARLY_RECV_PACKET_EVENT,L_recv__early_recv_packet_0, );
#define DEBUG_ITB 0
#endif


  /* Fairness handlers */

  SET_HANDLER (FAIR_SDMA_RDMA_EVENT, L_fair__sdma_rdma_0, );
#ifndef POLL_PENDING
  SET_HANDLER (FAIR_SDMA_EVENT, L_fair__sdma_0, );
  SET_HANDLER (FAIR_RDMA_EVENT, L_fair__rdma_0, );
```

```
#endif

  /* handlers for (de)activating polling */

  gm.poll.idle_handler = && L_idle;
  gm.poll.active_handler = && L_sdma__poll_for_sdma_0;

  gm_assert_p (gm.free_send_chunk_cnt < 2 || NOTICED_NOT (SEND_PENDING));
  /* gm_print_state ( ); */
  /* printf ("free_send_chunk_cnt: 0x\%x\n", free_send_chunk_cnt); */

  DISPATCH (146, "initial dispatch");
```

A.5 Fichero ./mcp/gm_bootstrap.h

En este fichero se definen variables concernientes a la arquitectura del interfaz de red. La versión implementada hace uso del bit de estado del LANai 7 *HEAD_INT_BIT* para detectar la entrada de un paquete desde la red.

A.6 Definiciones para LANai 7

```
#  elif L7
#undef POLL_PENDING _POLL_PENDING /* Do not remove: acts as feature switch */
enum gm_state_bits
  {

#if GM_ENABLE_ITB

    /* HEAD_INT_BIT  0x00000001 */
    /* RECV_INT_BIT                0x00000002 */
    /* BUFF_INT_BIT  0x00000004 */
    /* SEND_INT_BIT  0x00000008 */
    FREE_SEND_CHUNK = LAN8_SIG_BIT,/* 0x00000010 */
```

```
    ACK_PENDING = LAN9_SIG_BIT,/*  0x00000020 */
    TIME_INT_BIT = TIME0_INT_BIT, /* 0x00000040 */
    RDMA_PENDING = TIME1_INT_BIT,/* 0x00000080 */
    RDMAING = OFF_BY_1_BIT,/* 0x00000100 */
    RECEIVING = OFF_BY_2_BIT,/* 0x00000200 */
    SENDING = OFF_BY_4_BIT,/* 0x00000400 */
    SDMAING = NRES_INT_BIT,/* 0x00000800 */
    /* WAKE_INT_BIT                    0x00001000 */
    SEND_PENDING = TIME2_INT_BIT,/*  0x00002000 */
    SDMA_PENDING = MEMORY_INT_BIT,/*  0x00004000 */
    POLL_PENDING = PARITY_INT_BIT,/* 0x00008000 */
    FREE_RECV_CHUNK = LAN0_SIG_BIT,/*   0x00010000 */

    ITB_PACKET_PENDING = LAN1_SIG_BIT,/*0x00020000 */
    ITB_PACKET_SENDING = LAN2_SIG_BIT,/*0x00040000 */


  };
#define BIGGEST_STATE_BIT ITB_PACKET_SENDING

#else

    FREE_RECV_CHUNK = HEAD_INT_BIT, /* 0x00000001 */
    /* RECV_INT_BIT                0x00000002 */
    /* BUFF_INT_BIT  0x00000004 */
    /* SEND_INT_BIT  0x00000008 */
    FREE_SEND_CHUNK = LAN8_SIG_BIT,/* 0x00000010 */
    ACK_PENDING = LAN9_SIG_BIT,/*  0x00000020 */
    TIME_INT_BIT = TIME0_INT_BIT, /* 0x00000040 */
    RDMA_PENDING = TIME1_INT_BIT,/* 0x00000080 */
    RDMAING = OFF_BY_1_BIT,/* 0x00000100 */
    RECEIVING = OFF_BY_2_BIT,/* 0x00000200 */
    SENDING = OFF_BY_4_BIT,/* 0x00000400 */
    SDMAING = NRES_INT_BIT,/* 0x00000800 */
    /* WAKE_INT_BIT                    0x00001000 */
    SEND_PENDING = TIME2_INT_BIT,/*  0x00002000 */
    SDMA_PENDING = MEMORY_INT_BIT,/*  0x00004000 */
```

```
    POLL_PENDING = PARITY_INT_BIT,/* 0x00008000 */
  };
#define BIGGEST_STATE_BIT POLL_PENDING
#endif


#ifndef IMR
#define IMR  IMR_INIT_VAL
#endif
#  else
#    error bogus GM_LANAI_MAJOR_VERSION
#  endif


#if GM_ENABLE_ITB
#define IMR_INIT_VAL (TIME_INT_BIT  \
 | DMA_INT_BIT \
 | SEND_INT_BIT \
 | RECV_INT_BIT \
 | BUFF_INT_BIT \
 | HEAD_INT_BIT)
#else
#define IMR_INIT_VAL (TIME_INT_BIT  \
 | DMA_INT_BIT \
 | SEND_INT_BIT \
 | RECV_INT_BIT \
 | BUFF_INT_BIT)
#endif


#endif /* ifdef GM_LANAI_NATIVE */

...
```

A.7 Ficheros ./acconfig.h y ./configure.h

En estos ficheros se indica la macro de compilación condicional con ITBs y ciertas variables para su correcta compilación.

A.7.1 Fichero ./acconfig.h

```
/* Should we support ITB? */

#define GM_ENABLE_ITB 0
```

A.7.2 Fichero ./configure.in

```
dnl Used to compile MCP with ITB support
AC_ARG_ENABLE(itb,  --enable-itb         Support ITB,
              enable_itb=$enableval, enable_itb=no)

...

dnl Used to compile MCP with ITB support
if test ${enable_itb} = yes ; then
  AC_DEFINE(GM_ENABLE_ITB)
fi
```

Bibliografía

[Alc96] R. Alcover, *Optimización de redes de interconexión para multicomputadores mediante técnicas de Diseño Robusto de Taguchi*, tesis doctoral.

[And87] T.E. Anderson, D.E. Culler, D.A. Patterson et al., "A case for NOW (Networks of Workstations)," *IEEE Micro*, pp. 56-64, February 1995.

[Bod95] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J. Seizovic and W. Su, "Myrinet - A gigabit per second local area network," *IEEE Micro*, pp. 29–36, February 1995.

[Bop93] R.V. Bopana, and S. Chalasani, "A comparison of adaptive wormhole routing algorithms," *Proceedings 20th Annual International Symposium on Computer Architecture*, May 1993.

[Buy99] R. Buyya, *High Performance Cluster Computing*, Prentice Hall, 1999.

[Che95] L. Cherkasova, V. Kotov, and T. Rokicki, "Fibre channel fabrics: Evaluation and design," *29th International Conference on System Sciences*, February 1995.

[Chi92] A.A. Chien and J.H. Kim, "Planar-adaptive routing: Low-cost adaptive routing for multiprocessors," *Proceedings 19th Annual International Symposium Computer Architecture*, May 1992.

[Col01] S. Coll, J. Flich, M.P. Malumbres, P. López, J. Duato, and F.J. Mora, "A First Implementation of In-Transit Buffers on Myrinet GM Software," aceptado para publicar en *Communication Architecture for Clusters*, April 2001.

[Cul99] D.E. Culler, J.P. Singh, *Parallel Computer Architecture*, Morgan Kaufmann, 1999.

[Dal87] W.J. Dally and C.L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, Vol. C-36, No. 5, pp. 547-553, May 1987.

[Dal91] W.J. Dally, "Express Cubes: Improving the Performance of k-ary n-cube Interconnection Networks," *IEEE Transactions on Computers*, Vol. 40, No. 9, September 1991.

[Dal92] W.J. Dally, "Virtual-channel flow control," *IEEE Transactions on Parallel Distributed Systems*, vol. 3, No. 2, pp. 194–205, March 1992.

[Dav97] D. Garcia and W. Watson. "ServerNet II," *Proceedings of the 1997 Parallel Computer, Routing, and Communication Workshop*, pp. 119-135, June 1997.

[Dua93] J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 12, pp. 1320-1331, December 1993.

[Dua94] J. Duato and P. López, "Performance Evaluation of Adaptive Routing Algorithms for k-ary n-cubes," *Proceedings Parallel Computer Routing and Communication Workshop*, May 1994.

[Dua95] J. Duato, "A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 10, pp. 1055-1067, October 1995.

[Dua97] J. Duato, S. Yalamanchili, and L. Ni, "Interconnection Networks, An Engineering Approach," *IEEE Computer Society Press*, 1997.

[Fel94] R. Felderman, A. DeSchon, D. Cohen, and G. Finn, "ATOMIC: a high-speed local communication architecture," *Journal of High Speed Networks*, 3(1):1-28, January 1994.

[Fil97] M. Fillo and R. B. Gillet, "Architecture and Implementation of Memory Channel 2," *Digital Technical Journal*, vol. 9(1), 1997.

[Fli98a] J. Flich, P. López, M.P. Malumbres, and J. Duato, "EDINET: An Execution Driven Interconnection Network Simulator for DSM Systems," *Proceedings of 10th International Conference on Modelling Techniques and*

Tools for Computer Performance Evaluation, Lecture Notes in Computer Science 1469, September 1998.

[Fli98b] J. Flich, P. López, M.P. Malumbres y J. Duato, "Un simulador de redes de interconexión dirigido por ejecución para sistemas DSM," *IX Jornadas de Paralelismo*, Septiembre 1998.

[Fli99a] J. Flich, M.P. Malumbres, P. López and J. Duato, "Performance Evaluation of Networks of Workstations with Hardware Shared Memory Model Using Execution-Driven Simulation," *International Conference on Parallel Processing*, September 1999.

[Fli99b] J. Flich, M.P. Malumbres, P. López, J. Duato, and R. Felderman, "Using Minimal Routing in Myrinet Networks," *X Jornadas de Paralelismo*, September 1999.

[Fli00a] J. Flich, M.P. Malumbres, P. López, and J. Duato, "Improving Routing Performance in Myrinet Networks," *International Parallel and Distributed Processing Symposium (IPDPS 2000)*, May 2000.

[Fli00b] J. Flich, M.P. Malumbres, P. López, and J. Duato, "Performance Evaluation of a New Routing Strategy for Irregular Networks with Source Routing," *International Conference on Supercomputing*, May 2000.

[Fli00c] J. Flich, M.P. Malumbres, P. López, and J. Duato, "In-Transit Buffers: A Mechanism to Support Minimal Routing in Myrinet," *IEEE Technical Committe on Computer Architecture Newsletter*, June 2000.

[Fli00d] J. Flich, P. López, M.P. Malumbres, and J. Duato, "Improving the Performance of Regular Networks with Source Routing," *International Conference on Parallel Processing*, August 2000.

[Fli00e] J. Flich, P. López, M.P. Malumbres, and J. Duato, "In-Transit Buffers: A Mechanism to Improve Performance of Networks with Source Routing," *XI Jornadas de Paralelismo*, September 2000.

[Fli00f] J. Flich, P. López, M.P. Malumbres, J. Duato, and T. Rokicki "Combining In-Transit Buffers with Optimized Routing Schemes to Boost the

Performance of Networks with Source Routing," *International Symposium on High Performance Computing*, October 2000.

[Fli01] J. Flich, P. López, M.P. Malumbres, J. Duato, and T. Rokicki, "Improving Network Performance by Reducing Network Contention in Source-Based COWs with a Low Path-Computation Overhead," aceptado para publicar en *International Parallel and Distributed Processing Symposium*, April 2001.

[Gei94] A. Geist et al. "PVM 3 User's Guide and Reference Manual," September 1994.

[HIP] The HIPPI Protocol, *http://www.cis.ohio-state.edu/jain/cis788-95/hippi*.

[SCI] M. Ibel, K.E. Schauser, C.T. Scheiman, and M. Weis, "High Performance Cluster Computing Using SCI," *Tech Report*, University of California, Santa Barbara, http://www.cs.ucsb.edu/research/sci.

[IEEE85] IEEE Standard, "802.3: Carrier Sense Multiple Access with Collision Detection," New York, 1985.

[IEEE95] IEEE Standard, "802.3u, Media Access Control (MAC) Parameters, Physical Layer, Medium Attachment Units, and Repeater for 100Mb/s Operation, Type 100BASE-T," 1995.

[IBA] InfiniBand Home Page, *http://www.infinibandta.org/*

[Jai91] R. Jain, *The art of computer systems performance analysis*, New York, Ed. John Wiley & Sons.

[Ker79] P. Kermani and L. Kleinrock, "Virtual cut-through: a new computer communication switching technique," *Computer Networks*, vol. 3, pp. 267–286, 1979.

[Kim92] J. Kim, A. Chien, "An evaluation of the planar/adaptive routing," *Proceedings 4th IEEE International Symposium on Parallel Distributed Processing*, 1992.

[Kon91] S. Konstantinidou and L. Snyder, "Chaos router: Architecture and performance," *Proceedings of the 18th International Symposium on Computer Architecture*, June 1991.

[Lar98] J. Larson, "The HAL Interconnect PCI Card," *Workshop on Communications and Architectural Support for Network-based Parallel Computing*, 1998.

[Lau98] M. Lauria, S. Pakin, and A. Chien, "Efficient Layering for High Speed Communication: Fast Messages 2.x," *Proceedings of 7th IEEE International Symposium HPDC-7*, Chicago, IL, July 1998.

[Mil91] P.R. Miller, *Efficient communications for fine-grain distributed computers*, Ph.D Thesis, Southamptom University, 1991.

[Mpi94] "Message Passing Interface Forum," *The Message-Passing Interface Standard*, May 1994.

[GM] Myricom, Inc, "Myrinet. Multi-platform GM software," disponible en *http://www.myri.com/GM.*

[SWCH] Myricom, Inc., "8-Port Myrinet Switch with 4 LAN ports and 4 SAN ports," disponible en *http://www.myri.com/myrinet/switches/m2fm-sw8.html.*

[lan10m] Myrinet, "M2-CB-35 LAN cables," *http://www.myri.com/myrinet/ product_list.html*

[MYRI] *http://www.myri.com.*

[MYR2K] *http://www.myri.com/news/00701/index.html.*

[Mar93] M. de Prycker, *Asynchronous Transfer Mode: Solution for broadband ISDN*, Second Edition, Ellis Horwood, 1993 (ISBN: 0-13-178542-7).

[Pry98] L. Prylli and B. Tourancheau, "BIP: a New Protocol Designed for High Performance Networking on Myrinet," *Proceedings Workshop PC-NOW, IPPS/SPDP'98*, Orlando, FL, April 1998.

[Qia96] W. Qiao and L.M. Ni, "Adaptive routing in irregular networks using cut-through switches," *Proceedings of the 1996 International Conference on Parallel Processing*, August 1996.

[Rie99] R. Riesen et al., "CPLANT, " *Proceedings of the Second Extreme Linux Workshop*, June 1999.

[Rob96] Robert W. Horst. "ServerNet deadlock avoidance and fractahedral topologies," *Proceedings of the International Parallel Processing Symposium*, April 1996.

[Ross89] F.E. Ross, "An overview of FDDI - the fiber distributed data interface," *IEEE Journal on Selected Areas of Communications*, September 1989.

[San00] J.C. Sancho, A. Robles, and J. Duato, "New Methodology to Compute Deadlock-Free Routing Tables for Irregular Networks," *Workshop on Communications and Architectural Support for Network-based Parallel Computing*, January 2000.

[SAN] *http://www.sandia.gov*.

[She98] R. Sheifert, *Gigabit Ethernet*, Addison-Wesley, ISBN: 0-201-18553-9, Abril 1998.

[Sch90] M.D. Schroeder et al., "Autonet: A high-speed, self-configuring local area network using point-to-point links," *IEEE Journal on Selected Areas in Communications*, 9(8):1318-1335, October 1991.

[Sco94] S.L. Scott and J.R. Goodman, "The Impact of Pipelined Channels on k-ary n-Cube Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 1, pp. 2–16, January 1994.

[Sei93] C.L. Seitz and W. Su, "A family of routing and communication chips based on the Mosaic," *Proceedings of the Washington Symposium on Integrated Systems*, 1993.

[Sil97a] F. Silla, M.P. Malumbres, A. Robles, P. López and J. Duato, "Efficient Adaptive Routing in Networks of Workstations with Irregular Topology," *Workshop on Communications and Architectural Support for Network-based Parallel Computing*, February 1997.

[Sil97b] F. Silla and J. Duato, "Improving the Efficiency of Adaptive Routing in Networks with Irregular Topology," *1997 International Conference on High Performance Computing*, December 1997.

[Sil97c] F. Silla and J. Duato, "On the Use of Virtual Channels in Networks of Workstations with Irregular Topology," *1997 Parallel Computer Routing and Communication Workshop*, June 1997.

[Sil97d] F. Silla, M.P. Malumbres, J. Duato, D. Dai, and D.K. Panda, "Impact of Adaptivity on the Behavior of Networks of Workstations under Bursty Traffic," *Proceedings of the 1998 International Confence on Parallel Processing*, August 1998.

[Sil00] F. Silla and J. Duato, "High Performance Routing in Networks of Workstations with Irregular Topology," *IEEE Transactions on Parallel and Distributed Computing*, vol. 11, no. 7, pp. 699-719, ISSN 1045-9219, July 2000.

[SPLA2] S. C. Woo et al., "The SPLASH-2 Programs: Characterization and Methodological Considerations," *International Symposium on Computer Architecture*, 1995.

[Tanen96] A.S. Tanenbaum, *Computer Networks*, 3a edición, Prentice-Hall, 1996.

[IRFC] *Transmission Control Protocol (TCP)*, Internet Request For Comment no. 793, 761 and 675.

[VIA] Virtual Interface Architecture Home Page, *http://www.viarch.org/*, 1998.

[Yoc97] K.G. Yocum, J.S. Chase, A.J. Gallatin, and A.R. Lebeck, "Cut-Through Delivery in Trapeze: An Exercise in Low Latency Messaging," *IEEE Symposium on High-Performance Distributed Computing*, August 1997.