Universidad de Castilla-La Mancha

Departamento de Informática

Diseño de un Encaminador Orientado a Tráfico Multimedia en Entornos LAN

Memoria que para optar al grado de
Doctor en Ingeniería Informática presenta
María Blanca Caminero Herráez

Dirigida por
Dr. D. Francisco J. Quiles Flor y Dr. D. José Duato Marín

Albacete, Julio 2002

Agradecimientos

Una vez finalizado el esfuerzo que supone realizar el trabajo plasmado en estas páginas, llega el momento de hacer un punto y aparte (nunca un punto y final) y dedicar aunque sólo sean unas pocas líneas a todos aquellos que lo han hecho posible.

En primer lugar, un recuerdo para mis directores de Tesis, Paco y Jose. Paco fue quien me brindó la oportunidad de entrar en este mundillo de la Universidad. Pero ya unos cuantos años antes, cuando estudiaba la Diplomatura en Informática en la entonces Escuela Universitaria Politécnica de Albacete, fueron él y compañeros como él quienes despertaron en mí el "gusanillo" de la investigación y la vocación para dedicarme a esta no siempre bien agradecida tarea de la docencia universitaria. Y qué decir de Jose... Muchas veces nos hemos encontrado con problemas que creíamos insalvables, y han bastado unos pocos minutos con él para descubrir que en no pocas ocasiones la solución era mucho más obvia y sencilla de lo que pretendíamos. Sus ideas y comentarios siempre han sido como un soplo de aire fresco que renueva las energías cuando éstas flaquean. Por todo ello, y por todo el apoyo ofrecido durante estos años de trabajo, muchas gracias a los dos.

También quisiera dar las gracias a todos los compañeros del Grupo de Redes y Arquitecturas de Altas Prestaciones, donde he podido desarrollar este trabajo. Trabajar con vosotros ha sido y será siempre un privilegio y todo un placer. Gracias a todos. Y gracias en especial a Carmen, con quien he tenido oportunidad de colaborar más estrechamente, y a Tere, esa vocecilla tras la pared de la izquierda (ánimo, compañera!!).

No quiero olvidar tampoco a los compañeros del Grupo de Arquitecturas Paralelas de la Universidad Politécnica de Valencia, en especial a Fede Silla, por estar siempre disponibles cuando he aparecido por allí de improviso y con un montón de preguntas bajo el brazo. Gracias.

Quisiera continuar dando las gracias a los compañeros del Departamento de Informática, por eso mismo: por ser los mejores compañeros durante estos años. Muchas gracias a todos.

Un recuerdo especial va para el grupo de compañeros, bueno, más que compañeros, amigos, que solemos compartir más momentos juntos. Durante estos años hemos pasado por muchos ratos buenos, y otros no tan buenos, pero siempre habéis estado ahí haciendo el trabajo más agradable y llevadero. Gracias a tod@s, Mazmorrer@s!!.

Y por último, pero desde luego no los últimos, quisiera dar las gracias a los más cercanos a mí. Mil gracias a mis padres y hermanos, por apoyarme siempre, y haberme ayudado a seguir mi vocación universitaria.

Y sobre todo, gracias a Kike, por ser como es, y por estar siempre ahí, animándome en los momentos de debilidad, soportándome estoicamente en los momentos de nerviosismo y compartiendo y aumentando los momentos de alegría. Quizás podría haberlo hecho sin ti, pero desde luego no hubiese sido igual. Gracias, mil gracias, ∞ millones de gracias `:o ;)` .

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Redes de altas prestaciones

Durante los últimos años, los avances en teoría, arquitectura y tecnología de las redes de computadores han permitido mejorar las prestaciones de éstas enormemente. Se han desarrollado algoritmos de encaminamiento más eficientes y fiables, mejores topologías de red, mecanismos de control de flujo y de la congestión, esquemas de utilización de los medios compartidos, y técnicas de conmutación más eficientes. Las redes de interconexión de multicomputadores también se han desarrollado en paralelo, siguiendo el camino marcado por las redes generales de alcance amplio, y mejorando su arquitectura. Más aún, durante la pasada década hemos presenciado como técnicas empleadas tradicionalmente en redes de interconexión de multicomputadores y multiprocesadores han saltado las barreras de dichas máquinas extendiendo su aplicación al ámbito de las redes de área local (LAN).

El motivo es el surgimiento de los *grupos de estaciones de trabajo* (*Clusters Of Workstations* o COWs), también denominados redes de estaciones de trabajo o NOWs, como alternativa de bajo coste a los computadores paralelos, para una extensa gama de aplicaciones que exhiben un grado de paralelismo medio [18]. En estas COWs, se emplean estaciones de trabajo u ordenadores personales de propósito general como elementos de procesamiento, unidos mediante una **red de altas prestaciones**. Estas redes también se han planteado como alternativa al tradicional bus del sistema dando lugar a las *redes de área de sistema* (SAN, *System Area Network*) [73, 132].

Tanto en el caso de las COWs como de las SAN, las redes de altas prestaciones proporcionan una alternativa a la clásica y barata topología de bus, trátese del bus del sistema (por ejemplo, un bus PCI), o de la típica configuración de red local Ethernet. El drástico incremento experimentado durante la última década tanto en velocidades de transmisión de información (10 Mbps se consideraba algo rápido), como en velocidad de funcionamiento de los circuitos integrados (los procesadores de 66 MHz eran el último grito), y más aún en las cantidades masivas de información a manejar (Internet era algo restringido a la comunidad científica, y el tratamiento masivo de datos daba sus primeros pasos), han puesto de manifiesto las carencias de las topologías basadas en bus. Concretamente, y refiriéndose al bus de sistema PCI, en [132] se citan los siguientes inconvenientes:

- Contención por recursos desordenada por parte de periféricos, memoria y CPUs. El desorden lleva a la ineficiencia y a prestaciones subóptimas.

- Único punto de fallo potencial en el sistema, puesto que todo el sistema se monta alrededor del bus, y si éste es único, un sólo fallo en el mismo deshabilita el sistema completo. Más aún, si una tarjeta conectada al bus falla, puede hacer que el sistema entero deje de funcionar, y la única manera de descubrir cuál fue la tarjeta causante del fallo es mediante el método de "ensayo y error". Esto es inadmisible en servidores que necesitan de alta disponibilidad.

- Importantes limitaciones físicas, puesto que al incrementar la longitud del bus para acomodar más dispositivos, las señales eléctricas pierden estabilidad. Lo mismo ocurre al incrementar la frecuencia de reloj: cuanto mayor sea la frecuencia, menor es la longitud posible del bus, y menos periféricos se pueden conectar. En un caso extremo, la especificación de PCI-X para 133 MHz sólo permite un único conector por bus.

Compaq [141] también señala las limitaciones de las típicas LAN que dieron auge a los sistemas distribuidos a mediados de los años 80:

- Todas las LAN estaban basadas en la difusión, donde cada mensaje transmitido se transporta a todos los hosts conectados a la red. Esto significa que la productividad de la red se ve limitada por el ancho de banda del medio. Cuando con el tiempo se van incrementando tanto el número de usuarios de la LAN como el volumen de información transferida, este tipo de red no puede soportar esa demanda.

- En sus configuraciones más simples, las redes LAN convencionales exhibían un único punto de fallo.

- En las redes basadas en bus, un incremento en velocidad implica decrementar el tamaño máximo de los segmentos

Tanto en la interconexión de máquinas para formar un *cluster* dedicado al procesamiento distribuido, como en la misma arquitectura de los computadores convencionales, el principal cuello de botella, al menos en lo que a capacidad de transferencia de información se refiere, se hallaba en la topología de bus. En el ámbito de las LAN, la industria empezó a trabajar en dispositivos como puentes, concentradores inteligentes y encaminadores que aliviasen el problema. Todos ellos se basaban en el uso de conmutadores o *switches*, que evitaban que todos los mensajes llegasen a todas partes, sino que cada mensaje se dirige únicamente hacia su destino. Como consecuencia, a principios de los años noventa se desarrolló en el *Systems Research Center* de Digital (actualmente, perteneciente a Compaq) una red basada enteramente en conmutadores, con enlaces punto-a-punto entre éstos y los *hosts*: la **red Autonet**, o **AN1** [142]. En esta red los mensajes eran encaminados por los conmutadores hacia su destino, a lo largo de un camino que era determinado en cada conmutador con arreglo a la dirección de destino almacenada en la cabecera del mensaje. De esta forma, muchos mensajes distintos pueden estar atravesando la red por enlaces distintos en un momento dado. Los conmutadores de esta red empleaban como técnica de conmutación *virtual cut-through*, es decir, tan pronto como el conmutador ha analizado la cabecera del mensaje y ha calculado el puerto de salida correspondiente, el mensaje se hace avanzar a través de ese puerto de salida, incluso aunque el mensaje completo no se haya recibido todavía en el conmutador. Esta forma de transmisión segmentada disminuye notablemente la latencia de los mensajes respecto a técnicas tipo *store-and-forward* [1].

[1] En [51] se describen y analizan detalladamente las diversas técnicas de conmutación empleadas en redes de interconexión

En el diseño de esta red, se aplicaron muchas de las ideas originalmente desarrolladas y aplicadas en el ámbito de los computadores paralelos: los enlaces punto-a-punto, la conmutación *cut-through*, conmutadores basados en *crossbar*, etc.. Sin embargo, también fue necesario desarrollar nuevas técnicas para afrontar las características diferenciales del nuevo entorno: algoritmo de encaminamiento *up*/down** para poder encaminar mensajes sin provocar bloqueos en la topología irregular, *buffers* de mayor tamaño y control de flujo *stop-and-go* para aprovechar mejor el ancho de banda de los enlaces largos, algoritmo de reconfiguración distribuido para reconfigurar las tablas de encaminamiento ante la conexión o desconexión de elementos de la red (que, al contrario de lo que sucedía en los computadores paralelos, ahora son independientes), etc.

Autonet fue la pionera de este tipo de redes, introduciendo muchos de los conceptos y principios presentes en otras redes posteriores, como Myrinet [25] o ServerNet [73].

La **red Myrinet**, de Myricom Inc., tuvo su germen en dos proyectos financiados por el gobierno de EE.UU.: los proyectos Mosaic, del CalTech (California Institute of Technology), y Atomic LAN, del USC/ISI (University of Southern California/Information Sciences Institute). El objeto del primero era el desarrollo de un multicomputador experimental de grano fino, mientras que en el segundo se perseguía construir una LAN empleando componentes Mosaic. Los orígenes de Myrinet explican el uso de técnicas empleadas en redes de interconexión de multicomputadores en su diseño: conmutación *wormhole*, conmutadores basados en *crossbar*, encaminamiento fuente, etc.

En el mismo año apareció la **red ServerNet** (originalmente denominada TNet), la primera propuesta para reemplazar el bus del sistema por una red conmutada hecha por Tandem para sus servidores de gama alta, con fuertes requerimientos de funcionamiento ininterrumpido. Se trata por tanto de la primera propuesta de SAN (*System Area Network*) aparecida en la literatura. En la red ServerNet se introducía por primera vez la idea de dotar a los periféricos de cierta inteligencia para ser capaces de enviar y recibir mensajes a través de la red, sin necesidad de que la CPU participe en todos los movimientos de datos. La idea subyacente era que empezaban a aparecer aplicaciones que requerían de una transferencia masiva de datos sin necesidad de ningún procesamiento (telecompra, educación a distancia, vídeo bajo demanda, ...). Por tanto, era necesario un servidor capaz de satisfacer esas demandas masivas de información, puesto que en la organización tradicional el bus del sistema y la CPU se convertían en cuellos de botella insalvables para ofrecer las prestaciones en cuanto a transferencia de datos de las nuevas aplicaciones.

Como se ha podido observar, los arquitectos de estas nuevas redes de altas prestaciones se centraron en cómo reservar y utilizar los restringidos recursos de la red de forma eficiente, con el objetivo de proporcionar mayor productividad y menor latencia media que las soluciones basadas en bus que se empleaban anteriormente.

Esta aproximación es plenamente válida cuando la información que fluye por la red no necesita obtener prestaciones "especiales", es decir, simplemente es necesario trasegar una cantidad de tráfico suficiente (alta productividad) y procurar que ese tráfico llegue a su destino con el menor retardo posible en término medio. Por ello, a este tipo de tráfico se le denomina como de "mejor esfuerzo" (*best-effort*), pues el sistema sólo se compromete a hacer lo que pueda para que llegue a su destino con buenas prestaciones, pero no garantiza nada.

Sin embargo, en los últimos años, hemos presenciado un crecimiento explosivo de las comunicaciones de datos continuos (audio/vídeo) sobre redes de computadores [165, 156, 32]. Las tecnologías de red actuales, como la fibra óptica y la conmutación ATM, hacen que estén disponibles a bajo coste redes de gran ancho de banda, capaces de proporcionar varios gigabits por segundo [137, 31]. Gracias a la disponibilidad de este tipo de redes de gran ancho

de banda, las aplicaciones multimedia surgen de forma natural como los principales clientes que aprovecharán las mayores capacidades de las redes de computadores. El mayor ancho de banda que requieren las nuevas aplicaciones puede ser satisfecho plenamente con las propuestas introducidas en la década de los noventa. Sin embargo, estas nuevas aplicaciones tienen otros requisitos adicionales completamente diferentes a los requerimientos de las aplicaciones tradicionales.

1.2. Nuevas aplicaciones, nuevas necesidades

Las aplicaciones multimedia integran varios tipos de medios, como vídeo, audio, imágenes fijas, gráficos y texto. La transmisión de estos medios, y sobre todo del audio y del vídeo, introducen nuevos requerimientos en las redes de transmisión. Concretamente, tanto el vídeo como audio generan un tráfico continuo, en el sentido de que requiere disponibilidad de ancho de banda durante toda la duración de la transmisión para que la aplicación no se vea interrumpida en el nodo receptor. En el caso de la señal de vídeo, se debe mostrar al receptor una imagen cada cierto tiempo de forma continua. Además, estas imágenes, una vez emitidas en el emisor, se deben presentar en el destino dentro de unos límites de retardo adecuados. Los valores de estos límites dependen del nivel de interacción de la aplicación, y pueden oscilar desde unos 20 milisegundos, hasta varios segundos. La técnica más común es imponer cotas al retardo de acuerdo con los límites perceptivos de la aplicación. Cuando el retardo sufrido por un paquete[2] de información es superior a dicho límite, el receptor lo descarta, provocando una pérdida en la calidad de la señal recibida (posiblemente acentuada por las dependencias entre paquetes introducidas por la codificación de la señal). Además, la transmisión de ese paquete finalmente descartado, también supone un desperdicio de ancho de banda [85].

Todos estos nuevos requisitos se reflejan en lo que se denomina *Calidad de Servicio* (QoS). El término *Calidad de Servicio* (QoS) fue empleado por primera vez hace unos veinte años en el **Modelo de Interconexión de Sistemas Abiertos** (OSI). Hacía referencia a la capacidad de un proveedor de servicios (un operador de red) para soportar los requerimientos de las aplicaciones de usuario con respecto a cuatro parámetros de servicio: ancho de banda, latencia o retardo, *jitter* o variación en la latencia y pérdida de datos [24].

La provisión de *ancho de banda* para una aplicación significa que la red tenga la suficiente capacidad para soportar los requerimientos de productividad de la aplicación.

La *latencia* o *retardo* describe el tiempo que se tarda en enviar un paquete de usuario desde un nodo origen a un nodo destino. Hay algunas aplicaciones, como la transmisión de vídeo o voz, que tienen marcados requerimientos en cuanto a latencia, puesto que si el paquete llega demasiado tarde, ya no es útil y se ignora. Por tanto, los paquetes que llegan tarde a su destino se traducen en ancho de banda desperdiciado y en una reducción de la QoS que recibe la aplicación. Sin embargo, los paquetes que llegan demasiado pronto también presentan problemas. En este tipo de aplicaciones multimedia, el receptor espera recibir la información a intervalos bastante regulares. Los paquetes prematuros deben almacenarse en el destino hasta que llegue su turno de ser procesados. Por tanto, puede ocurrir que el espacio de *buffers* en el receptor se desborde, viéndose obligado a descartar información que llegó demasiado antes de tiempo. Es decir, al contrario de lo que sucede en la transmisión de

[2]En este Capítulo empleamos el término "paquete" en un sentido muy general, indicando una unidad de transferencia de información entre dos usuarios a través de una red. Posteriormente concretaremos la terminología empleada a este respecto.

datos tradicional, ahora ya no se cumple que a menor retardo, necesariamente se obtengan mejores prestaciones, sino que un retardo excesivamente pequeño también puede provocar problemas.

En cuanto al *jitter*, se trata de la variación en los retardos entre la llegada al receptor de dos paquetes consecutivos. Estos retardos son causados por esperas provocadas a los paquetes por la contención por recursos que en un momento dado pueden hallarse ocupados. Una variación elevada en los retardos experimentados por los paquetes complica la reproducción en el destino de secuencias de voz o vídeo.

La *pérdida de datos* es el último de los parámetros generalmente considerados cuando se habla de QoS. La pérdida de información es importante en aplicaciones de voz y vídeo, pues puede afectar al proceso de decodificación, lo que se traduce en una degradación de la señal que recibe el usuario final. En aplicaciones tradicionales, la pérdida de datos también es un serio problema, pero puesto que en general estas aplicaciones no tienen requisitos temporales demasiado estrictos (a fin de cuentas el tráfico que generan se suele catalogar en su inmensa mayoría como de tipo *best-effort*), se pueden aplicar procedimientos de retransmisión para obtener la transferencia fiable de todos los datos. Por el contrario, cuando se trata de aplicaciones multimedia, la retransmisión no suele ser de utilidad debido a los retardos incurridos entre la detección del error y el reenvío de la información.

Los requerimientos de Calidad de Servicio para las aplicaciones multimedia se definen en función de los requerimientos perceptivos de las mismas. Es decir, se va a requerir que la red transfiera la información con unas determinadas prestaciones para que los usuarios perciban la señal sin degradaciones.

A continuación se comentarán brevemente los requerimientos perceptivos de algunas de las aplicaciones multimedia más representativas [35]:

Vídeoconferencia. La vídeoconferencia es una aplicación interactiva en tiempo real, y por tanto sus requerimientos en términos de retardo son muy exigentes. Un retardo en torno a los 200-500 milisegundos puede resultar molesto a los usuarios. En [59] se recomienda un retardo máximo de 150 milisegundos para que el retardo resulte imperceptible para los usuarios. En cuanto a sus requisitos de ancho de banda, las secuencias suelen contener simplemente el busto de una persona, con lo que el movimiento es escaso, y por tanto no requieren un gran consumo de ancho de banda. De hecho, la mayoría de aplicaciones de vídeoconferencia actuales emplean la red telefónica conmutada, y por tanto se les dedican canales de 64 Kbps.

Difusión de televisión digital. Estas aplicaciones incluyen películas, anuncios, noticias, que en general presentan un mayor grado de movimiento. Esto hace que los requerimientos de ancho de banda sean mayores que en el caso de la vídeoconferencia (en torno a los 4-10 Mbps para la TV convencional, y de 20 a 80 Mbps para la televisión de alta definición). Por el contrario, los requerimientos en términos de retardo no son tan estrictos como en el caso anterior. Generalmente, se toleran bien latencias de varios segundos para las transmisiones en directo. Incluso para las transmisiones de información en diferido no se requiere ningún retardo específico, pues el usuario no lo podrá apreciar. El único requisito al respecto es que una vez que la información empieza a visualizarse en el receptor, ésta debe ser mostrada de manera continua, sin interrupciones. Esto implica que la variación en el retardo o *jitter* será un parámetro importante a tener en cuenta.

Vídeo bajo demanda (VoD). En esta aplicación la señal de vídeo codificada está almacenada en un servidor, y se transmite al usuario bajo petición de éste. Es el usuario,

por tanto, quien controla el comienzo de la transmisión. Además, puede ser que el usuario tenga acceso a funciones de rebobinado o pausa de la señal. Si el usuario no dispone de ese control adicional, el retardo sólo afecta a la latencia con la que empieza a proporcionarse el servicio desde que el usuario lo solicita. Así, se toleran retardos de hasta unos pocos minutos. Por el contrario, si el usuario tiene control sobre el avance de la transmisión (rebobinados, pausas, etc.) la aplicación se vuelve más interactiva, y los requerimientos en términos de retardo pasan a ser de unos cientos de milisegundos [39].

Telemedicina. Esta aplicación es fuertemente interactiva, entre el centro de diagnóstico y el usuario, y por tanto, los requerimientos en términos de retardo son exigentes, en tornos a los 100 milisegundos o menos. La resolución de las imágenes es elevada, y las degradaciones en su calidad deben ser imperceptibles. Estas aplicaciones suelen requerir un ancho de banda de entre 1 y 20 Mbps.

Kurose y Ross [97] clasifican las aplicaciones multimedia existentes en las siguientes tres categorías:

- ***Streaming*:**
  - Los clientes solicitan ficheros de audio o de vídeo a servidores, y segmentan la recepción de los datos desde la red y su visualización.
  - Son interactivos: el usuario puede controlar la reproducción, de manera similar a como lo haría con un reproductor de vídeo convencional.
  - Retardo: desde la petición del cliente hasta el inicio de la visualización pueden transcurrir varios segundos.
  - Ejemplos: RealPlayer de RealNetworks [7], Windows Media de Microsoft [6].
- **Unidireccionales de tiempo real:**
  - Similares a las emisiones existentes de radio o televisión, pero empleando la red como medio de transmisión
  - No son interactivas, el usuario simplemente escucha o visualiza la señal.
  - Retardo: al igual que en el caso anterior, se toleran varios segundos de retardo desde que el cliente decide conectarse a la emisión.
  - Ejemplos: hoy en día, muchas emisoras de radio emiten sus programas vía Internet [3, 4].
- **Interactivas de tiempo real:**
  - Este tipo de aplicaciones permiten que los usuarios se comuniquen entre sí mediante audio o vídeo en tiempo real.
  - Requerimientos de retardo mucho más exigentes que las clases anteriores (valores aceptables son menos de 150 milisegundos para vídeo y menos de 400 milisegundos para el audio).
  - Ejemplos: vídeoconferencia [5] o conversación telefónica (VoIP) [2].

Todas estas aplicaciones son relativamente insensibles a la pérdida de datos, siempre y cuando no degraden seriamente la calidad de la señal a transmitir. Los niveles tolerables de pérdidas de datos dependen también de la robustez del destino, pues en muchos casos se aplican en el

destino técnicas que ocultan las pérdidas de información en la imagen. Un estudio completo sobre el tema aparece en [35].

Podemos obtener las siguientes conclusiones en cuanto a las necesidades de las aplicaciones multimedia:

- Los requerimientos en cuanto a retardo serán tanto más estrictos cuanto mayor sea el grado de interactividad de la aplicación.
- Los requerimientos en cuanto al ancho de banda dependen de la calidad de la señal, y de las características de la misma (imágenes estáticas o con mucho movimiento, resolución de la imagen, etc.).
- En cualquier caso es importante que, una vez iniciada la visualización de la señal en destino, ésta se haga de manera continua, para que el usuario la perciba sin interrupciones.

Las redes de altas prestaciones empleadas en COWs y SANs hasta la fecha no incluyen ningún soporte para estas nuevas aplicaciones. Sus objetivos se limitan a ofrecer alta productividad y baja latencia media al tráfico de usuario.

1.3. Conceptos previos

En este apartado, estableceremos un conjunto de conceptos que serán referidos posteriormente a lo largo de la presente Memoria.

1.3.1. Clases de servicio

Ya hemos hablado previamente de la *calidad de servicio* (*QoS*), y hemos introducido los parámetros que reflejan los requerimientos de las aplicaciones: productividad, retardo, *jitter* y pérdida de datos. Estos parámetros reflejan la percepción que tiene el usuario de un servicio de red. La red es la responsable de mantener el nivel de QoS esperado por sus usuarios.

Puesto que considerar por separado las necesidades de cada flujo de información que pueda generar cada aplicación distinta que pueda necesitar cada usuario, es algo completamente inviable dado el número de usuarios, y el amplísimo rango de aplicaciones presentes y futuras que pueden presentarse, es necesario definir un número limitado de **clases de servicio**. Así, cuando una aplicación desee transferir información escogerá aquella clase de servicio que mejor se adapte a sus necesidades. Si no existiese una diferenciación del tráfico en clases de QoS, la red debería de soportar los requerimientos de calidad más exigentes para todo el tráfico.

Cada clase se define mediante unos requisitos específicos de QoS. En la literatura existen diversas propuestas de clasificación del tráfico. Una clasificación clásica es la introducida en ATM [8], basada en cinco categorías de tráfico:

CBR (*Constant Bit Rate*). A esta clase de tráfico pertenecerían conexiones que necesitan un ancho de banda fijo, y unos fuertes requerimientos en cuanto a latencia. Este ancho de banda está siempre disponible mientras dure la conexión, por lo que se suele emplear para servicios de emulación de circuitos. Ejemplos de aplicaciones que pueden

emplear esta clase de servicio son las conexiones telefónicas, la vídeoconferencia y audio y vídeo sin comprimir, y en general, cualquier tipo de comunicaciones interactivas multimedia, para las que es necesario garantizar cotas en el ancho de banda y/o en los retardos.

rt-VBR (*real time-Variable Bit Rate*). Esta clase de tráfico está pensada para aplicaciones sensibles al retardo que generan el tráfico en ráfagas, es decir, a una tasa de inyección variable. Sobre estas fuentes la red puede aplicar el multiplexado estadístico, concepto que veremos en el apartado 1.3.2. Cuando se comprimen flujos de audio y vídeo, como por ejemplo mediante la codificación MPEG-2, se suele generar tráfico perteneciente a esta categoría. Por tanto, y debido a que cada vez existen más y mejores técnicas de compresión, este tipo de tráfico puede convertirse en el más extendido en un futuro próximo.

nrt-VBR (*non real time-Variable Bit Rate*). A esta categoría de tráfico pertenecerían aquellas fuentes que generan tráfico a ráfagas, pero que no tienen requisitos de retardo o de variación en el retardo. Un ejemplo típico de aplicaciones que generan tráfico de este tipo es la reproducción de clips de audio o películas de vídeo.

ABR (*Available Bit Rate*). Esta clase está diseñada para el tráfico normal de datos, tales como transferencia de ficheros y correo electrónico, que no requiere garantías de servicio. Aunque no se requiere garantizar ni el retardo máximo ni un ancho de banda mínimo, es deseable que los conmutadores intenten minimizarlos tanto como sea posible. Por ello, también se suele denominar como *tráfico de mejor esfuerzo* (*best-effort traffic*). A esta clase de tráfico se le aplica control de flujo en caso de que la red esté congestionada. Ejemplos de aplicaciones que pueden utilizar esta clase de servicio son todas las convencionales: transferencia de ficheros, servicios de noticias, etc.

UBR (*Unespecified Bit Rate*). Esta clase de tráfico está diseñada para aplicaciones que utilizan cualquier capacidad sobrante de la red, y que no son sensibles a la pérdida o al retardo de la información. No se le aplica control de flujo, y en caso de congestión sus paquetes son descartados y las fuentes no reducen su tasa de inyección. Las aplicaciones que pueden usar este tipo de servicio serían las mismas que en la clase de servicio ABR.

Diversos autores han propuesto alternativas a esta clasificación. Por ejemplo, en [92] sólo se consideran tres de estas clases, puesto que no hace distinción entre las dos clases de tráfico VBR, y no considera el tráfico UBR. Otro ejemplo aparece en [131], donde se habla de cuatro clases de tráfico:

DBTS (*Dedicated Bandwidth Time Sensitive*). Este tipo de tráfico requiere un ancho de banda mínimo, y una latencia máxima determinada. Sería por tanto equivalente a las clases CBR y rt-VBR de ATM. Ejemplos de esta categoría serían aplicaciones eminentemente interactivas, como la vídeoconferencia o *Voice over IP* (VoIP).

DB (*Dedicated Bandwidth*). Esta clase de tráfico requiere un ancho de banda mínimo, pero no es demasiado sensible a la latencia. Es por tanto similar a la clase nrt-VBR de ATM.

BE (*Best Effort*). Este tipo de tráfico suele exhibir ráfagas por naturaleza, y es bastante insensible tanto al ancho de banda como a la latencia. Se puede asimilar por tanto a la categoría ABR de ATM. En esta clase se pueden encuadrar el tráfico que generan la mayoría de servicios convencionales que se suelen ejecutar en redes de comunicaciones:

servicios de ficheros y de impresión, navegación en la web, transferencia de ficheros, correo electrónico, etc.

CH (*Challeged*). El servicio que recibe el tráfico perteneciente a esta clase se degrada intencionalmente para que no afecte al tráfico BE. La idea es que esta clase de servicio sea empleada por tareas de tipo administrativo, que no deben interferir con ninguna de las demás clases empleadas por el tráfico de usuarios. Un ejemplo de aplicación de esta clase de tráfico sería la realización de copias de seguridad.

1.3.2. Compartición estadística frente a aislamiento

En un momento dado, por una red circularan varios flujos de información generados por diferentes aplicaciones y usuarios. Un *aislamiento entre los flujos de tráfico* facilita la protección de la calidad de servicio requerida para cada conexión. Esto se consigue reservando para cada conexión su ancho de banda máximo o pico. Las fuentes VBR no transmiten continuamente con ese caudal, por tanto, la capacidad del enlace estará desaprovechada durante la mayor parte del tiempo.

El objetivo que se persigue con una *compartición estadística de recursos* es el de obtener una mayor eficiencia de la red. Esto se consigue al multiplexar de forma estadística varias conexiones VBR, siendo la suma de sus anchos de banda pico mayor que la capacidad física del enlace (esto siempre y cuando la suma de sus anchos de banda medios sea inferior al caudal del enlace). Si los flujos de tráfico son numerosos e independientes, puede darse el caso de que sus anchos de banda instantáneos totales sobrepasen la capacidad del enlace (esto es, cuando muchas fuentes transmiten simultáneamente a sus tasas máximas de inyección), pero esto sucede con una probabilidad muy pequeña. Esto se expresa mediante la llamada *Ley de los Números Grandes*, que afirma que el caudal total se aproximará al caudal medio total con probabilidad uno, cuando el número de flujos independientes se hace muy grande. Para hacer realidad la ganancia por la multiplexación estadística (relación entre el ancho de banda pico total y el ancho de banda del enlace), es deseable mantener un elevado factor de utilización, y maximizar el grado de compartición estadística de los recursos de la red.

Sin embargo, una consecuencia adversa de la multiplexación estadística es que la calidad de servicio de una conexión puede verse afectada por el tráfico de otras conexiones. Por ejemplo, una ráfaga en una conexión podría saturar un *buffer* y provocar un desbordamiento del mismo. La probabilidad de desbordamiento y de retrasos excesivos en las colas es mayor cuanto mayor es la carga. Por tanto, puede ser deseable mantener un factor de utilización bajo (que puede no ser posible por motivos económicos), o de otro modo, proporcionar aislamiento entre los flujos de tráfico para reducir el efecto de las ráfagas en un flujo sobre la QoS de otro flujo.

Desafortunadamente, estos enfoques entran en contradicción con el objetivo de la multiplexación estadística.

Una manera útil de aislar o modificar los efectos colaterales entre los flujos de tráfico son las *prioridades*. Podemos especificar las prioridades en base a distintos criterios:

Prioridades de retardo Establecen el orden en que los paquetes de las colas se planifican para su transmisión por un enlace compartido.

Prioridades de pérdidas Designan quién tiene preferencia a la hora de ocupar espacio en *buffers* compartidos.

Por ejemplo, los paquetes con prioridad de retardo alta tendrán preferencia en la transmisión sobre los paquetes con baja prioridad de retardo. Por tanto, el efecto del tráfico de baja prioridad sobre los retardos experimentados por el tráfico de alta prioridad es menor. Sin embargo, esto afecta en gran medida a los retardos experimentados por el tráfico de baja prioridad. Este ejemplo pone de relieve que las prioridades sólo especifican el trato preferencial de una clase de tráfico con respecto a otra: una clase de tráfico se beneficia a expensas de otra clase.

La red deberá trabajar necesariamente con una configuración que suponga un compromiso entre lograr una utilización eficiente y la protección de la QoS de los diferentes flujos de datos que la atraviesan. Este punto de trabajo dependerá también de la cantidad de ráfagas que tengan los flujos de datos (*burstiness*) y de lo predecibles que éstas sean.

1.3.3. Control reactivo y preventivo

En redes de conmutación de paquetes, existe la posibilidad de que la tasa agregada del tráfico que entra en la red (o en una parte de la red) sobrepase temporalmente la capacidad de la misma, en cuyo caso los paquetes pueden experimentar largos retardos, o ser descartados por la red. Esta situación se denomina *congestión*.

Se han propuesto varios tipos de algoritmos de control de la congestión o de gestión del tráfico en la literatura (en [83] hay una completa recopilación de ellos). Estas soluciones pueden clasificarse en dos clases: esquemas reactivos, o de control con realimentación, y esquemas preventivos, o algoritmos de reserva de recursos.

Las redes de datos convencionales emplean *mecanismos de control reactivos*. Estas técnicas están basadas en detectar una situación de congestión y tomar acciones encaminadas a reducirla. La congestión se detecta a través de la información de realimentación proporcionada por la red, y por tanto, actúan en una escala de tiempo del orden de varios retardos de propagación de ida y vuelta entre extremos (*round-trip time, RTT*). En la especificación del protocolo TCP se incluyen mecanismos de control de este estilo [16].

Sin embargo, estas técnicas presentan dos inconvenientes. En primer lugar, no son apropiadas para fuentes de tiempo real, que generalmente no pueden ser controladas por la red. En segundo lugar, la efectividad del control basado en realimentación se halla limitado fundamentalmente por el retardo de propagación. El tiempo empleado en transmitir un paquete es mucho más corto que el tiempo necesario para detectar una situación de congestión, y de notificárselo a la fuente para que reaccione. Las fuentes de alta velocidad son capaces de inyectar demasiados paquetes en la red antes de que la información de realimentación pueda propagarse a través de la red para controlarlas.

Los *métodos preventivos*, por el contrario, operan al menos a dos escalas de tiempo: a nivel de conexión y a nivel de paquete. Los *métodos preventivos a nivel de conexión* actúan cuando llega una nueva petición de conexión. Su misión es comprobar en cada conmutador una serie de condiciones para su admisión, encaminadas a verificar si hay suficientes recursos para atender tanto la nueva conexión como las conexiones existentes, al nivel de QoS requerido. Como resultado de estas comprobaciones, la nueva conexión será admitida, efectuando la reserva de los correspondientes recursos, o bien, será rechazada. Estos métodos se suelen denominar **Control de Admisión de Conexiones (CAC)**.

En los *métodos preventivos al nivel de paquetes*, la disciplina de servicio de paquetes de cada conmutador selecciona qué paquete se transmitirá a continuación según sus requisitos de prestaciones.

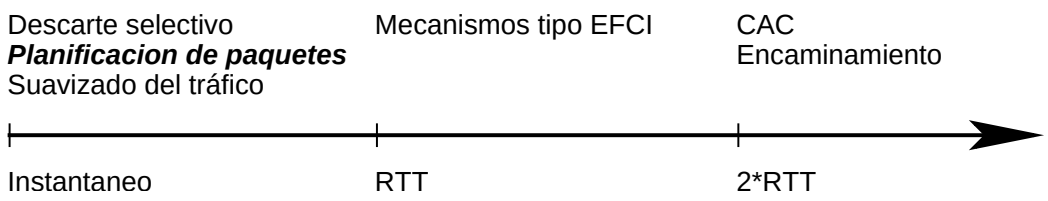


Figura 1.1: Mecanismos de control del tráfico.

Puesto que ambos niveles de control (a nivel de conexión y a nivel de paquete) se hallan muy relacionados entre sí, una solución completa debe especificar tanto la disciplina de servicio como las condiciones para la admisión de conexiones.

La disciplina de servicio se halla muy relacionada con el *control de parámetros de uso*. Este control se emplea una vez establecida con éxito una conexión, para regular la cantidad de tráfico que entra en la red. También se denomina *vigilancia de las entradas* (*input policing*), pues se trata de vigilar que una conexión inyecta tráfico en la red de acuerdo con lo especificado en su establecimiento.

Los controles de tipo reactivo son más adecuados para soportar servicio *best-effort* o ABR, mientras si se pretende dar garantías de calidad de servicio, es mejor adoptar métodos preventivos. Ambos enfoques pueden coexistir dentro de una red que pretenda soportar los distintos tipos de servicio descritos.

1.3.4. Niveles de control

En redes de comunicaciones orientadas a la conexión, los flujos de tráfico pueden verse a diferentes niveles, identificados por distintas entidades de tráfico:

- Llamadas
- Ráfagas (que consisten en conjuntos de paquetes consecutivos)
- Paquetes individuales

Cada mecanismo de control actúa en una escala de tiempo característica. Generalizando lo expuesto en la sección anterior, una clasificación de los mecanismos de control del tráfico conforme a sus escalas de tiempo se muestra en la Figura 1.1.

Los mecanismos que operan sobre paquetes individuales son los más instantáneos, pues las decisiones de control dependen sólo de las condiciones locales en un conmutador. Por ejemplo, el descarte selectivo de paquetes depende del nivel de congestión de los *buffers* del conmutador.

En cambio, otros mecanismos operan a través de la red sobre una escala de tiempo del orden de los retardos de propagación entre extremos. Estos mecanismos implican el paso de información en un sentido entre dos puntos situados sobre una conexión virtual. Un ejemplo es el mecanismo denominado EFCI (*Explicit Forward Contention Indicator*), empleado en ATM, donde la detección de la congestión se lleva dentro de las cabeceras de los paquetes que viajan por un camino virtual.

Otros mecanismos operan sobre otra escala de tiempo aún mayor, como el control de admisión de conexiones. En este caso, se intenta el establecimiento de una nueva conexión mediante

el intercambio de mensajes de señalización, y los nodos situados a lo largo de la ruta pueden aceptar o rechazar la petición.

En el desarrollo de esta Tesis se prestará especial atención al desarrollo de algoritmos de planificación apropiados para ofrecer las prestaciones requeridas por los distintos tipos de tráfico. Además, se perseguirá la obtención de algoritmos sencillos, que sean susceptibles de una eficiente implementación en hardware.

1.4. Desarrollo de la Tesis

En esta Tesis Doctoral se aborda el problema de ofrecer garantías de servicio a las aplicaciones multimedia mediante el diseño de un encaminador compacto, pensado para su uso en entornos de área local y de sistema.

En el **Capítulo 2** se expone con detalle la motivación subyacente tras el trabajo desarrollado en esta Tesis, señalando los problemas existentes y no resueltos todavía de forma completamente satisfactoria. También se concretan los objetivos a cubrir durante el desarrollo de la Tesis.

En el **Capítulo 3** se efectúa una exposición del estado del arte actual en lo que se refiere a algoritmos de planificación de conmutadores. Se describen las diferencias en la planificación según la organización del conmutador, más concretamente según la situación de los *buffers*. Se establece una clasificación de los mismos, y se expone con cierto nivel de detalle un ejemplo de cada clase. Para acabar el capítulo, se describen un par de algoritmos de planificación empleados en encaminadores para SAN y redes de multicomputadores.

En el **Capítulo 4** se introduce la arquitectura de conmutador propuesta para satisfacer las garantías de QoS que necesitan las nuevas aplicaciones multimedia: el *encaminador multimedia MMR*. Se discuten las distintas consideraciones tenidas en cuenta para su diseño. Los algoritmos de planificación se introducen en este Capítulo de un modo genérico, pues seguidamente se describen con más detalle los distintos algoritmos de planificación desarrollados para su uso en este nuevo encaminador. Esto sucede en el **Capítulo 5**.

Una vez descrita la arquitectura del nuevo encaminador, y los algoritmos de planificación del tráfico propuestos, es el turno de evaluar las prestaciones ofrecidas. En el **Capítulo 6** se lleva a cabo una evaluación detallada de la arquitectura propuesta, analizando todas las opciones posibles, y considerando cargas compuestas por distintos tipos de tráfico, tanto multimedia como convencional. El objetivo es verificar que el diseño de encaminador propuesto es capaz de proporcionar la QoS requerida por los flujos multimedia, independientemente de la presencia de tráfico convencional sin requerimientos de calidad. Así mismo, será deseable que este tráfico no multimedia obtenga una elevada productividad y buenas prestaciones medias.

Finalmente, se dedica el **Capítulo 7** a concretar las conclusiones que se pueden extraer del desarrollo de la Tesis. Asimismo, se detallarán brevemente los artículos publicados en diversos Congresos y Simposios tanto nacionales como internacionales, fruto del trabajo realizado. Y por último, se indican los caminos que se quedan abiertos en esta Tesis para la investigación futura.

Antes de continuar con el desarrollo de esta memoria, es necesario un pequeño inciso para clarificar la terminología empleada. A lo largo de esta memoria, se empleará el término "encaminador" para identificar un elemento de interconexión en una red, es decir, un elemento capaz de "encaminar" mensajes con el objeto de que lleguen a su destino. El término se emplea comúnmente en redes de multicomputadores y multiprocesadores (por ejemplo, al hablar de encaminadores *wormhole*), para designar aquellos elementos que implementan la

funcionalidad de una capa de conmutación de mensajes [51]. Este concepto no debe de confundirse con el que se emplea usualmente en el ámbito de las redes de computadores, donde un encaminador es un dispositivo de interconexión que funciona al nivel de red del modelo OSI [155], es decir, que trasiega paquetes entre redes distintas [34]. Un ejemplo de esto sería un encaminador IP. Se podría decir que el concepto de encaminador tal y como se entiende a lo largo de este trabajo, es asimilable a un dispositivo que trabaja al nivel OSI de enlace de datos.

Capítulo 2

Motivación y Objetivos

2.1. Antecedentes

En el Capítulo previo se ha resumido el proceso de migración de técnicas y métodos empleados en redes de multicomputadores y multiprocesadores al entorno de área local. El resultado es el uso de redes y grupos de estaciones de trabajo unidos por interconexiones de altas prestaciones como alternativa de bajo coste a los tradicionales supercomputadores o computadores masivamente paralelos. Una característica común a todos estos elementos de interconexión (conmutadores o encaminadores) es el empleo de técnicas de conmutación tipo *cut-through*, bien sea *wormhole* (como el *Reliable Router* [43], Cray T3D/E [144, 145], SGI Spider [57], ServerNet [73, 58] o Myrinet [25]) o *Virtual Cut-Through* (VCT) (como el *Chaos Router* [95], el S-Connect de Sun [122], o HAL Mercury [170]).

Los conmutadores y encaminadores del tipo de los reseñados anteriormente están optimizados para la obtención de alta productividad y baja latencia media cuando el tráfico se compone de datos que no requieren de ninguna garantía de Calidad de Servicio. Esto es, están diseñados con la transmisión eficiente de tráfico *best-effort* como objetivo. Sin embargo, también se ha puesto de manifiesto en el Capítulo anterior el gran auge experimentado por aplicaciones que manejan datos multimedia. Muchas de estas aplicaciones necesitan de los recursos computacionales de un grupo o *cluster* de estaciones de trabajo. Algunos ejemplos son los servidores de vídeo bajo demanda (*Video on Demand, VoD*), o los servidores de bases de datos con información gráfica de cualquier tipo, como los Sistemas de Información Geográfica (GIS). También es habitual que se trate de aplicaciones distribuidas por naturaleza (como vídeoconferencias o juegos multiusuario en red). En cualquier caso, es necesario que la red de interconexión subyacente ofrezca soporte a las necesidades de QoS de dichas aplicaciones multimedia.

En los últimos años, se han propuesto diversas arquitecturas de encaminadores de altas prestaciones. Por ejemplo, el **conmutador HIPIQS** (*HIgh-Performance Input Queued Switch*) [153] se basa en una organización con *buffers* a la entrada. Sin embargo, el objetivo perseguido en su diseño seguía consistiendo en maximizar la productividad y minimizar la latencia media del tráfico convencional de datos. En concreto, la idea era solventar los problemas de productividad inherentes a los conmutadores con *buffers* a la entrada debidos al efecto del bloqueo de la cabeza de línea (*HOL-blocking*), así como desarrollar un sistema lo suficientemente versátil como para ofrecer buenas prestaciones tanto en entornos de interconexión de sistemas paralelos como en entornos LAN.

De los encaminadores y conmutadores reseñados anteriormente, sólo los conmutadores de ServerNet incluyen un mecanismo de arbitraje (el arbitraje *ALU-biasing*) que intenta ejercer algún control sobre el reparto de ancho de banda del conmutador entre los distintos enlaces en entrada, pero no es suficiente como para soportar tráfico multimedia.

En los últimos años el creciente interés en ofrecer un soporte hardware a la QoS en redes de interconexión se ha plasmado en una serie de propuestas interesantes. En el diseño del **conmutador Switcherland** [52] se incluye algún soporte a la QoS de los flujos multimedia. Pensado como interconexión con memorias en sistemas multiprocesador, en su diseño se consideraron dos tipos de tráfico, que los autores denominan CBR y VBR. El tráfico CBR es similar a la clase de tráfico CBR tal y como lo plantea el ATM Fórum, con una tasa de generación constante. Aplicado al entorno para el que se pensó el Switcherland, la idea es permitir que los accesos a memoria se realicen a una tasa dada y en determinados momentos de tiempo. Esto sería útil por ejemplo para acceder a imágenes de vídeo almacenadas en memoria o en algún dispositivo de entrada/salida. Los autores consideran una segunda clase de tráfico, que no tiene requerimientos de ancho de banda ni de latencias, a la que denominan VBR (no confundir con la clase de tráfico VBR definida por el ATM Fórum). El conmutador da una prioridad absoluta al tráfico CBR sobre el VBR, lo cuál puede provocar retrasos indefinidos a esta última clase de tráfico.

El conmutador Switcherland realiza una aproximación tímida al problema del soporte de transmisiones con requerimientos de QoS, aunque se trata de una solución pensada para un entorno limitado, y que carece de la flexibilidad suficiente como para acomodar el tráfico de las diversas clases de servicio descritas en el Capítulo previo.

La arquitectura del **conmutador Yuni** [175] ofrece soporte a la Calidad de Servicio de las distintas aplicaciones mediante una sofisticada organización donde existen *buffers* asociados tanto a los puertos de entrada como a los de salida, con planificadores distintos en cada uno de ellos, para decidir las células o paquetes que han de atravesar el *crossbar* o ser retransmitidos por los puertos de salida. El *crossbar* funciona al doble de velocidad que los enlaces físicos. Además, este conmutador proporciona canales para el soporte de un servicio TDM (*Time Division Multiplexing*), con garantías absolutas de ancho de banda y *jitter* nulo. Se trata por tanto de un conmutador complejo, pensado para su uso como plataforma integradora de tráfico de conmutación de paquetes y de circuitos, o como conmutador multi-terabit válido para su uso dentro del núcleo de Internet. Por tanto, no resulta adecuado para las necesidades más modestas de un entorno LAN.

El **encaminador MediaWorm** [174, 173] es una interesante aproximación a la provisión de garantías de QoS empleando conmutación *wormhole*. Al contrario de lo que sucede en los enfoques orientados a la conexión, que reservan los recursos previamente a la conexión de datos, cuando se emplea conmutación *wormhole*, los recursos necesarios para la trasmisión de datos se van reservando progresivamente según avanza el mensaje por la red. En el caso de que dichos recursos estén ocupados, el mensaje detiene su avance en espera de la liberación de dichos recursos. Esto puede introducir tiempos de espera potencialmente no acotados y extremadamente variables, lo que perjudicaría sensiblemente las prestaciones y la QoS recibida por las aplicaciones multimedia.

Con el encaminador MediaWorm, se intenta paliar estos problemas adjudicando el ancho de banda de los enlaces a los distintos mensajes que lo atraviesan en un momento dado, en función de una indicación de los requerimientos de ancho de banda que necesita el mensaje. Esta indicación es un campo en la cabecera del mensaje. El encaminador aplica el algoritmo *Virtual Clock* [178] para seleccionar el canal virtual de cada puerto de entrada que va a transmitir sus flits a través del *crossbar* y del puerto de salida correspondiente. Al tráfico *best-effort* se le asignan valores de prioridad mínimos, con el objeto de que sus mensajes

sólo sean transmitidos cuando no hayan mensajes multimedia por transmitir. Sin embargo, y puesto que los mensajes correspondientes al mismo flujo de información se tratan de forma independiente, este enfoque basado en *wormhole* puede introducir problemas como entrega desordenada de dichos mensajes, o valores de *jitter* elevados. Además, existe otro problema, y es que puesto que *wormhole* no es una técnica orientada a la conexión, no existe una fase de reserva de recursos previa a la transmisión de los datos. Por tanto, la implementación de un control de admisión de conexiones resulta prácticamente inviable. Esto puede llegar a comprometer la QoS recibida por los flujos de información, especialmente cuando el ancho de banda que demandan los flujos de datos sobrepase la capacidad de los enlaces.

El problema de la provisión de distintas garantías de servicio a distintos mensajes dentro de encaminadores *wormhole* ha sido tratado por varios autores, que proponen diversas soluciones. En [139] se propone un esquema donde se adjudica una mayor prioridad a ciertos mensajes dentro del encaminador. Sin embargo, este método sólo resulta adecuado para soportar la transmisión de mensajes con necesidades de tiempo real, y no es válido cuando se trata de flujos continuos de información. En [99] se propone un esquema de reserva de ancho de banda en un encaminador *wormhole*, pero siempre aplicado al tráfico *best-effort*. El estudio de [63] analiza diversas estrategias para la provisión de QoS en redes *wormhole*. Una de ellas consiste en emplear una red exclusivamente para el tráfico de tiempo real, lo que probablemente sea prohibitivo en coste pues cada *host* debería de emplear dos interfaces de red: uno para la conexión a la red con QoS y otro para la conexión a la red de tráfico *best-effort* normal. Otra opción es implementar una red síncrona (por ejemplo, FDDI [172]) sobre una red asíncrona (como Myrinet [25], por ejemplo). Sin embargo, esta opción produce retardos innecesariamente elevados a cargas bajas, y además no es escalable, pues el aumento en el número de *hosts* perjudica el rendimiento de la red, y las garantías de QoS. Una última opción es disponer de conjuntos de canales virtuales distintos para tráfico con necesidades de QoS y sin ellas, de manera que los primeros tengan prioridad (preemptiva o no) sobre los últimos. Aún así, no se provee de la flexibilidad suficiente como para soportar los diferentes tipos de tráfico multimedia.

Recientemente, un grupo de empresas líderes en el campo de las Tecnologías de la Información ha propuesto lo que pretende ser un nuevo estándar de interconexión entre nodos de procesamiento y dispositivos de entrada/salida, y para la comunicación entre procesadores. Se trata de la **Arquitectura InfiniBand (IBA)** [1], y está siendo desarrollada por la *InfiniBand*^{TM} *Trade Association* (IBTA). La IBTA se constituyó en Agosto de 1999, y es un grupo de más de 200 empresas, lideradas por un comité compuesto por miembros de Dell, Compaq, HP, IBM, Intel, Microsoft, y Sun. Empresas como 3Com, Cisco Systems, Fujitsu-Siemens, Hitachi, Adaptec, Lucent Technologies, NEC y Nortel Networks actúan como patrocinadores. La IBTA publicó la primera especificación de la Arquitectura InfiniBand en Octubre del 2000 [19].

El objetivo perseguido por la Arquitectura InfiniBand (IBA en adelante) es proponer una alternativa a los buses estándar, para la interconexión de procesadores entre sí y con dispositivos de entrada/salida. Dichos buses se convierten en el cuello de botella de servidores, además de presentar problemas de baja disponibilidad [135]. Por tanto, IBA define una red de área de sistema (SAN) independiente de los sistemas operativos de los *hosts* y de la plataforma en concreto. La unidad básica de comunicación en IBA es el *mensaje*, los cuales se segmentan en *paquetes* para su transmisión a través de enlaces y conmutadores. En la especificación se definen varios tamaños de paquete, que oscilan entre los 256 bytes y los 4 KB, en los que se incluyen las cabeceras correspondientes. La Arquitectura de InfiniBand incluye una serie de mecanismos que, convenientemente usados, pueden servir para proveer de QoS a las aplicaciones multimedia. A continuación, se describen someramente estos mecanismos.

InfiniBand soporta el esquema de canales virtuales en forma de *Virtual Lanes* (VL). Los conmutadores pueden implementar entre 1 y 16 VLs, estando el VL 15 reservado para el tráfico de mantenimiento de la red. Al tráfico que circula por este canal se le adjudica siempre la máxima prioridad. Además, se incluyen hasta 16 Niveles de Servicio (SLs, *Service Levels*), pero se dejan sin especificar las características que debe tener el tráfico de cada uno de ellos. Cada paquete va marcado con un nivel de servicio, y en cada conmutador existe una tabla de correspondencias entre niveles de servicio y canales virtuales.

La relación de prioridades entre los distintos canales virtuales se establece mediante una *tabla de arbitraje*. Parte de dicha tabla se dedica a paquetes de "alta prioridad", y el resto a paquetes de "baja prioridad". Sin embargo, el estándar no define lo que es "alta" y "baja" prioridad. En esta tabla se incluyen una serie de pesos con los que debe ser atendido cada canal virtual, siguiendo un esquema cíclico ponderado (*weighted round-robin*).

La propuesta de mecanismos de provisión de QoS sobre la arquitectura propuesta por InfiniBand es un tema de interés muy reciente [131, 15, 14]. Los mecanismos propuestos hasta la fecha soportan de manera eficiente el tráfico CBR, pero su extrapolación a tráfico VBR es todavía tema de estudio.

2.2. Motivación

En la Sección previa se han revisado diversas propuestas de arquitecturas de encaminador orientadas a proveer Calidad de Servicio a aplicaciones multimedia. Sin embargo, se ha podido comprobar cómo no existe ninguna solución concluyente al problema.

Muchas arquitecturas de conmutadores de altas prestaciones están orientadas a maximizar la productividad máxima, sin tener en cuenta los diferentes requerimientos impuestos por los distintos flujos de datos. Es decir, se trata de encaminadores de altas prestaciones, pero orientados exclusivamente a tráfico convencional de datos *best-effort*.

Existen propuestas de conmutadores basados en *wormhole*, con un cierto soporte a la provisión de prestaciones diferenciadas a los distintos tipos de tráfico. No obstante, siguen presentando los problemas inherentes al uso de esta técnica de conmutación. Esto es, los valores de *jitter* siguen sin estar limitados, no se puede garantizar la entrega en orden de los mensajes pertenecientes al mismo flujo de información, y al no existir una reserva de recursos previa a la transmisión de datos, es posible sobrecargar la red, comprometiendo así la QoS recibida por las aplicaciones multimedia.

Por último, existen algunas arquitecturas de conmutador, como Yuni, que sí que ofrecen un soporte adecuado a un amplio rango de aplicaciones con diversos requerimientos de QoS. Sin embargo, se trata de encaminadores excesivamente complejos y costosos para un entorno local.

Por otro lado, dentro del marco global de la arquitectura, uno de los principales elementos involucrados en la provisión de QoS a los distintos flujos de tráfico es el algoritmo de planificación del tráfico. Hay que recordar que este algoritmo es el que decide qué paquetes de información se transmiten en cada instante de tiempo, luego es quien tiene potestad para adelantar o retrasar la transmisión de un paquete con el objeto de hacer cumplir las garantías de calidad de todos los flujos.

Tal y como se desarrollará con más detalle en el Capítulo 3, tampoco existe una propuesta definitiva de algoritmo de planificación del tráfico capaz de proveer la QoS suficiente a las aplicaciones, que además sea susceptible de ser implementado eficientemente en hardware. Hay que tener en cuenta que la mayoría de las soluciones existentes en la literatura, están

pensadas para su uso en conmutadores con *buffers* a la salida. Sin embargo, esta organización no resulta óptima para una red de alta velocidad pues requiere que internamente, el *switch fabric* funcione a una frecuencia igual a la de los enlaces físicos, multiplicada por el número de puertos. Cuando los enlaces físicos funcionan a frecuencias cercanas al límite impuesto por la tecnología, resulta inviable que el conmutador funcione internamente todavía a una frecuencia varias veces superior [107].

En los últimos años se han realizado propuestas de planificadores para otras organizaciones más convenientes en redes de alta velocidad, concretamente organizaciones que disponen de *buffers* asociados a los puertos de entrada. Aún así, la inmensa mayoría de ellas están orientadas a maximizar la utilización del conmutador y de los enlaces, y no a la provisión de QoS. Esto es, la inmensa mayoría de los algoritmos de planificación para conmutadores de redes de alta velocidad propuestos hasta la fecha están diseñados para soportar eficientemente el tráfico *best-effort* convencional. Por tanto, no son válidos cuando se trata no ya de conseguir transmitir la mayor cantidad de información posible, sino de que ésta llegue a su destino alcanzando una ciertas prestaciones definidas a priori.

Últimamente, se han realizado algunas propuestas de planificadores que intentan proporcionar Calidad de Servicio a las aplicaciones que lo necesitan. Sin embargo, éste es un tema de reciente interés, y no dejan de ser aproximaciones más o menos teóricas al problema, difícilmente implementables con un hardware rápido y compacto de alta velocidad.

2.3. Objetivos

Según lo expuesto en la Sección anterior, el primer objetivo global perseguido en el desarrollo de esta Tesis es la **propuesta de una arquitectura de encaminador**, diseñada especialmente para su uso en entornos de *clusters* o redes locales de estaciones de trabajo (COWs/NOWs), que sea capaz de soportar adecuadamente tanto el tráfico multimedia generado por las nuevas aplicaciones, como el tráfico convencional de datos generado por aplicaciones tradicionales. Además, se trata de ofrecer garantías de QoS al tráfico multimedia.

Los objetivos que debe cumplir el encaminador diseñado se pueden desglosar en los siguientes puntos:

- Provisión de garantías de QoS a las aplicaciones multimedia
- Adjudicación eficiente del ancho de banda no utilizado por los flujos multimedia al tráfico convencional de tipo *best-effort*
- Maximización de la utilización de los enlaces
- Diseño lo suficientemente simple como para ser implementado eficientemente en hardware de alta velocidad. Se debería tender a conseguir una implementación mono-chip.

Durante el desarrollo de esta Tesis, se concretarán las características que definen la arquitectura de encaminador propuesta: organización del conmutador, técnica de conmutación empleada, algoritmos de arbitraje y planificación del tráfico, algoritmos de encaminamiento, algoritmos de control de admisión de conexiones, etc.

El Capítulo 4 de esta memoria se dedica a la presentación de la arquitectura de encaminador propuesta, el Encaminador Multimedia, bautizado como **MMR (*MultiMedia Router*)**.

En él se desglosan y se justifican todas las decisiones tomadas en su diseño, ponderando debidamente las diferentes alternativas que se presentan.

Una vez establecido el marco global de la arquitectura, se prestará mayor atención a uno de los principales elementos involucrados en el diseño del Encaminador, el *algoritmo de planificación del tráfico*, pieza que juega un papel clave a la hora de proveer las garantías de QoS que necesitan las aplicaciones multimedia, y de maximizar la productividad global alcanzada por el encaminador.

Como segundo objetivo global de la Tesis se plantea el **diseño de un algoritmo planificador del tráfico**, capaz de proporcionar la QoS necesaria a los flujos multimedia, y lo suficientemente simple como para ser capaz de calcular la planificación del tráfico a velocidades comparables a la de conmutación en encaminadores *cut-through* convencionales.

En el Capítulo 5 se describen detalladamente los diversos algoritmos de planificación desarrollados para el Encaminador propuesto. También se incluye una estimación de los costes de su implementación en hardware, para justificar así la viabilidad de su implementación dentro de la arquitectura general propuesta.

Capítulo 3

El Problema de la Planificación del Tráfico en Conmutadores

3.1. Introducción

3.1.1. Algoritmo de planificación del tráfico

En el Capítulo 1 se comentaban los cuatro parámetros que generalmente se consideran para especificar la Calidad de Servicio: ancho de banda, latencia o retardo, *jitte*r o variación en la latencia y pérdida de datos. También se señalaban los requisitos de algunas de las aplicaciones multimedia más extendidas en términos de estos parámetros.

Los cuatro parámetros anteriores se ven afectados por un tema en común: la *congestión*, es decir, la contención por el uso de los recursos de un sistema que son limitados. El tráfico puede verse congestionado mientras espera a que se libere un recurso necesario. El control de la congestión ha sido un tema de intenso debate durante varios años. Existen diversas y variadas propuestas y aproximaciones a este tema (en [83] se puede encontrar una recopilación de diversas contribuciones al respecto). Sin embargo, es interesante hacer notar que la mayoría de estas propuestas requieren, o pueden mejorarse, con un diseño correcto de un pequeño pero crítico componente de una red: el **algoritmo de planificación del tráfico**. Veremos a continuación la razón de su importancia en cualquier esquema de control de la congestión.

Las redes telefónicas estaban basadas en la idea de la *conmutación de circuitos*. Cada aplicación debería ser capaz de determinar su caudal pico de transmisión, para establecer un camino dedicado entre los nodos extremos. La red se hace responsable de reservar suficientes recursos a lo largo de este camino, de manera que los datos puedan ser transmitidos como un flujo continuo de caudal igual al caudal pico de transmisión. Dicha reserva se lleva a cabo normalmente por medio de una multiplexación por división en el tiempo (TDM) o en frecuencia (FDM). La característica más sobresaliente de esta aproximación es que cuando la suma de los caudales pico de transmisión iguala la capacidad del enlace, ya no se pueden admitir más conexiones en dicho enlace.

Sin embargo, existen aplicaciones (por ejemplo, el tráfico de datos y algunas aplicaciones en tiempo real, como el vídeo comprimido) que tienen un caudal de transmisión pico mucho más elevado que su caudal medio. Por tanto, el empleo de conmutación de circuitos desperdiciaría los recursos de la red, obteniendo una utilización muy pobre del ancho de banda del enlace. Para solucionar este inconveniente, se introdujeron las redes de *conmutación de paquetes* o

store-and-forward como alternativa a las redes de conmutación de circuitos. Esta técnica se basa en el principio clave del *multiplexado estadístico*. La idea es que las distintas fuentes de datos que comparten un mismo canal compartido se reparten su ancho de banda en función de su demanda. Esto es, el canal nunca permanecerá ocioso mientras alguna fuente tenga algo que transmitir. Además, no se asigna de manera predeterminada en qué instantes de tiempo le toca transmitir a cada fuente. Para evitar que una fuente se haga con todo el ancho de banda del canal, se define el **paquete** como el tamaño del bloque de datos que una fuente puede transmitir en un momento dado. Si solamente una fuente tiene datos que transmitir, podrá enviarlos como una secuencia de paquetes sin interrupción. Pero si hay más de una fuente activa, sus paquetes se transmitirán de manera intercalada por el canal. La decisión sobre qué paquete será el siguiente a transmitir se puede llevar a cabo de muchas formas, y atendiendo a diferentes criterios [134].

Una consecuencia del empleo del multiplexado estadístico es que la red puede operar de forma eficiente en la mayoría de las situaciones, pero la congestión puede degradar sus prestaciones durante intervalos de tiempo más o menos cortos. Por ejemplo, los usuarios pueden llegar a sincronizarse y enviar simultáneamente ráfagas a su caudal pico en los mismos intervalos de tiempo, de tal forma que la red necesitará grandes *buffers* para absorber estas ráfagas. También puede ocurrir que los usuarios se comporten de forma anómala o sean avariciosos, intentando hacerse con más ancho de banda del que les corresponde.

Para evitar una degradación en sus prestaciones, y el desperdicio de sus recursos, la red debe soportar un mecanismo que prevenga la congestión, o bien se recupere de ella. Este mecanismo debe controlar la cantidad de servicio ofrecido a las diferentes conexiones en los conmutadores a lo largo del camino de comunicación. Para controlar el servicio ofrecido a cada conexión, uno de los aspectos más importantes es determinar cuál será el siguiente paquete que se transmita a través de un elemento de conmutación. Esta es precisamente la misión del *algoritmo de planificación del tráfico*.

3.1.2. Características de una disciplina de servicio

Las disciplinas de servicio o algoritmos de planificación de paquetes operan en la menor escala de tiempo, esto es, con la mayor frecuencia. Junto con el control de admisión de conexiones, proporcionan los dos componentes más importantes de una arquitectura preventiva de gestión del tráfico. Mientras que los algoritmos de control de admisión de conexiones *reservan* recursos durante el tiempo de establecimiento de una conexión, las disciplinas de servicio de paquetes *asignan* los recursos de acuerdo con la reserva, durante la fase de transmisión de datos. Incluso con esquemas de control del tráfico reactivos, una planificación adecuada en los conmutadores hará que el control entre extremos sea más efectivo.

Existen tres tipos de recursos a asignar:

- **Ancho de banda**: Qué paquetes se transmiten
- **Urgencia (*promptness*)**: Cuándo se transmiten dichos paquetes
- **Espacio de almacenamiento**: Qué paquetes se descartan

Estos recursos afectan a su vez a tres parámetros de prestaciones: productividad, retardo y tasa de pérdidas.

Las características deseables de una disciplina de servicio son la eficiencia, capacidad de protección, flexibilidad y simplicidad.

Eficiencia Si queremos garantizar ciertos requerimientos de prestaciones, necesitamos una política de admisión de conexiones que limite la carga de tráfico garantizado en la red, o que limite el número de conexiones que puedan aceptarse. Una disciplina de servicio es más eficiente que otra si puede proporcionar las mismas garantías de prestaciones entre extremos bajo una mayor carga de tráfico con servicio garantizado. Una disciplina de servicio eficiente da como resultado una mayor utilización de la red.

Protección Los servicios garantizados requieren que la red proteja a los clientes que se comportan ateniéndose a sus reservas frente a tres fuentes de variabilidad: usuarios con comportamiento anómalo, fluctuación en la carga de la red, y tráfico *best-effort*. Tanto los usuarios con comportamiento anómalo como los equipos que funcionan mal pueden inyectar paquetes en la red a mayor tasa que la declarada. Además, las fluctuaciones en la carga de la red pueden provocar que para una conexión, la tasa instantánea de llegada a algún conmutador sea mayor, incluso aunque cumpla con las restricciones de tráfico a la entrada de la red. Otra fuente de variación es el tráfico *best-effort*. Aunque la carga compuesta de tráfico garantizado se encuentra limitada por el control de admisión de conexiones, los paquetes *best-effort* no tienen restricciones. Es esencial que la disciplina de servicio sea capaz de satisfacer los requerimientos de prestaciones para los paquetes pertenecientes a los clientes que se comportan correctamente, incluso en presencia de usuarios anómalos, fluctuación en la carga de la red y tráfico *best-effort* no restringido.

Flexibilidad El servicio con garantías de prestaciones debe ser capaz de soportar aplicaciones con diversas características y requisitos de QoS. Por ejemplo, las características de la visualización científica son muy diferentes de las del vídeo; incluso para el vídeo, las aplicaciones de vídeoconferencia, cine y televisión de alta definición, necesitan distintas QoS. Además existen otros factores, como los diferentes algoritmos de codificación, diferentes resoluciones, etc. que contribuyen a la diversidad de requerimientos para fuentes de vídeo. Debido a la gran diversidad de características de tráfico y requerimientos de prestaciones de las aplicaciones existentes, así como el desconocimiento de las futuras, la disciplina de servicio debería ser lo suficientemente flexible como para ser capaz de asignar diferentes cantidades de retardo, ancho de banda y tasa de pérdidas a las diferentes conexiones con servicios garantizados.

Simplicidad La disciplina de servicio debería ser conceptualmente simple, para facilitar su análisis, y mecánicamente simple, de manera que permita una implementación de alta velocidad.

3.2. Clasificación de los algoritmos de planificación del tráfico

Los algoritmos de planificación del tráfico propuestos hasta la fecha pueden dividirse en dos categorías principales, *basados en prioridades* y *basados en tramas*, según la granularidad de la planificación [92].

En los *esquemas basados en prioridades*[1], la prioridad de cada paquete, establecida en base al ancho de banda reservado y las cotas de latencia requeridas, se compara para decidir cuál se transmite en primer lugar. Este enfoque proporciona mayor flexibilidad y mejores cotas de latencia, pero requiere una lógica de control más complicada en el conmutador.

[1] Estos esquemas también aparecen en la literatura como *mecanismos de prioridades ordenadas* o *esquemas basados en time-stamp*.

Por el contrario, los *esquemas basados en tramas* (*frames*) emplean tramas de tamaño fijo. Cada trama se divide en varios *slots* de paquetes. Al reservar un cierto número de *slots* por tiempo de trama, las conexiones tienen garantizados ancho de banda y cotas de latencia. Este enfoque permite un control más simple del conmutador, pero a veces proporciona solo una controlabilidad limitada.

La mayoría de los algoritmos de planificación basados en prioridades comparten disciplinas comunes, pero pueden parecer bastante diferentes según sus motivaciones y las aplicaciones para las que están orientados. Para una mejor comprensión, estos esquemas se subdividen a su vez en *esquemas orientados al retardo*, y *esquemas orientados al ancho de banda*, dependiendo de si su objetivo principal es proporcionar cotas de retardo o garantías de ancho de banda. Aunque las cotas en retardo y en ancho de banda están fuertemente relacionadas entre sí, estos distintos puntos de arranque pueden provocar que los algoritmos resultantes tengan un aspecto muy diferente.

En cambio, en los esquemas basados en tramas, las garantías de ancho de banda y las cotas a los retardos están acoplados a través de las tramas. Esta clase de esquemas se subdividen a su vez en *esquemas que conservan el trabajo* (*work-conserving*) y *esquemas que no conservan el trabajo* (*non-work-conserving*), dependiendo de si cada conmutador efectúa control del caudal. En los conmutadores que conservan el trabajo, si hay paquetes pendientes dirigidos a canales desocupados, se transmiten inmediatamente por los canales. En cambio, los conmutadores que no conservan el trabajo pueden provocar retrasos adicionales a paquetes incluso si los canales de salida a los que van dirigidos están libres. A pesar de que las disciplinas de servicio que no conservan el trabajo pueden desperdiciar ancho de banda, simplifican el control de los recursos de la red al limitar de forma estricta el caudal de salida del tráfico en cada conmutador.

En la Tabla 3.1 se muestra la clasificación que se ha expuesto en los párrafos anteriores, junto con algunos algoritmos de ejemplo para cada grupo. Parte de este Capítulo se dedicará a resumir sus características más distintivas, aunque se verá con más detalle los que aparecen marcados en negrita en la tabla.

Todas estas propuestas están planteadas para organizaciones de conmutador donde los paquetes que esperan su turno para ser transmitidos se almacenan en el puerto de salida correspondiente. Al asumir que los paquetes están listos para ser transmitidos hacia cada puerto de salida según llegan al conmutador, estos algoritmos son capaces de proporcionar garantías de Calidad de Servicio.

Sin embargo, la arquitectura con colas a la salida sufre de un grave problema de escalabilidad, acentuado por las grandes velocidades de transmisión de los enlaces actuales. Veremos a continuación las posibles alternativas para organizar los *buffers* en un conmutador.

3.3. Organización de los *buffers* en un conmutador

En la Figura 3.1 se muestra un modelo general de organización de un conmutador de $N \times N$ puertos. Se puede observar que consta de un conjunto de puertos de entrada y salida, con una cierta lógica asociada a cada uno de ellos, unidos entre sí por una red de interconexión entre las entradas y las salidas, que en la literatura se suele denominar "*switch fabric*".

Los tipos más comunes de "*switch fabric*" incluyen los basados en memoria compartida, en bus, en *crossbars* y en redes multietapa. Nosotros consideraremos únicamente arquitecturas de conmutador *no bloqueantes*, que se suelen implementar mediante un *crossbar*. Esto es, asumiremos que el conmutador está construido sobre un *fabric* capaz de transmitir cualquier

ALGORITMOS BASADOS EN PRIORIDADES

Orientados al retardo	Orientados al ancho de banda
CODA [164]	FQ [45]
RACE [98]	**VC** [178]
TDC [81]	PGPS [128]
DEDD [54]	SCFQ [68]
JEDD [168]	FFQ [157]
RCSD [177]	

ALGORITMOS BASADOS EN TRAMAS

Conservan el trabajo	No conservan el trabajo
SG [65, 66]	**WRR** [90]
HRR [82]	DRR [147]

Tabla 3.1: Clasificación de los algoritmos de planificación del tráfico.

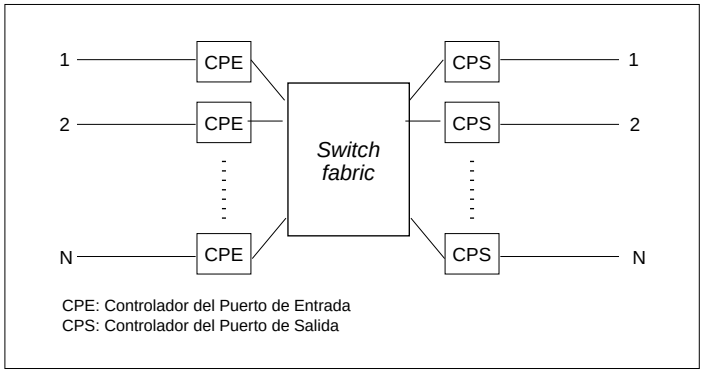


Figura 3.1: Un modelo general para un conmutador de $N \times N$ puertos.

conjunto de paquetes presentados a sus entradas sin bloqueo interno, siempre y cuando los puertos de salida de los paquetes sean distintos.

Pero incluso con una arquitectura del conmutador que no sea bloqueante, las peticiones de conexión deben competir con otras peticiones. Si dos paquetes llegan a diferentes puertos de entrada, y ambos van dirigidos al mismo puerto de salida, entonces sólo uno de ellos podrá ser enviado inmediatamente, mientras que el otro tendría que ser almacenado en un *buffer*. Esto se conoce como *contención de salida*. El emplazamiento de los *buffers* donde se almacenan los paquetes afectados por ella determinan la organización general del conmutador.

La situación de los *buffers* en un conmutador tiene una gran influencia tanto en el coste de implementación del mismo, como en las prestaciones que proporciona. Existen varias posibles organizaciones del espacio de almacenamiento en un conmutador, que describiremos brevemente a continuación [120].

Organización con *buffers* a la salida Cuando llega un paquete a un puerto de entrada, el paquete se almacena inmediatamente en el *buffer* del puerto de salida correspondiente. Debido a que puede darse el caso de que varios paquetes lleguen al conmutador simultáneamente por distintos puertos de entrada, y que todos ellos requieran el mismo puerto de salida, el *buffer* situado a la salida debe ser capaz de encolar el tráfico a una velocidad mucho mayor de la velocidad con que puede llegar el tráfico por un sólo enlace. En el peor de los casos, tanto el *buffer* de salida como el *switch fabric* deben funcionar a N veces la velocidad de los enlaces, siendo N el número de puertos de entrada. Esto impone una severa limitación a la escalabilidad del conmutador.

Organización con *buffers* centrales compartidos En esta organización hay un sólo *buffer* compartido por todos los puertos de entrada. Este *buffer* puede verse como una unidad de memoria compartida con soporte de N accesos concurrentes de escritura (que llevarán a cabo los N puertos de entrada), y hasta N accesos concurrentes de lectura (que llevarán a cabo los puertos de salida). Debido a que los paquetes destinados para el mismo puerto de salida pueden llegar simultáneamente desde muchos puertos de entrada, el puerto de salida debe leer el tráfico a una velocidad mucho mayor de la que lo escribe un sólo puerto. De nuevo, esto impone un severo límite al tamaño máximo del conmutador.

Organización con *buffers* a la entrada Esta organización no tiene los problemas de escalabilidad de las organizaciones anteriores. En esta arquitectura, cada puerto de entrada mantiene una cola FIFO (*First-In First-Out*) de paquetes, y sólo el primero de la cola es seleccionable para ser transmitido durante un ciclo de tiempo o *slot* dado. Esto conduce a un severo cuello de botella en cuanto a prestaciones, el *bloqueo de cabeza de línea (HOL-blocking)*. Este bloqueo se produce porque cuando el paquete situado a la cabeza del *buffer* no puede ser transmitido debido a contención, el resto de paquetes de la cola tampoco se pueden transmitir, incluso aunque los puertos de salida que solicitan estén desocupados. Se ha demostrado que la productividad máxima de cada puerto de entrada queda limitada al 58'6 % bajo tráfico uniforme (e incluso es menor con tráfico con ráfagas) debido a los efectos del *HOL-blocking* [87]. No obstante, y debido a la simplicidad y escalabilidad de esta organización frente a las anteriores, se ha investigado en cómo soslayar el problema del *HOL-blocking*. Se ha demostrado que, con un buen diseño del esquema de almacenamiento, el *HOL-blocking* puede solucionarse, incrementando drásticamente la productividad de estos conmutadores [123, 86]. La idea es proporcionar varias colas por cada puerto de entrada, de tal manera que nunca coincidan en la misma cola dos paquetes dirigidos a puertos de salida

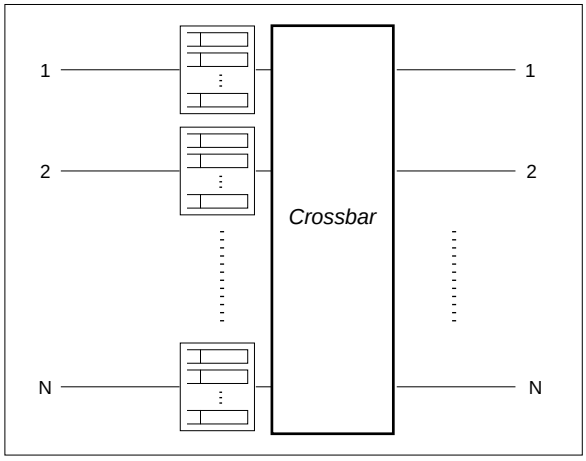


Figura 3.2: Modelo general para un conmutador de $N \times N$ puertos con *buffers* a la entrada y múltiples colas por puerto de entrada.

distintos (ver el esquema de la Figura 3.2). De esta manera, aun tratando cada cola en modo FIFO, se elimina por completo el problema del *HOL-blocking*. Estas colas se pueden organizar de varias maneras. Las alternativas más comunes son emplear una cola por cada puerto de salida (a esta variante se le denomina ***Virtual Output Queuing, VOQ***), o bien dedicar una cola a cada flujo (o agrupación de flujos) de información entre un origen y un destino, lo que se denomina **canal virtual**.

Organización con *buffers* combinados a la entrada y a la salida Esta organización es una combinación de los esquemas con *buffers* a la entrada y con *buffers* a la salida. Se trata de un buen compromiso entre las prestaciones de los conmutadores con *buffers* a la salida, y la escalabilidad de los conmutadores con *buffers* a la entrada. La idea es la siguiente: si en un conmutador con *buffers* a la entrada, como máximo un paquete puede ser transmitido hacia cada puerto de salida por unidad de tiempo, y en un conmutador con *buffers* a la salida se pueden transmitir hasta N paquetes por unidad de tiempo, el empleo de *buffers* a ambos lados permite escoger un valor intermedio razonable.

Aunque han aparecido recientemente algunas propuestas que implementan la organización con *buffers* combinados a la entrada y a la salida [30, 116, 38], es la organización basada en *buffers* a la entrada la que más interés ha despertado en los últimos años en la comunidad científica debido a que pueden trabajar a la misma velocidad que un sólo enlace [114, 153, 115, 103, 140, 84, 69, 102, 38]. Recordemos que la organización con *buffers* a la salida necesita que el conmutador funcione internamente a mayor velocidad que la de los enlaces. Para que cualquier paquete que llegue por una entrada pueda encolarse directamente en el correspondiente *buffer* de salida, en el peor de los casos, tanto el *buffer* de salida como el *switch fabric* y la lógica de control deben funcionar a N veces la velocidad de los enlaces, siendo N el número de puertos de entrada [103]. Esto no suponía un gran problema con redes convencionales, con anchos de banda de pocos Megabits/seg. Sin embargo, en los últimos años, la tendencia es hacer funcionar los enlaces al límite mismo de la tecnología, con lo que hacer que el conmutador funcione internamente a una velocidad todavía mayor que los puertos resulta inviable [107].

En las siguientes secciones se efectuará un breve repaso por los algoritmos de planificación del tráfico propuestos en la literatura para las distintas organizaciones de conmutador co-

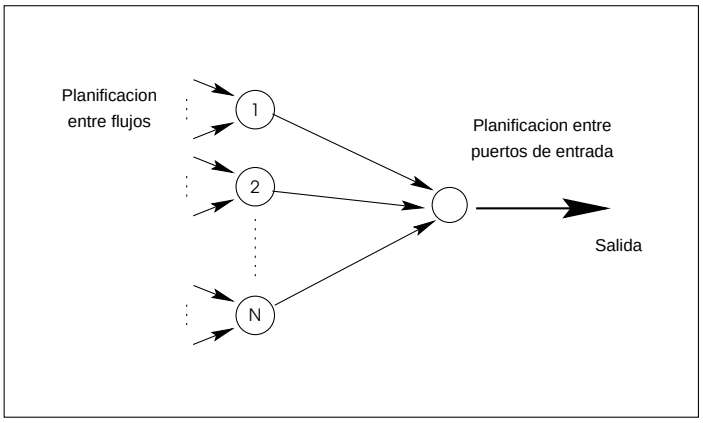


Figura 3.3: Los dos niveles de planificación en un conmutador con *buffers* a la entrada.

mentadas en este punto. Se abordará en primer lugar el problema de la planificación del tráfico para la organización con *buffers* a la entrada. A continuación se tratará el caso de la organización con *buffers* a la salida. Se finalizará comentando algunas propuestas aparecidas para conmutadores específicamente diseñados para redes de área de sistema (SANs).

3.4. Planificación del tráfico en conmutadores con *buffers* a la entrada

Se ha comentado anteriormente que el mayor problema que presenta un conmutador con los *buffers* situados en sus puertos de entrada es la baja productividad provocada por el *HOL-blocking*. También se ha señalado que este problema se puede solventar organizando los *buffers* en múltiples colas.

La planificación de paquetes en estos conmutadores se puede dividir conceptualmente en dos niveles (ver Figura 3.3) [158, 157]:

1. Planificar las transmisiones entre los puertos de entrada del conmutador que están transmitiendo hacia un puerto de salida común

2. Planificar un paquete de entre los disponibles en un mismo puerto de entrada

El segundo problema puede resolverse mediante cualquiera de los algoritmos que aparecen en la Tabla 3.1, y en general, empleando cualquiera de los existentes en la literatura para conmutadores con *buffers* en la salida. Dichos algoritmos serán tratados en detalle en la próxima sección. Por tanto, este apartado se centrará en el primer problema: la planificación de las transmisiones a través del *crossbar*. Se asumirá que hay disponibles para conmutar varios paquetes por cada puerto de entrada, no sólo el que se encuentre a la cabeza de cola. Es decir, que los *buffers* de entrada se han diseñado con algún mecanismo que evite el problema del *HOL-blocking* (probablemente, empleando *buffers* de acceso aleatorio).

El problema se puede enunciar como seleccionar un conjunto de paquetes para transmitir a través del *crossbar* de manera que cumpla las siguientes restricciones:

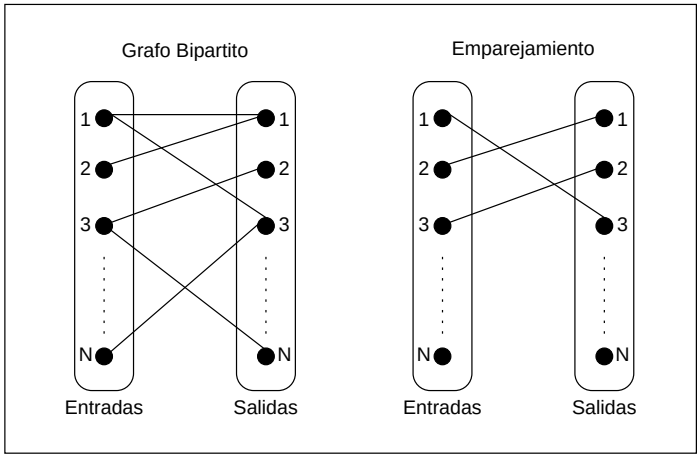


Figura 3.4: Ejemplo de grafo bipartito y de un emparejamiento válido.

1. Como máximo, un único paquete de cada puerto de entrada puede ser transmitido en cada *slot* o ciclo de tiempo.
2. Como máximo, un único paquete puede ser recibido por cada puerto de salida en cada *slot* o ciclo de tiempo.

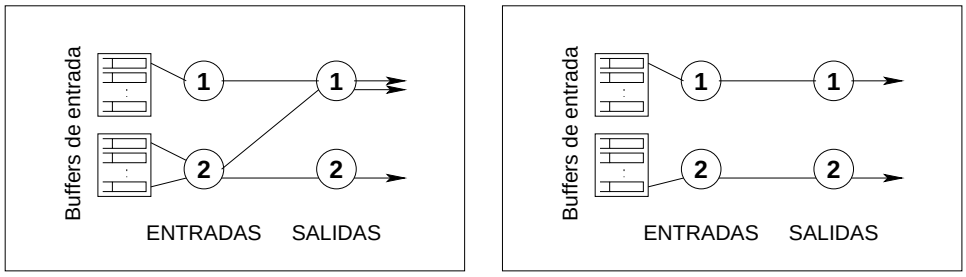
Este problema se puede reducir al *problema del emparejamiento en un grafo bipartito* [17]. Dicho grafo bipartito se construye al representar cada puerto de entrada mediante un vértice en el primer grupo, y cada puerto de salida con un vértice en el segundo grupo. Cada paquete accesible se representa mediante un arco que va desde el vértice que representa el puerto de entrada donde se halla almacenado, hasta el puerto de salida a donde va dirigido (ver la Figura 3.4). El problema de emparejamiento bipartito intenta maximizar el número de vértices del primer grupo conectados al segundo grupo, seleccionando un subconjunto de arcos de tal forma que ningún par de arcos tenga un vértice en común. La restricción de no más de un arco por vértice es equivalente a decir que no se puede transmitir más que un paquete desde cada puerto de entrada, y hacia cada puerto de salida, por ciclo.

Si el emparejamiento se realiza de tal forma que no se puede añadir un nuevo paquete sin modificar las asignaciones actuales, se denomina *emparejamiento maximal*. En este emparejamiento, todos los paquetes o están planificados para transmisión, o están bloqueados. Si el número de paquetes planificados por el emparejamiento es el mayor de entre todos los emparejamientos válidos, entonces se tiene un *emparejamiento máximo*.

También es posible asignar pesos a los arcos que componen el grafo, con lo que la maximización puede hacerse según peso, es decir, haciendo que la suma de los pesos de los arcos incluidos sea máxima. Dependiendo del valor que se asigne a dichos pesos, se pueden obtener distintos algoritmos de emparejamiento. A continuación se resumen algunos ejemplos de algoritmos de cada tipo.

3.4.1. Algoritmos que buscan emparejamiento maximal en tamaño

En un principio pueden parecer más interesantes los algoritmos para calcular un emparejamiento máximo en tamaño [88, 163], pues en cada ciclo se obtendría un aprovechamiento máximo del ancho de banda agregado del *crossbar*. Sin embargo, estos algoritmos presentan



(a) Peticiones de tráfico (b) Emparejamiento máximo

Figura 3.5: Ejemplo de inanición en un conmutador 2×2 cuando se emplea un algoritmo de emparejamiento máximo.

dos problemas: i) el tiempo de convergencia puede ser elevado, y ii) se puede provocar la inanición de algún flujo con ciertos patrones de tráfico [112]. En la Figura 3.5 se muestra un ejemplo extremadamente simple, donde un algoritmo de emparejamiento máximo nunca permitirá el avance de paquetes desde la entrada 2 a la salida 1, pues esto implicaría un emparejamiento de tamaño 1.

Por ello, los algoritmos que se proponen en la literatura para resolver el problema del emparejamiento suelen buscar uno que cumpla la propiedad de ser *maximal*. La idea es ir construyendo la planificación de forma iterativa, añadiendo en cada vuelta emparejamientos entre los enlaces que queden sin ocupar. Los emparejamientos se añaden de forma progresiva, pero nunca se eliminan. Es por ello que este enfoque no siempre da con el emparejamiento máximo, aunque en muchos casos se aproxime bastante. Además, suelen introducir algún mecanismo para evitar la inanición.

El algoritmo de emparejamiento propuesto por Anderson y otros en [17], denominado **Emparejamiento Paralelo Iterativo** o **PIM** (de *Parallel Iterative Matching*), se basa en la aleatoriedad para reducir el número de iteraciones necesarias, y evitar la inanición.

El algoritmo consiste en repetir los tres pasos mostrados en la Figura 3.6 un cierto número de veces. Inicialmente, todas las entradas y salidas están desemparejadas. Al inicio de cada iteración únicamente se consideran las entradas y salidas no emparejadas hasta el momento.

Cada uno de estos tres pasos se lleva a cabo en paralelo en cada puerto de entrada y de salida; no existe ningún planificador centralizado. En la Figura 3.7 se muestra un ejemplo de una iteración del algoritmo. En la fase de petición, cada puerto de entrada envía una petición (representada mediante una flecha) a los puertos de salida para los cuales tiene destinado algún paquete. Así, la entrada 1 envía sus peticiones a las salidas 1 y 2, la entrada 3 dirige sus peticiones a las salidas 2 y 4, y finalmente, la entrada 4 solicita la salida 4. En la fase de concesión, las salidas seleccionan de manera aleatoria una de las peticiones recibidas, y notifican a la entrada correspondiente de su concesión. De esta manera, la salida 1 selecciona la petición de la entrada 1, y tanto la salida 2 como la 4 escogen la de la entrada 3. En la última fase, cada entrada decide al azar qué concesión acepta, formando así los emparejamientos. Así, la entrada 1 se empareja con la salida 1 (es la única concesión que recibió), mientras que la entrada 3 escoge aleatoriamente emparejarse con la salida 2.

El protocolo se va repitiendo, almacenando los emparejamientos obtenidos en cada iteración, hasta que ya no se alcancen más. En el ejemplo de la Figura 3.7, en una iteración posterior, el algoritmo realizaría el único emparejamiento adicional posible, el de la entrada 4 con la salida 4.

Algoritmo PIM:

1. **Fase de petición** Cada entrada no emparejada envía una petición a cada una de las salidas para las que tiene un paquete almacenado. Este paso informa a cada salida de todas sus parejas potenciales.

2. **Fase de concesión** Si una salida no emparejada recibe alguna petición, escoge una aleatoriamente y la acepta, enviando un reconocimiento. La salida notifica a cada entrada si su petición fue aceptada o no.

3. **Fase de aceptación** Si una entrada recibe varios reconocimientos, escoge uno al azar, lo acepta y se lo notifica a esa salida.

Figura 3.6: Algoritmo de Emparejamiento Paralelo Iterativo (PIM).

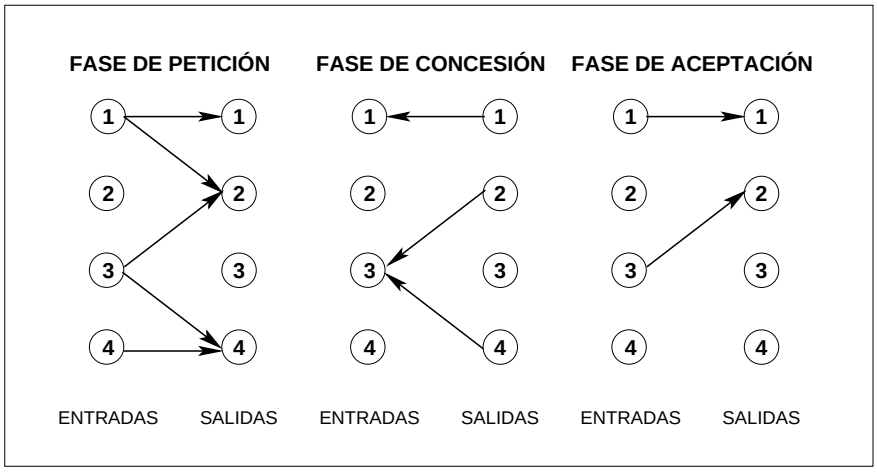


Figura 3.7: Una iteración del algoritmo de emparejamiento paralelo iterativo (PIM).

El empleo de selecciones aleatorias en los pasos 2 y 3 del algoritmo ofrece interesantes ventajas, pero también algunos inconvenientes [109, 112].

Las ventajas que proporciona la aleatoriedad son las siguientes:

- Los autores demuestran que gracias a las selecciones arbitrarias, en cada iteración, el algoritmo es capaz de eliminar o conectar al menos 3/4 de los posibles emparejamientos restantes, y por tanto, el algoritmo convergerá a un emparejamiento maximal tras $O(\log N)$ iteraciones (para un conmutador de tamaño $N \times N$) [17].
- La aleatoriedad también asegura que en algún momento todas las peticiones serán concedidas y emparejadas, eliminando el problema de la inanición de algún flujo.
- Se elimina la necesidad de mantener información de estado entre sucesivas ejecuciones del algoritmo. Esto simplifica su comprensión y análisis.

Y los problemas que presenta son básicamente tres:

- La implementación del mecanismo de selección arbitraria entre los miembros de un conjunto que varía con el tiempo es complicada y costosa de hacer en hardware.
- Bajo ciertas condiciones de carga, puede no tratar de forma equilibrada a todos los flujos [109].
- Aunque las prestaciones obtenidas son muy buenas con varias iteraciones (en [121] se demuestra que con 4 iteraciones se puede obtener una productividad del 99 % para cualquier tamaño de conmutador), las prestaciones cuando se hace una única iteración son muy pobres (la productividad está en torno al 63 % [109]). Esto puede ser un problema si no se dispone de tiempo para efectuar dicho número de iteraciones.

Tomando como referencia PIM, se han propuesto algoritmos similares que intentan abordar los problemas que presenta PIM. Estos nuevos algoritmos mantienen la idea de calcular el emparejamiento de forma iterativa, y con un proceso de petición-concesión-aceptación similar al que emplea PIM, pero varían el criterio empleado para seleccionar qué petición se concede (fase 2) y qué concesión se acepta (fase 3).

McKeown propone en [109, 110] el algoritmo de emparejamiento denominado **iSLIP**. Este algoritmo es muy similar a PIM, pero las decisiones a tomar no se basan en el azar, sino que siguen un esquema cíclico o *round-robin*. El algoritmo consiste en la repetición de tres fases como PIM, pero se modifica el criterio de selección. El proceso queda como se especifica en la Figura 3.8 (las modificaciones respecto a PIM aparecen en cursiva).

En la Figura 3.9 se muestra una iteración del algoritmo iSLIP sobre el mismo ejemplo empleado para PIM. El esquema cíclico considerado será un *round-robin* entre los puertos del conmutador, esto es, se considerarán las peticiones/concesiones en el orden $1, 2, 3, 4, 1, 2, \ldots$ Los cuadrados junto a cada puerto indican el valor del puntero al elemento más prioritario dentro del esquema cíclico. Inicialmente, todos los punteros contienen el valor de 1, el inicio del ciclo. Tras la fase de petición, las salidas deben escoger una entrada a la cuál otorgar su concesión. Puesto que se consideran las peticiones en orden a partir del 1, cada salida escogerá la entrada de menor número de orden. Así, la salida 2, que recibe dos peticiones (de las entradas 1 y 3), seleccionara la realizada por la entrada 1, y la salida 4 escogerá la de la entrada 3, pues aparece antes que la 4 en el esquema cíclico. La salida 1 otorga su concesión a la entrada 1, pues es la única petición que recibe. Hay que hacer notar que en

Algoritmo iSLIP:

1. **Fase de petición** *(idéntica a PIM)* Cada entrada no emparejada envía una petición a cada una de las salidas para las que tiene un paquete almacenado.

2. **Fase de concesión** Si una salida no emparejada recibe alguna petición, *escoge aquella que aparezca como siguiente según un esquema cíclico prefijado que empieza en el elemento más prioritario.* La salida envía el reconocimiento a la entrada correspondiente. Las salidas notifican a cada entrada si su petición fue aceptada o no. El puntero al elemento más prioritario de ese esquema cíclico se desplaza *si y sólo si* esta concesión se acepta (es decir, se llega a realizar el emparejamiento) en la fase 3 de la primera iteración.

3. **Fase de aceptación** Si una entrada recibe varios reconocimientos, escoge *el siguiente que aparezca según un esquema cíclico prefijado*, lo acepta y se lo notifica a esa salida. El puntero al elemento más prioritario dentro de ese esquema cíclico se incrementa (módulo N) a la siguiente posición tras la salida aceptada, sólo si el emparejamiento se produce en la primera iteración.

Figura 3.8: Algoritmo iSLIP.

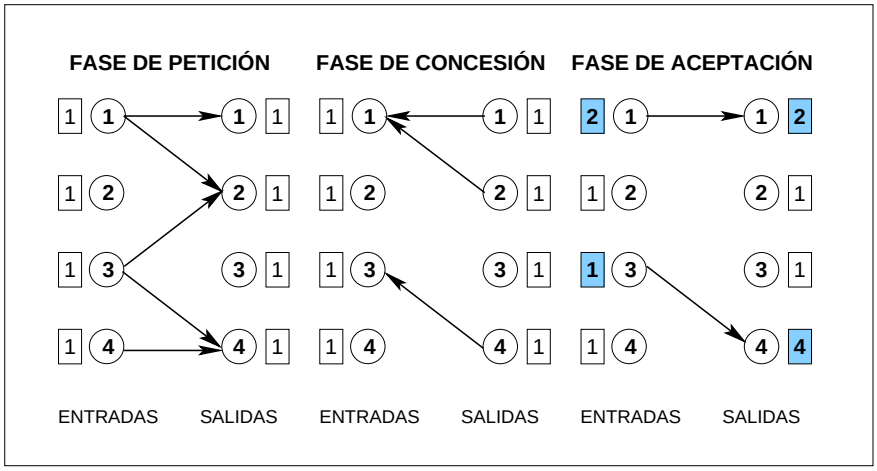


Figura 3.9: Una iteración del algoritmo de emparejamiento iSLIP.

esta fase no se modifican los punteros. Esto sucederá en la tercera fase, una vez realizados los emparejamientos, y por ser esta la primera iteración. En esta última fase, las entradas aplican el mismo criterio para seleccionar qué concesión de salida escogen. Así, la entrada 1, que recibe dos concesiones, escoge la proporcionada por la salida 1, según el esquema en *round-robin*. En este momento se actualizan los punteros, tanto de las entradas como de las salidas, a la posición siguiente a la del puerto con quien se ha emparejado. Los punteros que se modifican aparecen sombreados en la Figura 3.9. De esta forma, la entrada 1, que se ha emparejado con la salida 1, hace avanzar su puntero hasta el 2, y lo mismo hace la salida 1, mientras que la entrada 3 hace avanzar su puntero hasta la posición siguiente a la salida 4, que es la salida 1 (incremento módulo $N = 4$). De igual modo, la salida 4 actualiza su puntero al valor siguiente al 3, o sea, al 4.

Hay que señalar que el valor de los contadores no se reinicializa al principio de cada nueva ejecución del algoritmo. El efecto es que las entradas o salidas que acaban de ser emparejadas se convierten en las menos prioritarias en la siguiente ejecución del algoritmo. Además, el hecho de actualizar los punteros en las salidas únicamente cuando ha tenido éxito el emparejamiento, hace que los árbitros de las salidas se desincronicen, y tiendan a dar su concesión a peticiones distintas, maximizando así la productividad.

Otra de las propiedades del algoritmo es que se garantiza que ninguna conexión sufrirá de inanición. Esto es porque una entrada seguirá solicitando una salida hasta que obtenga éxito, y porque los punteros sólo se actualizan en la primera iteración. La salida en cuestión atenderá como máximo las otras $N - 1$ entradas primero, y como mucho tendrá que esperar N ciclos antes de ser aceptada por cada entrada. Por tanto, cada petición será servida en menos de N^2 ciclos. En en [109, 110] aparece un análisis detallado del algoritmo, junto con algunas variantes.

Por otro lado, Tamir y Chou proponen en [162] varios algoritmos de arbitraje para conmutadores con múltiples colas en las entradas. Hablan de *árbitros simétricos para crossbars*. La razón de este nombre es que cada puerto de entrada contiende por múltiples puertos de salida, pero sólo necesita conseguir uno para obtener una utilización completa, y de manera simétrica, cada puerto de salida contiende por más de un puerto de entrada, y sólo necesita uno para alcanzar una utilización completa.

Los algoritmos de Tamir y Chou se implementan como una matriz de "puntos de decisión"

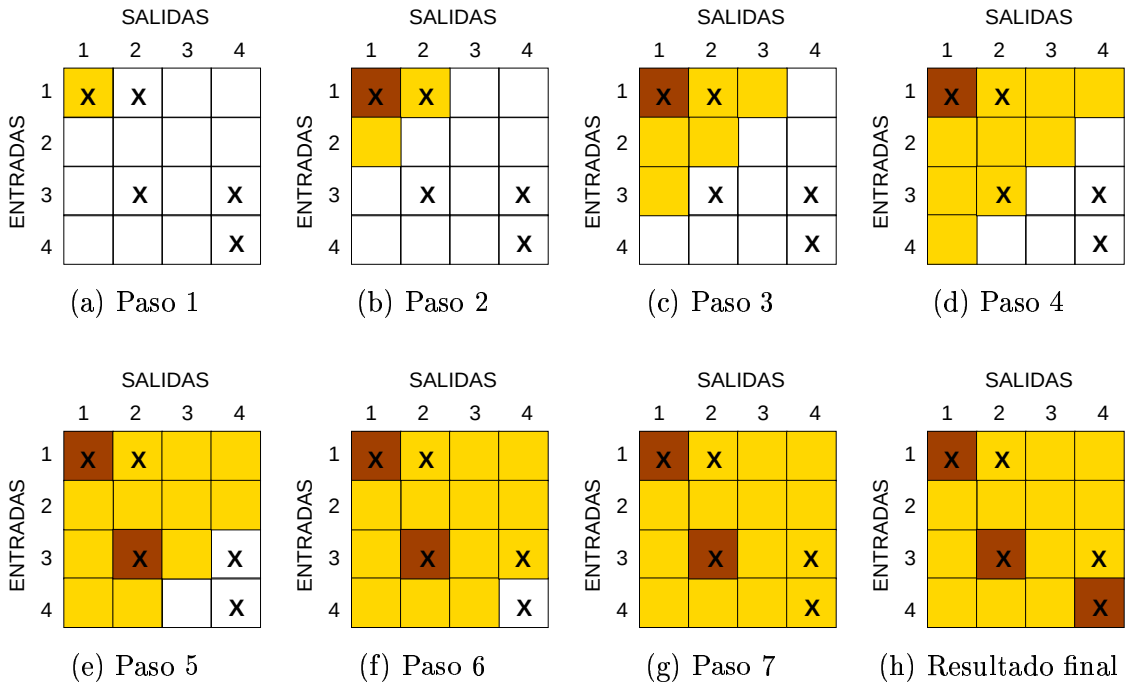


Figura 3.10: Ejemplo de ejecución del algoritmo WFA.

interconectados entre sí. Estos puntos calculan la planificación del *crossbar* pues tras la ejecución del algoritmo, en cada uno de ellos se obtiene una señal que habilita o no ese punto de cruce. Como entrada reciben la petición de habilitación de ese cruce, en caso de que la entrada correspondiente tenga algo que transmitir a la salida correspondiente. Cada punto de cruce actualiza su estado (habilitar el punto de cruce o no) en función del estado de los demás puntos de su fila, y columna, y de si fue solicitado o no. Dependiendo del orden de actualización, se obtienen distintos algoritmos.

Uno de los algoritmos propuestos en [162] es el denominado ***Wave Front Arbiter* (WFA)**. En este algoritmo, el orden de actualización es como un frente de onda que se propaga siguiendo las diagonales de la matriz de puntos de cruce. Se hace un emparejamiento en un punto de cruce siempre y cuando no exista ningún otro en los puntos situados por encima del actual en la misma columna, ni en la misma fila a su izquierda. Esto es, se realiza el emparejamiento si no se ha establecido previamente ningún otro en el que intervengan ese puerto de entrada y/o ese puerto de salida.

En la Figura 3.10 se muestra el avance del frente de onda sobre un ejemplo de una matriz de peticiones. Esta matriz se forma de la siguiente manera: si se solicita una conexión entre el puerto de entrada i y el de salida j, el punto de cruce correspondiente, representado por la casilla $\{i, j\}$, aparece marcado con una "**X**". La matriz de la Figura 3.10 refleja las mismas peticiones de los ejemplos de PIM y iSLIP mostrados en las Figuras 3.7 y 3.9.

El frente de onda se inicia en el punto de cruce $\{1, 1\}$. La diagonal considerada en cada momento por la actualización de los árbitros se marca con un sombreado claro, mientras que los emparejamientos realizados se marcan con un sombreado más oscuro. Hay que hacer notar que, una vez hecho un emparejamiento, las conexiones entre los puntos de cruce propagan la información para que no se haga ningún otro que involucre a esa entrada o a esa salida. Así, puesto que en el primer paso se realiza el emparejamiento $\{1, 1\}$, en el segundo paso (Figura 3.10-(b)), no se puede realizar el emparejamiento $\{1, 2\}$, puesto que el puerto de entrada

1 ya está ocupado. Lo mismo sucede en el paso 7 (Figura 3.10-(g)), con el emparejamiento {3, 4}. Cuando el frente de onda ha pasado por todas las diagonales, el algoritmo finaliza. En este ejemplo ha obtenido los tres emparejamientos posibles: {1, 1}, {3, 2} y {4, 4}.

La gran ventaja de este algoritmo es la simplicidad de su implementación hardware, pues la lógica de decisión en cada punto de cruce consiste en una combinación simple de puertas lógicas sencillas. También se pueden incluir otro par de módulos simples para decidir en qué punto se inicia el arbitraje, y no favorecer siempre a los mismos puertos por considerarlos siempre en el mismo orden.

Tanto PIM, como iSLIP o WFA ofrecen buenas prestaciones cuando el tráfico es uniforme. Sin embargo, cuando el patrón de tráfico no es uniforme, su productividad decae considerablemente [113]. Esto es debido a que, al tratarse de algoritmos que aproximan emparejamientos de tamaño máximo, no consideran el estado de las colas al tomar las decisiones de planificación. Mekkittikul propone en [113] un par de algoritmos que proporcionan buenas prestaciones con tráfico no uniforme, y que se pueden implementar de forma simple. Estos algoritmos consideran parámetros como la ocupación de las colas o el tiempo de espera en cola de un paquete durante el proceso de planificación. Por tratarse de algoritmos de emparejamiento de peso máximo, se tratarán en la siguiente sección.

Los algoritmos descritos hasta este momento están orientados únicamente a obtener productividad máxima, es decir, persiguen emparejar un gran número de puertos para trasegar la mayor cantidad de información posible a través del *crossbar*. Sin embargo, no se tienen en cuenta los distintos requisitos de ancho de banda de los distintos flujos, especialmente si varían con el tiempo o no cumplen con sus especificaciones hechas al establecerse. Debido a ello, surgen un par de modificaciones, sobre el algoritmo PIM concretamente, encaminadas a solucionar ese problema.

Los mismos autores de PIM proponen una modificación denominada **Emparejamiento Paralelo Estadístico** [17], para respetar el ancho de banda reservado por conexiones CBR frente a ráfagas de otros tipos de tráfico. En primer lugar, se divide el ancho de banda de los enlaces en ciclos de paquete o *slots*, que se agrupan en tramas. En cada trama, se reservan ciclos a cada conexión CBR en función de sus necesidades. El resto de tráfico ocuparía los ciclos restantes.

Si se aplica el algoritmo PIM tal cual, todos los flujos son tratados por igual, y puede que el tráfico CBR no reciba el ancho de banda reservado. En el emparejamiento estadístico, sin embargo, se divide el ancho de banda entre los flujos que lo necesitan, de acuerdo a sus reservas, de manera que un flujo que haya hecho una mayor reserva de ancho de banda tendrá mayor probabilidad de conseguir una asignación de entrada-salida que transmita sus paquetes. En la Figura 3.11 se muestra una versión simplificada del algoritmo.

Debido a que son los puertos de salida quienes tienen la iniciativa en el emparejamiento, puede que envíen una señal de concesión a un puerto de entrada sin paquetes pendientes. Por ese motivo, la productividad obtenida con este algoritmo está limitada a un 72 % del ancho de banda total. El resto se puede asignar mediante emparejamiento paralelo iterativo normal.

Stiliadis y Varma proponen en [158] otra modificación sobre PIM con el mismo objetivo de respetar las reservas de ancho de banda de los distintos flujos. Su algoritmo se denomina **Emparejamiento Paralelo Iterativo Ponderado (WPIM, *Weighted PIM*)**. Al igual que ocurría en el algoritmo de emparejamiento estadístico, en este caso el ancho de banda también se divide en tramas, compuestas de ciclos de paquete o *slots*. Cada flujo hará su reserva de ancho de banda como un cierto número de ciclos o *créditos*. El objetivo de WPIM es garantizar que al menos cada flujo es capaz de transmitir en cada trama tantos paquetes

Algoritmo de Emparejamiento Paralelo Estadístico:

1. **Fase de concesión** Cada salida j escoge una entrada i para otorgarle una concesión. con una probabilidad proporcional a la reserva de ancho de banda por parte de la entrada i.

2. **Fase de aceptación** Si una entrada recibe varias concesiones, escoge como máximo una para aceptar (puede que no acepte ninguna) en un proceso de dos pasos:

    1. La entrada reinterpreta cada concesión como cero o más concesiones virtuales, de manera que la probabilidad resultante de que la entrada i reciba k concesiones virtuales de la salida j sea proporcional a la reserva de ancho de banda para esa salida.
    2. Si una entrada recibe concesiones virtuales, escoge una aleatoriamente para aceptarla; la salida correspondiente a la concesión virtual aceptada se empareja con la entrada.

Figura 3.11: Algoritmo de Emparejamiento Paralelo Estadístico.

Algoritmo WPIM:

1. **Fase de petición** Cada puerto de entrada que todavía no ha sido emparejado envía una petición a cada uno de los puertos de salida para los cuales tiene paquetes almacenados en sus colas.

2. ***Fase de máscara*** Al recibir las peticiones efectuadas por los puertos de entrada, cada puerto de salida crea una máscara, que consiste en un bit por petición de la siguiente forma: para aquellas entradas que ya han transmitido hacia el puerto de salida por lo menos tantos paquetes como especifica su crédito durante la trama en curso, el bit se pone a 1; en otro caso, el bit se pone a 0. De entre las peticiones recibidas, sólo se emplean en el proceso de emparejamiento aquellas que fueron originadas por puertos de entrada no enmascarados (su bit de máscara vale 0). El resto, se ignoran.

3. **Fase de concesión** De entre las peticiones que quedan tras la fase anterior, el puerto de salida selecciona una de manera aleatoria con probabilidad uniforme, y envía una señal de concesión al puerto de entrada que la originó.

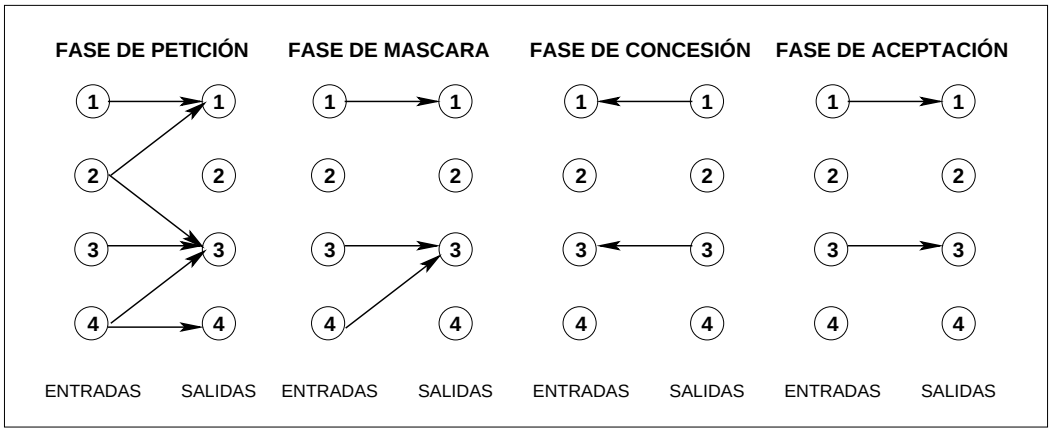
4. **Fase de aceptación** Cada puerto de entrada no emparejado que recibe una o más concesiones, selecciona una con igual probabilidad y lo notifica al puerto de salida correspondiente. Esos puertos de entrada y salida ya están emparejados y pueden eliminarse de sucesivas iteraciones.

Figura 3.12: Algoritmo de Emparejamiento Paralelo Iterativo Ponderado (WPIM).

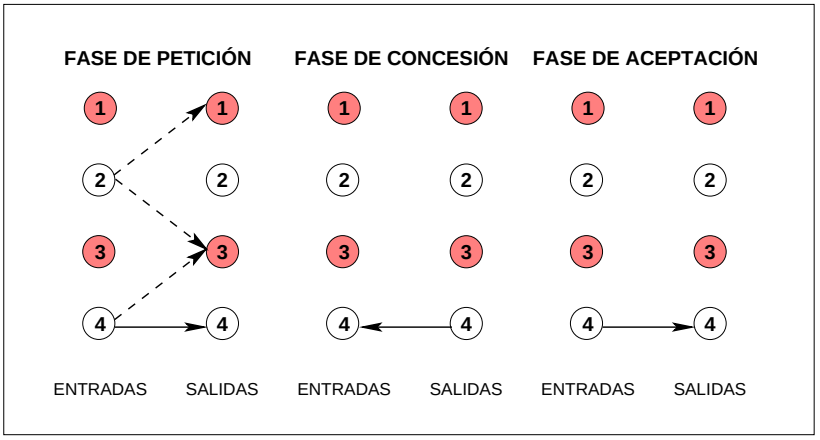
como ciclos reservó. Esto se consigue introduciendo una nueva fase en el algoritmo PIM que enmascara aquellas peticiones que ya han conseguido transmitir los paquetes especificados en su reserva. El algoritmo queda como se muestra en la Figura 3.12.

Este ciclo se repite un número determinado de veces, o bien hasta que no existen más peticiones no enmascaradas pendientes.

En la Figura 3.13-(a) se ilustran los cuatro pasos del algoritmo. Para la fase de máscara, se supone que el puerto de entrada 2 ya ha agotado sus créditos para las conexiones destinadas a los puertos 1 y 3 para la trama actual. Por tanto, ambas peticiones son enmascaradas en la segunda fase. De igual forma, la petición del puerto de entrada 4 al de salida 4 también se enmascara. Durante la fase de concesión, el puerto de salida 3 selecciona aleatoriamente el puerto de entrada 3. Puesto que esta entrada no recibe más concesiones, acepta la del puerto de salida 3. La salida 1 otorga su concesión a la entrada 1, pues es la única petición que recibe, y esta entrada acepta dicha concesión al ser también la única recibida. Así, quedan



(a) Una iteración de la primera fase



(b) Una iteración de la segunda fase

Figura 3.13: Ejemplo de funcionamiento del algoritmo WPIM.

emparejadas la entrada 1 con la salida 1, y la entrada 3 con la salida 3. De esta forma, se garantiza que las conexiones que todavía no han agotado sus créditos puedan conseguir la parte de ancho de banda que les corresponde.

Al finalizar esta primera fase, es posible que existan algunos pares de entrada/salida sin emparejar, incluso habiendo paquetes para enviar, debido a que las peticiones correspondientes fueron bloqueadas durante la fase de máscara del algoritmo. Con el objeto de maximizar el número de paquetes planificados, el algoritmo puede repetirse sobre los puertos de entrada que permanecen sin emparejar, tras reiniciar todos los bits de máscara a 0. Esto es, se puede emplear el algoritmo de emparejamiento paralelo normal sobre las peticiones no emparejadas para repartir el ancho de banda residual, después de haber satisfecho las reservas de ancho de banda. Esto se muestra en la Figura 3.13-(b). En este ejemplo, es fácil ver que la petición del puerto de entrada 4 hacia el puerto de salida 4 puede ser atendida también. Las peticiones dirigidas a los puertos de salida 1 y 3 son ignoradas, puesto que estos puertos ya están emparejados.

El objetivo principal de la fase de máscara es el de bloquear las peticiones iniciadas por las entradas que intentan utilizar más ancho de banda del enlace de salida que el acordado. De esta forma, se permite que el resto de entradas puedan recibir su parte. Cuando el tráfico real

hacia un puerto de salida desde alguno de los puertos de entrada es menor que el reservado, el resto del ancho de banda se reparte por igual entre todas las conexiones que comparten dicho enlace de salida.

Para finalizar esta sección, señalar la existencia de algunas propuestas que calculan el emparejamiento empleando **redes neuronales**, como por ejemplo, el propuesto por Park y Lee en [130]. Ese caso en concreto considera una cola con acceso aleatorio por cada puerto de entrada, y asigna mayor prioridad a aquella cola con mayor ocupación, con el objeto de minimizar la probabilidad de desbordamiento de los *buffers*. Asimismo, se añade la restricción de la transmisión ordenada de los paquetes dirigidos a un mismo destino. Otras propuestas en esta línea aparecen en [166, 64, 129].

3.4.2. Algoritmos que buscan emparejamiento de peso máximo

En los algoritmos anteriores, el objetivo era maximizar el número de conexiones entre puertos de entrada y salida de un *crossbar*. A la hora de calcular los emparejamientos, se consideraban todas las peticiones por igual, con el mismo peso. Pero también se puede ponderar cada petición con un peso, y buscar un emparejamiento que dé un valor de peso lo mayor posible. El motivo es incrementar la productividad al máximo, incluso ante patrones de tráfico no uniforme [113].

En este apartado se describirán algunos algoritmos que buscan emparejamientos maximales en peso. Dependiendo del criterio seguido para asignar esos pesos, se obtienen distintos algoritmos, con diferentes características.

Todos los algoritmos descritos a continuación emplean la técnica de *Virtual Output Queuing*, para evitar el *HOL-blocking*. Recordemos que esta técnica consiste en que cada puerto de entrada mantiene una cola distinta por cada puerto de salida. De esta forma, los paquetes dirigidos a un puerto de salida desocupado, no se verán retrasados innecesariamente por paquetes afectados por la contención de salida.

El algoritmo ***Longest Queue First* (LQF)** [111] se basa en asignar un peso $w_{i,j}$ a cada petición de la entrada i a la salida j, que se corresponde con la ocupación de la cola de paquetes destinados a la salida j en la entrada i, es decir, en el número de paquetes almacenados en dicha cola en espera de transmisión. El algoritmo selecciona un emparejamiento tal que la suma de las ocupaciones de las colas servidas, es máxima. Es decir, construye la planificación de forma que se transmiten paquetes de las colas más largas. Esto aumenta la productividad ante patrones de tráfico no uniformes.

Un inconveniente de LQF es que, aunque las prestaciones obtenidas son buenas, puede provocar inanición en algunas colas. Un ejemplo simple de esto se ilustra en la Figura 3.14.

Podemos observar que, en cada entrada, hay una cola por salida, siendo la de arriba la correspondiente a la salida 1, y la de abajo la correspondiente a la salida 2. Los arcos están etiquetados con sus pesos correspondientes: el número de paquetes que hay en cada puerto de entrada destinados a cada salida. Por ejemplo, el arco que va de la entrada 1 a la salida 1, está etiquetado con un 1, porque en la cola de la entrada 1, correspondiente a la salida 1 (que denominaremos $Q_{1,1}$), hay un sólo paquete esperando para ser transmitido. Al aplicar LQF, se transmitirán los paquetes situados a la cabeza de las colas $Q_{1,2}$ y $Q_{2,1}$, pues son las de mayor ocupación. Ahora bien, si suponemos que ya no llegan más células para las colas $Q_{1,1}$ y $Q_{2,2}$, y que para las otras dos colas, llega un paquete en cada ciclo de célula, las colas $Q_{1,1}$ y $Q_{2,2}$ nunca serán servidas, pues siempre tendrán menor ocupación que las otras dos.

El algoritmo ***Oldest Cell First* (OCF)** [114] es una alternativa a LQF que soluciona el problema de la inanición. Esto lo consigue asignando como pesos los tiempos de espera

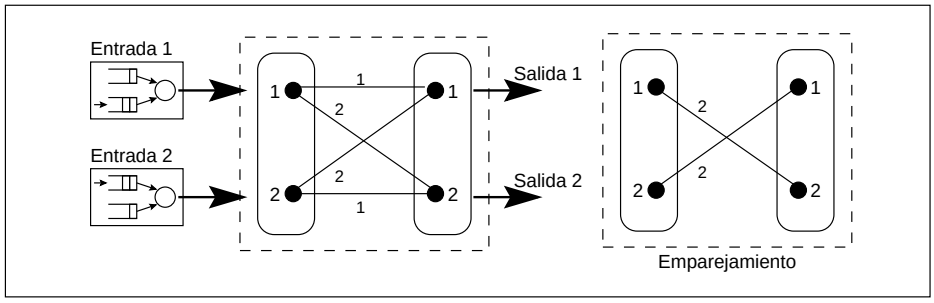


Figura 3.14: Ejemplo de inanición con LQF para un conmutador de 2×2.

de los paquetes almacenados en la cabeza de cada cola. De esta manera, en cada ciclo de transmisión, los paquetes no servidos se hacen más viejos, y así hasta que llegan a ser lo suficientemente viejos como para ser transmitidas. En [114] se prueba que OCF es capaz de obtener un 100 % de productividad para cualquier patrón de procesos de llegadas independientes.

Sin embargo, LQF y OCF presentan un grave problema, y es su dificultad de implementación en hardware a alta velocidad. La razón es que el mejor algoritmo conocido para calcular el emparejamiento según LQF o OCF tiene una complejidad temporal de $O(N^3 \log N)$ [163].

Ante esta perspectiva, los mismos autores de LQF y OCF proponen en [115] el algoritmo denominado ***Longest Port First* (LPF)** con la intención de combinar los beneficios de un emparejamiento maximal en tamaño (de cálculo eficiente) con los de un algoritmo de máximo peso (capaz de alcanzar una productividad máxima incluso con tráfico no uniforme), a la vez que permitiendo una simple implementación en hardware.

LPF encuentra el conjunto de emparejamientos de tamaño máximo, y de entre ellos, escoge la combinación con el mayor peso total. LPF puede lograr una productividad del 100 % ante tráfico tanto uniforme como no uniforme.

En LPF, los pesos se definen de la siguiente forma, para cada petición de la entrada i a la salida j:

$$w_{i,j} = \begin{cases} R_i + C_j, & L_{i,j} > 0 \\ 0, & \text{en otro caso} \end{cases} \tag{3.1}$$

donde R_i es la ocupación de entrada (número total de paquetes que esperan en la entrada i), C_j es la ocupación de salida (número total de paquetes, distribuidos entre todas las entradas, esperando a ser transmitidos por la salida j) y $L_{i,j}$ denota la ocupación de la cola $Q_{i,j}$.

R_i y C_j se calculan así:

$$R_i = \sum_{j}^{N} L_{i,j} \quad ; \quad C_j = \sum_{i}^{N} L_{i,j} \tag{3.2}$$

Al sumar las ocupaciones de entrada y de salida, se obtiene una medida de la carga o congestión que afronta un paquete en su competición por ser transmitido hacia su salida. Esta medida de carga es el peso que se emplea para calcular el emparejamiento.

En [115] se demuestra que LPF encuentra un emparejamiento máximo tanto en tamaño como en pesos. Esto se consigue empleando una versión modificada de un algoritmo para el cálculo de emparejamientos de tamaño máximo, más la adicción de un paso previo de ordenación de las peticiones de acuerdo con los pesos. La complejidad temporal del algoritmo resultante es de $O(\log N^{2,5})$.

Puesto que LPF sigue siendo complejo de implementar, los autores sugieren el empleo de una aproximación iterativa denominada **iLPF (*iterative LPF*)** [113]. La idea consiste en calcular un emparejamiento maximal en tamaño entre las peticiones, después de la ordenación de las mismas para que siempre se consideren las peticiones de mayor prioridad en primer lugar.

Como último ejemplo de algoritmos de emparejamiento de peso máximo, comentar brevemente el algoritmo denominado ***Earliest Due Date First Matching* (EDDFM)** [103]. El algoritmo también está pensado para conmutadores con una organización VOQ. Sus autores proponen asignar a cada paquete de cada cola un peso en función de la cota preestablecida para su retardo. En cada cola, los paquetes se ordenan según su peso, con el de más valor a la cabeza. Este peso se irá incrementando con el tiempo, en caso de que el paquete no sea seleccionado para ser transmitido. Para el algoritmo de emparejamiento, se considerará el peso del paquete que se encuentre a la cabeza de cada cola. De esta forma, al calcular el emparejamiento, se tiene en cuenta la "urgencia" con la que se debe transmitir cada paquete. Este algoritmo mejora la probabilidad de que los paquetes obtengan retardos menores que sus cotas con respecto a otras propuestas. Sin embargo, su complejidad de implementación es elevada, al menos $O(\log N^{2,5})$ por tratarse de un algoritmo similar a los anteriores. Sus autores, por el contrario, no proponen ninguna aproximación iterativa que simplifique su implementación.

3.4.3. Algoritmos de planificación y QoS

Todos los algoritmos comentados en las secciones anteriores están principalmente orientados a maximizar la productividad del conmutador, es decir, a planificar la mayor cantidad posible de paquetes a través del *crossbar* en cada ciclo, para diversos patrones de tráfico (uniforme o no, simétrico o no). Lo único que algunos tienen en consideración además del objetivo principal de ofrecer una productividad máxima es evitar la inanición de los flujos, e intentar tratar a todos con equidad.

Nong y Hamdi [120] resumen los esfuerzos realizados hasta la fecha en aras de ofrecer garantías de QoS en conmutadores con *buffers* a la entrada. Estos investigadores clasifican los enfoques al problema en tres categorías: algoritmos basados en asignación de ciclos de tiempo, algoritmos basados en emparejamiento maximal y algoritmos basados en emparejamiento estable.

Las propuestas basadas en la **asignación de ciclos de tiempo** se basan en planificar el momento en que los paquetes deben abandonar el conmutador, proporcionando una garantía de ancho de banda. Por tanto resultan muy apropiados para manejar el tráfico de tipo CBR, pero no tanto para soportar el tráfico VBR. Dentro de esta categoría, los autores del estudio incluyen al algoritmo de emparejamiento estadístico [17], y al WPIM [158]. Otras propuestas en esta línea se basan en el uso de tramas [74, 171].

Por ejemplo, en [74] el arbitraje consiste en rellenar una *matriz de asignación de ciclos*, con la planificación para una trama completa, en cada puerto de entrada. Además, los puertos deben ponerse de acuerdo para que no haya conflictos por los puertos de salida. El algoritmo de planificación no se ejecuta ciclo a ciclo, sino que se ejecuta una vez cada cierto tiempo,

Algoritmo LOOFA:

1. **Selección de la salida** Cada entrada sin emparejar envía una petición hacia la salida sin emparejar con la menor ocupación en todo el conmutador.

2. **Selección de la entrada** Cada salida, al recibir las peticiones de varias entradas, escoge una y le envía una concesión.

Figura 3.15: Algoritmo *Lowest Output Occupancy First* (LOOFA).

según la velocidad de la implementación del algoritmo de arbitraje (de complejidad $O(N^2 f)$, siendo N el número de puertos y f el tamaño de trama). Al menos, la planificación debe recalcularse al modificarse los requerimientos de ancho de banda de las conexiones, por ejemplo, en el establecimiento y cierre de las conexiones. Y en [171] se emplea un esquema similar al algoritmo de planificación *stop-and-go* [65], ideado para conmutadores con *buffers* a la salida (ver Sección 3.5.2). El problema de estos métodos es que la asignación se hace para tramas completas, con lo que se hace difícil la gestión de ráfagas. Por tanto, únicamente soportan de forma adecuada el tráfico de tipo CBR. Una ventaja que presentan es que siempre se pueden asignar los ciclos de tiempo de forma que se obtenga una productividad del 100 %. El mayor inconveniente es su falta de flexibilidad para acomodar ráfagas de tráfico.

Las propuestas basadas en **emparejamiento maximal** engloban algoritmos que se ejecutan de forma iterativa similar a PIM, pero que consideran parámetros tales como la ocupación de las colas o el tiempo de espera de los paquetes en las mismas a la hora de calcular el emparejamiento. Entre los ejemplos de estos algoritmos se halla el algoritmo denominado **LOOFA (*Lowest Output Occupancy First Algorithm*)** [96], donde se mantienen listas de preferencia globales para resolver los conflictos que ocurran en cada puerto de entrada. La idea es dar preferencia a los emparejamientos que involucren a los puertos de salida de menor ocupación, para hacer que en la medida de lo posible, nunca estén ociosos.

El algoritmo LOOFA consiste en la repetición de los pasos descritos en la Figura 3.15. Inicialmente, todas las entradas y salidas están sin emparejar. En el paso 2, se pueden emplear varios criterios de selección: cíclico, aleatorio, o el paquete que más tiempo lleve en el conmutador (OCF, *Oldest Cell First*).

Sin embargo, este algoritmo, y en general el resto de algoritmos de su clase, requieren que el conmutador funcione internamente a mayor velocidad que los enlaces para poder proporcionar garantías de QoS [96]. Esto también hace necesario el empleo de *buffers* a la salida, para adecuar la velocidad interna de conmutación con la externa de transmisión por los enlaces. Esto es, los algoritmos basados en emparejamiento maximal para proporcionar emparejamientos teniendo en cuenta los requerimientos de QoS de los flujos de datos, suelen estar diseñados para arquitecturas de conmutadores con *buffers* tanto a la entrada como a la salida (arquitectura denominada CIOQ, *Combined Input-Output Queuing*).

Otra posibilidad para el cálculo del emparejamiento es realizar los pares atendiendo a una lista de preferencias por parte de cada puerto de entrada y de salida, de manera que cada puerto se conecte con el mejor de los posibles. Esto no es más que una aplicación del problema del **emparejamiento estable**. Existe un algoritmo clásico para su resolución,

Algoritmo de Gale-Shapley:

Mientras quede una entrada sin emparejar, repetir:

1. Cada entrada sin emparejar envía una petición a la salida de mayor preferencia no solicitada previamente.
2. Cada salida se empareja con la entrada de mayor preferencia de entre las que le han enviado solicitud en el paso previo

Figura 3.16: Algoritmo de Gale-Shapley.

propuesto por Gale y Shapley en 1962 [56], y sobre él se basan las propuestas de algoritmos de emparejamiento de esta categoría. El algoritmo de Gale-Shapley se basa en repetir fases de solicitud-aceptación, con la diferencia respecto a los algoritmos iterativos anteriores de que esta vez sí se pueden romper emparejamientos previamente hechos para conseguir un emparejamiento mejor. Esto es, se trata de un algoritmo que emplea el paradigma de programación dinámica frente al enfoque voraz de los algoritmos PIM y similares. Por tanto, es capaz de dar siempre con el mejor emparejamiento posible.

Un emparejamiento se dice que es *inestable* si hay al menos un par entrada/salida que no están emparejados, pero que se prefieren mutuamente a su pareja actual. Un emparejamiento que no es inestable se denomina *estable*. Trasladado al contexto de proporcionar QoS en conmutadores VOQ/CIOQ, el problema del emparejamiento estable se puede enunciar de la siguiente forma.

> "Un paquete $P_{i,j}$ en un conmutador VOQ (*Virtual Output Queuing*) será planificado por un algoritmo de emparejamiento estable para ser transmitido a través del *switch fabric si y solo si* no se conmuta ningún paquete que esté por delante de $P_{i,j}$ en las listas de preferencia de la entrada i y la salida j"

donde $P_{i,j}$ denota un paquete dirigido del puerto de entrada i al puerto de salida j.

Y el algoritmo de Gale-Shapley se aplicaría de la forma indicada en la Figura 3.16. Inicialmente, todas las entradas y salidas están sin emparejar. Se presupone que cada entrada dispone de una lista de las salidas ordenada según algún criterio de preferencia, y viceversa. Hay que hacer notar que en el paso 2 una salida puede romper un emparejamiento hecho previamente, en caso de recibir una petición de emparejamiento por parte de un "mejor" candidato.

En concreto, el algoritmo denominado **MUCFA (*Most Urgent Cell First Algorithm*)** [136] establece una ordenación de los paquetes en función de su *urgencia*, definida como la posición teórica que ocuparían en la cola correspondiente, si el conmutador tuviese los *buffers* a la salida. Esta ordenación da como resultado las listas de preferencia empleadas en el algoritmo de Gale-Shapley. De esta forma, se consideran más prioritarios aquellos paquetes que estarían situados más cerca de la cabeza de su *buffer* correspondiente, esto es, más próximos a ser transmitidos. Otras propuestas consideran el orden de llegada de los paquetes o la ocupación de las respectivas colas de salida [160, 33].

Algoritmos	Complejidad	Productividad máxima	Tráfico mejor soportado
Asignación de ciclos de tiempo	$O(N^{2,5})$	100 %	CBR
Emparejamiento maximal	$O(N^2)$	50 %	CBR
Emparejamiento estable	$O(N^2)$	50 %	CBR, VBR

Tabla 3.2: Comparación de las características de las tres clases de algoritmos para la provisión de QoS en conmutadores con *buffers* a la entrada.

Realmente, la idea detrás de este algoritmo es emular el comportamiento de un conmutador con *buffers* FIFO a la salida, para lo cuál requiere que el conmutador funcione internamente a 4 veces la velocidad de los enlaces (esto es, se requiere un *speedup* igual a 4). Resultados posteriores indican que este *speedup* puede reducirse a un factor de 2 bajo ciertas condiciones [160, 33]. Para ello, se requieren *buffers* tanto a la entrada como a la salida del conmutador, dando lugar de nuevo a una arquitectura CIOQ. Esto implica una productividad máxima del 50 % si el conmutador funciona internamente a la misma velocidad que los enlaces.

En general, la provisión de garantías de QoS en un conmutador VOQ mediante emparejamiento estable, se especifica en términos de un conmutador con *buffers* a la salida. Esto es, el comportamiento de ese conmutador equivalente se simula en paralelo para obtener las listas de preferencia asociadas a los puertos de salida. De esta manera, es posible aplicar en estos conmutadores algoritmos diseñados para conmutadores con *buffers* a la salida.

Para finalizar, en la Tabla 3.2 (adaptada de [120]) se muestra un resumen de las características de las tres clases de algoritmos descritas. Los algoritmos de asignación de ciclos de tiempo son los más complejos de implementar aunque en principio pueden ofrecer una productividad del 100 % sin necesidad de acelerar el *switch fabric* con respecto a los enlaces. Esto es, se pueden implementar con una arquitectura pura de *buffers* a la entrada. Por el contrario, sólo soportan de forma eficiente el tráfico de tipo CBR. Los algoritmos de las otras dos clases ofrecen una productividad máxima del 50 % si no se emplea aceleración del *switch fabric*, lo cuál puede limitar su aplicabilidad cuando los enlaces son de gran ancho de banda. Se puede concluir que no existe ninguna aproximación de complejidad relativamente baja que sea capaz de ofrecer una cierta garantía de QoS a distintos tipos de tráfico, sin necesidad de acelerar internamente el conmutador con respecto a los enlaces.

3.5. Planificación del tráfico en conmutadores con *buffers* a la salida

El problema a resolver en este caso es el de, dado un conjunto de flujos de datos que compiten por el uso de un canal de salida, decidir cual será el siguiente paquete a transmitir, en base a ciertos criterios. Estos criterios dependerán de las garantías de calidad de servicio requeridas para cada uno de los flujos de datos, y vendrán determinados por el algoritmo concreto que se emplee.

Tal y como aparece en la Tabla 3.1 (página 25), los algoritmos de planificación para conmutadores con *buffers* a la salida se pueden agrupar en dos categorías: los basados en prioridades,

y los basados en tramas. En las siguientes secciones se repasan ejemplos de algoritmos de cada tipo.

3.5.1. Algoritmos basados en prioridades

Para controlar las cotas en el ancho de banda y la latencia de las conexiones, un conmutador debe ser capaz de planificar paquetes para su transmisión en orden distinto al de su llegada. En general, en algoritmos de este tipo, a los paquetes se les asigna una *prioridad* basada en sus requerimientos de servicio, y se planifican según el orden de sus prioridades cuando existe más de un paquete disponible. Dependiendo del mecanismo de asignación de prioridades, existen muchos algoritmos de planificación distintos, proporcionando un amplio rango de servicios, incluso para el mismo conjunto de tráfico.

A continuación, se describirán superficialmente algunos algoritmos de planificación basados en prioridades. En primer lugar, se tratarán ejemplos de algoritmos orientados al retardo; a continuación, ejemplos de algoritmos orientados al ancho de banda. Para cada grupo, se comentarán con cierto detalle un algoritmo: DEDD y VC, respectivamente.

Esquemas orientados al retardo

El esquema más simple *orientado al retardo* consiste en asignar prioridades a las conexiones, y dar la misma prioridad a todos los paquetes pertenecientes a la misma conexión. Al transmitir en todo momento los paquetes con mayores prioridades, el conmutador puede garantizar cotas de retardo en los nodos a las conexiones de alta prioridad.

Un ejemplo de esto es el encaminador de la **máquina paralela en tiempo real CODA** [164]. En este encaminador, a cada puerto de entrada se le asigna una cola de prioridad, de forma que los paquetes de mayor prioridad pueden adelantar a los de menor prioridad. Una técnica única de este encaminador es el *adelantamiento de prioridad* (*priority forwarding*): cuando los paquetes de mayor prioridad están bloqueados por los de menor, los valores de prioridad de estos últimos se actualizan al valor de los primeros. Así, se evita el problema de inversión de prioridades que se produce cuando los paquetes de alta prioridad pueden verse indirectamente bloqueados por otros de menor prioridad a lo largo de múltiples encaminadores.

Otro ejemplo interesante es la **red RACE** para sistemas paralelos empotrados [98], donde se emplea conmutación de circuitos y las peticiones de alta prioridad pueden expulsar (*preempt*) a los circuitos de baja prioridad.

Estas arquitecturas proporcionan redes de bajo coste y altas prestaciones, a la vez que soportan mensajes de control con fuertes restricciones de tiempo real del rango de unos pocos microsegundos. Sin embargo, las cotas en los retardos sólo están garantizadas para las conexiones de mayor prioridad. Esto es, estas redes están orientadas al soporte de mensajería en tiempo real, junto con el tráfico convencional tipo *best-effort*.

La idea básica de emplear un mecanismo de prioridad estática para las comunicaciones en tiempo real puede extenderse con controles del caudal de tráfico para cada nivel de prioridad, de manera que se puedan garantizar cotas de latencia a todas las conexiones, no sólo a las de mayor prioridad. Cuando se restringe la tasa de inyección de cada conexión a ciertos caudales máximos, cada conexión tiene garantizada su cota de retardo dependiendo de su prioridad (a menos que se hagan reservas sobrepasando la capacidad del enlace).

Un buen ejemplo de este esquema es el **conmutador *Time-Deterministic Communication* (TDC)** [81]. Una petición para una nueva conexión se acepta si existe un camino con suficiente ancho de banda libre; su retardo de paquete en el peor caso se determina en base a los caudales de tráfico de las clases existentes. Cada conmutador se limita a transmitir paquetes en el orden de las prioridades y el *jitter* se controla en el destino. Hay que destacar que, debido a que TDC no preserva la suavidad del tráfico en cada conmutador, las ráfagas de tráfico inducidas dentro de la red pueden hacer que las cotas de los retardos sean elevadas. Este mecanismo proporciona un marco para el soporte barato en hardware de comunicaciones en tiempo real. Sin embargo, con una restricción estricta de los patrones de inyección del tráfico, solo es capaz de soportar un estrecho rango de aplicaciones, del tipo de mensajes de control periódicos. Además, debido a que TDC depende del control del caudal a la llegada, un usuario final que exhiba un comportamiento anómalo puede interferir con las garantías de retardo de las demás conexiones. Más aún, no existe el concepto de distribución del ancho de banda, que es esencial para soportar flujos continuos de datos, tales como audio y vídeo.

Otros esquemas más fiables consisten en asignar las prioridades a los paquetes de forma dinámica, dependiendo de la historia del tráfico en lugar de asignar prioridades estáticas a las conexiones.

Un ejemplo bien conocido de estos esquemas de prioridad dinámica es el algoritmo ***Delay Earliest-Due-Date* (DEDD)**, propuesto por Ferrari y Verma en [54]. Este algoritmo está ideado para gestionar tráfico con diversos requerimientos de tiempo real dentro de una red de área amplia, aunque el criterio podría ser empleado igualmente en redes de menor alcance.

El criterio que se sigue a la hora de seleccionar los paquetes que se transmitirán está basado en el cálculo de límites de tiempo o *deadlines*. El *deadline* de un paquete se calcula en base al tiempo esperado de llegada del paquete, de acuerdo con la tasa de llegada especificada en el establecimiento de la conexión. A este tiempo esperado de llegada se le suma la cota de retardo local que proporciona el nodo. Por ejemplo, si una conexión especifica que enviará un paquete cada 0.2 seg. y la cota del retardo en un nodo es de 1 seg., entonces el paquete k-ésimo de esa conexión tendrá un *deadline* de $0{,}2k + 1$. Como veremos, cada nodo ofrece una cota local de retardo, calculada en base a los requisitos de la conexión.

Los autores consideran tres tipos de tráfico:

- Conexiones con requerimientos *deterministas* de temporización. Para ellos se proporciona una cota de su retardo a través de la red, que debe ser cumplida. Esta clase de tráfico está pensada para soportar aplicaciones con fuertes restricciones de tiempo real (*hard real-time applications*).
- Conexiones con requerimientos *estadísticos* de temporización. Para ellos se proporciona una cota de su retardo a través de la red, que debe ser cumplida con una cierta probabilidad, que también se especifica.
- Otras conexiones. No requieren garantías de tiempos.

Cada conmutador mantiene una cola distinta por cada tipo de conexiones. Las correspondientes a las dos primeras clases de tráfico, se ordenan según sus *deadlines*, en orden creciente. No pueden existir dos paquetes deterministas cuyos tiempos de servicio (el tiempo que dedica el conmutador a transmitirlos) puedan coincidir en el tiempo, en el caso de que ambos sean transmitidos lo más tarde posible. En ese caso, se reducirá el *deadline* de uno de ellos (es decir, se incrementaría su prioridad de forma adecuada).

El algoritmo adoptado a la hora de elegir un paquete para su transmisión consiste en lo siguiente. Se compara el tiempo final (el *deadline*) del paquete que ocupa la cabeza de la cola estadística con el tiempo de inicio (el *deadline* menos el tiempo de servicio) del paquete situado en la cabeza de la cola determinista. Si este último es menor que el primero, el paquete determinista se planifica para transmitirlo inmediatamente. En caso de igualdad, los paquetes deterministas siempre tienen mayor prioridad que los estadísticos. En caso contrario (el tiempo final del paquete estadístico es menor que el tiempo de inicio del paquete determinista), se repite la comparación entre el tiempo final del paquete situado en cabeza de la tercera cola y el tiempo de inicio del paquete en cabeza de la cola estadística.

En el caso de que haya usuarios maliciosos o funcionamiento erróneo en los *hosts*, que inserten paquetes en la red a mayor frecuencia que la debida, se intenta minimizar su influencia en el resto de tráfico incrementando de forma adecuada el *deadline* de los paquetes que rompen el contrato establecido. Si el espacio de almacenamiento está limitado, puede ocurrir que algunos de ellos sean descartados debido al desbordamiento del *buffer*.

A la hora de establecer las conexiones, se hacen una serie de comprobaciones en cada conmutador para comprobar si existe suficiente capacidad de cálculo para atender los paquetes de la nueva conexión, junto con los de las conexiones previamente establecidas. También se determina la cota mínima de retardo que puede asignarse a la conexión que se está estableciendo, de manera que se evite la saturación del planificador (ésta tiene lugar cuando se exigen restricciones de planificación imposibles de cumplir). Una vez todos los conmutadores que componen el camino entre la fuente y el destino de la conexión han pasado todos las comprobaciones, la conexión se acepta. Entonces, los requisitos de retardo y probabilidad (en su caso) de cada nodo, se calculan en función de los requerimientos globales especificados para la conexión.

Una ligera variante de DEDD también proporciona pequeños valores de *jitter* además de garantías de retardo. ***Jitter Earliest-Due-Date* (JEDD)** [168] añade reguladores a las entradas del conmutador con el objeto de provocar retardos extra a los paquetes que han llegado más temprano de lo esperado, de manera que cada paquete perteneciente a una conexión experimenta el mismo retardo de conmutador. JEDD puede requerir una mayor complejidad en el conmutador.

Esquemas orientados al ancho de banda

Al contrario que los algoritmos discutidos anteriormente, que se centran sobre las cotas en los retardos, muchos enfoques ampliamente aceptados están orientados principalmente hacia el control del ancho de banda. La idea básica fue desarrollada por Nagle [118] con el objeto de proporcionar un control de la congestión eficiente y servicios justos en las redes de datagramas convencionales. Para desacoplar el tráfico perteneciente a diferentes comunicaciones, se sugiere el empleo de colas separadas, las cuales se sirven cíclicamente (en *round-robin*). Si se compara con el esquema FCFS (*First Come-First Served*) convencional, este nuevo esquema de encolamiento y planificación mejora drásticamente la protección y la equidad (*fairness*) entre las conexiones.

La idea fue redefinida por Demers y otros, al proponer el algoritmo denominado ***Fair Queuing* (FQ)** [45] [2]. En este algoritmo se solucionan las limitaciones del original, que asumía paquetes tamaño único y cantidades de servicio idénticas para todos los usuarios finales. FQ garantiza un reparto del ancho de banda igual para cada conexión virtual, mediante la emulación bit a bit de la planificación *round-robin* a nivel de paquete. A cada

[2] FQ también se conoce como *Generalized Processor Sharing* (GPS).

paquete se le asigna un número de vuelta virtual, que sería el momento en que completaría su transmisión bajo planificación *round-robin* bit a bit. Se le asigna la mayor prioridad al paquete con el menor número de vuelta de finalización. La idea puede extenderse fácilmente para reservar diferente ancho de banda para cada conexión.

A pesar de que FQ garantiza servicios justos y el ancho de banda reservado a los usuarios finales, también tiene inconvenientes importantes en cuanto a su complejidad de implementación. En primer lugar, y para poder dar servicio a los paquetes en un orden diferente al de su llegada, FQ necesita colas distintas por cada conexión. En segundo lugar, los paquetes deben ser ordenados según sus prioridades; esto requiere de un soporte hardware especial para reducir el retardo de la planificación. Puesto que la complejidad de planificación se incrementa en proporción al número de conexiones que se han de soportar en un conmutador, estos esquemas no son viables si tenemos un elevado número de conexiones. Mientras que estas dos fuentes de complejidad son comunes a casi todos los esquemas basados en prioridades con una cola por cada conexión, una característica exclusiva de FQ que dificulta su implementación eficiente es el cálculo de la prioridad para cada paquete entrante. FQ necesita una gran sobrecarga para este cálculo debido a que emula un modelo fluido bit a bit.

En [128] se introduce un esquema muy similar a FQ, denominado ***Packet-by-Packet Generalized Processor Sharing* (PGPS)** [3]. Su aportación principal es demostrar que los mecanismos de planificación justos también pueden emplearse para garantizar cotas a los retardos, si se emplean controles bien definidos sobre el caudal de tráfico en las entradas a la red. Cuando las ráfagas de una conexión se restringen mediante un mecanismo de control del caudal en la fuente, como por ejemplo *token bucket* [124], la congestión dentro de la red en el peor de los casos está limitada, de manera que las cotas en los retardos pueden ser garantizadas. La cota del retardo de una conexión está determinada únicamente por la cantidad de ancho de banda reservado y el tamaño de ráfagas permitido. Además, si se emplea el control de caudal por la fuente, puede evitarse el desbordamiento de los *buffers*, aún disponiendo de una capacidad de almacenamiento limitada en el conmutador.

Un esquema similar a FQ y PGPS es ***Virtual Clock* (VC)**, desarrollado por Zhang [178]. VC lleva a cabo las siguientes funciones:

- Controla la tasa media de transmisión de los flujos de datos VBR.
- Hace que la utilización media de recursos por parte de cada usuario se corresponda con su productividad especificada.
- Proporciona protección entre los flujos individuales.
- Soporta servicios de prioridad multinivel.

En este algoritmo, cada flujo se modela mediante dos variables: *tasa media* (*AR*: *average rate*) e *intervalo medio* (*AI*: *average interval*). Esto significa que, al dividir la cantidad de datos recibidos durante cada periodo de tiempo AI, debería resultar en AR. El valor mínimo de AI es $1/AR$. En ese caso, estaríamos ante un flujo CBR. El valor máximo es evidentemente toda la duración del flujo, con lo que tendríamos un patrón de tráfico totalmente incontrolado. Un buen valor de AI debe permitir un control del flujo por parte de la red, a la vez que debe poder reflejar posibles variaciones en la llegada de paquetes, dentro de cada AI. De esta forma, la tasa media medida sobre cada periodo AI, permanecerá relativamente constante.

[3] PGPS también se conoce como *Weighted Fair Queuing*(WFQ).

El algoritmo VC está inspirado por las redes TDM (*Time Division Multiplexing*). En estas redes, cada usuario tiene garantizada una tasa de transmisión, al reservar ciertos *slots* determinados dentro de una trama para transmitir sus paquetes. Esta reserva es fija, con lo cual es posible que se desaprovechen *slots* al no haber paquetes pertenecientes al flujo correspondiente listos para ser transmitidos. Además, sólo soporta adecuadamente tráfico CBR. Por el contrario, los flujos están perfectamente aislados entre sí, y su productividad está garantizada.

VC intenta conseguir las mismas ventajas ofrecidas por TDM, a la vez que preservar las ventajas de la multiplexación estadística. Por tanto, la red debería asignar *slots* a los flujos bajo demanda. La red deberá regular el uso de los recursos únicamente cuando aparecen conflictos en la demanda, para garantizar la productividad que reservó cada flujo. Un sistema TDM regula el uso de recursos mediante un reloj de tiempo real: todos los usuarios de los canales envían datos por turnos cuando el reloj avanza. Un sistema multiplexado estadísticamente puede emplear un *reloj virtual* de una forma similar.

Para hacer que un flujo de datos VBR sea similar a un canal TDM, imaginamos que los paquetes del flujo llegan espaciados por un intervalo constante en tiempo virtual, de manera que la llegada de cada paquete indica que ha pasado un *slot* de tiempo. Se puede asignar a cada flujo de datos un *Reloj Virtual*, que avanza con cada llegada de un paquete de ese flujo. Si se hace que el paso de reloj se corresponda con el tiempo entre dos paquetes, el valor del Reloj Virtual denotará el tiempo de llegada esperado para el paquete que llega. Para imitar el orden de transmisión de un sistema TDM, se hace que cada nodo marque los paquetes de cada flujo con su valor de Reloj Virtual. Las transmisiones de paquetes se ordenan de acuerdo con los valores marcados, como si la marca de Reloj Virtual fuese el número de *slot* en tiempo real en un sistema TDM.

La implementación real del algoritmo emplea dos variables en lugar de una sola para representar el tiempo virtual: $VirtualClock$, y $auxVC$, para controlar que el flujo se atiene a los valores especificados para AI y AR. Cada conmutador lleva a cabo las dos siguientes funciones básicas:

Avance de datos: Hay una cola de paquetes por cada puerto de salida. Los paquetes se sirven de la forma siguiente:

1. Al recibir el primer paquete del flujo i:

$$VirtualClock_i \leftarrow auxVC_i \leftarrow tiempo\ real$$

2. Al recibir cada paquete del flujo i:
   a) $auxVC_i \leftarrow max(tiempo\ real, auxVC_i)$
   b) $VirtualClock_i \leftarrow (VirtualClock_i + Vtick_i)$;
   $auxVC_i \leftarrow (auxVC_i + Vtick_i)$
   (Para paquetes de tamaño constante, $Vtick_i = 1/AR_i$ paquetes/seg)
   c) Marcar el paquete con el valor de $auxVC_i$
3. Insertar el paquete en su cola de salida. Los paquetes se insertan y se sirven según el orden creciente de sus valores marcados.

Monitorización de los flujos: El conmutador calcula una variable de control

$$AIR_i = AR_i \times AI_i$$

cuando se establece cada flujo i. Al recibir cada conjunto de AIR_i paquetes del flujo i, el conmutador efectúa las siguientes comprobaciones:

- Si $(VirtualClock_i - tiempo\ real) > T$, donde T es un umbral de control, se envía un mensaje de aviso a la fuente del flujo. La diferencia entre $VirtualClock_i$ y el *tiempo real* es una medida de cuánto se ajusta la tasa a la que está transmitiendo un flujo a lo especificado. Cuanto mayor es esta diferencia, mayor es la diferencia entre la tasa especificada y la tasa efectiva de transmisión. Dependiendo de cómo reaccione la fuente, puede que sean necesarias más acciones de control.
- Si $(VirtualClock_i < tiempo\ real)$, $VirtualClock_i \leftarrow tiempo\ real$. En este caso, el flujo i ha transmitido por debajo de la tasa especificada. Al sincronizar $VirtualClock_i$ con el *tiempo real*, se evita que un flujo vaya acumulando indefinidamente sus *slots* de transmisión, para en un momento dado, saturar la red enviando toda la información de golpe. Esta acumulación de *slots* solo puede darse dentro de un intervalo AI, no entre varios, de manera que se permite una cierta variabilidad en la transmisión de los datos.

En este momento, el conmutador también sincroniza los valores de $VirtualClock$ y $auxVC$, siempre y cuando esto no haga que los paquetes del mismo flujo sean servidos fuera de orden (es decir, cuando el enlace de salida del flujo i está desocupado, o el paquete que se está transmitiendo tiene un valor de marca mayor que $auxVC_i$):

$$auxVC_i \leftarrow VirtualClock_i$$

Al encolar los paquetes según el orden de sus marcas de tiempo, hace que los paquetes procedentes de flujos distintos se sirvan muy intercalados entre sí, como en un sistema de servicio *round-robin*.

El espacio de almacenamiento en los conmutadores está completamente compartido. Cuando este espacio se agota, el algoritmo VC descarta el último paquete (es decir, el de mayor valor de marcado) de la cola más larga.

En la Figura 3.17 se muestra un ejemplo de funcionamiento de VC, comparándolo con el resultado obtenido con *First Come First Served*. Cada conexión especifica un tiempo entre llegadas de paquetes de 2, 5 y 5 unidades de tiempo, respectivamente. Las conexiones 2 y 3 transmiten a una mayor tasa que lo especificado, incumpliendo así su contrato con la red. Si se emplea FCFS como criterio de transmisión de paquetes, se observa que el flujo 1, que transmite ateniéndose a lo especificado, resulta perjudicado. En cambio, aplicando VC, cada conexión tiene prefijado un tiempo de transmisión, que se emplea cuando existen conflictos por el uso del enlace de salida. Así, podemos comprobar que la conexión 1 consigue transmitir sus paquetes a su tiempo, independientemente del comportamiento de los demás flujos.

La red puede establecer un orden de prioridades entre los flujos simplemente sustituyendo *real time* por *real time* $- P$ en el algoritmo anterior, siendo P un cierto valor que representa la prioridad. Este valor debe escogerse lo suficientemente grande como para que sea capaz de separar los paquetes pertenecientes a flujos de mayor prioridad de los de menor en la cola de servicio. Para los paquetes de tráfico *best-effort* se escoge un valor de P de $-\infty$, de manera que su marca vale ∞, y por tanto siempre estarán almacenados al final de la cola de servicio. Así, estos paquetes solo utilizarán los recursos que sobran tras atender los flujos que necesitan garantías de prestaciones.

Hemos visto que VC ofrece buenas propiedades de productividad, y aislamiento entre flujos, así como un reparto justo del ancho de banda. Sin embargo, su mayor inconveniente es su complejidad de implementación. Otro problema es que el uso de un reloj de referencia implica que éste no puede ser puesto a cero hasta que el sistema esté inactivo. Por tanto,

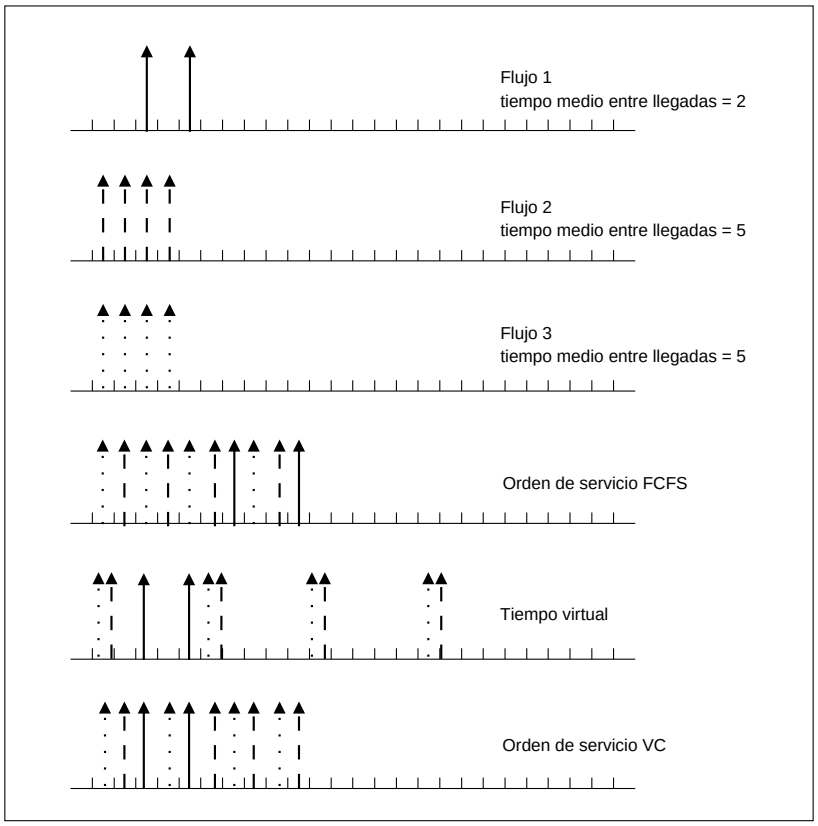


Figura 3.17: Ejemplo de funcionamiento de VC, frente a FCFS.

a menos que se empleen registros muy grandes para almacenar los tiempos virtuales, existe un problema potencial de desbordamiento numérico de los *buffers* si el sistema permanece ocupado durante un periodo largo de tiempo. En ese caso, el reloj virtual vuelve a contar desde cero, y esto lleva a una inestabilidad del algoritmo.

El algoritmo ***Credit-Based Fair Queuing*(CBFQ)** [22] se basa en un planteamiento similar al de VC, pero evita el uso de un reloj de referencia que pueda llegar a desbordarse. CBFQ emplea contadores para llevar la cuenta de de los créditos acumulados por cada flujo de tráfico. El algoritmo decide cuál será el siguiente paquete a enviar según el valor de los contadores, los tamaños de los paquetes situados a la cabeza de los distintos flujos (en el caso de admitir paquetes de tamaño variable), y la porción de ancho de banda reservada a los distintos flujos. El tamaño de los contadores se halla acotado por el tamaño del paquete más largo. Existe una optimización para el caso de tener paquetes de tamaño fijo, en la que los contadores se incrementan en valores menores de la unidad, y cuando alcanzan un valor mayor que uno, se reinicializan a cero, tras transmitir un paquete. De esta forma, los contadores no crecen de forma desmesurada.

Existen más aproximaciones a FQ que pueden reducir su complejidad de implementación con un sacrificio mínimo de sus beneficios (equidad y garantías de ancho de banda y, a veces, de latencia). Por ejemplo, ***Stochastic Fair Queuing* (SFQ)** [108] es una variante probabilística de FQ, pensada para redes de datagramas. Emplea funciones de *hashing* sencillas para mapear un par de direcciones fuente-destino, como en los puentes de Internet, a su cola correspondiente. Con el gran número de colas que se proporciona, la probabilidad de colisión entre múltiples conexiones es muy baja. Además, aproxima la planificación de FQ con un algoritmo *round-robin* estricto. SFQ es un equilibrio entre la equidad y el aislamiento del tráfico, para permitir una implementación simple.

En [68], Golestani propone un mecanismo simple capaz de proporcionar la equidad de FQ, pero con una complejidad mucho menor. En lugar de emular un modelo de flujo fluido para calcular la prioridad de los paquetes entrantes, él sugiere tomar el tiempo virtual de la conexión que se está sirviendo actualmente como tiempo virtual del sistema. De ahí su nombre: ***Self-Clocked Fair Queuing* (SCFQ)**.

Más recientemente, Stiliadis y Varma propusieron un enfoque distinto con el mismo propósito. Su nuevo algoritmo de planificación, ***Frame-based Fair Queuing* (FFQ)** [157, 159], es muy similar a SCFQ, excepto que emplea la noción de trama para gestionar el tiempo virtual del sistema. Esto puede garantizar cotas de latencia mucho menores que SCFQ. Aparte de eso, el mecanismo de planificación es prácticamente el mismo que los otros esquemas basados en prioridad.

Algoritmos de planificación por ráfagas

En toda red de conmutación de paquetes, éstos tienen un tamaño máximo. La mayoría de las unidades de datos del nivel de aplicación son demasiado grandes como para poder ser transportadas en un único paquete; por tanto, se hace necesario segmentarlas. Desde el punto de vista de la aplicación, los retardos entre extremos y la tasa de pérdidas de sus unidades de datos son más importantes que los que se especifican para paquetes individuales.

Partiendo de la observación anterior, en [100] se introduce el concepto de *ráfaga*, como la secuencia de paquetes que encapsula una unidad de datos de una aplicación. En este modelo, un flujo de tráfico sigue siendo una secuencia de paquetes, pero además varios de esos paquetes irán agrupados formando ráfagas. Además, las garantías de QoS se proporcionan a

las ráfagas en lugar de a paquetes individuales, y la reserva de recursos se efectúa en base a las ráfagas (*burst-based rate allocation*).

La idea para diseñar un algoritmo de planificación adecuado consiste en tomar una disciplina de planificación de paquetes para cada canal, y modificarla de manera que proporcione garantías de retardo al nivel de ráfaga. Se parte de un algoritmo basado en prioridades, donde cada paquete tiene asignado un cierto *deadline* o prioridad. A la hora de transmitir un paquete, el planificador selecciona el de menor *deadline* (mayor prioridad) de entre los que se encuentran a la cabeza de sus respectivos flujos.

En [100], sus autores escogen *Virtual Clock* como algoritmo de partida. Lo que hacen es modificar los valores de marcado de los paquetes para que dependan de los parámetros de la ráfaga.

Una optimización propuesta en [101] consiste en modificar el planificador de canales, para que implemente *prioridad de grupos*. La idea consiste en dividir en grupos los paquetes pertenecientes al mismo flujo, según van llegando. El mayor *deadline* de entre ellos, es el que se asigna a todos los paquetes del grupo. Por tanto, todos los paquetes del grupo excepto uno, tienen *deadlines* relajados. Para cada ráfaga m, el tamaño de grupo g_m es un parámetro cuyo valor ha de escogerse de manera que el retardo total de ráfaga de un flujo en el peor caso no se vea afectado por el uso de la prioridad de grupos.

La prioridad de grupos tiene dos ventajas. En primer lugar, el planificador tiene que llevar a cabo mucho menos trabajo, sobre todo cuando la utilización del canal es muy alta. Esto es debido a que la prioridad de un flujo solo cambia una vez por grupo, en lugar de una vez por paquete; por tanto, el planificador ha de actualizar sus estructuras de datos de prioridad con menos frecuencia. En segundo lugar, el uso de la prioridad de grupos ofrece mejores prestaciones estadísticas (retardo, tamaño de las colas, probabilidad de pérdidas) en redes donde se utilizan mucho algunos canales, pues se utiliza un tamaño de grupo grande para una ráfaga larga con una elevada tasa reservada. Esto se traduce en una relajación de los *deadlines* para muchos de los paquetes de dicha ráfaga, y mejores prestaciones estadísticas en un canal altamente utilizado.

3.5.2. Algoritmos basados en tramas

Una aproximación más simple y directa para obtener garantías de ancho de banda es emplear multiplexación por división en el tiempo sobre los enlaces físicos compartidos. La idea básica consiste en reservar para cada conexión varias ranuras (*slots*) de paquetes (que son el tiempo que se tarda en enviar un paquete) dentro un periodo fijo de tiempo, llamado *trama*. Esta reserva se hace en función de las demandas de ancho de banda de la conexión. Sin embargo, y debido a que este sencillo esquema desperdicia ancho de banda para el tráfico de ráfagas, no es el modelo adecuado para soportar redes de servicios integrados. Se han propuesto muchos algoritmos que pueden solucionar sus limitaciones, a la vez que mantienen su idea básica y sus beneficios para determinados servicios, dando lugar a los denominados algoritmos de planificación basados en tramas.

Como se comentó previamente, existen dos clases de algoritmos basados en tramas, los que conservan el trabajo y los que no, dependiendo de si el algoritmo introduce retardos adicionales al tráfico para preservar su forma, y desperdiciando así parte del ancho de banda. A continuación se describen algunos ejemplos de cada tipo.

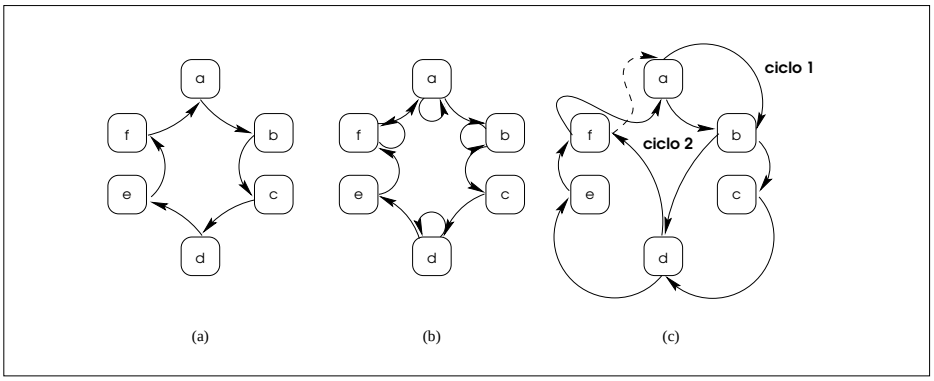


Figura 3.18: (a) Round-robin. (b) Round-robin ponderado. (c) Orden de las visitas.

Algoritmos que conservan el trabajo

Este tipo de algoritmos emplean la simplicidad del control de ancho de banda trama a trama sin desperdiciar recursos. Una forma de hacerlo es permitir que las conexiones utilicen cualquier ranura en lugar de tener que usar ranuras fijas, y permitir que el tamaño de las tramas varíe con el tiempo.

En [90], Katevenis y otros introducen un simple mecanismo hardware que implementa una planificación uniforme *round-robin* entre las conexiones, con diferentes pesos de servicio. El algoritmo se denomina ***Round-Robin* Ponderado (WRR, *Weighted Round-Robin*)**, y se basa en distribuir el ancho de banda disponible en un enlace entre los circuitos virtuales (CV) que lo utilizan, en proporción a unos pesos arbitrarios preestablecidos.

WRR consiste en lo siguiente. En *round-robin* convencional (RR), se recorren circularmente todos los CV, enviando un paquete de cada uno de entre los que tengan paquetes por enviar. De esta forma, en cada vuelta del algoritmo, todos los CVs tienen una oportunidad para enviar un paquete. Este procedimiento distribuye el ancho de banda a partes iguales entre todos los CV, asumiendo que el tamaño de paquete es el mismo para todos ellos. Además, si algún CV no utiliza todo su ancho de banda, el exceso se reparte por igual entre el resto de CV que pueden utilizarlo.

Ahora bien, se pretende distribuir el ancho de banda de un enlace de forma distinta, de forma que unos CV dispongan de más ancho de banda que otros, en base a unos pesos designados al establecer el CV. Una forma de hacerlo es, en cada vuelta del algoritmo, "visitar" cada CV un número de veces proporcional a su peso. Esto es básicamente lo que propone WRR. La idea se muestra de forma intuitiva en la Figura 3.18, donde los cuadrados simbolizan CVs con paquetes destinados a un puerto de salida común.

En la parte (a) de la Figura 3.18 se ilustra el algoritmo clásico RR, donde en cada ciclo de servicio, cada cliente es visitado exactamente una vez. La parte (b) es una primera aproximación al algoritmo WRR. Los clientes a, b, d y f reciben servicio dos veces más frecuentemente que los clientes c y e, puesto que los primeros son visitados 2 veces por cada 10 visitas, y los últimos sólo una. En la parte (c), se mantienen las frecuencias de las visitas, pero las que se hacen a los clientes "frecuentes" se espacian más en el tiempo. En [90] se adopta este último estilo: el ciclo de planificación se divide en subciclos, etiquetados como 1, 2, ..., N-1, N. Cada CV tendrá asociado un cierto peso, que hará que reciba una unidad de servicio durante cada ciclo cuya etiqueta corresponda a un subconjunto específico de los números 1, 2, ..., N. En [90] aparecen más detalles sobre la implementación en hardware del algoritmo.

Otra disciplina de servicio flexible y simple es ***Deficit Round Robin* (DRR)** [147], que puede soportar tamaños de paquete variables, y diferentes cantidades de reserva de ancho de banda para distintas conexiones. A cada conexión se le asigna el valor de un contador que se incrementa por un valor propio o *quantum* al principio de cada vuelta, y se decrementa por el tamaño de cada paquete transmitido. Las partes del ancho de banda que corresponden a cada conexión vienen determinadas por sus valores de *quantum*. En cada ronda, se envían paquetes pertenecientes a una conexión mientras que su valor del contador sea mayor que el tamaño del paquete situado a la cabeza de su cola. De esta forma, se lleva la cuenta de la cantidad de bits transmitidos por cada conexión, y se evita un reparto injusto del ancho de banda en el caso de que existan diferencias sustanciales en el tamaño de los paquetes pertenecientes a distintos flujos[4]. DRR puede ser implementado a un bajo coste porque las conexiones activas se atienden en estricto orden *round-robin*.

Algoritmos que no conservan el trabajo

Al contrario que los algoritmos anteriores, que permiten una utilización más eficiente de los recursos de la red, algunos investigadores proponen efectuar un control del caudal a nivel de conmutador añadiendo retardos intencionados en cada salto. Este control del caudal a nivel de conmutador conlleva el desperdicio de ancho de banda. Hay dos motivos que lo justifican:

1. La regulación del tráfico a la entrada de la red, como por ejemplo mediante un algoritmo *leacky bucket* [167, 29], no es suficiente. Esto es porque la suavidad del nivel de inyección no puede preservarse dentro de la red, debido a las interacciones irregulares e impredecibles de un conjunto de flujos de tráfico.

2. Muchas comunicaciones en tiempo real sólo requieren que los paquetes lleguen dentro de la cota de retardo, no antes, de forma que un ancho de banda extra puede que no les sea útil.

***Stop-and-Go* (SG)** [65, 66] es un ejemplo bien conocido de las disciplinas de servicio que no conservan el trabajo. Consiste en regular de forma estricta la inyección de paquetes en la red, de forma que la fuente de una conexión sólo puede enviar, como máximo, tantos paquetes como ranuras reservó. Además, cada conmutador transmite *todos* los paquetes que llegaron durante la última trama de tiempo, y *solamente esos*.

En [66] se propone una estrategia de control que proporciona comunicación sin pérdidas a la vez que una productividad garantizada, mantiene un retardo entre extremos por conexión que es igual a una constante más un pequeño término de *jitter* acotado, y es sencillo de implementar.

Esta estrategia tiene dos partes: una política de admisión aplicada a todas las conexiones en el nodo fuente, y un esquema de gestión de colas particular, efectuado en los conmutadores, que se denomina ***Stop-and-Go* (SG)**.

La política de admisión consiste en que, una vez que se establece una conexión, con una tasa de transmisión r_k, su patrón de entrada a la red ha de cumplir que, en todo periodo de tiempo T (T es el tamaño de trama), no lleguen más de $r_k \times T$ bits en total. Esto es, si los paquetes son de tamaño fijo S, no se admitirán en la red más de $(1/S) \times r_K \times T$ paquetes.

[4]Si se emplea *round-robin* junto a paquetes de tamaño variable, los flujos de datos que empleen un mayor tamaño de paquete obtienen en la práctica una mayor fracción del ancho de banda. En [99] se ha empleado esta propiedad para ofrecer garantías de ancho de banda a tráfico *best-effort* en redes *wormhole*.

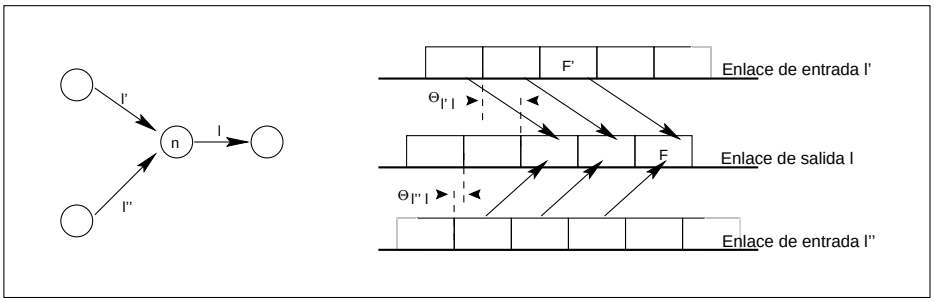


Figura 3.19: Ilustración de la disciplina *Stop-and-Go*.

Eso equivale en la práctica a reservar en cada trama el número necesario de *slots* necesarios para transmitir esa cantidad de paquetes, como máximo.

Conceptualmente, se considera que las tramas viajan por los enlaces, junto con los paquetes, de tal forma que en los enlaces de entrada a un conmutador habrá *tramas entrantes*, y en los de salida *tramas salientes*, separadas entre sí un tiempo T_l, que es el tiempo de propagación por el enlace, más el tiempo consumido al recibir y procesar un paquete en el conmutador. Las tramas entrantes correspondientes a los distintos enlaces de entrada no tienen por qué estar sincronizadas entre sí. Se dice además que dos tramas F y F' son *adyacentes* entre sí si, en un nodo, F es la primera trama saliente que empieza tras finalizar la trama entrante F' en dicho nodo. En un nodo, existe una *discordancia de fase* (*phase mismatch*) entre un enlace de entrada l y uno de salida l' que se denota como $\Theta_{l,l}$, y se define como el tiempo transcurrido entre el final de una trama entrante por l y el principio de la trama adyacente saliente por l'. El valor de la discordancia de fase entre dos enlaces se halla comprendido entre 0 y el tamaño de la trama T. En la Figura 3.19 se muestran los pares de tramas adyacentes y las discordancias de fase entre los enlaces de entrada y salida en un nodo.

El esquema de planificación *Stop-and-Go* se basa en las siguientes dos reglas:

Regla A Sea un nodo n, un enlace de salida l, uno de entrada l', y un paquete que llega durante una trama F' del enlace l', y que debe salir por el enlace l. Sea F la trama saliente del enlace l que es adyacente a la trama entrante F'. Entonces, la transmisión del paquete no debe empezar antes del inicio de la trama F.

Regla B Un enlace no debe permanecer desocupado mientras queden paquetes elegibles en la cola. Un paquete es *elegible* si puede ser transmitido sin violar la Regla A.

Los paquetes nuevos se designan como elegibles en un enlace solamente al inicio de las tramas salientes, y durante cualquier trama saliente no se amplía el conjunto de paquetes elegibles. De esta forma, se puede asegurar que un paquete, una vez se marca como elegible, será transmitido en un tiempo T, como máximo.

Además, SG no influye en la forma del tráfico, es decir, no introduce ráfagas en los flujos de datos. Esto permite limitar el tamaño de *buffers* necesario en cada nodo para evitar desbordamientos.

Por otro lado, si se compara SG con la disciplina de servicio FIFO, con ésta última se obtienen menores retardos en media, debido a que no se aplica ningún retardo adicional a los paquetes en los conmutadores. Sin embargo, es posible que se formen ráfagas en la red, lo cual puede disparar los retardos de forma no acotada. SG obtiene un mayor retardo

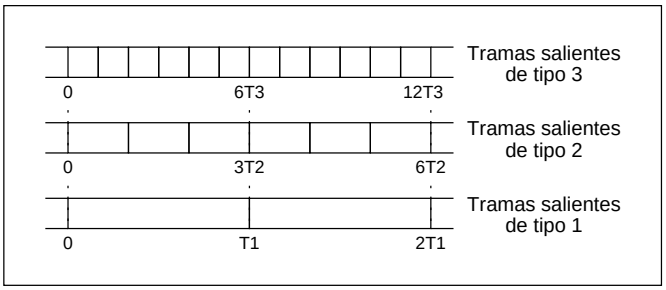


Figura 3.20: Algoritmo *Stop-and-Go*: Tramas salientes en un nodo para la versión multitrama, con 3 tamaños de trama, siendo $T_1 = 3T_2$, y $T_2 = 2T_3$.

medio, pero sin embargo, garantiza que los retardos posibles están comprendidos dentro de un pequeño rango. De esta forma, se pueden ofrecer garantías de servicio a los flujos de datos.

Un problema de SG, y en general de todos los algoritmos basados en tramas, es el acoplamiento existente entre la granularidad con que se puede reservar ancho de banda, y la cota del retardo que se puede garantizar a las conexiones. Ambos dependen del tamaño de trama T, pero de manera inversa. A mayor tamaño, menor es la granularidad de reserva de ancho de banda. Así, es posible ajustar mejor el ancho de banda reservado al realmente utilizado, mejorando la utilización de los enlaces. Por tanto, desde este punto de vista, resulta más provechoso un valor de T elevado. Por el contrario, el valor de la cota del retardo es mayor cuanto mayor es T, con lo cual serían deseables valores de T pequeños. Así debe buscarse una solución de compromiso para el valor de T que suponga un equilibrio entre ambas consideraciones.

En [65], el mismo autor propone como mejora el empleo de distintos tamaños de trama para mejorar la granularidad de la reserva de ancho de banda, sin perjudicar a las cotas de retardo. Se definen, por tanto, G tamaños de trama T_1, T_2, ..., T_G, con $T_g > T_{g+1}$, comunes a toda la red. Además, cada T_g es un múltiplo de T_{g-1} (Figura 3.20), aunque también es posible emplear tamaños de trama arbitrarios.

La idea es que cada conexión realice su reserva en una de las tramas disponibles. A la hora de planificar los paquetes, se establece un esquema de preferencia entre los distintos tipos de trama, de manera que se planifican antes los paquetes de conexiones que hicieron su reserva en las tramas de tipo mayor. Se proveen así varias "clases de servicio". Cada conexión, al establecerse, elige la trama donde efectúa su reserva, es decir, indica su "clase de servicio". Este esquema se puede ampliar para soportar tráfico *best-effort*, adjudicándole un tipo de trama igual a 0, de manera que únicamente sea servido en el caso de que no existan paquetes de tipos superiores.

Tanto la estrategia propuesta en [65] como la multitrama de [66], se pueden implementar fácilmente como modificaciones a una estructura de colas FIFO.

Por otro lado, con el algoritmo tal y como se plantea en [65], no existe flexibilidad para soportar eficientemente varios tipos de tráfico, puesto que siempre debe efectuarse la reserva del ancho de banda en base a los requisitos máximos. Esto funciona bien para tráfico CBR, pero provoca una baja utilización de los enlaces con tráfico VBR.

En [67] se propone otra modificación que intenta dar soporte a tráfico VBR, mejorando la utilización de la red, e intentando alcanzar un equilibrio entre ganancia estadística y probabilidad de pérdida de paquetes. A la vez, se preservan las cotas garantizadas en los retardos.

Otra disciplina que se encuadra en este tipo de algoritmos es ***Hierarchical Round Robin* (HRR)** [82]. HRR es parecida a SG, en el sentido de que el ancho de banda se controla mediante el número de ranuras reservadas en una trama de tiempo. Igual que hace SG, cada conexión reserva un número fijo de ranuras por trama de tiempo. Ahora bien, HRR soporta una jerarquía de niveles de clases, cada una de las cuales se atiende con un peso diferente en cada vuelta. La cantidad de ancho de banda reservado se controla mediante el nivel de clase y el número de ranuras reservadas.

HRR, al igual que SG, ofrece interesantes ventajas, como su simplicidad y la garantía de servicio. Además, HRR proporciona protección frente a usuarios con comportamiento anómalo, puesto que a cada conexión solo se le permite utilizar las ranuras que reservó. Sin embargo, este beneficio adicional se acompaña de una política de colas orientada a la conexión, que es necesaria para planificar paquetes en orden distinto al de su llegada.

Otra diferencia de HRR respecto a SG es que no provoca retardos mínimos: los paquetes que llegan al conmutador justo antes que sus ranuras reservadas, pueden atravesarlo inmediatamente. Esto puede ser bueno para algunas aplicaciones de reparto rápido de paquetes, pero no tanto para otras aplicaciones por la gran variación provocada en el retardo (*jitter*).

3.5.3. Resumen

En general, los algoritmos *basados en tramas*, conserven o no el trabajo, aventajan a los algoritmos *basados en prioridades* en que tanto las cotas en los retardos, como el ancho de banda, se garantizan de forma determinista al garantizar *una cantidad fija de tráfico dentro de cierto intervalo de tiempo.*

Más aún, en los algoritmos *basados en tramas*, se pueden analizar los retardos en los nodos de manera independiente, y luego sólo hay que sumarlos para determinar las cotas de retardo entre extremos. Estas propiedades hacen el análisis de la QoS, la predicción de servicios, e incluso los procesos de establecimiento de la conexión mucho más simples en comparación con los esquemas *basados en prioridades.*

Por contra, los algoritmos *basados en tramas* tienen una limitación: el acoplamiento entre el retardo y la granularidad de reserva del ancho de banda. Tanto la cota del retardo como la unidad de reserva de ancho de banda vienen determinadas por el tamaño de la trama, que a su vez viene determinado por la capacidad del enlace y la unidad mínima de reserva de ancho de banda. Con mayores tamaños de trama, podemos soportar conexiones con un rango más amplio de requerimientos de ancho de banda, pero las cotas en los retardos en cada conmutador aumentan proporcionalmente. En un intento de llegar a una solución de compromiso, algunos investigadores sugieren el empleo conjunto de varios tamaños de trama para reducir este problema de acoplamiento [65, 82]. Esto puede eliminar la limitación en cierta medida, al precio de necesitar un control más complejo.

Por último, los *algoritmos de planificación por ráfagas* suponen una variante interesante sobre los algoritmos basados en prioridades, puesto que tienen en cuenta también los requisitos de QoS que necesitan las aplicaciones. De esta forma, son capaces de optimizar los recursos de la red para transmitir la información de forma que sea de utilidad a las aplicaciones que la reciban, así como de aprovechar la información acerca de la estructura de los flujos para optimizar la velocidad a la que se realiza la planificación.

3.6. Planificación del tráfico en conmutadores para SANs

Como se comentó en el Capítulo primero de Introducción, una SAN (*System Area Network*) es una forma alternativa al clásico bus de sistema para combinar elementos hardware estándar para lograr sistemas verdaderamente escalables que proporcionan prestaciones y productividad superiores, de manera fiable. En este apartado se comentarán un par de propuestas de planificación desarrolladas para este entorno, que tienen en cuenta cierta provisión de garantías de QoS.

3.6.1. Algoritmo de arbitraje *ALU-biasing*

La primera SAN introducida en el mercado fue la red ServerNet [73]. ServerNet emplea de encaminadores *wormhole*, implementados con un *crossbar* con *buffers* FIFO en sus entradas. Estos encaminadores introducen un algoritmo de arbitraje, denominado *ALU-biasing* que persigue objetivos como la reserva y reparto del ancho de banda de los enlaces entre distintos flujos de información entre conmutadores y distribuir el ancho de banda sobrante de forma proporcional entre otras conexiones, todo ello con prestaciones y coste comparables a los de los encaminadores de multicomputadores existentes.

El *arbitraje ALU-biasing* reparte el ancho de banda de salida entre los enlaces de entrada de forma proporcional a su ancho de banda reservado. Los encaminadores pueden ser programados en línea para reservar fracciones arbitrarias del ancho de banda de los enlaces de salida para distintos enlaces de entrada.

La idea básica consiste en que al controlar el ancho de banda de cada enlace en el encaminador, se puede controlar el ancho de banda acumulado entre dos nodos que se comunican. Mediante el arbitraje *ALU-biasing*, a cada puerto de entrada se le asigna un contador que mantiene la historia del tráfico, y un valor de incremento fijo que determina la fracción del ancho de banda del enlace de salida reservada para ese puerto de entrada. El encaminador controla la distribución del ancho de banda por medio de simples reglas de actualización del contador:

1. El puerto de entrada con mayor valor en su contador siempre gana la transmisión por un puerto de salida. Sólo los puertos de entrada con paquetes pendientes para ese puerto de salida participan en el arbitraje.

2. Los perdedores aumentan sus prioridades incrementando sus contadores en el valor de cada incremento, y el ganador disminuye su prioridad restando a su propio contador la suma de los incrementos de los perdedores.

De esta manera, *ALU-biasing* mantiene limitados los valores de los contadores dentro de un estrecho rango. El algoritmo aparece más detallado en la Figura 3.21.

Un ejemplo de asignación del ancho de banda en un conmutador ServerNet se muestra en la Figura 3.22. En la Tabla 3.3 se muestra una traza de la ejecución del algoritmo para esa configuración.

Efectivamente, se puede comprobar que de los 7 slots de tiempo planificados, la entrada C resulta ganadora para 4 de ellos, la entrada B para 2 y la entrada A para 1, con lo que se cumple la adjudicación proporcional de ancho de banda según los valores de los incrementos I_i, que se muestra en la Figura 3.22.

Algoritmo *ALU-biasing*:

Definiciones previas: Dado el puerto de entrada i:

- I_i : valor del incremento (cuanto más grande, más ancho de banda se reserva)
- C_i : contador (inicialmente, cero)
- S : suma de los valores de los incrementos de todos los puertos de entrada que compiten por el mismo puerto de salida
- $S_i = S - I_i$

Algoritmo de arbitraje: Cuando uno o más puertos de entrada compiten por un puerto de salida desocupado repetir:

- Puerto de entrada ganador $\leftarrow$ puerto de entrada con mayor C_i
- Para cada puerto de entrada i:
  - Si ganó, $C_i \leftarrow C_i - S_i$
  - Si perdió, $C_i \leftarrow C_i + I_i$
  - Si desocupado, $C_i \leftarrow C_i$

Figura 3.21: Algoritmo *ALU-biasing*.

Tiempo (*slots*)	Entrada A $I_A = 2;\ S_A = 12;\ C_A = 0$	Entrada B $I_B = 4;\ S_B = 10;\ C_B = 0$	Entrada C $I_C = 8;\ S_C = 6;\ C_C = 0$
1	$C_A = 2$	$C_B = 4$	$\mathbf{C_C = 8}$
2	$C_A = 2 + 2 = 4$	$\mathbf{C_B = 4 + 4 = 8}$	$C_C = 8 - 6 = 2$
3	$C_A = 4 + 2 = 6$	$C_B = 8 - 10 = -2$	$\mathbf{C_C = 2 + 8 = 10}$
4	$\mathbf{C_A = 6 + 2 = 8}$	$C_B = -2 + 4 = 2$	$C_C = 10 - 6 = 4$
5	$C_A = 8 - 12 = -4$	$C_B = 2 + 4 = 6$	$\mathbf{C_C = 4 + 8 = 12}$
6	$C_A = -4 + 2 = -2$	$\mathbf{C_B = 6 + 4 = 10}$	$C_C = 12 - 6 = 6$
7	$C_A = -2 + 2 = 0$	$C_B = 10 - 10 = 0$	$\mathbf{C_C = 6 + 8 = 14}$
...	...	...	...

Tabla 3.3: Ejemplo de ejecución del algoritmo *ALU-biasing*.

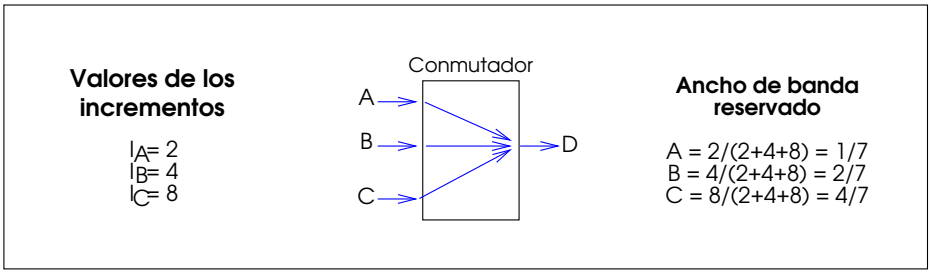


Figura 3.22: Ejemplo de reserva de ancho de banda con *ALU-biasing*.

Este algoritmo es una buena aproximación para garantizar calidad de servicio a las aplicaciones, sobre todo teniendo en cuenta que el coste de su implementación es bastante bajo: menos del 10 % del total de las puertas lógicas del ASIC que implementa el encaminador de ServerNet corresponden a la lógica de control del algoritmo. La velocidad de operación del encaminador tampoco se ve penalizada, pues la lógica de arbitraje se puede ejecutar en paralelo con la transmisión del paquete previo, de manera que el impacto en la velocidad de reloj es mínimo. El encaminador ServerNet opera a 50 MHz, velocidad comparable a la de otros conmutadores *wormhole* comerciales, como el SP1/2 de IBM, el CS-2 de Meiko o el del Paragon de Intel.

3.6.2. Algoritmo *Rotating Combined Queuing*

Otro algoritmo de planificación diseñado para su uso en entornos de SAN y multicomputadores es el denominado ***Rotating Combined Queuing*(RCQ)** [93]. Al contrario que la mayoría de conmutadores empleados en redes de multicomputadores, RCQ está diseñado para conmutadores con colas a la salida.

Los objetivos de diseño de RCQ se diferencian respecto a los perseguidos por *ALU-biasing* en que en este caso las garantías de servicio se van a ofrecer a conexiones individuales, y no simplemente a flujos que circulen por un mismo camino. Por lo demás, también se intenta obtener un algoritmo que se pueda implementar de forma simple y barata.

La idea básica de RCQ consiste en reducir los costes que conlleva la gestión del tráfico combinando múltiples colas desacopladas entre sí y que por tanto no pueden provocar bloqueo de cabeza de línea (*HOL blocking*). En la Figura 3.23, se muestra la organización de las colas, situadas en los módulos de salida del conmutador. Las colas paralelas son idénticas, y no implican ningún nivel de prioridad.

RCQ es similar a SQ [66] excepto en que RCQ puede soportar el tráfico a ráfagas de forma más eficiente, empleando varias colas FIFO por puerto de salida. RCQ también emplea el concepto de trama, donde cada conexión reserva un cierto número fijo de *slots* de paquete, dentro de un tiempo fijo de trama. Sin embargo, RCQ proporciona colas extra destinadas a almacenar paquetes que deben ser transmitidos en tramas futuras, con lo que las fuentes pueden inyectar paquetes a una tasa mayor que la reservada. Así, las ráfagas de tráfico pueden aprovechar eficientemente el ancho de banda que no se esté utilizando, mientras que el tráfico de tiempo real puede obtener sus garantías de servicio. El tamaño de cada cola es lo suficientemente grande como para contener una trama completa.

La notación que se va a emplear es la siguiente:

- $VCI(p)$: Identificador de la conexión virtual del paquete p.

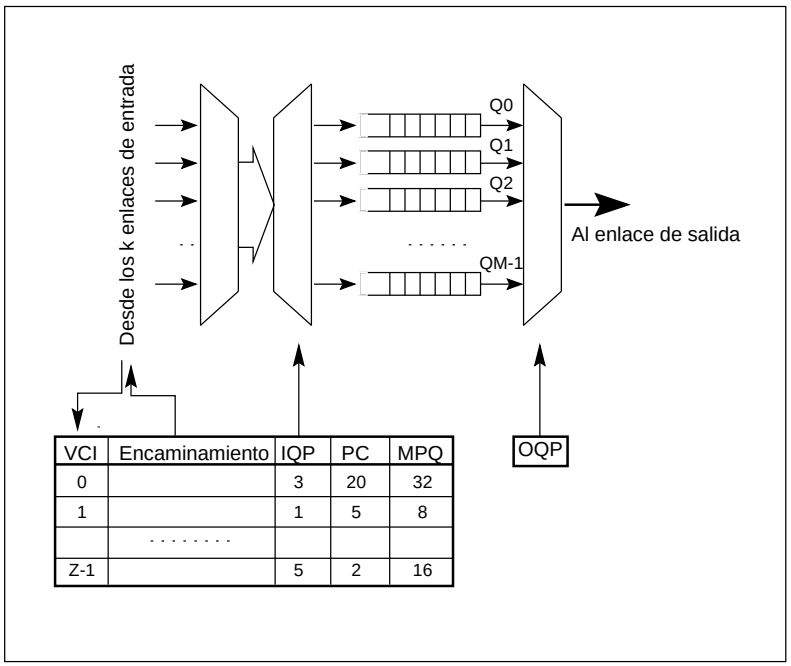


Figura 3.23: Un módulo de salida en un conmutador RCQ.

- Q_i : La cola i-ésima de un modulo de salida.
- IQP_i : Puntero a la cola de entrada actual, para la conexión i. Por ejemplo, si $IQP_5 = 3$, entonces los paquetes con $VCI = 5$ se almacenan en la cola 3.
- OQP : Puntero a la cola de salida actual. Siempre que el canal de salida está desocupado, se extrae un paquete de la cola indicada por OQP.
- PC_i : El número total de paquetes de la conexión i almacenados en su cola de entrada actual ($PC_i \leq MPQ_i$).
- MPQ_i : El número máximo de paquetes de la conexión i que se pueden almacenar en una cola.
- M : El número total de colas en cada módulo de salida.

Y el algoritmo RCQ funciona como se muestra en la Figura 3.24.

Inicialmente, se selecciona la primera cola para transmitir sus paquetes de forma inmediata. En la fase de establecimiento de cada conexión i, se inicializa su puntero a la cola de entrada IQP_i a OQP, para que sus paquetes se transmitan lo antes posible. Asimismo, el número máximo de paquetes por cola (MPQ_i) se inicializa con el valor del ancho de banda reservado para la conexión.

Por cada paquete que llega (P_{in}), se busca información referente a la conexión a la cual pertenece en la tabla de que dispone el conmutador. El paquete se almacena en la cola apuntada por IQP_i, y se incrementa el contador de paquetes PC_i. Si hemos alcanzado así el máximo número de paquetes permitidos en la cola actual ($PC_i == MPQ_i$), entonces se incrementa IQP_i, y el contador de paquetes se pone a 0. Esto indica que hemos transmitido ya los paquetes que corresponden dentro de la trama actual, y los siguientes paquetes que lleguen se transmitirán dentro de la trama siguiente.

Algoritmo *Rotating Combined Queuing* :

- Inicialización:
  - $OQP \leftarrow 0$
  - Para cada nueva conexión $i \leftarrow 0 \ldots Z-1$:
    1. $IQP_i \leftarrow *$
    2. $PC_i \leftarrow 0$
    3. $MPQ_i \leftarrow BW_i$
- Para cada paquete entrante P_{in}, repetir:
  1. $i \leftarrow VCI(P_{in})$
  2. $iqp \leftarrow IQP_i$; Si $iqp = 0$, entonces $iqp \leftarrow OQP$
  3. Añadir el paquete a la cola Q_{iqp}
  4. $PC_i \leftarrow PC_i + 1$
  5. Si $PC_i = MPQ_i$, entonces $IQP_i \leftarrow (iqp{+}1)\ mod\ M$, $PC_i \leftarrow 0$
- En cada ciclo vacío del canal de salida, repetir:

  Si hay algún paquete pendiente P_{out} en Q_{OQP},

  entonces
  1. transmitirlo
  2. Si $IQP_{VCI(P_out)} = OQP$, entonces $IQP_{VCI(P_out)} \leftarrow *$

  sino $OQP \leftarrow (OQP + 1)\ mod\ M$

Figura 3.24: Algoritmo *Rotating Combined Queuing* (RCQ).

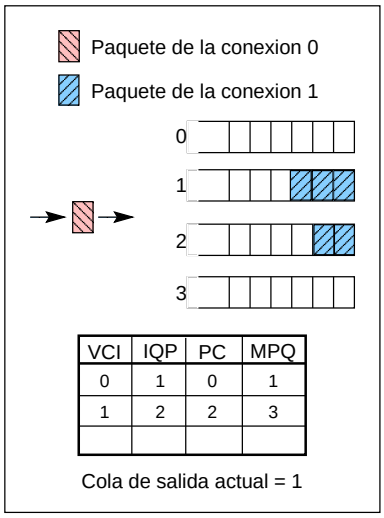


Figura 3.25: Ejemplo de funcionamiento de RCQ.

Cada ciclo que el canal de salida esté libre, se transmite el paquete que se encuentre a la cabeza de la cola apuntada por OQP. Si la cola se queda vacía tras transmitir ese paquete, el puntero OQP se avanza hasta la siguiente cola.

De esta forma, todo paquete que llegue dentro de la reserva de ancho de banda establecida para su conexión, sufrirá un retardo máximo del tiempo de una trama, puesto que sólo le afectan los paquetes que están en la cola de salida actual (OQP), y siempre tendrá disponibles en ésta el espacio que reservó (BW_i). Por tanto, el retardo máximo que puede sufrir un paquete en el peor de los casos está acotado por la distancia que ha de recorrer, multiplicada por el tiempo de una trama.

Si una cola se queda vacía, independientemente de que todas las conexiones hayan hecho uso de su ancho de banda o no, RCQ adjudica inmediatamente el ancho de banda sobrante a las ráfagas de datos, simplemente incrementando OQP. De esta forma, se salta los *slots* reservados por conexiones que no los han utilizado.

Veamos hora un ejemplo sencillo del funcionamiento de RCQ. En la Figura 3.25 se muestra la situación de la que partimos. Hay dos conexiones establecidas a través de este puerto de salida. La conexión 0 tiene reservado un *slot* por trama ($MPQ_0 = 1$) y transmite sin sobrepasar esa reserva. Por el contrario, la conexión 1 tiene reservados 3 *slots* ($MPQ_1 = 3$), y transmite con mayor tasa que la reservada. La cola de salida actual es la 1. Podemos ver que la conexión 1 ha agotado el espacio de almacenamiento de que dispone en la cola 1, y ha empezado a ocupar la cola 2 ($IQP_1 = 2$). La conexión 0 todavía no ha gastado su *slot* reservado. Supongamos que llega un paquete de la conexión 1. Si todavía no se ha vaciado la cola 1, se almacenará ahí en espera de su transmisión, puesto que sigue teniendo su sitio reservado y libre. En cambio, si se vacía la cola 1 antes de que llegue este paquete, la cola de salida pasará a ser la 2 y se empezarán a transmitir los paquetes que la conexión 1 almacenó ahí. Si en ese momento llega el paquete de la conexión 0, se almacenará en la cola 2, que es la cola de salida actual. Así, se podrá transmitir dentro de la trama actual.

RCQ mejora a otros algoritmos basados en tramas, como SG, empleando varias colas para prevenir el desaprovechamiento del ancho de banda. Al contrario que SG, que controla estrictamente la tasa de salida de cada conexión, RCQ permite que variaciones en el tráfico utilicen el ancho de banda reservado por otras conexiones pero que no está siendo utilizado.

Los paquetes de sobra que llegan a mayor tasa que la reservada se van encolando en las colas paralelas, de forma que no bloquean a los paquetes de otras conexiones que llegan a su tiempo. De esta forma, las colas múltiples no sólo soportan la planificación de paquetes en orden distinto al de llegada, sino que también permiten que las ráfagas de tráfico exploten el ancho de banda no utilizado. Al contrario que si se emplean colas independientes por conexión, el número de colas necesarias en RCQ es independiente del tamaño de la red y del número de conexiones virtuales. Viene determinado únicamente por el control de flujo o la cantidad de ráfagas que se quiera soportar[5].

Con el objeto de reducir la complejidad de la planificación a la vez que proporcionar aislamiento del tráfico con mínimas sobrecargas debidas al almacenamiento en colas, RCQ integra fuertemente la gestión de las colas y la planificación en un solo mecanismo. Este enfoque introduce un cierto acoplamiento del tráfico en contrapartida a su simplicidad y reducido coste, pero dicho acoplamiento está controlado de forma que se pueden ofrecer servicios garantizados al tráfico de tiempo real, penalizando sólo ligeramente al tráfico de ráfagas en cuanto a la productividad pico que puede alcanzar.

[5] O bien, al contrario: la cantidad máxima de ráfagas permitida para cada conexión viene determinada por el número de colas paralelas por puerto

Capítulo 4

MMR: Un Encaminador para Tráfico Multimedia

Tal y como se ha puesto de manifiesto en los Capítulos anteriores, existe una necesidad de provisión de Calidad de Servicio (QoS) en entornos locales debido al gran auge experimentado en los últimos años por aplicaciones que hacen uso intensivo de la transmisión de datos multimedia.

La motivación viene en dos sentidos. Por un lado, la necesidad de disponer de potentes servidores capaces de proporcionar los flujos multimedia requeridos por los usuarios. Cada vez más, estos servidores se basan en una arquitectura distribuida, empleando lo que se ha dado en llamar SAN (*System Area Network*) para mejorar las prestaciones, la fiabilidad y la escalabilidad del sistema. Por otro lado, muchas aplicaciones son intrínsecamente distribuidas, ejecutándose en redes de corto alcance o LAN. Dentro de esta categoría se podría incluir reuniones virtuales o juegos multiusuario. En cualquier caso, es necesaria una red de interconexión que ofrezca las prestaciones adecuadas.

Tradicionalmente, los avances en la tecnología de interconexión de redes de multicomputadores y SAN/LAN se han centrado en la obtención de baja latencia y alta productividad para el tráfico *best-effort* [142, 25, 73, 51, 152, 149]. Por tanto, estas redes no están preparadas para ofrecer garantías de servicio a múltiples aplicaciones simultáneamente.

El abanico de nuevas aplicaciones surgidas recientemente, exhibe diferentes mezclas de cálculo, comunicaciones y coordinación distribuida, de manera que imponen tanto una mayor demanda de prestaciones de la red (un cambio cuantitativo), como unos determinados requerimientos de servicio (un cambio cualitativo) [32].

Como se señaló en el Capítulo de Introducción, los requisitos de QoS impuestos por la mayoría de las aplicaciones multimedia necesitan de mecanismos específicos para proporcionarles un soporte adecuado. La mayoría de encaminadores empleados en multicomputadores y redes de estaciones de trabajo (NOWs) emplean conmutación *wormhole* o *virtual cut-through*. Estas técnicas son capaces de proporcionar bajas latencias a los mensajes en media, pero no proporcionan ningún soporte a la QoS.

En el Capítulo anterior, se han revisado algunas propuestas para soportar comunicaciones en tiempo real en redes de multicomputadores, a la vez que proporcionan comunicación con baja latencia para tráfico *best-effort* (página 46). Sin embargo, en estas propuestas se asume que los mensajes de tiempo real son muy cortos y solo consumen una pequeña parte del ancho

de banda. Por tanto, no son apropiados para proporcionar QoS a largos flujos de datos que pueden solicitar más ancho de banda del disponible en un cierto enlace.

La única técnica de conmutación diseñada para proporcionar QoS, soportar un gran número de conexiones, y alcanzar una latencia relativamente baja es ATM [137]. Sin embargo, ATM es fruto de un compromiso entre muchos intereses enfrentados, y está adaptada principalmente para redes de área amplia (WANs). Las principales limitaciones de ATM cuando se emplea en redes de área local (LANs) para soportar aplicaciones multimedia son las siguientes:

- Todo el tráfico en ATM es orientado a la conexión. Existe tráfico que por naturaleza no es orientado a la conexión, como los mensajes cortos de control generados por aplicaciones multimedia. Este tráfico, por tanto, necesita un servicio sin conexiones. Al emplear conexiones incluso para este tipo de tráfico, su latencia se incrementa de manera innecesaria, los recursos se emplean de manera ineficiente, y por tanto, puede convertirse en un cuello de botella del sistema.
- ATM no emplea control de flujo. Por el contrario, se basa en protocolos de control de admisión y en el multiplexado estadístico para mantener la contención sobre los enlaces de salida a niveles aceptables. Teniendo en cuenta que el tráfico puede ser irregular y que no existe ningún tipo de control de la utilización del ancho de banda de los enlaces en los encaminadores intermedios, se requieren grandes *buffers* para minimizar la pérdida de células. Esta solución es adecuada en entornos WAN, pero no en LANs. En una LAN, se puede emplear el control de flujo de manera eficiente porque los enlaces son más cortos. Además, puede reducirse el espacio dedicado a *buffers* si se emplea control de flujo. Algunas propuestas recientes, sugieren el uso de control de flujo cuando ATM se implementa en una LAN [89].
- ATM proporciona un soporte mínimo para la distribución del ancho de banda de los enlaces entre varias clases de tráfico de prioridades diferentes.
- ATM no ofrece soporte para el control del *jitter*. Más aún, la mayoría de los conmutadores ATM actuales exhiben latencias muy elevadas en comparación con encaminadores *wormhole*, degradando las prestaciones de las aplicaciones que confían en una transmisión rápida de los mensajes de control.

Por otro lado, los algoritmos de planificación del conmutador y los enlaces empleados comúnmente con el objeto de ofrecer QoS a las aplicaciones que lo requieren (revisados en el Capítulo anterior), suelen ser bastante complejos como para admitir una implementación monochip de alta velocidad.

4.1. Requerimientos de las aplicaciones

En el Capítulo 1 ya se introdujeron algunos ejemplos de aplicaciones multimedia actuales. Como se pudo apreciar, el tráfico generado por las aplicaciones multimedia exhibe un comportamiento diferente del de otras aplicaciones tradicionales, como las de computación paralela, de tiempo real, servidores remotos, etc. A continuación se resumen sus características más distintivas.

Flujos de datos muy largos Las transmisiones de audio y vídeo suelen estar compuestas de flujos de datos muy largos, que duran desde unos pocos segundos hasta horas. Estos flujos de datos se transmiten mejor empleando un esquema orientado a la conexión.

Amplio rango de requisitos de ancho de banda Conexiones distintas pueden tener necesidades de ancho de banda muy diferentes. Por ejemplo, las transmisiones de vídeo requieren mucho más ancho de banda que las de audio. Además, estos requisitos pueden no ser constantes a lo largo de toda la duración de la conexión. La transmisión de audio y vídeo requiere transmitir tramas de información a intervalos regulares, provocando un tráfico a ráfagas. Los flujos de vídeo se suelen comprimir para reducir sus necesidades de ancho de banda. Como la tasa de compresión depende de los datos de entrada, los requerimientos de ancho de banda varían con el tiempo, dando lugar a tráfico VBR (*Variable Bit Rate*).

Gran número de conexiones En cada nodo de la red, pueden estar ejecutándose simultáneamente varias aplicaciones multimedia. Además, cada una de estas aplicaciones puede requerir la transmisión de varios flujos de datos. Como los requisitos de QoS pueden satisfacerse mejor si se emplea un esquema orientado a la conexión, entonces se debería ser capaz de soportar un gran número de conexiones de forma concurrente. Sin embargo, este número es mucho menor que el que deben soportar los encaminadores empleados en entornos de área amplia (WAN), orientados a las conexiones de voz. Para un entorno LAN, este número estará en el rango de unos cientos de conexiones.

Sensibilidad al *jitter* El tráfico multimedia es muy sensible a la variación de la latencia (*jitter*). Por ejemplo, las tramas de vídeo deben llegar a intervalos regulares. Si una trama llega demasiado pronto debe ser almacenada en el nodo destino, incrementando por tanto la necesidad de espacio de almacenamiento. Por el contrario, si llega demasiado tarde, no puede ser visualizada y debe descartarse. En este caso, se muestra de nuevo la trama anterior, y se reduce la calidad del vídeo.

Tolerancia a la latencia En general, las aplicaciones multimedia no son demasiado sensibles a la latencia. Algunas incluso toleran latencias elevadas, como el vídeo bajo demanda. En cambio, aplicaciones como el videoteléfono necesitan una respuesta del sistema más rápida. En cualquier caso, se admiten latencias elevadas en el establecimiento de la conexión.

Mensajes de control cortos Muchas aplicaciones multimedia requieren transmitir cortos mensajes de control además de los largos flujos de datos. Este tráfico no es sensible al *jitter*, pero las prestaciones de las aplicaciones multimedia pueden ser sensibles a la latencia de los mensajes de control. En particular, estos mensajes no deberían verse bloqueados por los flujos de datos.

Algunas aplicaciones generan tráfico CBR, pues el proceso de compresión y descompresión de la información puede reducir inaceptablemente la calidad de la señal. Por otro lado, también circulará por la red tráfico de tipo *best-effort* generado por aplicaciones tradicionales. En este caso, el tráfico *best-effort* no debe interferir en las garantías de QoS del tráfico multimedia. Sin embargo, el ancho de banda debe adjudicarse con cuidado para evitar que el tráfico *best-effort* se quede sin recursos.

4.2. Arquitectura del encaminador

Con la arquitectura de conmutador propuesta inicialmente en [50], y descrita en el resto del Capítulo, se pretenden conseguir los siguientes objetivos:

1. Soportar un gran número de conexiones multimedia, satisfaciendo sus necesidades de Calidad de Servicio (QoS)

2. Adjudicar el ancho de banda sobrante a tráfico *best-effort*, de manera que su latencia media sea mínima

3. Maximizar la utilización de los enlaces cuando la red alcanza la saturación

Para lograr esos objetivos, hay que seleccionar cuidadosamente varios parámetros de diseño de la red, tanto cualitativos como cuantitativos, que se muestran en la Tabla 4.1.

Parámetros cualitativos	**Parámetros cuantitativos**
Topología de la red	Tamaño de la red
Técnica de conmutación	Ancho de banda de los enlaces
Organización de los *buffers*	Número de puertos del encaminador
Gestión de los *buffers*	Frecuencia de reloj del encaminador
Organización del *crossbar*	Tamaño de los *buffers*
Algoritmos de planificación	Número de canales virtuales
del enlace y del conmutador	

Tabla 4.1: Parámetros a considerar en el diseño del Encaminador Multimedia.

Tanto el tamaño de la red como el patrón de interconexión de los nodos (esto es, la topología) vienen definidos por las necesidades de los usuarios. Por tanto, no son parámetros que deban abordarse en el diseño del Encaminador. En el resto del Capítulo, se analizará la selección de los parámetros cualitativos, para que sean capaces de satisfacer los requerimientos de las aplicaciones, a la vez que maximizando la utilización de los enlaces, y minimizando la complejidad del encaminador. Para cada parámetro, se expondrán las opciones existentes y cuál es la elegida para el encaminador. Se partirá de una descripción de encaminador canónico, para luego ir concretando cada uno de los aspectos de su diseño.

4.2.1. Arquitectura canónica de un encaminador

La arquitectura de un encaminador va a estar muy condicionada por la técnica de conmutación a la que tenga que dar soporte. La mayoría de los encaminadores comerciales empleados en multiprocesadores y SAN/LAN utilizan alguna variante de la técnica de conmutación *cut-through*.

Se va a tomar como punto de partida el modelo canónico de arquitectura de un encaminador *wormhole* básico [51], tal y como aparece en la Figura 4.1. Los componentes de la misma son los siguientes:

Conmutador o *Switch Fabric* Este componente es responsable de la conectividad entre los enlaces de entrada y los de salida. En encaminadores de alta velocidad se suele utilizar un *crossbar* con total conectividad, si bien los encaminadores comerciales recientes utilizan un *crossbar* multiplexado. En esta decisión también influirá el número de canales virtuales por puerto que implemente el conmutador.

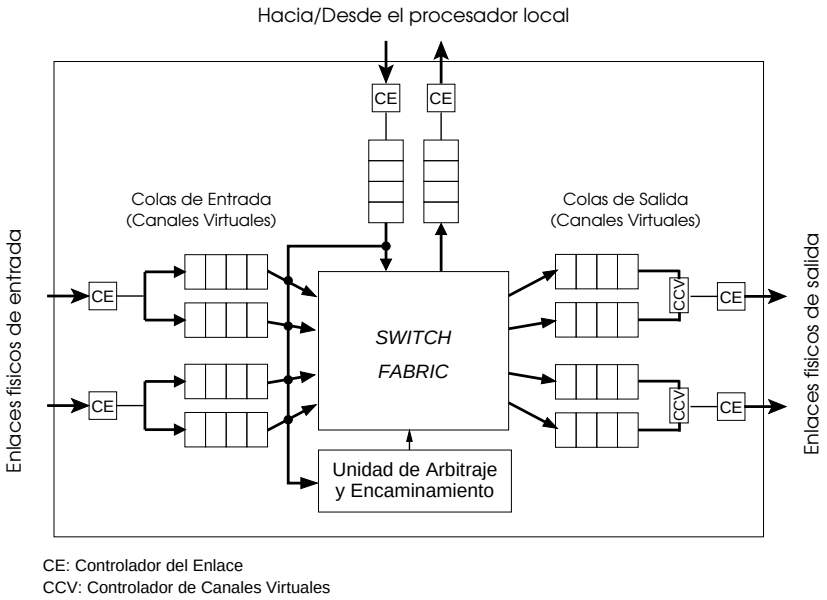


Figura 4.1: Arquitectura canónica de un encaminador.

Controlador de Enlace (CE) Esta unidad ejerce el control de flujo de los canales físicos entre encaminadores adyacentes, coordinando la transferencia de las unidades de control de flujo. Debe estar prevista la suficiente capacidad en los *buffers* para recibir y almacenar toda la información en tránsito, mientras se hacen efectivas las señales de control de flujo con los retrasos que tengan. Si hay canales virtuales, también es el responsable de decodificar el canal de destino de cada phit[1].

Controlador de Canal Virtual (CCV) Este componente es responsable de multiplexar los contenidos de los canales virtuales sobre los canales físicos.

Unidad de Arbitraje y Encaminamiento Esta unidad ejecuta la función de encaminamiento. Si realiza encaminamiento adaptativo, procesa la cabecera del mensaje para conocer el conjunto de canales de salida candidatos y genera las peticiones de estos canales. Si el encaminamiento es determinista tan sólo realiza una operación de actualización de cabecera. Esta unidad también ejecuta la función de selección del canal de salida para un mensaje de entrada. En caso de conflicto debe arbitrar a quién le es concedido el canal de salida.

Buffers Es el espacio de almacenamiento necesario para mantener los mensajes en tránsito. El tamaño del *buffer* es un número entero de unidades de control de flujo. En este modelo, cada *buffer* está asociado a canales virtuales de entrada o salida, existiendo diseños alternativos que sólo los mantiene en la entrada y/o en la salida, e incluso que incorporan *buffers* centrales, como es el caso del encaminador empleado en el multicomputador SP-2 de IBM [161].

Interfaz del Procesador En los multicomputadores se trata del interfaz del canal físico que comunica con el procesador adyacente al encaminador. En LANs/SANs no suele existir esta unidad de procesador-encaminador, con lo cual cada máquina se comunica a través de uno de los enlaces de entrada/salida disponibles en el encaminador.

[1]Definimos el *phit* como la unidad de transmisión por el enlace físico. Un *flit* o unidad de control de flujo estará compuesto por uno o más phits.

De la complejidad de cada uno de estos componentes dependerá la latencia del control de flujo, que determinará a qué velocidad pueden ser transmitidos los flits a lo largo del camino. Esta latencia es la suma del retraso del control de flujo al atravesar el canal, el tiempo de conducir el flit a través del *crossbar* y el retraso de multiplexación al atravesar el controlador de enlace, además del tiempo de lectura y escritura en los *buffers*. El principal factor en el retraso global lo va a constituir el tamaño y la estructura del *switch fabric*. Esto condicionará en gran medida el que para rendimientos parecidos se prefiera una técnica de conmutación sobre otra, si la primera impone menos requerimientos hardware en el *switch*.

4.2.2. Técnica de conmutación

La conmutación es el método empleado para asignar recursos en los encaminadores, conectando los canales de entrada y de salida. El mecanismo de conmutación empleado suele definir asimismo un método de control de flujo, esto es, un método para regular el tráfico en la red de interconexión.

Los primeros multicomputadores [146] emplearon la técnica de **conmutación de paquetes**, heredada de las redes de computadores. Con esta técnica, los recursos se asignan a los paquetes completos, de modo que hasta que no se ha recibido la totalidad del mismo en un nodo, no se retransmite por el siguiente canal. La latencia resulta ser directamente proporcional a la distancia recorrida por el mensaje.

El mecanismo empleado cuando dos o más mensajes pretenden utilizar simultáneamente un mismo canal es muy simple. A uno de los mensajes se le asigna el canal (empleando la política que se considere más apropiada para la selección del mensaje, tal y como se verá en un apartado posterior), y los otros simplemente esperan en sus *buffers*.

La introducción del mecanismo denominado ***wormhole*** (WH) [42] permitió una reducción drástica de la latencia. En efecto, con esta técnica, los mensajes se dividen en unidades más pequeñas, denominadas *flits* (*flow control digits*), sobre las cuales se realiza el control de flujo. Es decir, los recursos se asignan a los flits, en lugar de asignarlos a mensajes enteros. Por lo tanto, no es necesario esperar a recibir el mensaje completo para transmitirlo por el siguiente canal. En cuanto se ha recibido un flit, este puede transmitirse inmediatamente por el siguiente canal.

Sin embargo, de todos los flits en los que se descompone un mensaje, sólo el primero, denominado *flit de cabecera*, posee información para el encaminamiento. Por lo tanto, el resto de flits deben seguir, sin despegarse, al de cabecera para conseguir llegar a su destino. Esto se consigue haciendo que los recursos se reserven para todo el mensaje. Es decir, el flit de cabecera reserva los canales conforme los va cruzando, mientras que el último flit del mensaje o *flit de cola* los libera. Con esta estrategia, la latencia de un mensaje está compuesta de un término dependiente de la distancia recorrida y de otro término dependiente de la longitud del mensaje. Si la longitud del mensaje es grande la latencia será prácticamente independiente de la distancia recorrida.

Otra importante ventaja de este mecanismo es que no es preciso disponer de grandes *buffers* en cada nodo para almacenar los mensajes. En efecto, sólo es preciso disponer de elementos de almacenamiento para un sólo flit, permitiendo obtener un controlador de comunicaciones rápido y de reducido tamaño. En la práctica, sin embargo, se utilizan tamaños mayores en los *buffers* para facilitar el avance de los flits en ausencia de un reloj único en la red, y para aumentar las prestaciones [41].

En lo que respecta al modo de resolver los conflictos entre mensajes, en *wormhole* se utiliza una técnica consistente en bloquear todos los mensajes excepto uno, al cual se le asigna el

canal. Nótese que los mensajes bloqueados permanecen en la red, manteniendo los recursos que se les habían asignado.

Los flits tienen un tamaño generalmente igual al del ancho del canal de comunicación, aunque pueden ser mayores o menores que éste. Existe una cota inferior para el tamaño de flit, puesto que debe alojar la información para el encaminamiento.

Wormhole está inspirado en una técnica anterior denominada ***Virtual Cut-Through*** (VCT) [91] desarrollada para las redes de computadores. La diferencia básica con *wormhole* radica en la estrategia utilizada para resolver los conflictos entre los mensajes. En este caso, a uno de los mensajes se le permite continuar, pero el resto es almacenado temporalmente en el nodo, liberando los recursos asignados en la red. Los mensajes son inyectados nuevamente en la red en cuanto el canal está libre. Esta estrategia posee también la ventaja de *wormhole* en lo que respecta a la latencia de los mensajes, con el inconveniente de necesitar *buffers* mayores para almacenar los mensajes que no pueden avanzar. Sin embargo, aumenta la productividad de la red y mejora la utilización de los canales. La latencia en ausencia de conflictos tiene la misma expresión que en *wormhole*.

La **conmutación de circuitos**, técnica comúnmente utilizada en las redes telefónicas, también puede emplearse en las redes de interconexión de los multicomputadores (algunos ejemplos son el iPSC-2 o el iPSC/860 [77]). Con este mecanismo, la comunicación tiene lugar en tres fases. En la primera fase, denominada *fase de establecimiento de la conexión*, se reserva un circuito formado por varios canales físicos entre el nodo origen y el destino del mensaje. En la segunda fase, denominada *fase de transmisión*, el mensaje se transmite directamente del nodo origen al nodo destino a través del circuito establecido en la primera fase. Puesto que los canales reservados se observan como un canal único, no es necesario disponer de buffers en los nodos intermedios. La tercera fase, denominada *fase de finalización de la conexión*, consiste en eliminar el circuito conforme la cola del mensaje lo atraviesa.

La latencia de un mensaje con esta estrategia es ligeramente superior a la obtenida con el mecanismo *wormhole*, puesto que hay que propagar el éxito en el establecimiento de la ruta hasta el nodo origen antes de enviar el mensaje.

Si se encuentra un canal ocupado cuando se está estableciendo el circuito, se dice que el circuito está bloqueado. Habitualmente se cierra el circuito y se intenta establecer de nuevo más tarde. En [71] se presenta un estudio muy detallado de diferentes técnicas de conmutación de circuitos. En [60] se ha propuesto una variante de esta técnica con el objetivo de aumentar la tolerancia a fallos de la red.

Como ya se ha indicado, en la práctica se utiliza un mecanismo de control de flujo tipo *wormhole* con tamaños de *buffer* superiores a un flit. Por supuesto, esta estrategia está a medio camino entre el *wormhole* puro y el *virtual cut-through*.

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 un conflicto. Este mensaje ha reservado una serie de *buffers* y canales, y los mantiene durante todo el tiempo que está detenido. Como consecuencia, si otros mensajes desean utilizar algún canal de los reservados por el mensaje detenido, se producirán nuevos conflictos, que, a su vez, originarían más conflictos con otros mensajes. Todo ello conlleva una pobre utilización de los canales, lo que se traduce en una baja productividad de la red de interconexión.

Dally [40] ha propuesto una solución a este problema. En primer lugar, en vez de asociar un único *buffer* a cada canal físico, se asocian múltiples *buffers*, reduciendo el tamaño de cada uno. 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 detenido, los flits 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**.

La utilización del mecanismo de control de flujo basado en 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. Por lo tanto, un grado de multiplexación muy elevado puede conducir a un aumento de la latencia de los mensajes.

La disponibilidad y flexibilidad de los canales virtuales dio lugar a la aparición de varias **técnicas de conmutación híbridas**. Estas técnicas han sido motivadas por un intento de combinar las ventajas de varias aproximaciones básicas, además de obtener mejoras en aspectos distintos a la latencia y la productividad, aspectos como la tolerancia a fallos, la fiabilidad, etc.

La **conmutación de circuitos segmentada** es una aproximación que mezcla *wormhole* con conmutación de circuitos. En *wormhole*, el flit de cabecera que contiene la información de encaminamiento establece un camino a través de la red entre la fuente y el destino. Los flits de datos se transmiten de forma segmentada a lo largo del camino, inmediatamente después del flit cabecera. Si la cabecera no puede avanzar, el mensaje queda bloqueado, y los recursos ocupados por este mensaje también. Pues bien, en conmutación de circuitos segmentada, antes de comenzar a viajar los flits de datos, el flit cabecera establece el camino completo entre la fuente y el destino. Una vez que el camino está establecido, comienzan a viajar los datos.

La ventaja fundamental de que viaje solo el flit de cabecera es una mayor flexibilidad. Por ejemplo, en caso de quedar bloqueado tendría posibilidad de volver sobre sus pasos y reintentar el encaminamiento por algún camino alternativo. La diferencia básica con conmutación de circuitos es que el camino está constituido por canales virtuales y no por canales físicos. Cuando finalmente el flit de cabecera alcanza el destino, tiene que enviar un flit de reconocimiento para que los datos comiencen el viaje de manera segmentada.

Esta técnica introduce una mayor latencia al tener que establecer el camino completo antes de comenzar a transmitir datos, pero hereda las ventajas de ser una técnica orientada a la conexión, ya que una vez establecida la conexión, el camino queda libre para los datos, pudiendo asegurar a partir de ese instante la latencia de la transmisión.

Las características de ATM ya fueron comentadas al inicio del presente Capítulo (página 68), donde se expusieron sus ventajas e inconvenientes como técnica capaz de dar soporte a la QoS, pero pensada para redes de área extensa (WANs) y con graves deficiencias para entornos de tipo LAN/SAN. En este entorno, es posible el utilizar el control de flujo de manera eficiente dada la longitud de los enlaces, reduciendo el tamaño de los *buffers* e impidiendo la pérdida de información que nos genera la multiplexación estadística en ATM. Además la mayoría de los conmutadores ATM comerciales tienen latencias muy elevadas en comparación con los encaminadores desarrollados para multicomputadores, degradando las prestaciones de las aplicaciones.

Una vez realizado un recordatorio de varias técnicas de conmutación, y a modo de resumen, en la Tabla 4.2 se muestran sus características en cuanto a la Unidad de Control de Flujo (UCF) utilizada, estar o no orientada a la conexión y por tanto ser capaz o no de soportar garantía de QoS.

Técnica de conmutación	**U.C.F.**	**Orientado a la conexión**	**QoS**
Conmutación de circuitos	mensaje	sí	sí
Conmutación de paquetes	paquete	no	no
ATM	célula	sí	sí
Virtual Cut-Through	paquete	no	no
Wormhole	flit	no	no
Conm. de circuitos segmentada	flit	sí	sí

Tabla 4.2: Características de las técnicas de conmutación.

De estas características se pueden extraer algunas consecuencias. La primera es que para poder garantizar valores de latencia o *jitter* acotados en las conexiones multimedia, es necesario emplear un esquema orientado a la conexión. Las técnicas de conmutación que reservan recursos sobre la marcha, como *wormhole* o *virtual cut-through*, no son apropiadas, porque el tiempo que un mensaje puede permanecer bloqueado esperando la liberación de un recurso ocupado, no está acotado.

Por otro lado, tanto los mensajes de control como el tráfico *best-effort* se pueden beneficiar de técnicas que proporcionan baja latencia, como *wormhole* o *virtual cut-through*. En cambio, en este caso, un esquema orientado a la conexión no es apropiado, pues la alta latencia que conlleva la fase de establecimiento de la conexión puede ser un orden de magnitud superior a la latencia obtenida con conmutación *wormhole*.

Teniendo en cuenta las necesidades de cada tráfico y las posibilidades de cada técnica de conmutación se puede realizar la siguiente clasificación que puede verse en la Tabla 4.3. Se puede observar que no existe ninguna técnica que satisfaga las necesidades de todos los tipos de tráfico, es decir que sea apropiada para el tráfico multimedia y a la vez dé soporte al tráfico de control y al tráfico convencional *best-effort*.

Un buen compromiso consistiría en utilizar una técnica híbrida [47]. Los flujos de datos se transmitirían mediante alguna técnica orientada a la conexión (conmutación de circuitos,

Tipos de tráfico	**Características**	**Técnica de conmutación más apropiada**
Flujos multimedia	*Jitter* acotado	Conmutación de circuitos
	Tolerancia a la latencia inicial	ATM
	Flujos largos	Conm. de circuitos segmentada
Mensajes de control	*Jitter* no acotado	Conmutación de paquetes
Tráfico *best-effort*	Baja latencia media	*Virtual cut-through*
	Flujos cortos	*Wormhole*

Tabla 4.3: Tipos de tráfico y técnicas de conmutación más apropiadas.

ATM, PCS), de manera que primero se establece una conexión entre origen y destino, y por ella se transmiten los datos. Los mensajes de control y el tráfico *best-effort* emplearían *wormhole* o *virtual cut-through*. Además, los recursos utilizados por cada técnica de conmutación han de estar separados, o bien se han de utilizar dos técnicas de conmutación que sean capaces de compartir los recursos. Esto es, se deberían utilizar dos técnicas compatibles a nivel hardware.

Habrá que tener en cuenta las compatibilidades hardware, pero también los estudios realizados sobre el rendimiento de estas técnicas de conmutación, ideadas para entornos de multiprocesadores, en entornos COW. Por ejemplo, en entornos de multiprocesadores la técnica de conmutación *wormhole* con la implementación de un pequeño número de canales virtuales [40], se ha mostrado claramente superior a *virtual cut-through* [51]. Por este motivo lo implementa el encaminador empleado en el Cray T3E [145]. En cambio, en un entorno COW es *virtual cut-through* la que obtiene una mayor productividad, ya que las tradicionales desventajas de *virtual cut-through*, como son las necesidades de mayores *buffers* y la sobrecarga generada en la paquetización de los datos desaparecen [126, 48].

En concreto, en el diseño del Encaminador Multimedia se ha optado por el empleo de ***Pipelined Circuit Switching* (PCS)** [44], debido a varios motivos:

- Su simplicidad
- Puede combinarse fácilmente con *wormhole* o *virtual cut-through*
- Utiliza control de flujo para prevenir la pérdida de datos
- Requiere poco espacio de *buffers* asociado a cada canal virtual
- La sobrecarga introducida por la información de control es mucho menor que en ATM

Por estas razones, PCS parece más adecuado que ATM para integrar un encaminador en un solo chip, y proporcionar soporte para un entorno LAN.

En cuanto a la técnica de conmutación seleccionada para transmitir los mensajes de control, se ha preferido ***virtual cut-through*** sobre *wormhole* porque dichos mensajes de control suelen ser pequeños. *Wormhole* y *virtual cut-through* se comportan de forma idéntica en ausencia de contención. Sin embargo, cuando aparece la congestión, *virtual cut-through* ofrece mejores prestaciones porque los paquetes bloqueados se extraen de la red. Esto beneficia al tráfico multimedia porque reducirá la interferencia con los flujos de datos. El principal inconveniente respecto a *wormhole* es la necesidad de *buffers* lo bastante grandes como para poder almacenar paquetes enteros. Como los paquetes con mensajes de control serán en su mayoría pequeños, esto no supone un problema. Cuando haga falta transmitir mensajes largos de tráfico *best-effort*, dichos mensajes pueden dividirse en pequeños paquetes de tamaño fijo. Esto no introduce ninguna sobrecarga adicional, puesto que los modernos niveles software de mensajes, como AM [169] o FM [127], ya fraccionan los mensajes en trozos de tamaño fijo, con el objeto de segmentar su transmisión a través del interfaz de red, e incrementar así las prestaciones obtenidas.

Por otro lado, se ha demostrado que, cuando se trata de entornos NOW/COW, un encaminador *virtual cut-through* ofrece las mismas o incluso mejores prestaciones que otro basado en *wormhole*, con una menor complejidad de implementación [48, 49]. Por ejemplo, puesto que los enlaces son largos, *wormhole* necesita *buffers* grandes en este entorno para realizar la transmisión de forma segmentada (técnica denominada *segmentación del canal*, *channel pipelining* [154]), y aprovechar así totalmente el ancho de banda del canal [25]. Más aún, la

longitud de los enlaces está limitada por el tamaño de los *buffers* [25]. Por el contrario, con *virtual cut-through*, aun cuando los *buffers* han de tener capacidad como para almacenar uno o varios paquetes completos, su tamaño no condiciona la longitud de los enlaces. Además, el control de flujo en redes *wormhole* con segmentación del canal, y canales virtuales es relativamente complejo, debido a que la unidad de control de flujo es el flit [150]. En cambio, con *virtual cut-through* el control de flujo se simplifica, debido a que trabaja sobre paquetes completos, en lugar de sobre flits.

4.2.3. Organización de los *buffers*

Cuando se establece una conexión, se reserva un canal virtual en cada enlace perteneciente al camino entre origen y destino. Con el objeto de poder soportar un gran número de conexiones simultáneas, necesarias en un entorno de soporte a aplicaciones multimedia (ver Sección 4.1), los *buffers* de cada enlace deben organizarse como un gran conjunto de canales virtuales. Se asignará uno por conexión, con el objeto de considerar los requisitos de QoS de cada flujo de información de forma independiente del resto.

Cuando los canales físicos se multiplexan en un número elevado de canales virtuales, el retardo del encaminador se incrementa drásticamente. Esto se debe a los retardos introducidos por el multiplexor y el controlador de los canales virtuales. Además, el área de silicio dedicada a *buffers* se incrementa rápidamente con el número de canales virtuales. Más aún, los *crossbar* completamente demultiplexados llegan a ser prohibitivos al aumentar el número de canales virtuales. Por tanto, hace falta una organización distinta de los *buffers* si se desea soportar un gran número de canales virtuales.

Si se considera únicamente el espacio dedicado a *buffers*, los *buffers* centrales son la mejor opción, puesto que todos los enlaces comparten esos *buffers*. Pero esta organización tiene algunos inconvenientes:

1. Introduce cuellos de botella, y limita en gran medida el número de puertos del encaminador.
2. El control de flujo se hace más complejo.
3. Aunque sería posible combinar los *buffers* centrales con otros a la entrada, esto sólo es eficiente cuando la mayor parte de los datos puede ser transmitida de inmediato, sin necesidad de almacenamiento en los *buffers* centrales. En nuestro caso, esto no es posible, porque habrá ocasiones en que debamos retrasar la transmisión de algunos paquetes, en favor de otros datos de mayor prioridad.
4. Puesto que el espacio dedicado a *buffers* es común a todos los enlaces, puede ocurrir que la ocurrencia de ráfagas en uno de ellos agote el espacio disponible para otros enlaces, provocando pérdidas de información o retrasos elevados debido al control de flujo. En cualquier caso, se produciría una pérdida de QoS en las aplicaciones.

Por estos motivos, una organización del encaminador basada en *buffers* centrales no resulta adecuada para los propósitos perseguidos por el MMR.

Los *buffers* a la salida han sido los preferidos tradicionalmente en conmutadores para LANs y WANs, pues son normalmente más eficientes que a la entrada porque eliminan el tiempo de espera de cabeza de línea (*HOL-blocking*), y además la provisión de garantías de QoS es más sencilla en esta organización. Sin embargo, los paquetes entrantes deben ser procesados inmediatamente para decidir el *buffer* de salida donde deben almacenarse. El problema

principal que tiene esta organización de *buffers* es que existe contención entre los paquetes entrantes por los *buffers* de salida. Cuando llegan varias ráfagas de datos por distintos enlaces de entrada, y van destinadas al mismo enlace de salida, aparece la contención, y pueden llegar a descartarse algunos paquetes. Esto ocurre incluso si todos los datos son de alta prioridad. Para este problema existen varias soluciones, como se comentó en la Sección 3.3: uso de una organización de *buffers* multipuerto, muy cara y compleja, o la aceleración del *crossbar* con respecto a los enlaces de E/S, inviable en la práctica dada la elevada frecuencia de funcionamiento de los enlaces actuales. Otra opción consiste en aplicar control de flujo. Sin embargo, el control de flujo es muy complicado cuando solo se emplean *buffers* a la salida.

El enfoque tradicional utilizado en los encaminadores tipo *wormhole* utiliza *buffers* situados en los enlaces de entrada. Esta organización simplifica considerablemente el control de flujo y es capaz de conseguir latencias bajas para cargas bajas y medias. Sin embargo, presenta dos problemas graves cuando se emplea en un entorno LAN/SAN:

1. Sufre de bloqueo de cabeza de línea (*HOL-blocking*) cuando a la red se le aplica una carga media-alta, como ya se comentó en la Sección 3.3. En dicha Sección también se señalaron soluciones propuestas a este problema, como el empleo de canales virtuales independientes para cada conexión, o el uso de una organización VOQ (*Virtual Output Queuing*).

2. El retardo de ida y vuelta de la información de control de flujo implica el uso de grandes *buffers* con el objeto de prevenir pérdidas de datos cuando los enlaces son largos, si se emplea control de flujo tipo *stop-and-go* [25]. Esto se puede solucionar empleando *control de flujo basado en créditos* [89], al precio de aumentar el tráfico de control, perdiendo ancho de banda útil para el tráfico de usuarios. Esta sobrecarga se podrá amortizar mejor empleando unidades de control de flujo de tamaño elevado.

El Encaminador Multimedia MMR empleará *buffers* a la entrada, pues son los empleados originalmente en las técnicas de conmutación que se implementarán en el encaminador (PCS y *virtual cut-through*) y facilitan el control de flujo. Pero puesto que los pequeños *buffers* FIFO utilizados en multicomputadores resultan ineficientes para trabajar en entornos LAN/SAN, será necesaria una organización alternativa que posibilite el manejo de cientos de conexiones simultáneas. Además, se debe evitar el problema del *HOL-blocking*, que limita severamente la productividad máxima del encaminador [87].

Tradicionalmente, los canales virtuales se han organizado como un conjunto de colas unidas por un multiplexor. Esta organización es adecuada para un número reducido de canales virtuales, pero ya no resulta viable si se incrementa el número de canales virtuales a unos cientos por cada enlace físico.

En el Encaminador Multimedia, los canales virtuales se organizan como un conjunto de módulos de memoria RAM entrelazados según un esquema simple, al estilo del empleado en computadores vectoriales y segmentados [75, 72]. Cada módulo de memoria tiene un puerto de entrada y otro de salida. Cada flit se almacena como un conjunto de palabras consecutivas en memoria. Los flits que pertenecen al mismo canal virtual se almacenan en conjuntos de posiciones de memoria adyacentes. Antes de transmitir cada flit, se transmite una palabra de control indicando el número de canal virtual al cuál pertenece el flit. Las palabras de control se distinguen de las de datos mediante un bit de control, como en [25]. El número de canal virtual se emplea para direccionar el *buffer* correspondiente dentro de la memoria RAM. Después de almacenar cada palabra, la dirección se incrementa automáticamente. El entrelazado se lleva a cabo mediante los bits menos significativos. El número de módulos de

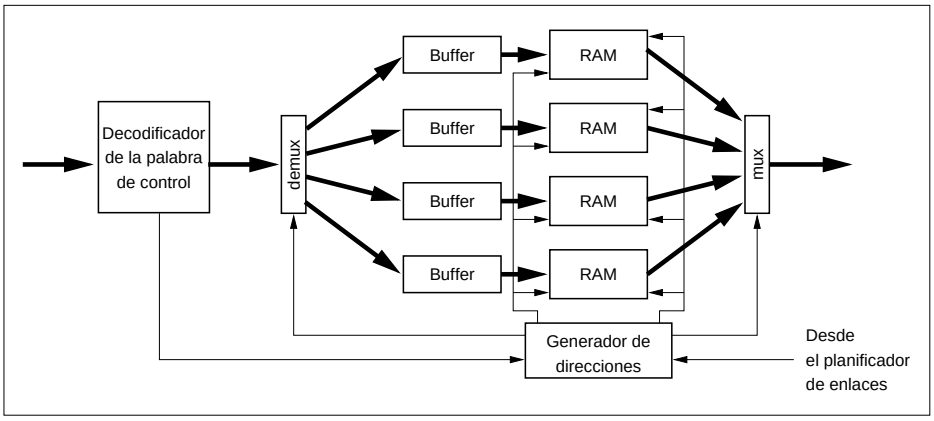


Figura 4.2: Organización de los *buffers* en el MMR.

memoria y el tamaño de los flits se seleccionan de manera que concuerden con el tiempo de acceso a memoria en ciclos de reloj.

La organización de *buffers* propuesta se muestra en la Figura 4.2. La dirección de lectura que se necesita para recuperar los flits la proporciona el planificador de enlaces, una vez que la planificación para ese ciclo ha sido calculada. Debe señalarse que una palabra puede ser leída tan pronto como sea almacenada, sin tener que esperar a la recepción del flit completo. De esta forma, se puede segmentar la transmisión al nivel de palabra. El tiempo de acceso a memoria incrementa la latencia en unos pocos ciclos de reloj, pero las aplicaciones multimedia normalmente toleran bien la latencia.

En el Encaminador Multimedia se va a evitar la pérdida de información debida a desbordamiento de los *buffers* mediante el empleo de control de flujo. El esquema escogido es el denominado **control de flujo basado en créditos**. Se ha preferido frente al control de flujo tipo *stop-and-go*, usualmente empleado en redes de multicomputadores, por varios motivos:

- El retardo de la información de control de flujo a través de los enlaces hace necesario el empleo de *buffers* de tamaño elevado por cada canal virtual, lo que, dado el elevado número de canales virtuales que se planea soportar en el MMR, elevaría de forma considerable su complejidad hardware y su tamaño en área de silicio.
- Es necesaria una mayor sobrecarga de control, como se verá más adelante. Sin embargo, puesto que los flujos multimedia son muy largos, el tamaño de la unidad de control de flujo o flit se puede hacer lo suficientemente grande como para amortizarla.
- Se puede argumentar que hasta que la información de control de flujo llega al emisor, los enlaces permanecen inutilizados perjudicando la tasa de utilización de los mismos. Sin embargo, el control de flujo se establece por cada canal virtual, y no por cada enlace físico. Por tanto, al existir un gran número de ellos, la probabilidad de que haya al menos uno preparado para transmitir (es decir, con créditos disponibles) es bastante elevada, especialmente con cargas medias-altas.

El **control de flujo basado en créditos** consiste en lo siguiente. El emisor dispone de un cierto número de créditos, que se corresponde con el número de *buffers*[2] libres en el receptor.

[2]Cada *buffer* se entiende con capacidad para una unidad de control de flujo o flit.

Este número de créditos se decrementa cada vez que se transmite un flit. El receptor envía un crédito al emisor cada vez que consigue enviar un flit.

Esta técnica puede ser menos eficiente que *stop-and-go* porque por cada flit que se transmite, se requiere el envío de un crédito en la dirección opuesta. Para reducir esta sobrecarga de control, se deben emplear tamaños de flit grandes. El tamaño de flit es un parámetro que habrá que calibrar.

Además, otra ventaja del uso de flits grandes es que posibilitará basar el diseño del encaminador en un *crossbar* multiplexado, así como utilizar algoritmos sofisticados en la planificación de los enlaces y el conmutador. Esto se verá con más detalle en las siguientes secciones.

Sin embargo, el uso de flits de gran tamaño también plantea nuevos problemas. Los flits grandes incrementan la latencia y los requerimientos de espacio de almacenamiento. La latencia puede reducirse si se segmenta la transmisión de los flits con una granularidad menor, por ejemplo, a nivel de phit[3].

4.2.4. Organización del conmutador

La mayoría de conmutadores se implementan como *crossbars*. Existen varias organizaciones posibles: multiplexados, parcialmente multiplexados y completamente demultiplexados [51].

Un ***crossbar* completamente demultiplexado** conecta entre sí todos los canales virtuales de entrada con todos los canales virtuales de salida. Esto es viable si se tienen pocos canales virtuales y pocos enlaces, pero si el número de canales o de enlaces crece, es necesario buscar alternativas. El problema se agudiza si consideramos que el ancho de palabra interno del conmutador suele ser de 8 o 16 bits. La Figura 4.1 (página. 71) representa una configuración de este tipo.

Una alternativa es un ***crossbar* multiplexado**, es decir, un *crossbar* con tantos puertos como enlaces de E/S tenga el encaminador. La tasa de datos que atraviesan el conmutador está limitada por el ancho de banda de los enlaces físicos, mientras que los canales virtuales desacoplan los recursos de *buffers* del enlace físico. De esta manera, la utilización del canal permanece elevada, y la construcción de tales *crossbars* resulta viable. Por contra, ahora es necesario el arbitraje tanto en los puertos de entrada como en los de salida.

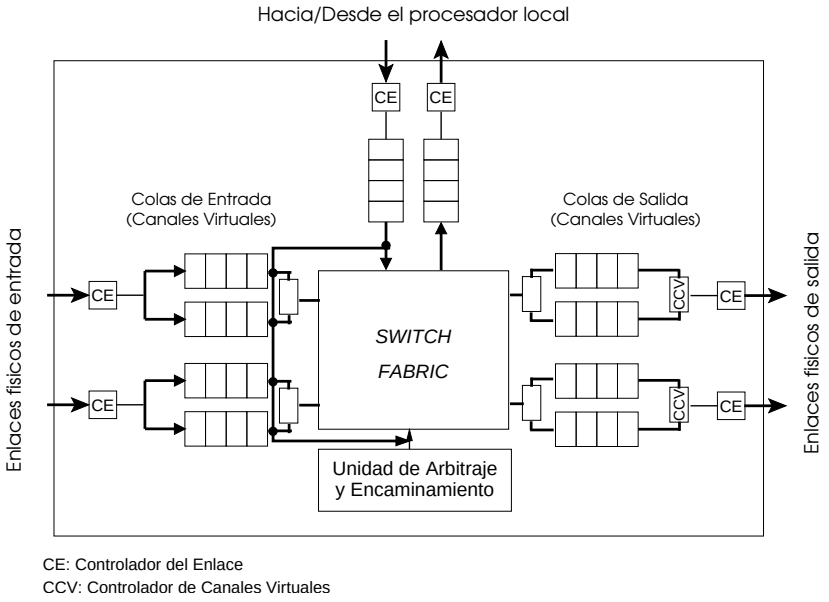
Y por último, cabe la posibilidad intermedia, el ***crossbar* parcialmente demultiplexado**, donde existe una entrada en el *crossbar* por cada canal virtual de entrada, pero sólo hay una salida por enlace físico. De esta forma, el arbitraje se limita a las salidas del conmutador. Estas dos últimas posibilidades se ilustran en la Figura 4.3.

Como se ha puesto de manifiesto, la organización basada en *crossbar* completamente demultiplexado llega a ser prohibitiva cuando tenemos un número elevado de canales virtuales. Incluso para un número relativamente pequeño de canales virtuales, se suelen emplear *crossbars* multiplexados. Por tanto, el Encaminador MMR utilizará un ***crossbar* multiplexado**.

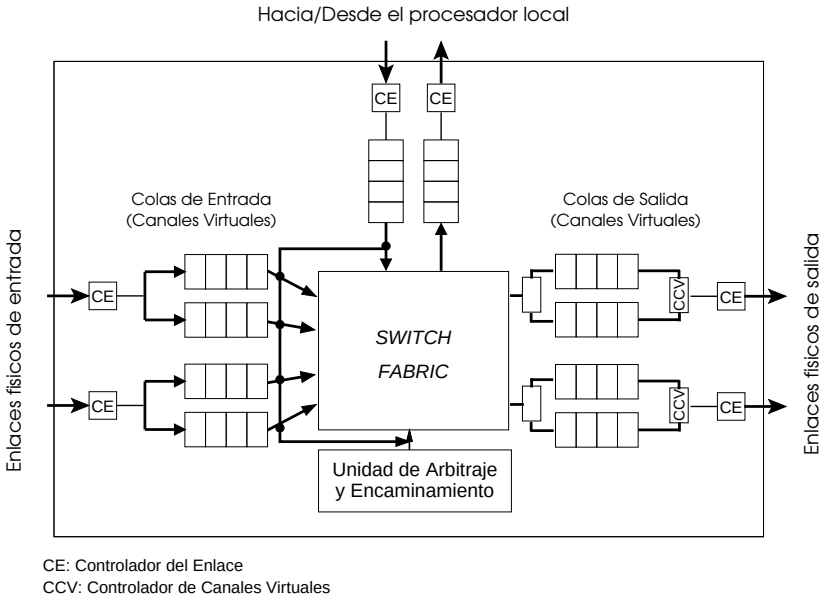
Como se ha señalado con anterioridad, el principal inconveniente de un *crossbar* multiplexado consiste en que se necesita arbitrar cada vez que un canal de entrada cambia de un canal virtual a otro. El arbitraje se requiere en el lado de la entrada para seleccionar un canal virtual de cada enlace físico de entrada. También es necesario un arbitraje en el lado de la salida porque varios canales virtuales pueden solicitar el mismo enlace de salida.

Puesto que el diseño de algoritmos de algoritmos de arbitraje del conmutador y de los enlaces apropiados para el MMR es uno de los objetivos principales de esta Tesis Doctoral, se va a

[3]Un *phit* es la cantidad de información que puede transmitirse en paralelo a través de un enlace



(a) *Crossbar* multiplexado



(b) *Crossbar* parcialmente demultiplexado

Figura 4.3: Organizaciones alternativas del *crossbar* de un conmutador.

dedicar el Capítulo siguiente a su descripción detallada. En este Capítulo se va a describir su funcionalidad a grandes rasgos, para completar la descripción general de la arquitectura de encaminador propuesta.

Debe hacerse notar que el arbitraje puede ocultarse durante la transmisión del flit anterior, si los flits son lo bastante grandes. Este es precisamente el caso del MMR. Además, de esta manera, la sobrecarga introducida por la reconfiguración del *crossbar* se amortiza sobre una mayor cantidad de datos.

Una característica interesante de los *crossbar* multiplexados es que no son necesarios los *buffers* a la salida. Como los puertos de salida del conmutador van conectados directamente a los enlaces de salida, los flits se transmiten directamente a través del conmutador y del correspondiente enlace de salida. Sin embargo, pueden ser necesarios unos *buffers* con capacidad para unos pocos phits para segmentar la transmisión de información a través del conmutador y del enlace. Por último, será necesario un serializador en el caso de que el camino de datos interno sea más ancho que los enlaces físicos externos.

Una vez definidos la situación y disposición de los *buffers*, y la configuración del *switch fabric*, queda prácticamente completa la arquitectura propuesta para el Encaminador Multimedia MMR. Un esquema de la misma se muestra en la Figura 4.4.

Los *buffers* que soportan los canales virtuales se organizan de la forma descrita la Sección 4.2.3, en módulos denominados **Memoria del Canal Virtual (MCV)**. Hay pequeños *buffers* de tamaño de un phit que van almacenando los flits que llegan. Sin pérdida de generalidad, se asume que el tamaño de phit es igual al tamaño de palabra. En caso contrario, habría un deserializador encargado de empaquetar varios phits en una sola palabra. Los phits de control se decodifican, ejecutando la acción correspondiente: incrementar un contador de créditos, seleccionar la dirección inicial de memoria para almacenar un flit que llega, etc. Los *buffers* de phits son lo bastante profundos como para ser capaces de almacenar todos los phits que puedan llegar durante la decodificación de un phit de control (esto es, durante el cálculo de la dirección de memoria donde deben de almacenarse esos phits). Los *buffers* de phit también permiten un procesamiento rápido de las sondas y reconocimientos cuando se establece una conexión.

Junto con los módulos de memoria que implementan los *buffers* se haya el **Planificador del Enlace**, que lleva a cabo el arbitraje entre los distintos canales virtuales que comparten cada enlace físico. Los planificadores de cada enlace funcionan en paralelo, obteniendo uno o más canales virtuales candidatos a transmitir un flit a través del *crossbar*. La información relativa a estos canales virtuales candidatos se pasa al **Planificador del Conmutador**, módulo encargado de calcular una planificación del *crossbar* libre de conflictos. Estos módulos, junto con las alternativas para su diseño e implementación, serán objeto de un estudio más detallado en el Capítulo siguiente. En los siguientes puntos se detallará el funcionamiento del resto de módulos.

4.2.5. Transmisión de paquetes y flits

El arbitraje en un conmutador con *buffers* a la entrada y *crossbar* multiplexado es difícil de realizar si las peticiones llegan de forma asíncrona, sobre todo si se pretende ofrecer garantías de QoS a los flujos de información. En ese caso, las peticiones deben almacenarse y sincronizarse de manera que se dé preferencia a las peticiones de mayor prioridad, aunque lleguen más tarde. Además, cuando un enlace de entrada dado gana el arbitraje para transmitir el siguiente flit, tiene que esperar hasta que el enlace de salida correspondiente queda libre,

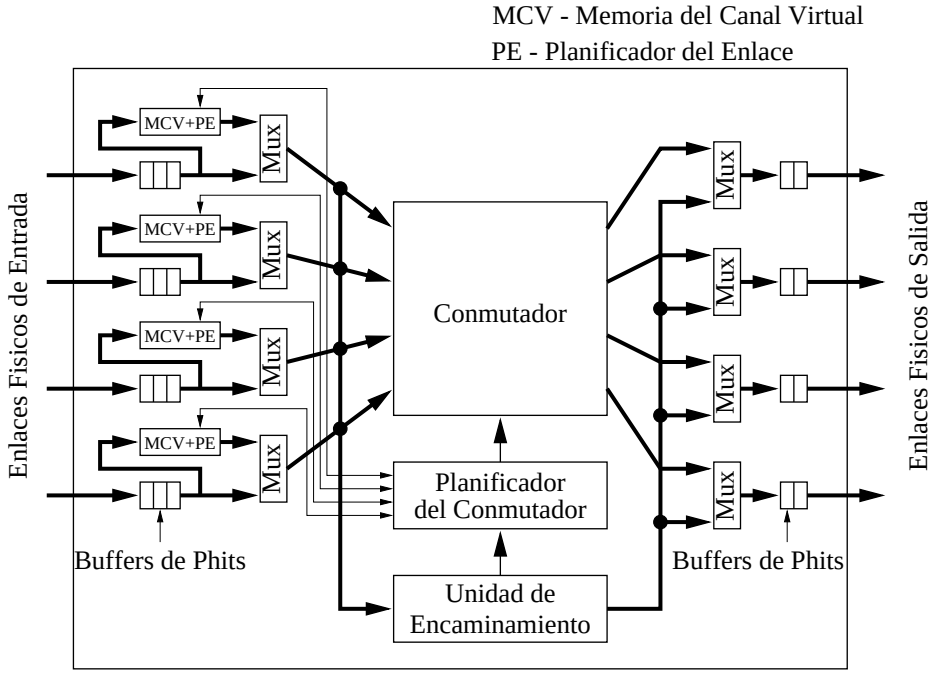


Figura 4.4: Arquitectura del Encaminador Multimedia MMR.

incluso si ya ha acabado de transmitir el flit anterior. En consecuencia, se desperdicia una parte del ancho de banda.

Por consiguiente, con el objeto de aprovechar completamente el ancho de banda del conmutador y de los enlaces, a la vez que simplificar el diseño del encaminador, el MMR asignará síncronamente los puertos del conmutador y los canales de salida a los canales virtuales con peticiones pendientes.

La transmisión de flits se organiza como una secuencia de ciclos de flit. Durante cada uno de éstos, todos los enlaces de entrada con flits listos comienzan transmitiendo una palabra de control con el identificador del canal virtual al que pertenece el flit. Después, transmiten síncronamente un flit a través del *crossbar* y los enlaces de salida. De manera simultánea se lleva a cabo el arbitraje para asignar los puertos del conmutador y los enlaces de salida para el siguiente ciclo de flit.

Una vez que ha finalizado la transmisión del flit en curso, se reconfigura el *crossbar*. Esta operación requiere un sólo ciclo de reloj. Durante la reconfiguración, no se transmiten flits de datos. El MMR aprovecha este ciclo para las transmisiones de sondas de encaminamiento, sondas en retroceso (*backtracking*), en el caso de que el retroceso esté permitido, y reconocimientos para el establecimiento de conexiones que puedan haber pendientes.

Una vez que el *crossbar* se ha reconfigurado, comienza la transmisión del siguiente flit, es decir, empieza el siguiente ciclo de flit. Hay que resaltar que, aunque dentro de cada encaminador la transmisión es síncrona, diferentes encaminadores trabajan asíncronamente entre sí.

Los mensajes de control y de tráfico *best-effort* se transmiten empleando conmutación *virtual cut-through*, es decir, se evita el establecimiento de conexiones que multiplican la latencia de estos mensajes de manera importante. Cuando un mensaje es mayor que un flit, se divide en paquetes de tamaño fijo, tal y como se ha explicado anteriormente. La unidad de control de flujo en *virtual cut-through* es un paquete. Dado el gran tamaño de los flits que se

emplean en la transmisión de flujos multimedia, se asume que el tamaño de paquete es igual a un flit. Por tanto, las unidades de control de flujo tienen el mismo tamaño en PCS y en *virtual cut-through*, con lo que el diseño del encaminador se simplifica. Las cabeceras de los paquetes se encaminan tan pronto como llegan a un encaminador, a través de la unidad de encaminamiento.

El uso de canales virtuales por parte de las transmisiones no multimedia se hace de la siguiente manera. Por un lado, para transmitir los mensajes de control se empleará un canal virtual específico, de un modo similar al que se emplea en el encaminador MediaWorm [173]. Y puesto que han de tener prioridad sobre los demás tipos de tráfico, se asigna siempre la máxima prioridad a los flits que circulan por este canal a la hora de realizar el arbitraje. En el MMR, se ha dedicado el canal virtual 0 a este uso. Puesto que este canal virtual va a ser empleado por todos los mensajes de control que circulen por el mismo enlace físico, independientemente del puerto de salida, se puede pensar que el *HOL-blocking* puede afectar sus prestaciones. Sin embargo, puesto que la utilización de este canal va a ser baja, debido a la baja frecuencia con que se generan los mensajes de control, y debido a que los mensajes se van a transmitir con la máxima prioridad, es poco probable que esto suceda. De todas formas, esto ha de verificarse a la hora de evaluar las prestaciones del encaminador.

Por otro lado, para la transmisión de mensajes *best-effort* y con el objeto de garantizarles una buena productividad, se van a dedicar tantos canales virtuales como puertos de salida tenga el encaminador. Así, los mensajes dirigidos a puertos de salida distintos, emplearán canales virtuales diferentes. Esto es, se implementa una organización *Virtual Output Queuing* mediante canales virtuales para la transmisión de los mensajes *best-effort*. Además, y puesto que este tráfico no debe afectar a las prestaciones de los demás tipos de tráfico soportados, a los flits *best-effort* se les asignará siempre un valor de prioridad mínimo, con el objeto de que únicamente utilicen la capacidad de transmisión no aprovechada por los flujos multimedia y los mensajes de control. La idea es que cuando no haya tráfico multimedia ni de control para transmitir, la capacidad del enlace se aproveche transmitiendo tráfico *best-effort*.

4.2.6. Unidad de encaminamiento

La Unidad de encaminamiento, como su propio nombre indica, se encarga de ejecutar el **algoritmo de encaminamiento**. La misión de este algoritmo es determinar el camino seguido por las sondas cuando se establece una conexión, y por los paquetes *best-effort* y de control en su avance a través de la red.

Un algoritmo completamente adaptativo puede ayudar considerablemente a encontrar un camino libre cuando la red se congestiona o cuando algunas regiones están muy cargadas. Sin embargo, el encaminamiento completamente adaptativo puede producir bloqueos. Este problema es difícil de resolver, especialmente cuando la topología de la red es irregular, y no tiene más restricciones que la de ser conexa. Sin embargo, ya existe una propuesta de un algoritmo completamente adaptativo para redes *wormhole* con topología irregular [148, 151]. Este algoritmo necesita dos canales virtuales por enlace, y también es válido para conmutación *virtual cut-through* y PCS. Por tanto, lo emplearemos para encaminar paquetes mediante conmutación *virtual cut-through*.

En cambio, para PCS podemos incrementar la flexibilidad de encaminamiento si empleamos algoritmos con *backtracking*. Estos algoritmos evitan los bloqueos liberando los recursos que fueron previamente reservados cuando la cabecera se bloquea debido a que no hay recursos disponibles. Al establecer las conexiones, emplearemos ***Exhaustive Profitable Backtracking* (EPB)** [61]. Este algoritmo lleva a cabo una búsqueda exhaustiva sobre los

caminos mínimos de la red hasta que encuentra un camino válido, o hasta que la sonda retrocede hasta el nodo fuente. Para evitar explorar dos veces los mismos caminos, cada canal virtual tiene asociado un registro histórico con los enlaces de salida que ya se han explorado.

El proceso de establecimiento de conexiones de PCS se verá ligeramente modificado en el Encaminador MMR. En PCS una conexión se establece encaminando una sonda desde el origen al destino. Esta sonda contiene información de control así como la dirección destino [62]. En el MMR, además, la sonda lleva información acerca de los requisitos de ancho de banda. Cada vez que la sonda se encamina con éxito, se reservan algunos recursos en el encaminador correspondiente. En particular, se reservan un canal virtual y sus *buffers* asociados (al igual que en PCS), y además se establece una reserva de ancho de banda para la conexión. Esto último se verá con más detalle en el siguiente apartado. Una vez que la sonda alcanza el nodo destino, se devuelve un reconocimiento a la fuente, siguiendo el mismo camino que la sonda, pero en sentido contrario. Cuando la fuente recibe este reconocimiento, la conexión está establecida. Si la sonda no puede reservar ningún camino, puede llegar a retroceder hasta el nodo fuente.

La unidad encaminamiento mantiene las **correspondencias de canales** (*channel mappings*) entre los canales virtuales de entrada y salida para las conexiones establecidas. Los canales virtuales se especifican mediante el canal físico al que pertenecen, y el número de canal virtual dentro del mismo. Es necesario mantener correspondencias directas e inversas entre canales. Las directas se necesitan para hacer avanzar los flits de datos, mientras que las inversas son utilizadas por las sondas en retroceso y por los reconocimientos. Estas correspondencias también se emplean para propagar información de estado (por ejemplo, créditos, modificaciones dinámicas en la reserva de ancho de banda, etc.).

4.3. Algoritmos de planificación de recursos

La red debe de proporcionar mecanismos para garantizar los requerimientos de QoS de distintas aplicaciones. El encaminador MMR soporta una abstracción de una conexión virtual llevada a cabo mediante canales virtuales. Se emplea un protocolo flexible de reserva de recursos, junto con cierto soporte para modificar las características de conexiones ya existentes, mediante palabras de control.

Los distintos tipos de QoS se garantizan mediante dos tipos de mecanismos:

1. El control de admisión de conexiones.
2. Mecanismos de vigilancia durante la transmisión de los datos, situados en los encaminadores, en los interfaces de red o en ambos.

El soporte para ofrecer garantías de QoS dentro del encaminador MMR se implementa como las soluciones a tres problemas básicos:

1. Reserva del ancho de banda
2. Planificación de los enlaces
3. Planificación del conmutador

El **esquema de reserva de ancho de banda** adjudica una porción adecuada del mismo para cada conexión al tiempo que ésta se establece, mientras que el algoritmo de planificación del enlace garantiza que el ancho de banda reservado, y solo el reservado, está disponible durante la transmisión de los datos.

La **planificación de los enlaces** tiene como objeto atender las necesidades tanto del tráfico multimedia como del *best-effort*. En concreto, sus objetivos son:

1. Satisfacer los requisitos de QoS del tráfico multimedia
2. Adjudicar el ancho de banda sobrante a los paquetes *best-effort*
3. Maximizar la utilización del enlace

En cuanto a la **planificación del conmutador**, su finalidad es tratar de minimizar los conflictos internos por causa de los puertos del encaminador, entre las conexiones planificadas. Un algoritmo de planificación del conmutador pobre ofrece una baja utilización del ancho de banda a través del encaminador, y por tanto, provoca una baja utilización de los enlaces.

Es decir, los algoritmos de planificación de enlaces y conmutador actúan de manera conjunta para satisfacer las garantías de QoS en función de la reserva hecha por parte de las conexiones multimedia. Son quienes se encargan de repartir efectivamente el ancho de banda disponible entre las distintas conexiones, en función de su reserva.

Las estrategias de planificación del conmutador y de los enlaces deben actuar de una manera estrechamente acoplada para utilizar el ancho de banda de la manera más efectiva posible. El mayor desafío en un encaminador MMR de un solo chip es que estas estrategias deben tener implementaciones rápidas y compactas, aunque el hecho de emplear flits grandes y simultanear el cálculo de la planificación con la transmisión de flits permite tener un margen de trabajo relativamente amplio.

En los siguientes apartados se va a tratar el problema de la reserva de ancho de banda por parte de los flujos de datos. Para ello, es necesario concretar de qué modo éstos especifican sus necesidades de acuerdo con sus características. El problema de la planificación de enlaces y conmutador se tratará con más detalle en el Capítulo siguiente, donde se describirán diferentes alternativas para su diseño.

4.3.1. Características de los flujos de datos

A grandes rasgos, y tomando como referencia la clasificación hecha por el ATM Forum [8], los flujos de datos multimedia se suelen clasificar como de **caudal constante** (CBR, *constant bit rate*) o **caudal variable** (VBR, *variable bit rate*). Las conexiones CBR son relativamente fáciles de manejar. Los requisitos de ancho de banda pueden ser satisfechos si se reserva el ancho de banda necesario al establecer la conexión.

El caso de las conexiones VBR es más complejo de manejar. Cuando se establece una conexión VBR, una aplicación puede tener un conocimiento de los requerimientos de ancho de banda medios y máximos exacto o sólo aproximado. Por ejemplo, es posible calcular el ancho de banda medio y máximo que necesita una película comprimida que está almacenada, pero no lo es si se trata de una transmisión en tiempo real de vídeo comprimido. En ese caso, únicamente se puede estimar.

Los flujos de datos que necesitan QoS pueden experimentar un *jitter* excesivo durante su transmisión. Los fragmentos que llegan a su destino demasiado tarde, normalmente son

descartados. Pero en algunas aplicaciones, no toda la información es igual de importante. Por ejemplo, en el caso del vídeo MPEG [12], las tramas de tipo I son más importantes que las de tipos B o P, porque estas últimas necesitan información codificada en las primeras. En este caso, es posible establecer prioridades distintas para tramas distintas. Otra posibilidad consiste en asignar prioridades distintas a diferentes trozos de información dentro de cada trama [37]. Las prioridades pueden emplearse a la hora de reservar ancho de banda o de planificar los enlaces.

Por último, hay conexiones "fiables" y "no fiables". Las **conexiones fiables** son aquellas que no intentan utilizar más ancho de banda del que han reservado. En otro caso, se trata de una **conexión no fiable**. Entonces, se necesitan comprobaciones sobre la marcha (**mecanismos de vigilancia**, o **Control de Parámetros de Uso** en la nomenclatura de ATM) para garantizar que una conexión no supera su parte de ancho de banda de la red, de manera que la red pueda seguir ofreciendo sus niveles de QoS al resto de conexiones.

4.3.2. Detalles de implementación

El ancho de banda de los enlaces y de los puertos del conmutador se divide en **ciclos de flit**. Estos ciclos se agrupan en **"vueltas"** o ***frames***. El número de ciclos de flit que hay en una vuelta es un múltiplo entero K ($K > 1$) del número de canales virtuales que tiene cada enlace físico.

El ancho de banda de una conexión se reserva como un número entero de ciclos de flit. Estos ciclos de flit serán asignados a la conexión durante cada vuelta, según los vaya pidiendo. Por tanto, un mayor valor de K proporciona una mayor granularidad para reservar ancho de banda. Sin embargo, el *jitter* podría llegar a incrementarse, porque las vueltas tardan más en acabar. Por tanto, el valor que se selecciona para K es un compromiso entre flexibilidad y *jitter*.

La estructura de datos empleada para soportar decisiones rápidas de planificación consiste en un conjunto de **vectores de bits de estado**, donde cada bit está asociado con un solo canal virtual. Estos vectores de bit proporcionan información sobre diferentes condiciones para todos los canales virtuales del encaminador. Por ejemplo, se pueden considerar los vectores de estado que se muestran en la Tabla 4.4.

Conexión CBR	Flits disponibles
Ancho de banda CBR servido	*Buffer* de entrada lleno
Conexión VBR	Créditos disponibles
Ancho de banda VBR servido	...

Tabla 4.4: Posibles vectores de bits de estado en el MMR.

Los bits de estos vectores se van actualizando cuando cambia el estado del canal virtual correspondiente. Por ejemplo, si en un canal virtual dado no hay créditos, y llega uno procedente del siguiente encaminador, el bit correspondiente del vector "créditos disponibles" se actualiza.

Los bits de estado están asociados a los canales virtuales a la entrada. Puede que hagan falta las correspondencias entre canales para poder actualizar los bits de estado. Por ejemplo, cuando llega un crédito hay que buscar el canal virtual de entrada que corresponde con el canal virtual de salida por el que ha llegado el flit.

Estos vectores de bits de estado pueden simplificar de forma considerable la ejecución concurrente del algoritmo de planificación de los enlaces a través de los puertos. Por ejemplo, en cada enlace físico de entrada se puede determinar rápidamente el conjunto de canales virtuales de entrada que hay en ese enlace con flits y créditos disponibles, servicio CBR pedido y no servido del todo en la vuelta actual, simplemente calculando el "Y" lógico de los vectores de estado correspondientes. De manera análoga, se puede calcular rápidamente el conjunto de canales que satisfacen otras condiciones.

4.3.3. Reserva del ancho de banda

Tal y como se ha apuntado previamente, cuando se está estableciendo una conexión en el encaminador MMR, el nodo fuente genera una sonda de encaminamiento que lleva información sobre los requerimientos de ancho de banda. La sonda intenta establecer una conexión construyendo un camino desde la fuente hacia el destino, reservando a su paso el ancho de banda en los enlaces, y los *buffers* correspondientes.

Si la reserva de recursos se lleva a cabo con éxito, la conexión queda establecida, y se admite la petición. Si los recursos necesarios no están disponibles en algún punto del camino, la conexión falla, y se liberan todos los recursos reservados hasta ese momento. Si se utiliza una búsqueda con vuelta atrás (*backtrack*), se pueden seguir otros caminos a través de la red.

Para **conexiones CBR**, este mecanismo es sencillo de implementar. La sonda debe transportar información sobre el ancho de banda que necesita la conexión c, medido en ciclos de flit por vuelta: lo llamaremos BW_m^c. Cada enlace de salida L necesita tener asociado un registro $resBW_L$ que lleve la cuenta del número total de ciclos de flit por vuelta que han sido reservados hasta el momento. Este registro se incrementa en BW_m^c cuando se admite una conexión, y se decrementa en esa misma cantidad cuando dicha conexión se libera.

Una conexión CBR c sólo puede ser admitida si se cumple la siguiente condición:

$$resBW_L + BW_m^c < numero\ de\ ciclos\ de\ flit\ en\ una\ vuelta \tag{4.1}$$

Para las **conexiones VBR**, el problema es más difícil de resolver. Algunas conexiones VBR envían un flujo de datos continuo, pero la cantidad de datos por unidad de tiempo varía con el tiempo. Este podría ser el caso de flujos de audio comprimido. Otras conexiones VBR envían ráfagas de datos de tamaño variable, pero no envían datos entre ráfagas. Este sería el caso de las tramas de vídeo comprimido. Para poder tratar con los diferentes requisitos de las conexiones, una sonda que establezca una conexión VBR v llevará los anchos de banda permanente BW_m^v y pico BW_p^v para esa conexión. En el caso de que se trate de un flujo continuo de datos, el ancho de banda permanente puede hacerse igual al ancho de banda medio que requiere esa conexión. En el caso del tráfico a ráfagas, el ancho de banda permanente puede establecer también como el ancho de banda medio, o bien ponerse a cero. El significado del ancho de banda pico es evidente en ambos casos. Estos valores pueden ser estimados, según el conocimiento de que se disponga del comportamiento de la conexión.

Para soportar la reserva de ancho de banda para conexiones VBR, cada enlace de salida necesita dos registros asociados: $resBW_L$ mantiene el número de ciclos de flit por vuelta que han sido reservados (es el mismo registro que en el caso de conexiones CBR), y $resBWp_L$ almacena el ancho de banda pico total requerido por todas las conexiones que utilizan el enlace L. Estos contadores se incrementan y decrementan respectivamente con los valores de BW_m^v y BW_p^v, cuando las conexiones se establecen y se eliminan.

Una conexión VBR v solo se aceptará si se cumplen dos condiciones:

$$resBW_L + BW_m^v < numero\ de\ ciclos\ de\ flit\ en\ una\ vuelta \quad (4.2)$$

$$resBWp_L + BW_p^v < CF \times numero\ de\ ciclos\ de\ flit\ en\ una\ vuelta \quad (4.3)$$

Es decir, en cada enlace de salida L, el registro $resBW_L$ lleva la cuenta del ancho de banda reservado para conexiones CBR y para el ancho de banda medio o permanente de las conexiones VBR. El segundo registro asociado a cada enlace de salida L, $resBWp_L$, almacena la cantidad acumulada de ancho de banda consumido por los picos de las conexiones VBR admitidas.

El parámetro CF se denomina **factor de concurrencia**. Este parámetro se almacena en un registro separado, y su valor se establece durante la inicialización del encaminador. Hay que hacer notar que el mecanismo de reserva propuesto no garantiza que la conexión obtendrá el ancho de banda pico en caso de necesidad. Si se proporcionase tal garantía, se podría desperdiciar una gran parte del ancho de banda. Sin embargo, ese ancho de banda será asignado a esa conexión con una cierta probabilidad. Esa probabilidad depende del factor de concurrencia CF. Un factor de concurrencia mayor significa que el ancho de banda del enlace será compartido entre más conexiones VBR, decrementando así la probabilidad de cumplir con las garantías de QoS. El factor de concurrencia es un compromiso entre las garantías de QoS y el número de conexiones que pueden ser atendidas concurrentemente para optimizar la utilización de los enlaces.

El protocolo de control de admisión en el MMR es similar al empleado en redes ATM. La diferencia principal consiste en que en el MMR, el control de flujo evita que los flits sean descartados. Además, los *buffers* de flit son relativamente pequeños, provocando una rápida propagación de la información de control de flujo. Finalmente, el control de flujo puede propagarse hasta el interfaz de red del nodo fuente, limitando la inyección de nuevos flits. La implementación de políticas de vigilancia dentro del MMR solamente es necesaria si no hay control en el interfaz, y se permiten conexiones no fiables.

Por último, señalar que empleando palabras de control a lo largo de una conexión, se pueden variar dinámicamente los requisitos de ancho de banda de la misma. Esto puede iniciarse en el interfaz fuente de una conexión, en respuesta a eventos externos o al rendimiento actual experimentado por la conexión. La respuesta puede implicar un cambio en la tasa de transmisión de datos, el descarte selectivo de paquetes de datos, o la limitación de la inyección. Empleando comandos de control similares a los de Myrinet [25], se puede encapsular la gestión dinámica del ancho de banda dentro del mecanismo de control de flujo.

La idea es que las funciones complejas de control del ancho de banda se implementen en los interfaces de red o en la CPU del nodo fuente, donde se puede obtener más flexibilidad, para que los encaminadores se mantengan compactos y rápidos.

Capítulo 5

Algoritmos de Planificación en el MMR

En el Capítulo anterior, se ha introducido brevemente la misión de los algoritmos de planificación dentro de la concepción global del Encaminador Multimedia MMR. Su misión fundamental consiste en adjudicar a cada conexión establecida a través del MMR el ancho de banda reservado, cumpliendo así con sus requisitos de Calidad de Servicio (QoS).

Como también se ha señalado, el MMR basa su arquitectura en un *crossbar* multiplexado con *buffers* situados a la entrada. Ello es debido a que el gran número de canales virtuales soportados hace inviable el uso de un *crossbar* demultiplexado.

El hecho de que el núcleo que posibilita el avance de los paquetes, el *crossbar*, disponga únicamente de un puerto por enlace físico, hace necesario un arbitraje tanto entre canales virtuales que comparten enlace físico, como entre canales virtuales de diferentes enlaces. En el primer caso, se trata de decidir cuál de esos canales va a ocupar un puerto de entrada al *crossbar*, mientras que en el segundo se trata de resolver conflictos por el uso de los puertos de salida del mismo. Esta idea es similar a la propuesta por Stiliadis y Varma en [158, 157] (ver Sección 3.4). Al primer arbitraje se le denomina **planificación del enlace** mientras que el segundo es la **planificación del conmutador**.

En el caso del MMR, se van a tener en cuenta las necesidades de QoS de cada flujo de datos a la hora de diseñar los algoritmos de arbitraje. Los algoritmos de planificación del enlace tienen como misión seleccionar uno o varios canales virtuales por cada enlace físico para pasarlos al planificador del conmutador. La selección se llevará a cabo considerando las necesidades de QoS de cada conexión. La misión del planificador del conmutador es calcular un emparejamiento libre de conflictos entre los puertos de entrada y de salida del *crossbar*, partiendo del conjunto de canales virtuales "candidatos" facilitado por los planificadores de enlace. Este algoritmo debe maximizar la productividad en términos de ser capaz de trasegar la mayor cantidad de información posible, pero teniendo en cuenta que las garantías de QoS de las distintas conexiones han de ser respetadas.

Los siguientes apartados tratarán con más detalle los algoritmos implicados en la planificación del tráfico en el MMR, exponiendo diferentes alternativas para su diseño.

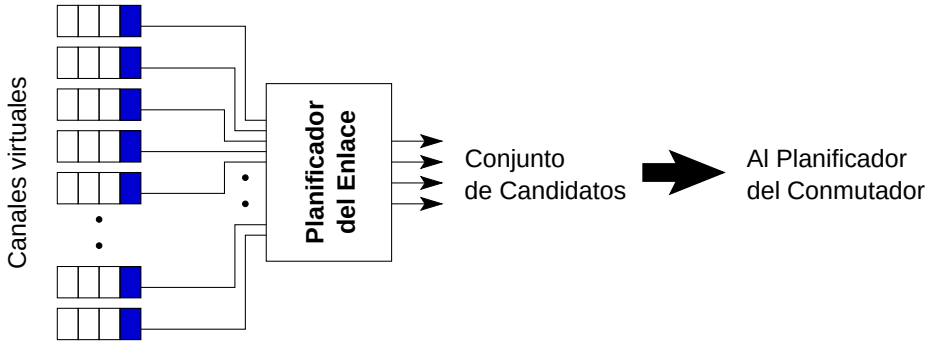


Figura 5.1: Representación conceptual del funcionamiento del Planificador del Enlace.

5.1. Algoritmos de planificación de enlaces

En el Encaminador Multimedia MMR, se implementan un número relativamente elevado de canales virtuales para proporcionar recursos independientes a los distintos flujos multimedia que atraviesan el Encaminador. Puesto que el *fabric* de interconexión está multiplexado, en cada ciclo de flit solamente se puede transmitir un flit por enlace físico. De ahí que sea necesario adoptar algún criterio para decidir a qué canal virtual se le va a adjudicar el uso del puerto del *crossbar*.

El objetivo del **algoritmo de planificación de enlace** es precisamente implementar dicho criterio en la forma del arbitraje entre los distintos flujos de datos que comparten un enlace físico. El arbitraje debe realizarse teniendo en cuenta las necesidades de QoS de los flujos multimedia, con el objeto de maximizar la utilización del enlace a la vez que respetando las garantías de QoS del tráfico multimedia. Además, se debe procurar adjudicar el ancho de banda sobrante al tráfico convencional *best-effort* que pueda coexistir con los flujos multimedia.

La planificación de los enlaces debe realizarse como un paso previo a la planificación del conmutador. Hay que recordar que la misión de éste último es encontrar un emparejamiento libre de conflictos entre los puertos de entrada y de salida del *crossbar*, que maximice la productividad a la vez que garantiza la QoS de los flujos multimedia. Debido al elevado número de canales virtuales soportados en el MMR, sería inviable que el planificador del conmutador tuviese en cuenta a todos y cada uno de los canales virtuales con flits listos para transmitir. Por tanto, el planificador de enlaces seleccionará por cada enlace físico, un cierto número (pequeño) de canales virtuales "candidatos" sobre los que trabajará el algoritmo de planificación del conmutador (ver Figura 5.1). Esta selección se realizará según un criterio de **prioridad dinámica** (*biased priority*). A los flits en cabeza de cada canal virtual se les asocia un valor de prioridad que se irá actualizando con el tiempo, de la manera que se describirá posteriormente. Para cada ciclo de flit, se seleccionará el canal virtual cuyo flit de cabeza tenga el valor de prioridad más elevado.

Para maximizar las posibilidades de encontrar un emparejamiento que maximice la utilización de los enlaces, se amplía la selección a un conjunto de C canales virtuales candidatos, aquellos cuyos flits de cabeza tengan los mayores valores de prioridad. El valor de C se mantiene pequeño, entre 1 y el número de puertos del *crossbar*. De esta manera, el planificador del conmutador dispondrá de más opciones para calcular el emparejamiento de forma que se maximice la utilización media del *crossbar*, pero sin llegar a la complejidad excesiva de considerar todos los canales virtuales activos en un momento dado.

En párrafos anteriores se ha apuntado que las prioridades que se asocian a los flits de cabeza de cada canal virtual van a variar con el tiempo, en una manera similar a otros algoritmos propuestos en la literatura [58, 32, 92]. La idea es calcular la prioridad como una función de la QoS requerida por la conexión y de la QoS que realmente está recibiendo. De esta manera, se relaciona la QoS que la conexión solicitó en su momento, con una estimación del tipo de servicio que realmente está recibiendo. Así, la prioridad está relacionada directamente con las necesidades de servicio de las aplicaciones.

La operación de actualización acopla el efecto del planificador (medido por ejemplo, como el retardo en cola), con la demanda de QoS (por ejemplo, ancho de banda requerido). Esto distingue esta aproximación de otros mecanismos como contadores de antigüedad, que tratan a todas las conexiones por igual, sin tener en cuenta sus requisitos de QoS. Mediante los mecanismos ideados siguiendo este razonamiento inicial, se consigue que la prioridad de las conexiones evolucione más rápidamente cuanto mayores sean sus requerimientos de QoS, esto es, cuanto más exigentes sean.

Como se indicó en el Capítulo 4, el Encaminador Multimedia está diseñado para soportar otros tipos de tráfico además de los flujos multimedia. Estos son el tráfico *best-effort* convencional, y el tráfico de control. Es necesario por tanto integrar de alguna manera a estos tipos de tráfico en el sistema de prioridades diseñado para planificar los enlaces del MMR. Para ello, hay que analizar que prestaciones se espera ofrecer a cada tipo de tráfico.

En primer lugar, el tráfico *best-effort* no requiere de ninguna garantía de QoS. Únicamente se persigue alta productividad y baja latencia medias, pero no debe interferir en ningún momento con las garantías de QoS que se ofrecen a las conexiones multimedia. La idea es que este tipo de tráfico ocupe el ancho de banda de los enlaces que no llegan a ocupar los flujos multimedia. Por tanto, en el sistema de prioridades, a los flits de tipo *best-effort* se les va a asignar un valor de prioridad mínimo, que además no crezca con el tiempo. De esta forma, los flits *best-effort* serán elegidos como candidatos en un enlace cuando no haya flits pertenecientes a conexiones multimedia listos para ser transmitidos en dicho enlace.

El caso de los mensajes de control es opuesto. Estos mensajes son muy cortos, y suelen aparecer con baja frecuencia, por lo que apenas ocupan ancho de banda, pero deben de ser retransmitidos lo antes posible hacia su destino, pues pueden variar el comportamiento de los flujos multimedia, o bien, modificar el uso de los recursos de la red. Por ello, se les asigna un valor de prioridad máximo, asegurando así que siempre serán elegidos como candidatos de primer nivel en cada enlace físico que atraviesen. Esto maximiza las posibilidades de atravesar el *crossbar* con un retardo mínimo.

Una vez establecido el marco general de la planificación del enlace, se van a presentar algunos algoritmos de cálculo de la prioridad basados en estas ideas, junto con algunos detalles de implementación que han de ser tenidos en cuenta.

5.1.1. Prioridad basada en ancho de banda

En una conexión tipo CBR, los requisitos de QoS están directamente relacionados con la reserva de ancho de banda. Por tanto, se puede calcular la prioridad de un flit como el cociente entre el retardo experimentado en cola por el mismo, y el tiempo entre llegadas de flits consecutivos establecido para la conexión (IAT o *Inter Arrival Time*). Esta es precisamente la idea del algoritmo IABP (*Inter Arrival Biased Priority*) [104]. Matemáticamente, se puede expresar así:

$$prioridad = \frac{retardo\ en\ cola}{IAT} \tag{5.1}$$

El algoritmo funciona de la siguiente forma. Cuando un flit tarda en ser transmitido, su retardo en cola crece, y por tanto así lo hace también su prioridad. Pero el ritmo de crecimiento del valor de la prioridad será tanto más rápido cuanto mayores sean los requisitos de ancho de banda de la conexión. De esta manera, el mecanismo acopla el efecto introducido por el planificador (el retardo en cola) con una medida de las demandas efectuadas por la aplicación (ancho de banda). Cuando el impacto negativo del planificador crece, también lo hace la prioridad, actualizándola con el tiempo, y con velocidades diferentes. Esto es, las conexiones de mayor ancho de banda (que generan los flits más rápidamente, con menor separación entre sí) ven incrementada su prioridad más rápidamente.

Esta aproximación también es válida para conexiones VBR. En este caso, los requerimientos de ancho de banda se pueden expresar en media, o irán variándose dinámicamente mediante palabras de control.

En el primer caso, se tomaría el tiempo *medio* entre llegadas para el cálculo de la prioridad. Esto provocará que algunos flits disfruten de más prioridad de la necesaria (cuando la fuente transmita por debajo de esa media) mientras que otros se vean perjudicados por necesitar una prioridad más alta (cuando la fuente transmita algún pico de tráfico). Un efecto beneficioso de esto puede ser el suavizado de los picos en el tráfico.

En el segundo caso, se puede aprovechar el conocimiento de las características del tráfico generado para variar sobre la marcha los requisitos de la conexión. Esto sucede por ejemplo, con tráfico de vídeo comprimido MPEG-2 [12]. Tal y como se verá posteriormente con mayor detalle, cuando se trate de los tipos de tráfico empleados en la evaluación de prestaciones, el tráfico de vídeo MPEG-2 consiste a grandes rasgos en la transmisión de una secuencia de imágenes. Cada una de esas imágenes tiene unos requisitos de ancho de banda diferentes. En ese caso, cuando la fuente vaya a inyectar una imagen en la red, conoce cuánto ancho de banda va a ocupar, y por tanto, puede comunicarlo a los encaminadores mediante una palabra de control. Así, a los flits de cada imagen se les tratará de forma adecuada a sus necesidades: los flits pertenecientes a imágenes con mayores requerimientos verán incrementada su prioridad con más rapidez que aquellos flits pertenecientes a imágenes con menores requerimientos.

5.1.2. Prioridad basada en *jitter*

Una segunda posibilidad para la función de actualización de la prioridad es el algoritmo denominado JBP *Jitter-Biased Priority* [104]. Según este algoritmo, la prioridad de un flit se calcula como el cociente entre el *jitter* experimentado por el flit hasta ese instante, más el *jitter* acumulado para la conexión, y el tiempo entre llegadas establecido para la conexión. El *jitter* acumulado no es más que la suma de todas las diferencias de retardo sucesivas. Expresado matemáticamente sería algo así:

$$prioridad = \frac{jitter\ acumulado + jitter\ actual}{IAT} \tag{5.2}$$

La idea detrás de este algoritmo es hacer que los flits de una conexión experimenten el mismo tiempo de servicio, y por tanto se minimice el *jitter*. Con ese objeto, esta aproximación reduce la prioridad de un flit si todavía es demasiado pronto para ser transmitido, con respecto al retardo medio experimentado por los flits anteriores de la conexión. Cuando el flit haya experimentado un retardo igual o superior a la media de la conexión, verá incrementada su prioridad con una velocidad que depende de los requerimientos de ancho de banda de la conexión.

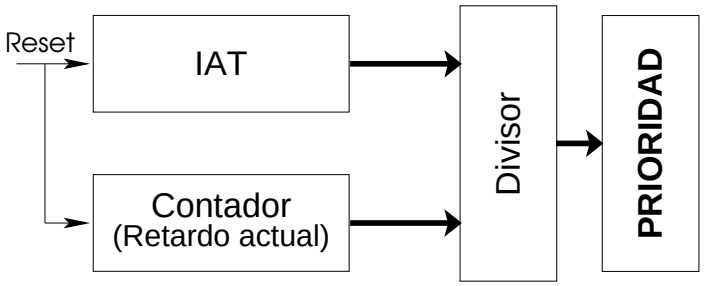


Figura 5.2: Diagrama de bloques del algoritmo IABP.

Cuando finalmente, el flit se transfiere a través del *crossbar*, el retardo sufrido a través del encaminador se registra, y se actualiza el valor del *jitter* acumulado para esa conexión.

5.1.3. Detalles de implementación

Las funciones de prioridad anteriormente comentadas necesitan de una circuitería asociada a cada canal virtual, pues es necesario calcular y actualizar el valor de la prioridad para los flits en cabeza de cada canal virtual, en cada ciclo de flit. Debido al elevado número de éstos que implementa el MMR, hay que tener muy en cuenta la complejidad de estos módulos, pues pueden hacer inviable la consecución de un diseño compacto y rápido.

En concreto, y tanto para IABP como para JBP, es necesario almacenar el retardo experimentado por cada flit, bien para emplearlo directamente (caso del IABP) o para calcular el *jitter* (caso del JBP). Esto hace necesario disponer de un contador por cada flit almacenado en el MMR. También es común la necesidad de un registro por conexión que almacene el IAT de la misma. Y en ambos casos es necesario implementar un divisor para calcular el cociente que da el valor de la prioridad.

Además, en la implementación del JBP sería necesario mantener por cada conexión un registro para almacenar el *jitter* acumulado por la misma, otro registro para mantener el retardo sufrido por el último flit transmitido, y la circuitería necesaria para implementar la diferencia entre este retardo, y el experimentado hasta el momento por el flit que está actualmente a la cabeza de la cola, en espera de ser transmitido. Así se calcularía el *jitter* que sufriría el flit actual. Y todavía es necesario un sumador para actualizar el valor del *jitter* acumulado cuando el flit sea transmitido.

En las Figuras 5.2 y 5.3 se muestran sendos diagramas de bloques de la implementación de ambos algoritmos. Se muestra únicamente la circuitería necesaria para los flits en cabeza de cada canal virtual. Por lo tanto, faltan en dichos esquemas los contadores que almacenan el retardo experimentado por cada flit dentro del encaminador.

Respecto al algoritmo IABP, únicamente es necesario disponer para cada canal virtual del registro que almacena el tiempo entre llegadas de los flits de la conexión. Este valor lo determina la fuente de la conexión, y puede incluirse en la sonda de establecimiento de la conexión. Si se trata de un flujo VBR, la conexión puede variar sus requerimientos dinámicamente, lo cual se traducirá en una modificación de este registro. La implementación se limita a calcular el cociente entre el retardo experimentado por el flit en la cola del MMR, almacenado en un contador que avanza con cada ciclo de reloj del encaminador, y el IAT, dando como resultado el valor de la prioridad.

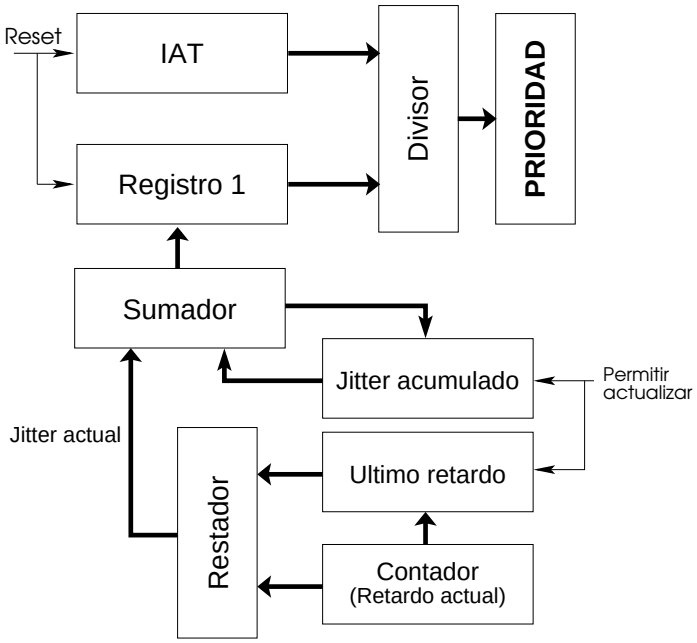


Figura 5.3: Diagrama de bloques del algoritmo JBP.

Por el contrario, el algoritmo JBP necesita de más circuitería auxiliar, como puede comprobarse en la Figura 5.3. Ello es debido a que es necesario calcular una estimación del *jitter* sufrido por el flit actual, que ha de sumarse con el valor del *jitter* acumulado para la conexión para componer el "dividendo" del cociente que dará el valor de la prioridad. El registro que almacena dicho "dividendo" aparece etiquetado como "Registro 1" en la Figura. En resumen, se puede observar que la implementación de este algoritmo requiere de 3 registros, un sumador y un restador (o una única unidad que realice las dos funciones) adicionales sobre la circuitería que necesita IABP. Los registros "Jitter acumulado" y "Último retardo" serán actualizados cuando el flit en cabeza de este canal virtual resulte planificado para su transmisión a través del *crossbar*.

Es evidente que la complejidad de implementación del algoritmo JBP es algo mayor que la del algoritmo IABP. En el Capítulo siguiente, dedicado a evaluar las prestaciones de la arquitectura de encaminador propuesta en esta Tesis y sus opciones de diseño, se compararán las prestaciones de ambos algoritmos, y se determinará si el incremento en complejidad introducido por JBP se traduce en una mejora importante de prestaciones que justifiquen el mayor coste.

Sin embargo, hay un elemento importante en ambos algoritmos que incrementa de manera muy notable la complejidad del circuito de cálculo de la prioridad. Se trata del divisor necesario para calcular el cociente de las magnitudes específicas de cada algoritmo con el IAT de la conexión. Un divisor es un elemento caro y complejo de implementar, y por tanto, sería interesante idear alguna forma de reemplazarlo por algún mecanismo más simple y menos costoso de implementar. Se trataría de aplicar el mismo razonamiento introducido por IABP y JBP, pero sustituyendo la división por alguna otra operación más simple de implementar y más rápida.

Conceptualmente, IABP es un algoritmo más sencillo e intuitivo que JBP. Además, se ha demostrado que el Encaminador Multimedia con el algoritmo IABP como planificador de enlaces, ofrece mejores prestaciones que si se emplea el algoritmo JBP [104]. Por todo ello,

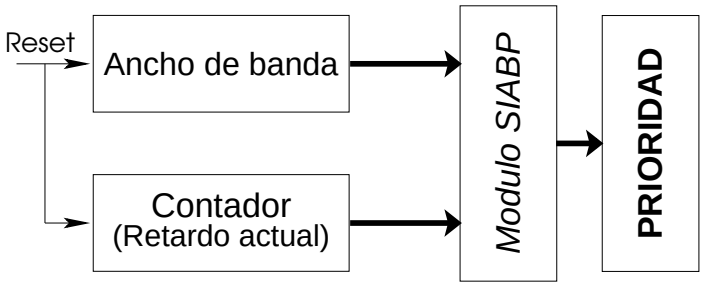


Figura 5.4: Diagrama de bloques del algoritmo SIABP.

se ha escogido la idea que introduce IABP como punto de partida para la realización de un algoritmo de planificación de enlaces más eficiente en términos de hardware. A continuación se describe el nuevo algoritmo.

En IABP, se relaciona el retardo experimentado por el flit en cabeza de cada canal virtual con sus necesidades de ancho de banda expresadas como el tiempo entre llegadas de dos flits consecutivos de la conexión. En el nuevo algoritmo, denominado **SIABP** (*Simple-IABP*), el valor de la prioridad de cada flit en cabeza se calcula de forma diferente. El retardo en cola del flit, al igual que sucede en IABP, se almacena en un contador que se incrementa en cada ciclo, esto es, el retardo en cola se expresa como un número entero de ciclos del encaminador.

Pero ahora la función de prioridad evoluciona de la siguiente forma. El valor inicial de la prioridad es el ancho de banda requerido por la conexión. En lugar de representarlo mediante el IAT, se emplea el número de *slots* por vuelta reservados para dar servicio al ancho de banda medio de la conexión. Esta magnitud es siempre un valor entero. Hay que hacer notar que calcular el cociente entre el retardo en cola y el IAT (como hace IABP) es equivalente a calcular el producto entre el retardo en cola y el ancho de banda. Si se aproxima el producto con operaciones de desplazamiento, la implementación se simplifica considerablemente. Lo que se hace es desplazar a la izquierda el valor de prioridad (es decir, multiplicarlo por dos) cada vez que el retardo en cola se hace mayor que 1, 2, 4, ..., 2^n, es decir, cada vez que un bit del contador del retardo se pone a uno por primera vez desde la última vez que se puso a cero. De esta forma, la QoS necesaria (representada por el valor inicial de la prioridad) también se relaciona con la QoS recibida por el flit (el retardo en cola).

Para comprobar la mejora en simplicidad de implementación del algoritmo SIABP frente al IABP se ha realizado una descripción en VHDL [11] para ambos algoritmos.

En la Figura 5.2 ya se ha mostrado el diagrama de bloques del algoritmo IABP. En la Figura 5.4 se muestran los bloques de hardware necesarios para el nuevo mecanismo de cálculo de la prioridad. Al igual que IABP, SIABP necesita del contador para almacenar el retardo del flit en cola. Sin embargo, SIABP almacena los requisitos de ancho de banda de la conexión de manera diferente, pues utiliza el número de *slots* especificado para la reserva de ancho de banda en la sonda de establecimiento de la conexión en lugar del IAT. La ventaja es que se trata de una magnitud entera.

Además, para calcular las prioridades, el algoritmo IABP emplea un módulo divisor estándar, mientras que el SIABP emplea un módulo especial denominado **módulo SIABP**. Los pines de entrada y salida de este módulo se muestran en la Figura 5.5 y un esquema de su lógica interna se muestra en la Figura 5.6, para valores de prioridad de 4 bits. Este módulo se encarga de efectuar los desplazamientos cuando sean necesarios. Se ha comprobado que mediante registros de 16 bits se puede representar sin problemas el rango de prioridades

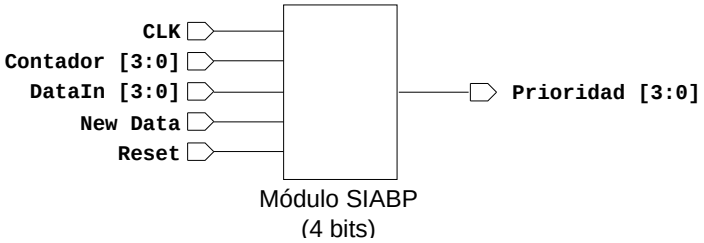


Figura 5.5: Distribución de pines para el módulo SIABP de 4 bits.

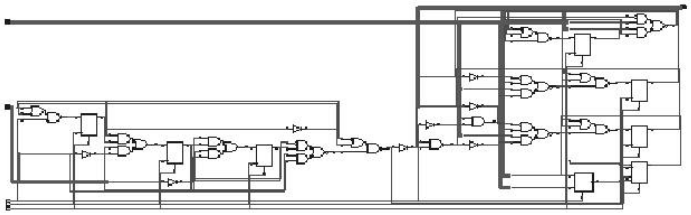


Figura 5.6: Lógica interna para el módulo SIABP de 4 bits.

necesario [26].

Estos módulos se han sintetizado con la herramienta de diseño Synopsis [10], para obtener detalles de la implementación. El proceso de síntesis ha empleado las librerías proporcionadas por Europractice, MIETEC 0.35 μm. La implementación del algoritmo IABP se basó en un divisor entero de 32 bits, proporcionado por el kit de diseño DesignWare [9], para obtener una estimación a la baja de la complejidad del algoritmo IABP (éste está formulado para trabajar con magnitudes en punto flotante).

El retardo estimado para la lógica IABP es de 155'87 ns, mientras que la implementación del SIABP ofrece un retardo de 4'07 ns. Es decir, el algoritmo SIABP ofrece un retardo 38 veces menor que la estimación entera de IABP. En términos de área de silicio, la reducción es también espectacular: 272.56 μm para el SIABP frente a los 6101.33 μm que necesita el IABP. Esto implica una reducción de unas 22 veces en área de silicio.

De nuevo, en el Capítulo dedicado a la evaluación de prestaciones se analizará si el ahorro en términos de hardware que representa SIABP sobre IABP, no se ve contrarrestado por una pérdida en las garantías de QoS que reciben los flujos multimedia.

Una vez calculado el valor de la prioridad de cada flit en cabeza de cada canal virtual, es necesario proceder a una ordenación de los mismos para obtener aquellos canales virtuales con mayor valor de prioridad. En este proceso, se han de enmascarar aquellos canales virtuales que no dispongan de créditos para enviar un flit al siguiente encaminador. Para ello, se hará uso de los vectores de bit de estado correspondientes (ver Sección 4.3.2).

La ordenación de un conjunto de valores que ocupan un número relativamente elevado de bits (los últimos resultados obtenidos en el MMR se han realizado con valores de prioridad de 16 bits [27, 26]), no es un problema trivial.

Existen diversas aproximaciones en la literatura para la implementación mediante hardware de las denominadas **colas de prioridad** [138, 176, 117, 23]. Estas colas almacenan en orden de prioridad un conjunto de paquetes o punteros a paquetes, y están ideadas para su uso en conmutadores con *buffers* a la salida. La principal diferencia que impide su aplicación directa

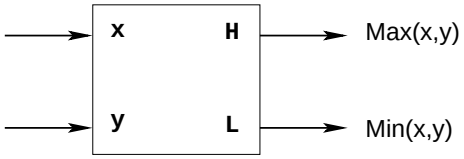


Figura 5.7: Conmutador de comparación para una red de ordenación.

en el entorno del MMR es que las colas de prioridad tratan con paquetes cuya prioridad es *estática*, es decir, mediante algún mecanismo se ha asignado un valor de prioridad a cada paquete, y este valor permanece inalterable mientras el paquete permanece en el conmutador. Por tanto, las operaciones críticas en estas colas son la adicción o encolado de nuevos paquetes en el lugar correspondiente de acuerdo a su prioridad, y la extracción o desencolado del elemento de mayor prioridad para su transmisión por el enlace de salida. El objetivo de las propuestas que aparecen es que estas operaciones se puedan realizar en un orden de tiempo constante, e independiente del tamaño de la cola y del número de niveles de prioridad empleados [117].

Este no es el caso del MMR, donde las prioridades asignadas al flit de cabeza de cada canal virtual se modifican en cada ciclo de flit, y por tanto el orden relativo entre los distintos canales virtuales puede variar. Es decir, en el MMR, para cada ciclo de flit, es necesario reordenar los valores de prioridad de todos los canales virtuales con posibilidad de enviar un flit para obtener el conjunto de candidatos que se pasarán al planificador del conmutador. En el peor de los casos, se dispone de tantos valores a ordenar como canales virtuales por enlace físico tiene el MMR, esto es, hay que ordenar cada ciclo de flit del orden de 100-200 valores de unos 16 bits cada uno.

La solución para la ordenación mediante hardware de un conjunto de valores pasa por el empleo de una **red de ordenación** (*sorting network*) [21]. Una red de ordenación no es más que una clase especial de algoritmos de ordenación, en los que la secuencia de comparaciones no depende de los datos. Esto las hace apropiadas para ser implementadas mediante hardware.

Una red de ordenación está formada por un conjunto de elementos básicos, los *conmutadores/comparadores* (C/C), interconectados según cierto patrón, dependiente del tipo de red. Un conmutador/comparador tiene dos entradas y dos salidas, y ejecuta una operación de comparación/intercamio. En la Figura 5.7 se representa un esquema de un C/C. Las líneas de entrada se denominan `x` e `y`, y las líneas de salida `H` y `L`. La lógica interna de este elemento es la siguiente:

```
si x > y
   entonces
      L = y, H = x
   sino
      L = x, H = y
```

Existen varios patrones para interconectar estos elementos de comparación simples que producen una ordenación de los valores introducidos en las entradas. Por ejemplo, una de las

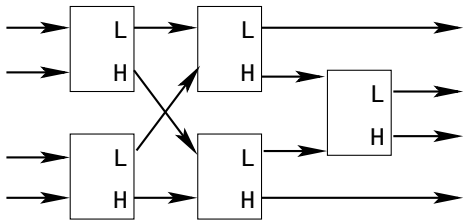


Figura 5.8: Red de ordenación por mezcla par-impar para 4 valores.

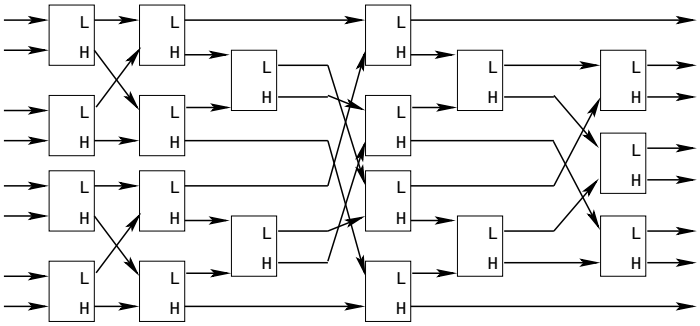


Figura 5.9: Red de ordenación por mezcla par-impar para 8 valores.

primeras redes de ordenación propuestas es la *red de ordenación por mezcla par-impar* (*odd-even sort merging network*) [21]. En las Figuras 5.8 y 5.9 se muestran redes par-impar para ordenar respectivamente 4 y 8 valores. La idea es, dada una secuencia desordenada de números, dividirla en dos mitades, los valores que ocupan los lugares pares en la secuencia inicial, y los valores que ocupan los lugares impares. Estas dos mitades se ordenan, aplicando el mismo criterio de forma recursiva, teniendo en cuenta que dos valores se ordenan con un único C/C. Finalmente, las dos mitades ordenadas se mezclan para obtener la secuencia total ordenada. Esto se representa para N entradas en la Figura 5.10, donde aparecen dos módulos para ordenar $N/2$ valores, y una etapa final de mezcla.

Existen muchas otras formas de construir una red de ordenación: bitónica [21], por burbuja y Shellsort [94], balanceada periódica [46], cada una con un orden de complejidad diferente (ver Tabla 5.1). Como se puede observar, como mínimo el orden de complejidad es $O(n \cdot log(n)^2)$ en cuanto al número de elementos C/C necesarios y $O(log(n)^2)$ en cuanto al número de etapas, siendo n el número de entradas a la red, es decir, el número de valores a ordenar. El primer parámetro (número de elementos C/C) está relacionado con el área de silicio que ocupará la red de ordenación, mientras que el segundo (número de etapas) lo está con el retardo en que se incurrirá para obtener una ordenación válida.

Señalar que existe una propuesta de red de ordenación de complejidad $O(n \cdot log(n))$ en cuanto al número de elementos C/C, la red AKS [13], pero su constante es tan grande que resulta más lenta que las redes de complejidad $O(n \cdot log(n)^2)$ para todos los tamaños de problema prácticos.

En el caso del Encaminador Multimedia, el número de valores a ordenar es relativamente elevado (del orden de 100-200). Por ejemplo, con una configuración de 128 canales virtuales por enlace físico, se debe implementar una red capaz de ordenar los 128 posibles valores de prioridad. Empleando, una red de búsqueda bitónica, ésta necesitaría del orden de

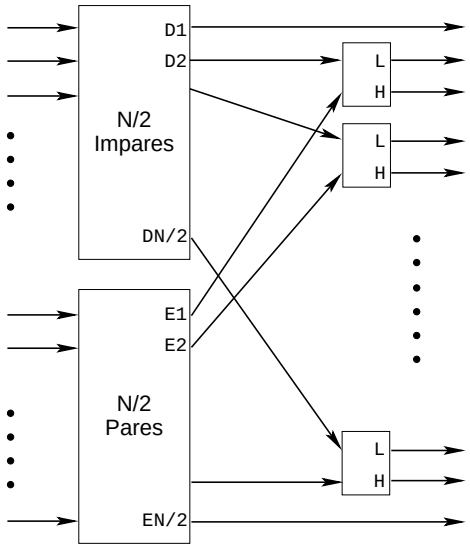


Figura 5.10: Red de ordenación por mezcla par-impar para N valores.

Algoritmo	**Número de etapas**	**Número de C/C**
Par-impar	$O(n)$	$O(n^2)$
Burbuja	$O(n)$	$O(n^2)$
Bitónica	$O(log(n)^2)$	$O(n \cdot log(n)^2)$
Shellsort	$O(log(n)^2)$	$O(n \cdot log(n)^2)$
Balanceada periódica	$O(log(n)^2)$	$O(\frac{n}{2} \cdot log(n)^2)$

Tabla 5.1: Resumen de la complejidad de distintas redes de ordenación.

$log(128)^2 = 7^2 = 49$ etapas, y $128 \cdot log(128)^2 = 128 \cdot 49 = 6272$ comparadores. Un elemento comparador de valores de 10 bits se puede implementar con un retardo menor de 1 nanosegundo, empleando tecnología CMOS de 0.25 μm [113]. Por tanto, el retardo introducido por un comparador de valores de 16 bits podría estimarse en 2 nanosegundos.

Según lo anterior, el retardo consumido por esta fase se puede estimar holgadamente en 100 nanosegundos (esto es, 49 etapas, con 2 ns de retardo cada una), añadiendo la lógica de multiplexación necesaria para implementar la ruta de datos. Para el caso de un Encaminador con enlaces de 1.24 Gbps, y un ancho de palabra de 16 bits, el período de reloj interno es de 12'9 ns. Por tanto, la fase de ordenación para la obtención de los candidatos supone algo menos de 8 ciclos de reloj.

En [125] se propone una arquitectura capaz de ordenar un número indeterminado N de valores mediante una red de ordenación con un número de entradas p fijo y pequeño. La idea es aprovechar la estructura en fases de una red de ordenación, para emplearla de forma segmentada y obtener en varios pasos una ordenación de todo el conjunto de N elementos. Esta aproximación podría reducir el área de silicio dedicada a la ordenación, aunque no hay que olvidar que es necesaria cierta lógica de control para manejar las distintas fases segmentadas.

Por otro lado, en el Encaminador Multimedia no interesa obtener la ordenación de todos los

Número de candidatos	Número de C/C eliminados
$C = 1$	12
$C = 2$	4
$C = 4$	1

Tabla 5.2: Reducción en el número de elementos C/C sobre una red de ordenación par-impar de 8 entradas, según el número de candidatos a extraer.

valores de prioridad, sino únicamente conocer cuáles son los C valores más prioritarios, siendo C un valor entre 1 y el número de puertos del encaminador. Los canales virtuales a los que corresponden esos C valores de prioridad serán los *candidatos* que se pasarán al planificador del conmutador. Por tanto, para reducir la complejidad del circuito de ordenación, se puede "podar" éste, para obtener únicamente aquellas C entradas más prioritarias. Hay que señalar que se trata de una reducción en términos de área ocupada por el circuito, pues el número de etapas necesarias para ordenar los valores no se puede reducir.

Por ejemplo, la red de ordenación de la Figura 5.9, que ordena 8 valores, puede podarse tal y como se muestra en la Figura 5.11, si únicamente se necesita un número C (menor que 8) de valores más prioritarios. En dicha Figura se muestra como quedaría la red original una vez "podada" convenientemente, para valores de C de 1, 2 y 4.

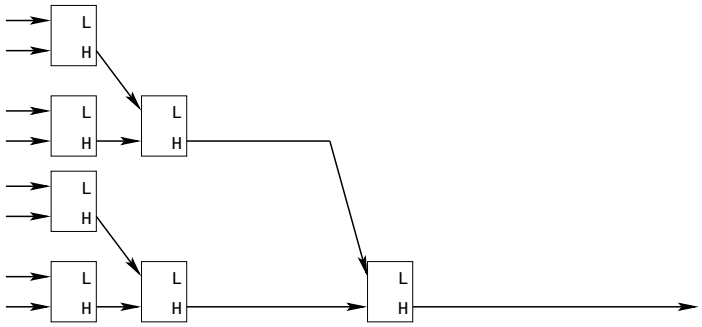
De los ejemplos mostrados en la Figura 5.11, se puede deducir que el número de elementos C/C "podados" es inversamente proporcional a la relación entre el número de candidatos a obtener, y el número total de canales virtuales. Esto es, cuando menor sea el número de candidatos en relación al número total de canales virtuales, mayor número de elementos C/C se pueden eliminar. En la Tabla 5.2 se muestra cuántos elementos C/C se eliminan en cada caso de los mostrados en la Figura 5.11.

Una vez descrita la forma de planificar los enlaces en el Encaminador Multimedia, es el turno de abordar cómo se tratan los candidatos seleccionados en este paso, para obtener una planificación del *crossbar* libre de conflictos, que maximice la productividad a la vez que tenga en cuenta las garantías de QoS de las conexiones multimedia. Esta tarea corresponde al *algoritmo de planificación del conmutador*, que será descrito en detalle en la siguiente Sección.

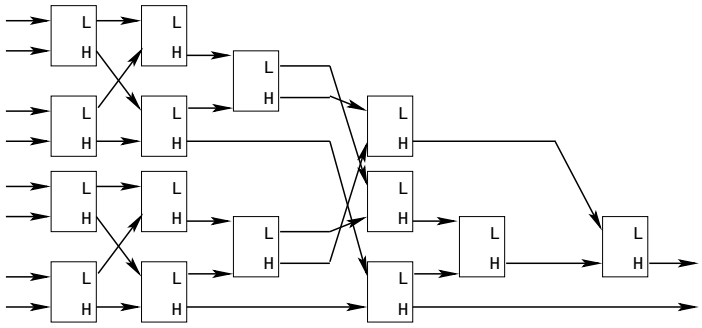
5.2. Algoritmos de planificación del conmutador

Como ya se comentó en el Capítulo anterior, el Encaminador Multimedia basa su arquitectura en un *crossbar* multiplexado, con tantos puertos como enlaces físicos tiene el Encaminador. Dado que el MMR soporta un gran número de canales virtuales, en cada ciclo de flit, existen varios canales virtuales por cada puerto de entrada con flits listos para ser transmitidos. El *algoritmo de planificación de enlaces* hace una primera selección entre todos estos canales, atendiendo a criterios relacionados con los requisitos de QoS de las distintas conexiones, tal y como se ha descrito en la Sección anterior. Ahora bien, todavía pueden existir conflictos por el uso de los puertos del *crossbar* entre los canales virtuales seleccionados.

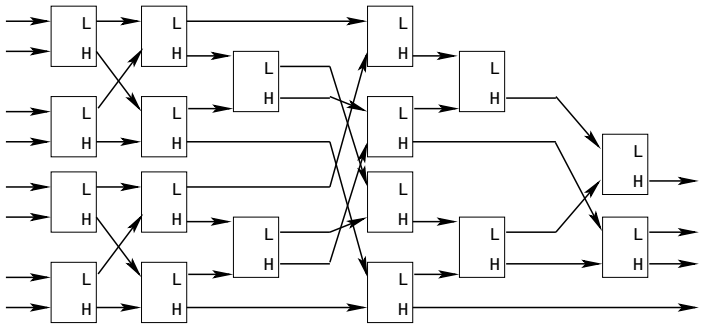
En este punto es donde entra en juego el **algoritmo de planificación del conmutador**. Su misión es encontrar un emparejamiento libre de conflictos entre canales virtuales de entrada y de salida, que cumpla dos objetivos:



(a) 1 candidato



(b) 2 candidatos



(c) 4 candidatos

Figura 5.11: Reducciones sobre la red de ordenación por mezcla par-impar para obtener distintos números de candidatos.

1. Maximizar la utilización media del *crossbar*, con el objeto de trasegar la mayor cantidad posible de información sin saturarse

2. Respetar en todo momento las garantías de QoS de las distintas conexiones multimedia

Como se ha puesto de manifiesto en el Capítulo 3, la inmensa mayoría de algoritmos de emparejamiento propuestos hasta la fecha únicamente tienen en cuenta el primer objetivo, maximizar la productividad. En los últimos años, están apareciendo algunas propuestas encaminadas a cumplir ambos objetivos, pero no está clara la posibilidad de ser implementados en un encaminador simple de alta velocidad.

En este apartado se trata por tanto de describir varias propuestas de algoritmos de planificación del conmutador, ideadas para la arquitectura de encaminador multimedia propuesta,

que persigan cumplir con los dos objetivos marcados, a la vez que permitiendo una implementación compacta y rápida.

La clave está en considerar el "grado de competencia" por el uso de los enlaces de salida con el objeto de maximizar productividad, a la vez que se aprovechan las prioridades calculadas por el planificador de enlaces para realizar emparejamientos que tengan en cuenta los requerimientos de las conexiones en cuanto a QoS.

Esto se plasma en dos decisiones críticas que han de ser tomadas dentro del planificador: el orden en que se examinan los puertos de salida para resolver los conflictos, y la política de arbitraje. El primero es importante porque cuando un puerto de entrada se empareja con un puerto de salida, descarta las peticiones para el resto de sus candidatos. Esto puede modificar el número de conflictos en otros puertos. Como resultado el orden en que se consideran los puertos puede influir en el número de peticiones satisfechas, y por tanto, en la utilización del conmutador. Esto se va a plasmar en una **función de ordenación**.

En cuanto a la política de arbitraje, se trata de definir una **función de arbitraje**, que consistirá simplemente en aprovechar las prioridades calculadas para los flits por el planificador de enlaces, y seleccionar en su caso el flit de mayor prioridad.

Los algoritmos que serán descritos a continuación comparten la función de arbitraje: se trata de la definida en el párrafo anterior. Sin embargo, difieren en la función de ordenación planteada.

5.2.1. Algoritmo basado en el orden de los candidatos

La primera propuesta establece una función de ordenación de puertos de salida en primer lugar, por niveles, y en segundo lugar por número creciente de conflictos. Esto es, se considera que los candidatos más prioritarios de cada enlace son los candidatos de nivel 1, los segundos más prioritarios son los candidatos de nivel 2, etc. Entonces, el algoritmo de planificación trata de establecer los emparejamientos considerando en primer lugar únicamente los candidatos de nivel 1. Si quedan puertos por emparejar, pasa a considerar los candidatos de nivel 2, y así sucesivamente, hasta que se emparejen todos los puertos, o no se puedan hacer más emparejamientos.

Debido a que en este proceso se van considerando las peticiones según el orden establecido por los niveles de candidatos, a este algoritmo de planificación del *crossbar* se le ha bautizado como **COA (*Candidate Order Arbiter*)** [28].

La idea adelantada en el párrafo anterior se plasma en el algoritmo descrito en la Figura 5.12. En dicha descripción no se han concretado las estructuras de datos auxiliares que emplea el algoritmo. Es necesario organizar la información involucrada en el proceso de emparejamientos de forma conveniente para agilizar el proceso.

En primer lugar, los planificadores de enlace generan la información de los canales virtuales seleccionados como candidatos en el denominado **vector de candidatos**. Un ejemplo se muestra en la Figura 5.13. Este vector contiene los valores de prioridad y el enlace de salida requerido para cada candidato. Concretamente, en el ejemplo de la Figura, el candidato de nivel 1, el más prioritario, tiene un valor de prioridad de 55, y está dirigido al puerto de salida 0. El siguiente candidato en orden de prioridad, el de nivel 2, tiene una prioridad de 43, y está dirigido al puerto de salida 2, etc.

Los planificadores de enlace generarán un vector de candidatos por cada enlace de entrada. Con el objeto de que la información de estos vectores sea más manejable, se representa en forma de una **matriz de selección**.

Algoritmo de Planificación COA:

Fase 1: Selección de candidatos Esta fase no es más que la *planificación de enlace*, que se ejecuta en paralelo en todos los enlaces y da como resultado un conjunto de candidatos. Estos candidatos se organizan de forma conveniente por niveles, para agilizar su empleo en las siguientes fases.

Fase 2: Ordenación de puertos Esta fase consiste en considerar los puertos de salida primero por nivel, y dentro de cada nivel, en orden creciente de conflictos, para intentar maximizar la probabilidad de emparejar el mayor número posible de entradas y salidas. De esta manera, los puertos de salida con más peticiones se seleccionan los últimos dentro de su nivel, puesto que esos puertos tienen más posibilidades de ser emparejados. Y se intenta realizar emparejamientos entre los candidatos más prioritarios de cada enlace en primer lugar, para poder maximizar las garantías de QoS. Una vez agotados los candidatos de un nivel, si todavía hay puertos libres, se pasa al siguiente nivel para intentar rellenar los "huecos" que queden. Este proceso se repite para todos los niveles de candidatos. En caso de que exista más de un candidato para el puerto de salida seleccionado en ese nivel, se aplicaría la tercera fase.

Fase 3: Arbitraje El arbitraje consiste en decidir, una vez seleccionado un puerto de salida en la Fase 2, y en caso de que existan para él distintos candidatos procedentes de enlaces de entrada diferentes, cuál de esos canales de entrada va a emparejarse con él. El criterio a emplear será escoger al más prioritario.

Figura 5.12: Algoritmo de planificación *Candidate Order Arbiter* (COA).

Figura 5.13: Algoritmo COA: Ejemplo de vector de candidatos con 4 candidatos.

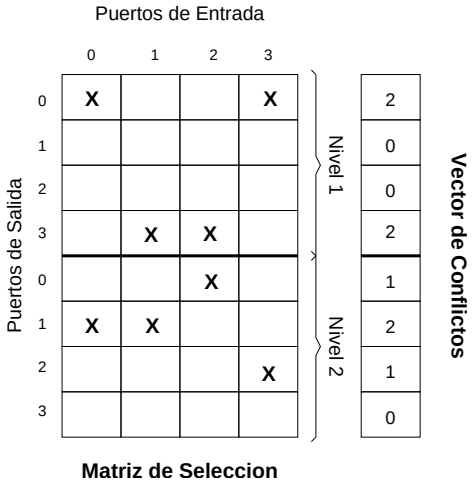


Figura 5.14: Algoritmo COA: Ejemplo de matriz de selección y vector de conflictos para un encaminador 4×4 y 2 niveles de candidatos.

Para un encaminador de $N \times N$ puertos, y C candidatos por enlace físico ($1 \leq C \leq N$), la matriz de selección tendrá $N \times C$ filas, y N columnas. Las primeras N filas de la matriz almacenan las peticiones de los candidatos de mayor prioridad de cada puerto de entrada (los candidatos de nivel 1). Las siguientes N filas almacenan las peticiones correspondientes a los candidatos de nivel 2 de cada entrada, y así sucesivamente. Esto es, la entrada (i, j) está activa si el flit de mayor prioridad de la entrada j requiere el puerto i. La información para el resto de niveles de candidatos se almacena de forma análoga. De esta forma, se tiene de forma compacta toda la información obtenida por los planificadores de enlaces. En la Figura 5.14 se muestra un ejemplo de matriz de selección, para un encaminador de 4 puertos, y 2 candidatos. Las filas y columnas se etiquetan con el número de puerto de salida y de entrada correspondientes, respectivamente. En dicha figura se puede observar como de los candidatos seleccionados en el puerto de entrada 0, el más prioritario (nivel 1) está dirigido al puerto de salida 0, mientras que el segundo más prioritario (nivel 2) está dirigido al puerto de salida 1. Del mismo modo, el puerto de entrada 1 tiene dos candidatos, dirigidos a los puertos de salida 3 y 1, respectivamente. Y así con el resto de puertos de entrada. Puede darse el caso de que para algún puerto de entrada sólo haya un canal virtual con flits listos para transmitir. En tal caso, no existiría ningún candidato de nivel dos para ese puerto.

De esta matriz, se extrae información del grado de competencia por los enlaces de salida en cada nivel para forma el denominado **vector de conflictos**. Este vector tiene $N \times C$ componentes, y almacena el número de entradas no nulas en cada fila de la matriz de selección. Esto es, el vector de conflictos identifica el número de conflictos que hay por cada puerto de salida. Así, queda disponible la información necesaria para llevar a cabo la segunda fase del algoritmo, la ordenación de los puertos. Dentro de cada nivel, se considerarán en primer lugar aquellos puertos de salida cuya posición correspondiente en este vector tenga el valor mínimo. En caso de empate, se seleccionará uno de los puertos aleatoriamente. En la parte derecha de la Figura 5.14 se muestra el vector de conflictos correspondiente a la matriz de selección de la misma Figura.

Seguidamente, y con el objeto de clarificar su comportamiento, se va a mostrar paso a paso una ejecución del algoritmo descrito, para un encaminador 4×4, con 2 niveles de candidatos. Se supone que los planificadores de enlaces han generado previamente los candidatos, cuya información se refleja en la matriz de selección de la Figura 5.14. Por claridad, las casillas de

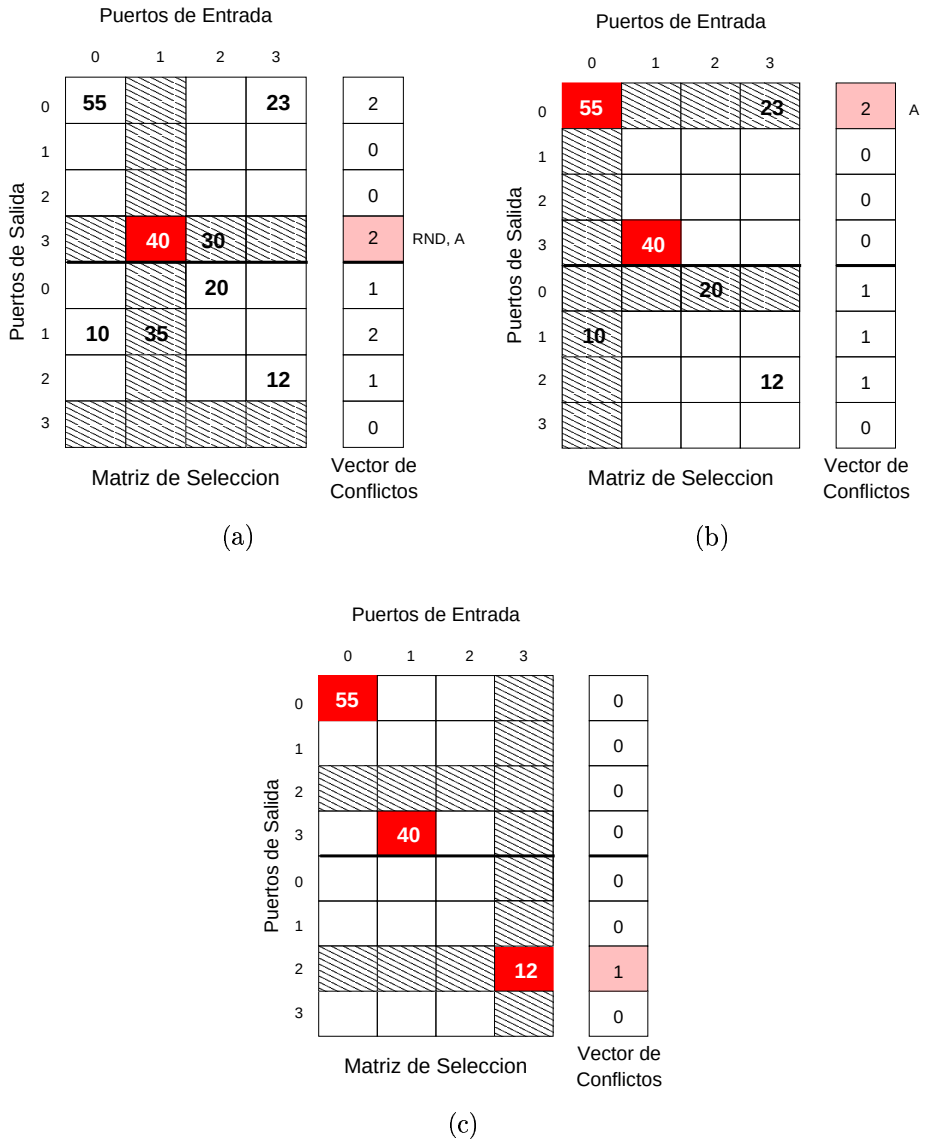


Figura 5.15: Algoritmo COA: Ejemplo de ejecución.

la matriz de selección activas se van a marcar con el valor de la prioridad del flit candidato correspondiente, pues este valor será necesario en la fase de arbitraje. El proceso consistirá en aplicar sucesivamente las fases 2 y 3 (ordenación de puertos y arbitraje) hasta conseguir emparejar todos los puertos posibles.

En primer lugar, el algoritmo trabaja sobre los *candidatos de nivel 1* (parte superior de la matriz). De ellos, escogerá aquel puerto de salida con menor número de conflictos (pero mayor que 0, evidentemente). En la Figura 5.15-(a), hay dos puertos (1 y 2) que no tienen peticiones, y por tanto, su valor en el vector de conflictos es 0, mientras que los otros dos puertos (0 y 3) tienen 2 conflictos cada uno. Por tanto, se ha de escoger aleatoriamente uno de ellos, por ejemplo, el 3. Esta elección aparece señalada como RND en la Figura. A continuación, dado que para el puerto 3 hay dos peticiones, hay que aplicar la fase de *arbitraje*, para seleccionar la más prioritaria. Esto se señala con la etiqueta A. De este proceso, resulta elegido el candidato proveniente del puerto de entrada 1. Esto es, el canal virtual correspondiente al candidato más prioritario del puerto de entrada 1, enviará su flit de cabeza al canal virtual

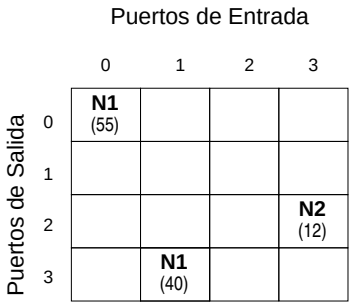


Figura 5.16: Algoritmo COA: Planificación obtenida tras la ejecución del algoritmo.

correspondiente del puerto de salida 3. Cada emparejamiento conseguido se marca con un sombreado oscuro en la Figura 5.15.

Cada vez que se realiza un emparejamiento, la matriz de selección debe de "limpiarse" descartando aquellas peticiones que involucran a los puertos comprometidos en el emparejamiento realizado. Por tanto, en este caso se deben limpiar las filas correspondientes al puerto de salida 3, y la columna correspondiente al puerto de entrada 1, que aparecen rayadas en la Figura 5.15-(a). Esto da como resultado la matriz de selección que se muestra en la Figura 5.15-(b). Obsérvese que también se ha recalculado el vector de conflictos.

De nuevo, hay que determinar el puerto de salida con menor número de conflictos en este primer nivel. Ahora es el puerto 0 el elegido, pues es el único que todavía tiene peticiones en este nivel de candidatos. De nuevo este puerto tiene dos candidatos, y hay que ejecutar la fase de arbitraje. Resulta escogido el candidato de prioridad 55, que proviene del puerto de entrada 0. De esta manera, se realiza el segundo emparejamiento, puerto de entrada 0 con puerto de salida 0, y se limpian la columna y filas correspondientes (rayadas en la Figura). Tras este paso, ya no queda ningún candidato en el primer nivel (la parte superior del vector de conflictos queda igual a 0's tras la pertinente actualización).

Si se considerase un único nivel de candidatos, el algoritmo terminaría en este punto. Se han emparejado 2 puertos, con lo que los otros dos pares de puertos entrada/salida permanecerían desocupados durante este ciclo de flit, desperdiciando la mitad del ancho de banda del *crossbar*. Ya que ese no es el caso, el planificador pasa entonces a considerar el nivel 2 de candidatos, por si pudiera rellenar algún hueco más. Puesto que quedan puertos libres, cabe la posibilidad de realizar todavía algún emparejamiento más, y mejorar así la utilización del *crossbar*.

Entonces, el proceso se repite para el *segundo nivel de candidatos*. Ahora, sólo hay un puerto de salida con número de conflictos mayor que 0, el puerto 2, y por tanto se selecciona para ser planificado (Figura 5.15-(c)). Puesto que hay un único candidato para ese puerto, el emparejamiento está claro: puerto de salida 2 con puerto de entrada 3. Después de este paso, la matriz de selección queda vacía, y el vector de conflictos es totalmente nulo, lo que indica la finalización del algoritmo, y la obtención de una planificación válida. Esta planificación se muestra en la Figura 5.16.

Se puede comprobar como en esta ejecución del algoritmo se han conseguido planificar 3 de los 4 los puertos del conmutador, con lo que se pierde la cuarta parte del ancho de banda del *crossbar* en este ciclo de flit. Además, de los 3 flits planificados para transmitirse, 2 son los más prioritarios de sus respectivos enlaces de entrada, lo que implica que el planificador tiene en cuenta las necesidades de las aplicaciones en cuanto a QoS.

Algoritmo de Planificación CCA:

Fase 1: Selección de candidatos Es idéntica al algoritmo anterior.

Fase 2: Ordenación de puertos Esta fase consiste en considerar los puertos de salida en orden creciente de conflictos, para intentar maximizar la probabilidad de emparejar el mayor número posible de entradas y salidas. De esta manera, los puertos de salida con más peticiones se seleccionan los últimos, puesto que esos puertos tienen más posibilidades de ser emparejados. En caso de que exista más de un candidato para el puerto de salida seleccionado en ese nivel, se aplicaría la tercera fase.

Fase 3: Arbitraje El arbitraje consiste en decidir, una vez seleccionado un puerto de salida en la Fase 2, y en caso de que existan para él distintos candidatos procedentes de enlaces de entrada diferentes, cuál de esos canales de entrada va a emparejarse con él. El criterio a emplear será escoger al más prioritario.

Figura 5.17: Algoritmo de planificación *Candidate Conflict Arbiter* (CCA).

5.2.2. Algoritmo basado en el grado de conflictos

El algoritmo anterior considera los candidatos por orden de nivel a la hora de realizar los emparejamientos a través del *crossbar*. El objetivo es intentar que en lo posible, sean los candidatos más prioritarios de cada enlace los que resulten emparejados. Para soportar esto, se emplea una matriz de selección organizada por niveles. Sin embargo, esto puede incrementar la complejidad de implementación del algoritmo, así como su tiempo de ejecución, al tener que ir considerando sucesivamente los distintos niveles de candidatos generados por los planificadores de enlaces.

Por tanto, se plantea otra posibilidad, que consiste en calcular una matriz de selección de dimensiones $N \times N$, donde se mezclen las peticiones de los candidatos de los distintos niveles. Esto implica que todos los candidatos se considerarán a la vez a la hora de realizar la planificación. El criterio de selección de un puerto de salida para emparejar será exclusivamente el número de conflictos, teniendo en cuenta para ello a los candidatos de todos los niveles. Por ello, a este algoritmo se le ha denominado **CCA (*Candidate Conflict Arbiter*)**.

De esta manera, se simplifica la implementación del algoritmo, pero cabe la posibilidad de que las garantías de QoS se vean comprometidas. Esto es algo que deberá analizarse en el Capítulo siguiente dedicado a Evaluación de Prestaciones.

El algoritmo queda tal y como se refleja en la Figura 5.17. Las estructuras auxiliares empleadas por el algoritmo CCA son similares a las del caso anterior: vector de candidatos, matriz de selección y vector de conflictos. Mientras que el **vector de candidatos** mantiene el mismo formato (ver Figura 5.14), las otras dos estructuras difieren, tal y como se indica a continuación.

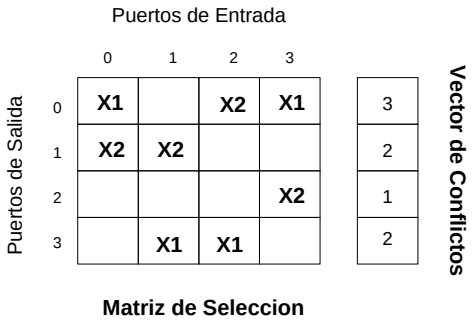


Figura 5.18: Algoritmo CCA: Ejemplo de matriz de selección y vector de conflictos para un encaminador 4×4 y 2 niveles de candidatos.

La **matriz de selección** tiene dimensión $N \times N$, debido a que ahora se consideran todos los candidatos a la vez, y por tanto, no es necesario organizarla por niveles. Cada entrada (i, j) está activa si al menos uno de los flits candidatos de la entrada j requiere el puerto i. En el caso de que dos o más candidatos de un mismo enlace estén dirigidos al mismo enlace de salida, únicamente se considerará el de menor nivel, es decir, el más prioritario.

De igual modo, el **vector de conflictos** tendrá N componentes, cada uno de los cuales indica el número de conflictos por cada puerto de salida, *considerando todos los niveles de candidatos.*

En la Figura 5.18 se muestran la matriz de selección y el vector de conflictos equivalentes a los que se mostraban en la Figura 5.14 para el algoritmo COA.

Para clarificar el funcionamiento del algoritmo, se mostrará a continuación un ejemplo de su ejecución paso a paso, partiendo de la misma situación empleada en el algoritmo anterior, y reflejada en la Figura 5.18 sobre las estructuras de este nuevo algoritmo. De nuevo, las casillas activas se van a representar en el ejemplo por el valor de prioridad del candidato correspondiente. Cuando haya que realizar alguna elección aleatoria del puerto de salida, se señalará con RND, y cuando sea necesario aplicar la fase de arbitraje, se marcará con A.

En primer lugar, de la fase de ordenación de puertos establecida, se obtiene que el puerto de salida con menor número de conflictos es el 2. Puesto que sólo tiene un candidato, no hay que aplicar la fase de arbitraje. De esta forma, se obtiene el primer emparejamiento: puerto de entrada 3 con puerto de salida 2. En este momento, y al igual que con el algoritmo anterior, se "limpian" la fila y columna correspondientes (Figura 5.19-(a)), obteniéndose la matriz y vector de conflictos actualizados de la Figura 5.19-(b).

Se vuelve a aplicar el criterio de ordenación, y esta vez hay empate entre los puertos 0, 1 y 3. Todos ellos tienen 2 conflictos. Por tanto, hay que escoger uno aleatoriamente, por ejemplo, el puerto 0. Puesto que existen dos candidatos destinados a ese puerto, hay que emplear la fase de arbitraje para escoger al más prioritario. El elegido es el candidato con un valor de prioridad de 55. De esta manera, se obtiene el segundo emparejamiento: puerto de entrada 0, con puerto de salida 0. Se limpian la fila y columna correspondientes, y se actualiza el vector de conflictos (Figura 5.19-(c)).

En este caso, la ordenación de puertos da como enlace de salida con menor número de conflictos al puerto de salida 1, con un único candidato, generado por el puerto de entrada 1. Por tanto, se realiza directamente el emparejamiento entre dicho par de puertos, actualizándose de nuevo convenientemente la matriz de selección y el vector de conflictos (Figura 5.19-(d)).

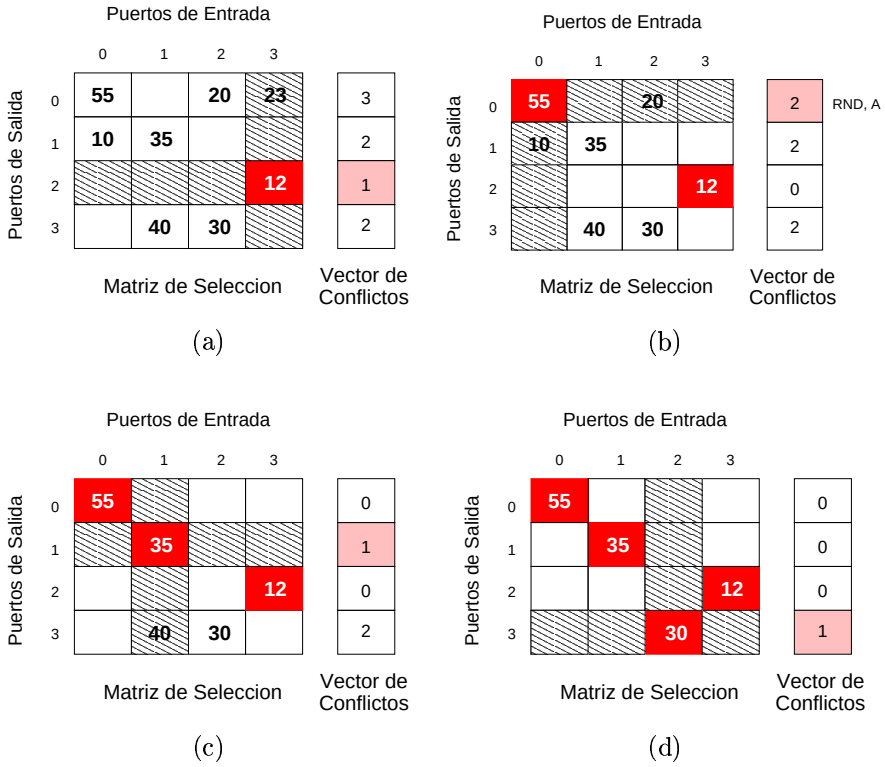


Figura 5.19: Algoritmo CCA: Ejemplo de ejecución.

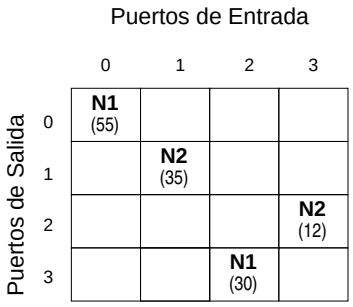


Figura 5.20: Algoritmo CCA: Planificación obtenida tras la ejecución del algoritmo.

Por último, sólo queda un puerto de salida con conflictos, el puerto 3, se selecciona, y puesto que de nuevo sólo tiene un candidato, no es necesario aplicar el arbitraje. Después de todo el proceso, la planificación calculada es la que se muestra en la Figura 5.20. Puede observarse como se ha obtenido una planificación que aprovecha completamente el ancho de banda del *crossbar*.

Comparando esta planificación con la obtenida mediante el algoritmo COA, se aprecia como se han planificado más flits, pero a costa de dejar de planificar un flit de prioridad mayor (candidato de nivel 1 del puerto de entrada 1, con prioridad 40), en favor de dos flits de menor prioridad (candidatos de los puertos de entrada 1 y 2, de nivel 2 y 1, y con prioridades 35 y 30, respectivamente). En principio, parece que con el algoritmo CCA efectivamente se aprovecha mejor el ancho de banda del *crossbar*, obteniendo una mejor utilización. Sin embargo, esto puede suceder a costa de imponer mayores retardos a flits más prioritarios, lo cuál puede comprometer sus garantías de QoS. Todo esto será analizado en el Capítulo siguiente, dedicado a la Evaluación de Prestaciones de la arquitectura y los algoritmos propuestos.

5.2.3. Planificación del conmutador y mensajes de control

Ya se ha señalado que en el Encaminador Multimedia los mensajes de control deben ser transmitidos con el menor retardo posible. Para ello, el planificador de enlace, independientemente del algoritmo de cálculo de la prioridad que se implemente, asignará a los flits de este tipo de tráfico la máxima prioridad. De esta forma, un flit de control siempre resultará elegido como candidato de primer nivel en la planificación del enlace.

Cuando se pasan los candidatos de los enlaces al planificador del conmutador, en cualquiera de los dos algoritmos propuestos, es posible que un flit de control no resulte planificado de inmediato. Esto es porque en ambos, el primer criterio de selección de un enlace de salida para planificar es el número de conflictos. Por tanto, puede darse el caso de que un mensaje de control esté destinado a un puerto de salida para el que haya muchos conflictos, y por no ser de los primeros elegidos en la planificación, finalmente resulte descartado. Esto impide la transmisión con mínimo retardo de los mensajes de control, independientemente de que se les haya asignado la mayor prioridad en la planificación del enlace.

Para solucionar este problema, se debe realizar en ambos casos un paso previo, que consiste simplemente en realizar un barrido por el primer nivel de la matriz de selección en el primer caso, y por toda la matriz en el segundo caso (siempre se trata de comprobar $N \times N$ casillas), y en caso de que haya algún candidato correspondiente a un mensaje de control, emparejarlo inmediatamente. De esta forma, se asegura que la máxima prioridad asignada en el enlace se mantiene a la hora de atravesar el *crossbar*. Esto es sencillo de implementar además porque en el MMR todos los mensajes de control emplean el canal virtual 0 de cada enlace, reservado para este uso. Y puesto que los mensajes de control ocurren con poca frecuencia, basta con un algoritmo sencillo y rápido, por ejemplo, WFA (*Wave Front Arbiter*) [162].

De esta forma, los flits de control sólo se verán retrasados por conflictos provocados por otros mensajes de control. Puesto que se suele considerar que dichos mensajes ocurren con una frecuencia bastante baja (algunos estudios generan un mensaje de control cada 33 milisegundos [173]), la probabilidad de conflictos entre ellos no es elevada. Y del mismo modo, a pesar de que solo se les dedique un único canal virtual, el efecto del *HOL-blocking* que se podría dar al emplear una sola FIFO (el *buffer* de ese canal virtual) no llega a afectar las prestaciones, puesto que la utilización del canal será baja. Concretamente, las simulaciones realizadas han demostrado que la utilización de los enlaces por mensajes de control se haya en torno al 0.01 %.

En el Capítulo siguiente, dedicado a Evaluación de Prestaciones, se analizará en detalle si lo aventurado en este apartado se cumple realmente.

5.2.4. Detalles de implementación

Una vez descrito el funcionamiento de los algoritmos propuestos, llega el turno de comprobar la viabilidad de su implementación dentro de las restricciones temporales que impone la arquitectura del Encaminador Multimedia. Hay que recordar que el tiempo de que se dispone para calcular una planificación válida es de *un ciclo de flit*, esto es, el tiempo que se tarda en transmitir un flit completo desde los *buffers* situados en los puertos de entrada, a través del *crossbar* y los puertos de salida. Esto es debido a que la planificación para un ciclo de flit se calcula en paralelo con la transmisión de flits del ciclo anterior.

Por ejemplo, en el caso de un Encaminador con enlaces a 1'24 Gbps, y un ancho de palabra interno de 16 bits, el ciclo de reloj interno es de 12'9 ns. Este reloj marca la frecuencia con que se debe de retransmitir cada palabra. Hay que determinar por tanto cuántos ciclos de reloj del Encaminador son necesarios para ejecutar los algoritmos de planificación. Esto dará el mínimo tamaño de flit necesario para soportar el cálculo concurrente de la planificación con la transmisión de flits. En este punto hay que recordar que en el Encaminador Multimedia los flits grandes no suponen un problema, pues las aplicaciones generan flujos de datos largos. Además, mediante flits grandes se amortiza mejor la sobrecarga de control introducida al transmitir cada flit, y la información del control de flujo. En el Capítulo dedicado al análisis de prestaciones se demostrará también que tamaños de flit más grandes no influyen negativamente en su latencia, sino más bien al contrario.

Para determinar el número de ciclos necesarios para el cálculo de la planificación, se van a desglosar los distintos bloques funcionales de una posible implementación hardware, y sobre ellos se va a estimar el retardo que introduce el cálculo de la planificación del conmutador.

Como referencia a la hora de estimar algunos de los retardos introducidos por cada módulo, se tomarán los cálculos presentados en [113]. En dicho trabajo, se describen con detalle los módulos involucrados en el diseño de un planificador orientado a soportar eficientemente tráfico *best-effort*. En este planificador hay algunos módulos básicos (como el de ordenación de valores) que se hayan también presentes en el diseño de los algoritmos de planificación presentados en las secciones anteriores.

El retardo de los módulos que no tienen ninguna equivalencia en el trabajo referenciado se estimará en base al número de niveles de puertas lógicas que se requieren para su implementación. Puesto que los datos tomados de [113] están calculados para tecnología CMOS de 0.25 μm, la tecnología de referencia para el cálculo del resto de estimaciones será la misma. Concretamente, el retardo típico de una puerta simple en dicha tecnología se ha estimado en 100 picosegundos (0'1 ns) [119]. Hay que señalar que la tecnología CMOS admite ya resoluciones de 0.18 μm [78]. El uso de esta tecnología permitiría obtener retardos inferiores a los que se han detallado en este punto.

Por tanto, el procedimiento a seguir para la estimación del retardo de los algoritmos de planificación propuestos será el siguiente. En primer lugar, se obtendrá un diagrama de módulos funcionales de la implementación. Posteriormente, cada uno de los módulos funcionales se detallará hasta llegar a un nivel de componentes básicos. Una vez determinados estos componentes, su retardo se estimará en base a los resultados presentados en el citado trabajo para componentes similares, y en base al número de niveles de puertas lógicas necesarios para su implementación, en los demás casos. Con esos resultados, ya se puede obtener la estimación del retardo global impuesto por los algoritmos de planificación.

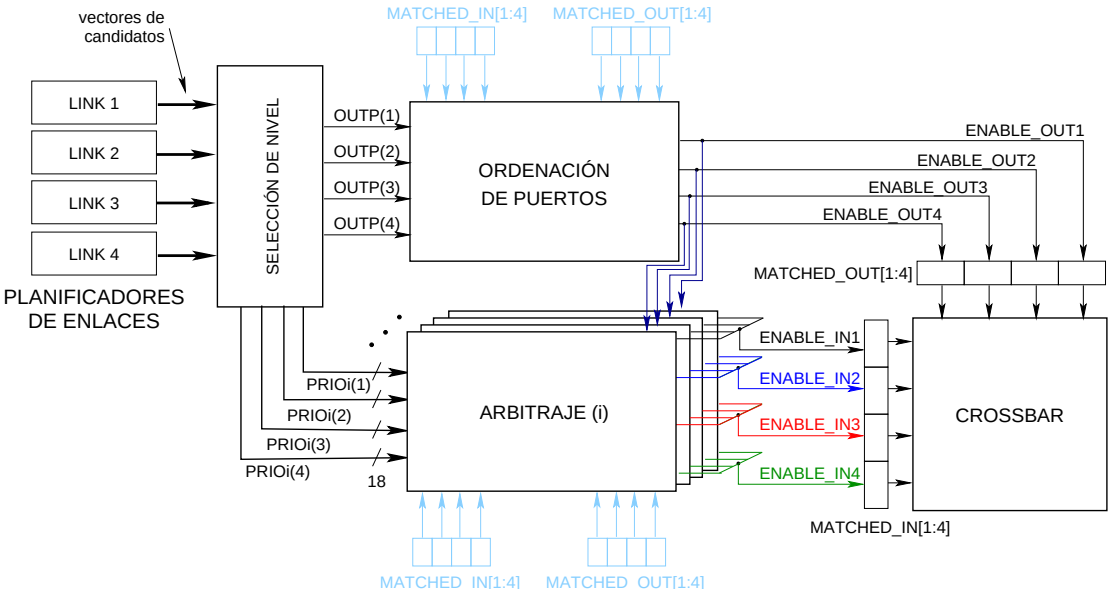


Figura 5.21: Implementación del algoritmo COA: Diagrama de bloques general.

El diseño de los algoritmos se ha hecho para un Encaminador Multimedia 4×4, con valores de prioridad de 16 bits. Los diagramas de bloques aquí descritos parten de que los planificadores de los enlaces físicos ya han generado los correspondientes vectores de candidatos.

Algoritmo COA

El algoritmo COA (*Candidate Order Arbiter*) considera las peticiones de cada nivel de candidatos sucesivamente, efectuando emparejamientos primero entre los candidatos de nivel 1 propuestos por cada enlace físico, para continuar con los de nivel 2, y así sucesivamente hasta que no se puedan realizar más emparejamientos.

Para efectuar un emparejamiento, se selecciona el puerto de salida con menor número de conflictos en la fase de *ordenación de puertos*. Esto conlleva hacer un recuento del número de peticiones dirigidas a cada puerto de salida, y ordenarlas de menor a mayor. El número de peticiones será 0 como mínimo, y 4 como máximo, esto es, hay 5 valores posibles distintos (0, 1, 2, 3 o 4). Por tanto, se necesitan 3 bits para codificar el número de peticiones por puerto, y la ordenación hay que hacerla sobre esos 3 bits. Entonces, de entre las peticiones dirigidas al puerto de salida seleccionado, se escoge la de mayor prioridad en la fase de *arbitraje*. Esta fase implica la ordenación de como máximo 4 valores de prioridad de 16 bits.

La Figura 5.21 muestra el diagrama de bloques general de la implementación desarrollada. En ella se aprecian dos grandes módulos, etiquetados como ORDENACIÓN DE PUERTOS y ARBITRAJE, cuya misión es calcular los puertos de salida y los puertos de entrada a emparejar, respectivamente. En la implementación desarrollada se ha optado por que estas dos fases se ejecuten en paralelo, de tal manera que cuando se seleccione el puerto de salida correspondiente, ya se disponga de la ordenación entre los puertos de entrada para dicha salida, y se pueda realizar el emparejamiento. Además, los módulos de ARBITRAJE están replicados para cada puerto de salida. La salida de cada módulo es del tipo triestado, de manera que únicamente se habilitará la del módulo correspondiente al puerto de salida elegido en cada iteración del algoritmo.

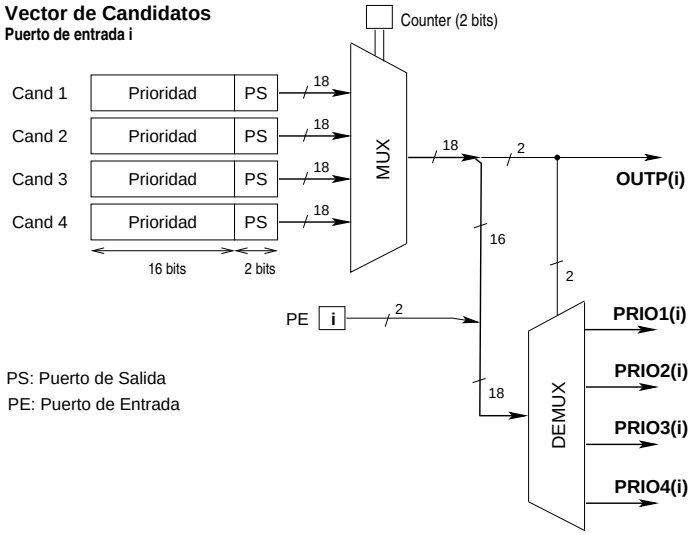


Figura 5.22: Implementación del algoritmo COA: Diagrama de bloques del módulo SELECCIÓN DE NIVEL, para el enlace de entrada i

Los emparejamientos que se van consiguiendo se van almacenando en un par de registros, MATCHED_IN y MATCHED_OUT, cada uno con tantos bits como puertos tenga el *crossbar*. Cada vez que se consigue un emparejamiento, el *crossbar* incorpora en una matriz de puntos de decisión interna los datos de puerto de entrada y salida correspondientes al nuevo emparejamiento. Cuando acabe el ciclo de flit en curso, la información de esa matriz de puntos de decisión servirá para reconfigurar el *crossbar*.

La información de entrada a los módulos comentados en párrafos anteriores se debe extraer a partir de los **vectores de candidatos**, generados por cada planificador de enlace. De esta tarea se encarga el **módulo SELECCIÓN DE NIVEL**. Concretamente, este módulo extrae los siguientes datos a partir de los vectores de candidatos:

- Los puertos de salida requeridos por cada candidato del nivel que esté siendo considerado en ese momento (OUTP(1) ... OUTP(4))
- Los valores de prioridad, aumentados con la codificación del puerto de entrada al que pertenecen, de los candidatos del nivel correspondiente hacia cada puerto de salida (PRIO1(i) ... PRIO4(i), para i = 1...4[1]).

Este módulo consiste básicamente en una lógica de multiplexación/demultiplexación sencilla, para dejar pasar la información de los candidatos correspondientes al nivel que se está considerando, hacia los módulos adecuados de las siguientes etapas. En la Figura 5.22 se representa el esquema básico del módulo, para un enlace físico. Este esquema se repetirá por cada puerto de entrada, para obtener la información de los candidatos pertenecientes a cada enlace físico.

Se puede observar como mediante un multiplexor se selecciona el candidato adecuado al nivel que se está considerando. Este multiplexor está gobernado por un contador que se incrementa cada vez que se cambia de nivel de candidatos. De esta manera se va dejando pasar la información de cada uno de los candidatos generados para el enlace. Ya se ha comentado que

[1] PRIOj(i) denota la información de un candidato del puerto de entrada i, que requiere el puerto de salida j

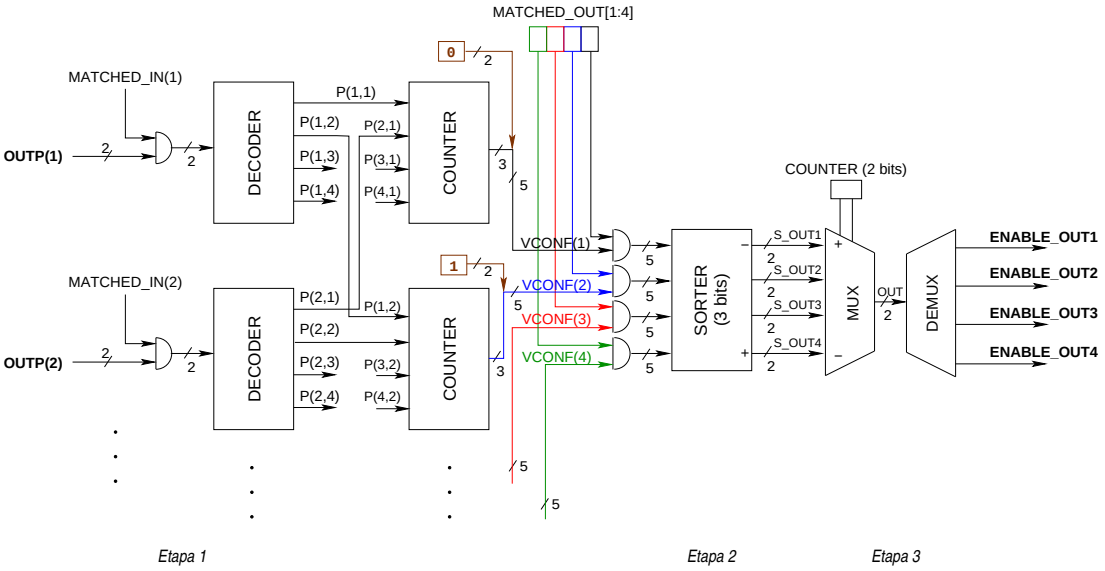


Figura 5.23: Implementación del algoritmo COA: Diagrama de bloques del módulo ORDENACIÓN DE PUERTOS.

esta información consiste en dos elementos: el puerto de salida requerido, que debe pasarse como entrada el módulo ORDENACIÓN DE PUERTOS (señal OUTP(i)), y la prioridad de dicho candidato, que debe de introducirse en el módulo ARBITRAJE(j) correspondiente al enlace de salida (j) solicitado por el candidato (señales PRIOj(i), j = 1...4). Por este motivo, es necesario introducir un demultiplexor para dirigir el valor de la prioridad (junto con el número de puerto de entrada al que corresponde) hacia el módulo adecuado. Este demultiplexor está gobernado por la codificación del puerto de salida requerido por el candidato.

Una vez que el módulo SELECCIÓN DE NIVEL ha extraído la información necesaria a partir de los candidatos generados por cada planificador de enlace, el módulo ORDENACIÓN DE PUERTOS selecciona un puerto de salida en función del número de peticiones por cada uno de ellos: en cada caso, escoge el puerto con menor número de conflictos. De manera simultánea, cada módulo ARBITRAJE realiza la ordenación de los valores de prioridad asociados a los candidatos destinados a cada uno de los puertos de salida. Así, cuando se haya obtenido un puerto de salida en la fase de ordenación, inmediatamente se habilita la selección del candidato más prioritario de entre todos los que van destinados a esa salida, en ese nivel.

La información de los puertos de entrada y salida ya emparejados se realimenta a estos módulos con el objeto de enmascarar las peticiones que los involucren. De esta manera se implementa el borrado de filas y columnas en la matriz de selección tras cada emparejamiento. Una vez se haya emparejado cada puerto de entrada, se puede deshabilitar el módulo SELECCIÓN DE NIVEL correspondiente, y de manera similar, cuando se haya emparejado un puerto de salida, se puede deshabilitar el módulo ARBITRAJE correspondiente.

En la Figura 5.23 se muestra un esquema de la implementación del **módulo ORDENACIÓN DE PUERTOS**. Por claridad, no se muestran todas las líneas de datos y todos los módulos. Sólo se muestra una parte para ilustrar la idea de su implementación.

Este módulo se puede describir como un conjunto de etapas. La *primera etapa* consiste en un decodificador más un contador (o codificador) por cada puerto de entrada. De esta manera, se obtiene una línea de petición (P(i, j)) a partir del puerto de salida especificado por los candidatos de cada enlace de entrada (OUTP(1) ... OUTP(4)), y se hace un recuento de

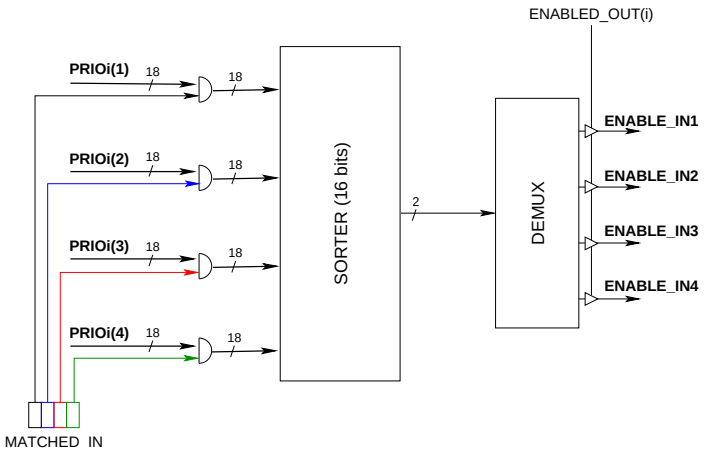


Figura 5.24: Implementación del algoritmo COA: Diagrama de bloques del módulo ARBITRAJE(i).

peticiones por cada puerto de salida. La entrada a esta etapa no es más que la información de los sucesivos candidatos de cada nivel, según avanza la ejecución del algoritmo. Las entradas a los decodificadores (es decir, las peticiones) se enmascaran en caso de que ese puerto de entrada ya esté emparejado (líneas desde MATCHED_IN(i)). Las líneas de peticiones desde cada puerto de entrada hacia cada puerto de salida aparecen etiquetadas en el esquema como P(i, j), indicando una petición desde la entrada i hacia la salida j.

Los contadores que aparecen a continuación en esta etapa llevan a cabo el recuento de peticiones realizadas hacia cada puerto de salida, esto es, en esta etapa se calcula el **vector de conflictos** (líneas etiquetadas como VCONF(1) ... VCONF(4)). El número de conflictos se codifica en 3 bits (hay 5 valores posibles: 0, 1, 2, 3 ó 4). Al igual que los decodificadores anteriores, los contadores también están replicados para cada puerto de salida. A la salida de cada contador, se le añade la codificación del puerto de salida al cuál corresponde el recuento, que supone dos bits más. Antes de pasar a la siguiente etapa, se enmascaran los valores del vector de conflictos que corresponden a puertos de salida ya emparejados.

En la *segunda etapa* se ordenan los valores anteriormente obtenidos, considerando sólo los 3 bits correspondientes al número de conflictos. La salida del módulo de ordenación son los números de puerto ordenados de menor a mayor, según su número de conflictos.

En la *tercera etapa* se obtiene el número del puerto de salida con menor número de conflictos en cada momento mediante un multiplexor. Este multiplexor está gobernado por un contador de 2 bits, que selecciona en cada instante el siguiente puerto en orden de conflictos. Dicho contador avanzará al ritmo en que se vayan emparejando puertos dentro de cada nivel de conflictos, y se pondrá a 0 cuando se cambie de nivel. Por último, el número de puerto de salida codificado se decodifica mediante un demultiplexor para habilitar la línea de salida correspondiente (ENABLE_OUT1 ... ENABLE_OUT4).

Hay que señalar que el orden relativo de los puertos de salida no se altera dentro de cada nivel de candidatos. Por tanto, dentro de cada nivel, no es necesario recalcular la ordenación, sino que mediante el contador de 2 bits se va seleccionando el siguiente puerto en orden de conflictos. Esto es, para calcular el primer emparejamiento dentro de un nivel hay que ejecutar las tres etapas de este módulo, mientras que para los siguientes emparejamientos dentro del mismo nivel, sólo es necesario emplear la tercera etapa.

La organización interna de cada **módulo ARBITRAJE** se muestra en la Figura 5.24. Como se ha indicado anteriormente, a cada uno de estos módulos se le facilitan los datos de prioridad

y de puerto de entrada de los candidatos dirigidos a un puerto de salida dado, pertenecientes al nivel que está siendo considerado. Esto es, la entrada a cada módulo correspondería a los datos almacenados en una fila de la matriz de selección. Las peticiones que involucran a los puertos de entrada ya emparejados se enmascaran mediante la información almacenada en el registro MATCHED_IN.

La parte central de cada módulo ARBITRAJE la compone un SORTER de 16 bits. Su misión es establecer cuál es la petición más prioritaria para ese puerto de salida. La salida del SORTER es simplemente la codificación del puerto de entrada correspondiente. Este número de puerto de entrada a emparejar se decodifica para generar la señal de activación del *crossbar* correspondiente, que se almacena en la posición correspondiente del registro MATCHED_IN para componer la planificación. La señal de salida de este módulo se activa únicamente cuando el puerto de salida correspondiente haya sido seleccionado para ser emparejado.

Es interesante señalar que, mientras se estén realizando emparejamientos dentro de un mismo nivel, el orden de los valores de prioridad de los candidatos dirigidos al mismo puerto de salida se mantiene. Esto es debido a que sólo hay una petición por cada puerto de entrada por cada nivel. Por tanto, cuando se consiga un emparejamiento, la actualización de la matriz de selección no enmascara ningún otro valor situado en la misma columna que el puerto de entrada emparejado, dentro del mismo nivel de candidatos (véase la Figura 5.15, en la página 107). Así, el más prioritario de entre los candidatos dirigidos hacia un puerto de salida determinado será siempre el mismo, mientras se permanezca en el mismo nivel de candidatos. Esto implica que únicamente es necesario calcular la ordenación de los valores de prioridad cuando se avanza al siguiente nivel de candidatos.

Los bloques funcionales de ordenación de valores empleados en el diseño descrito, tanto en la ordenación del número de conflictos por puerto de salida en el módulo ORDENACIÓN DE PUERTOS, como en la ordenación de los valores de prioridad dentro del módulo ARBITRAJE, no son módulos de ordenación convencionales. En ambos casos, cada entrada se compone de un número de bits mayor del número de bits considerado para la ordenación.

En el caso de la ordenación del número de conflictos, cada entrada (5 bits) se compone del recuento de conflictos (3 bits) más el identificador del puerto de salida al cuál pertenece dicho recuento (2 bits). La magnitud a ordenar es el número de conflictos, y el valor que interesa obtener a la salida de ese módulo no es el número de conflictos del puerto ganador, sino el puerto de salida al cuál corresponde.

Algo muy similar sucede dentro del módulo ARBITRAJE. Cada entrada se compone de 18 bits: 16 bits que codifican el valor de la prioridad, y 2 bits adicionales que indican el número de puerto de entrada al cuál corresponde ese valor. De nuevo, la ordenación debe hacerse en base a los valores de prioridad (16 bits), mientras que a la salida el valor que interesa obtener es el número de puerto asociado al valor ganador.

La implementación de estos módulos de ordenación implica modificar ligeramente la estructura de un módulo de ordenación convencional [113]. Cada elemento de comparación/conmutación estará diseñado tal y como se representa en la Figura 5.25. Los valores sobre los que se debe hacer la ordenación aparecen etiquetados como VALOR_i. Dichos valores ocupan n bits. Los valores asociados a la magnitud a comparar ocupan m bits.

Como se puede observar, el comparador sólo tiene en cuenta los n bits de la magnitud a ordenar. La idea también se puede expresar de la siguiente forma: el camino de datos del módulo de ordenación tiene $n + m$ bits de ancho, mientras que la lógica de control que lo gobierna sólo tiene en cuenta los n bits del valor a ordenar. En [113], se estima el retardo de un módulo similar, para $n = 10$ bits, en menos de 1 nanosegundo, empleando tecnología CMOS de 0.25 μm.

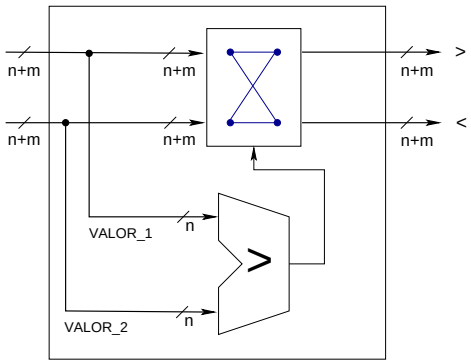


Figura 5.25: Implementación del algoritmo COA: Bloque de comparación modificado.

Por otro lado, el elemento mostrado en la Figura 5.25 proporciona a la salida los $n+m$ bits, pues serán la entrada de la siguiente etapa en la red de ordenación. A la salida de la última etapa, se puede prescindir de los n bits correspondientes al valor a ordenar.

Una vez descritos los módulos empleados, se puede estimar el tiempo que se tarda en obtener una planificación. Puesto que los módulos ORDENACIÓN DE PUERTOS y ARBITRAJE funcionan en paralelo, el retardo del cálculo de cada emparejamiento será igual al retardo experimentado por el más lento de los dos. A este valor habría que añadir el retardo introducido por el módulo SELECCIÓN DE NIVEL al cambiar de nivel de candidatos.

En el retardo del módulo SELECCIÓN DE NIVEL intervienen el retardo del contador, del multiplexor y del demultiplexor. Estos módulos se pueden implementar empleando entre 2 y 3 niveles de puertas lógicas [106, 105]. Por tanto, si el retardo de una puerta lógica simple en tecnología CMOS de 0.25 μm es de 100 ps [119], se puede acotar el retardo de cada componente en medio nanosegundo. Esto da un retardo global del módulo de 1'5 nanosegundos. Este tiempo es el que se tardaría en extraer la información de cada vector de candidatos. Este proceso se debe hacer siempre que se cambie de nivel de candidatos.

En lo que respecta al módulo ORDENACIÓN DE PUERTOS, su retardo total es la suma de los retardos introducidos por los distintos elementos que lo componen: decodificador, contador, SORTER de 4 valores de 3 bits, multiplexor y demultiplexor.

Al igual que en el caso anterior, los módulos decodificador, contador, multiplexor y demultiplexor también se pueden implementar empleando entre 2 y 3 niveles de puertas lógicas [106, 105]. Por tanto, se puede asumir que estos módulos también introducen retardos inferiores a medio nanosegundo.

En cuanto al bloque SORTER, debe obtener la ordenación de los 4 valores introducidos como entradas. Por tanto, es necesario utilizar 3 etapas de elementos comparadores (ver la Figura 5.8, página 100). El retardo de cada uno de estos elementos se puede estimar en 1 nanosegundo [113]. Esto da un retardo total del bloque SORTER de 3 ns.

De todo lo anterior, se puede concluir que el retardo total desde la entrada del módulo ORDENACIÓN DE PUERTOS se puede estimar en 5 ns (0'5 ns $\times$ 4 + 3 ns). Este retardo se produce en el peor de los casos, esto es, cuando se cambia de nivel de candidatos. Para emparejamientos adicionales dentro de un mismo nivel, sólo se emplean el multiplexor y el demultiplexor que activan la correspondiente salida del módulo, lo cuál supone un retardo de 1 ns.

Por último, queda analizar el retardo que introduce el módulo ARBITRAJE. La principal contribución al retardo del módulo viene dada por el SORTER de 4 valores de 16 bits. Para

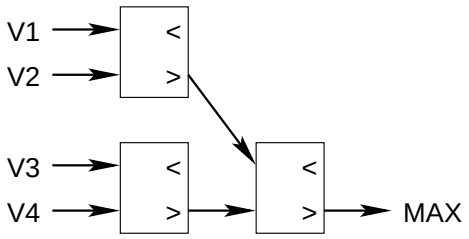


Figura 5.26: Implementación del algoritmo COA: Obtención del máximo de entre 4 valores.

ordenar 4 valores es necesario emplear una red de ordenación con 3 etapas (véase la Figura 5.8). Pero como sólo es necesario obtener el valor más prioritario, dos etapas bastarían (véase la Figura 5.26). Cada elemento comparador debe actuar sobre valores de 16 bits. A partir de la estimación de 1 nanosegundo presentada en [113] para estos elementos, donde se consideran valores de 10 bits, se puede estimar al alza el retardo de los elementos necesarios en 2 nanosegundos. Esto da un retardo global de este módulo de ordenación de 4 nanosegundos.

Al valor anterior, hay que añadir el retardo introducido por el demultiplexor, que será de medio nanosegundo, según se ha analizado para los casos anteriores.

Por tanto, el retardo del módulo ARBITRAJE será de 4'5 ns en el peor de los casos, esto es, cuando se cambia de nivel de candidatos, pues entonces es cuando hay que ordenar los valores de prioridad. Mientras no se cambie de nivel de candidatos, el retardo de este módulo es prácticamente nulo, pues los valores ya están ordenados, y el demultiplexor ya ha activado la señal correspondiente. Únicamente quedaría por habilitar la salida correspondiente, mediante la señal ENABLE_OUT(i).

De todo lo anterior, se concluye que el retardo en que se incurre cada vez que se obtiene un emparejamiento, en el peor de los casos (que haya un cambio en el nivel de candidatos considerado), es el siguiente:

Módulo		**Retardo (ns)**
SELECCIÓN DE NIVEL	$\rightarrow$	1,5
ORDENACIÓN DE PUERTOS	$\rightarrow$	5
ARBITRAJE	$\rightarrow$	4,5

Puesto que las módulos ORDENACIÓN DE PUERTOS y ARBITRAJE se ejecutan en paralelo, en el cómputo del retardo global sólo interviene el módulo más lento, que es el de ORDENACIÓN DE PUERTOS, con 5 ns. A esto hay que añadir el retardo del módulo SELECCIÓN DE NIVEL, pues se encarga de preparar la información de entrada a los otros dos módulos.

Ahora bien, es posible ocultar el retardo de este módulo si se simultanea la extracción de la información de los vectores de candidatos con el procesamiento de la misma. Esto es, mientras los módulos ORDENACIÓN DE PUERTOS y ARBITRAJE calculan emparejamientos entre los candidatos de un nivel, el módulo SELECCIÓN DE NIVEL puede estar extrayendo la información de los candidatos del siguiente nivel. De esta forma, cuando se vaya a cambiar de nivel, la información del siguiente conjunto de candidatos ya habrá sido extraída por el módulo SELECCIÓN DE NIVEL. Evidentemente, esto sólo podrá hacerse a partir del segundo nivel de candidatos.

La idea se puede ilustrar tal y como se muestra en la Figura 5.27, donde se representan los solapamientos entre los distintos módulos, para 4 niveles de candidatos, y considerando que en cada nivel de candidatos sólo se consigue un emparejamiento.

Así, el retardo introducido por el módulo SELECCIÓN DE NIVEL sólo hay que contabilizarlo al inicio de la ejecución del algoritmo.

El número máximo de emparejamientos en el caso de un Encaminador 4×4 es precisamente 4. Y como mucho se puede cambiar 4 veces de nivel. Tras el retardo inicial introducido por el módulo SELECCIÓN DE NIVEL, que supone 1'5 ns, se puede obtener un emparejamiento cada 5 ns. Por tanto, *el retardo máximo que introduce el cálculo de un emparejamiento completo es de 21'5 ns* ($5ns \times 4 + 1'5ns$).

Para el caso concreto de un Encaminador con enlaces a 1.24 Gbps, y ancho de palabra interno de 16 bits, donde su ciclo de reloj interno es de 12'9 ns, *se puede obtener una planificación del conmutador cada 2 ciclos de reloj.*

Hay que señalar que tanto los borrados de la matriz de selección, como el control del cambio de nivel cuando no haya más peticiones pendientes en un nivel, así como el control del final de la planificación (cuando se hayan recorrido todos los niveles de candidatos, o al haber emparejado todos los puertos, o bien al no haber más peticiones pendientes), se puede realizar de forma simple mediante sencillas operaciones combinacionales, por tanto no introducen retardos significativos. Además, en los cálculos realizados existe margen suficiente como para asumirlos.

Algoritmo CCA

El algoritmo CCA (*Candidate Conflict Arbiter*) es muy similar al algoritmo COA. Únicamente varía el criterio empleado para la ordenación de los puertos. En el algoritmo COA, se organizan los candidatos según su nivel, y por cada nivel se escogía aquel puerto con menor número de puertos. Los niveles se recorren desde el de más prioridad al de menor prioridad.

En el algoritmo CCA también se escoge el puerto de salida con menor número de conflictos, pero esta vez se consideran todos los candidatos de los distintos niveles a la vez. Es decir, el vector de conflictos almacena el número de peticiones dirigidas hacia cada puerto de salida por cada puerto de entrada, teniendo en cuenta los candidatos de todos los niveles. En el algoritmo anterior, por el contrario, el vector de candidatos se calculaba considerando únicamente a los candidatos de cada nivel.

Puesto que ambos algoritmos son similares, comparten muchos detalles en su implementación. La organización global del diseño del algoritmo CCA es la misma que la mostrada en la Figura 5.21 para el algoritmo COA. Sólo hay que modificar ligeramente el comportamiento de algunos módulos. Esto es lo que se va a detallar a continuación.

En primer lugar, el comportamiento del **módulo SELECCIÓN DE NIVEL** será ligeramente diferente. En el caso de COA, este módulo va alimentando a los módulos ORDENACIÓN DE PUERTOS y ARBITRAJE con la información extraída de los candidatos nivel por nivel, según va avanzando la ejecución del algoritmo. En el caso de CCA, todos los candidatos de todos los niveles se han de considerar a la vez. Por tanto, el módulo de selección de nivel alimentará los puertos de salida solicitados por los candidatos de los distintos niveles al módulo ORDENACIÓN DE PUERTOS previamente a la ejecución del algoritmo, para que se puedan contabilizar las peticiones de cada candidato hacia cada puerto de salida. Además, las prioridades de cada candidato se pasan al módulo ARBITRAJE asociado al puerto de salida correspondiente.

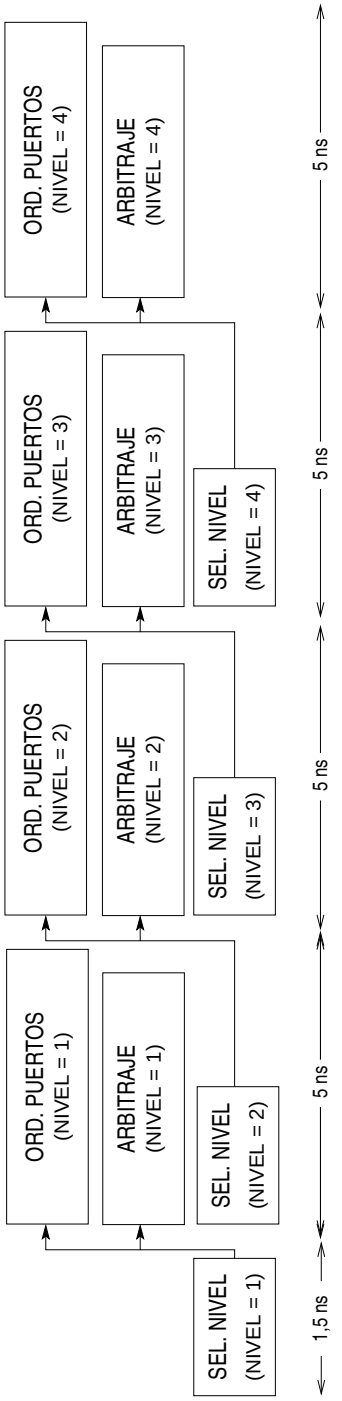


Figura 5.27: Implementación del algoritmo COA: Solapamiento de la ejecución de los distintos módulos.

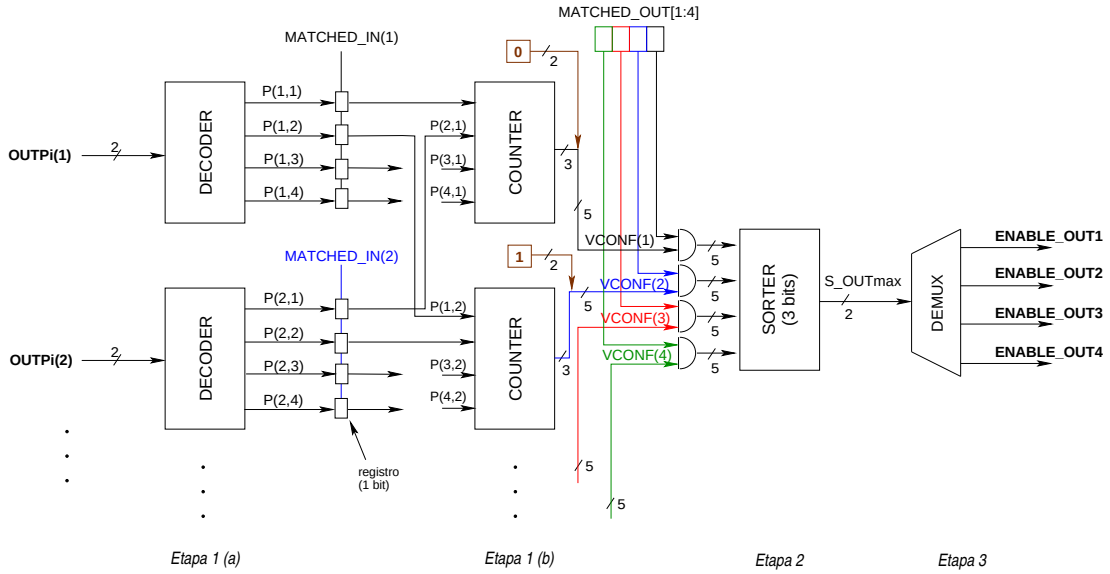


Figura 5.28: Implementación del algoritmo CCA: Diagrama de bloques del módulo ORDENACIÓN DE PUERTOS.

Esto es, el módulo SELECCIÓN DE NIVEL realiza la misma función que antes, pero con una frecuencia diferente. Antes el paso de un nivel a otro estaba gobernado por el hecho de que no quedasen más peticiones en el nivel en curso (aunque tal y como se señaló anteriormente, se puede adelantar ese cambio de nivel con el objeto de solapar el funcionamiento del módulo SELECCIÓN DE NIVEL con los otros dos). Ahora, el cambio de nivel vendrá impuesto por un lado, por la frecuencia con que se puedan ir decodificando los puertos de salida en el módulo ORDENACIÓN DE PUERTOS, y por otro por el propio retardo interno del módulo SELECCIÓN DE NIVEL.

El **módulo ORDENACIÓN DE PUERTOS** verá ligeramente modificada su estructura. En la Figura 5.28 se muestran las modificaciones que hay que introducir. Por un lado, se han introducido unos registros donde se van almacenando las peticiones hacia los distintos puertos de salida (líneas P(i,j)), que se van obteniendo sucesivamente a partir de la información suministrada por el módulo de SELECCIÓN DE NIVEL. Estos registros se muestran entre las etapas 1(a) y 1(b) en la Figura 5.28.

Cuando se han decodificado las peticiones de los candidatos de todos los niveles, es cuando se activan los contadores que generan los componentes del vector de conflictos (etapa 1(b) en la Figura 5.28). Seguidamente, estos valores se ordenan, y se calcula aquél con menor número de conflictos, de manera análoga a como se hacía en el algoritmo COA. Ahora bien, en el algoritmo CCA, por cada puerto de entrada puede haber más de una petición, ya que se consideran a la vez todos los candidatos. Por tanto, cada vez que se realiza un emparejamiento, y se borra una columna de la matriz de selección, puede descartarse alguna otra petición del puerto de entrada emparejado. Esto puede dar lugar a que el número de conflictos de otros puertos de salida se modifique, variando el orden relativo de los distintos puertos.

Por tanto, tras cada emparejamiento, es necesario volver a contar las peticiones, y repetir la ordenación. Una ventaja es que sólo es necesario obtener el puerto con menor número conflictos, no la ordenación de los 4 posibles valores (esto es, el SORTER sólo tiene una salida, y ya no es necesario el multiplexor). Pero por contra, para cada calcular cada emparejamiento,

ahora siempre hay que ejecutar las etapas 1(b), 2 y 3.

La única modificación necesaria dentro del **módulo ARBITRAJE** respecto al diseño mostrado en a Figura 5.24 para el algoritmo COA consiste en que hay que almacenar en registros los distintos valores de prioridad que genere el módulo SELECCIÓN DE NIVEL al ir recorriendo los distintos niveles de candidatos. Para ello, sólo es necesario añadir un registro de 18 bits en cada línea de entrada al módulo (PRIOi(1) ... PRIOi(4)). De ahí el SORTER leerá los valores de prioridad para efectuar la ordenación, así como el número de puerto de entrada del cuál proviene el candidato correspondiente. Por otro lado, en el algoritmo CCA se consideran a la vez todos los candidatos generados por cada puerto de entrada. Por tanto, si se efectúa un emparejamiento que involucre a ese puerto de entrada, puede descartarse el candidato más prioritario destinado a una salida dada, distinta de la emparejada. Como consecuencia, siempre que se consiga un emparejamiento, hay que volver a ordenar los valores de prioridad correspondientes al resto de puertos de salida, para volver a obtener el candidato más prioritario.

Una vez puestas de manifiesto las diferencias de la implementación de este algoritmo con respecto al anterior, se va a estimar el retardo impuesto por cada módulo.

El módulo SELECCIÓN DE NIVEL incurre en el mismo retardo que en el caso anterior, 1'5 ns, pues únicamente varía la frecuencia a la que debe funcionar. Esta frecuencia debe considerar el retardo intrínseco del módulo, este es, como máximo, se extraerá la información de un nivel de candidatos cada 1'5 ns.

Respecto al módulo ORDENACIÓN DE PUERTOS, se compone de algunos bloques comunes con respecto a su análogo para el algoritmo COA: el contador de peticiones y el demultiplexor. El retardo de cada uno de estos módulos se estimó en medio nanosegundo.

El bloque SORTER, por el contrario, es ligeramente diferente: ahora no se trata de ordenar los 4 valores introducidos en sus entradas, sino que en este caso basta con encontrar el máximo de entre los 4. Esto permite implementar el bloque con sólo 2 etapas de elementos comparadores/conmutadores (véase la Figura 5.26), lo que da un retardo de 2 ns para este bloque.

Los cálculos anteriores se refieren a las etapas denominadas 1(b), 2 y 3 en la Figura 5.28. Estas tres etapas son las que tendrían que ser repetidas para obtener cada emparejamiento. Por tanto, el tiempo necesario para seleccionar un puerto de salida para emparejar es de 3 ns.

La etapa 1(a) realiza la decodificación de los puertos de salida solicitados por los candidatos de cada nivel, sucesivamente. Ya se indicó que dicha decodificación se puede realizar en 0'5 ns. Sin embargo, la entrada a esta etapa es la salida del módulo SELECCIÓN DE NIVEL, cuyo retardo interno es de 1'5 ns. Por tanto, el retardo de la etapa viene condicionado por la velocidad a la que dicho módulo puede generar las entradas a los decodificadores.

El módulo ARBITRAJE introduce siempre el mismo retardo que el módulo correspondiente del algoritmo COA en el peor de los casos (cuando le toca reordenar los valores al cambiar de nivel de candidatos), esto es, 4'5 ns. A este valor habría que añadir el retardo impuesto por los registros introducidos en las entradas. Sin embargo, este retardo sólo hay que tenerlo en cuenta al inicio de la ejecución del algoritmo, que es cuando el módulo SELECCIÓN DE NIVEL está estableciendo sus valores. Por tanto, este retardo, de décimas de nanosegundo, también queda oculto por el retardo del módulo SELECCIÓN DE NIVEL.

Resumiendo todo lo anterior, se concluye que el retardo en que incurre cada módulo es el siguiente:

Módulo		Retardo (ns)
SELECCIÓN DE NIVEL	→	1,5
ORDENACIÓN DE PUERTOS (etapa 1(a))	→	0,5
ORDENACIÓN DE PUERTOS (resto)	→	3
ARBITRAJE (registro a la entrada)	→	0,5
ARBITRAJE (resto)	→	4,5

En la Figura 5.29 se ilustra la manera en que se puede solapar la ejecución de los distintos módulos. Al igual que se hizo para el algoritmo COA, se muestra un ejemplo con 4 candidatos, y 4 emparejamientos.

Se pueden distinguir dos fases diferenciadas: una fase inicial donde se extraen los datos necesarios de los vectores de candidatos, y se preparan para la ejecución del algoritmo, y otra fase donde se van calculando los emparejamientos. En la primera fase interviene el módulo SELECCIÓN DE NIVEL, la etapa 1(a) del módulo ORDENACIÓN DE PUERTOS, donde se decodifican las peticiones de puertos de salida, y los registros a la entrada del módulo ARBITRAJE. En la segunda fase, intervienen el resto de etapas del módulo ORDENACIÓN DE PUERTOS, y el resto del módulo ARBITRAJE. Hay que señalar que en la primera fase, la información de los distintos niveles de candidatos se va recorriendo en orden inverso a su prioridad, esto es, se obtiene en primer lugar la información de los candidatos de nivel 4, luego la de los de nivel 3, etc. De esta forma, en el caso de que en un enlace de entrada más de un candidato solicite el mismo enlace de salida, sólo se registrará la prioridad del de menor nivel, esto es, el más prioritario.

A partir de la Figura 5.29 se puede deducir que *el retardo del cálculo de la planificación del conmutador mediante el algoritmo CCA, en el peor de los casos es de 24'5 ns* ($1'5ns \times 4 + 0'5ns + 4'5ns \times 4$). Se trata del retardo máximo, el cuál se produce cuando se consigue el máximo número de emparejamientos, 4 en este caso.

Si el Encaminador tiene enlaces de 1.24 Gbps, y ancho de palabra interno de 16 bits, esto es, un ciclo de reloj interno de 12'9 ns, el tiempo invertido en la ejecución del algoritmo de planificación del conmutador supone algo más de 2 ciclos de reloj. Por tanto, *se obtendrá una nueva planificación del conmutador cada 3 ciclos de reloj.*

En la sección 5.1.3, se estimó el tiempo necesario para el cálculo de la planificación de los enlaces en 8 ciclos de reloj para la configuración considerada. Añadiendo los 2 ó 3 ciclos que se tarda en obtener una planificación del conmutador, según el algoritmo empleado, se obtiene que *son necesarios 10 u 11 ciclos de reloj del encaminador para obtener una planificación válida.* De ahí de deduce que el tamaño del flit debe ser de al menos 10 u 11 phits, esto es, 160-176 bits.

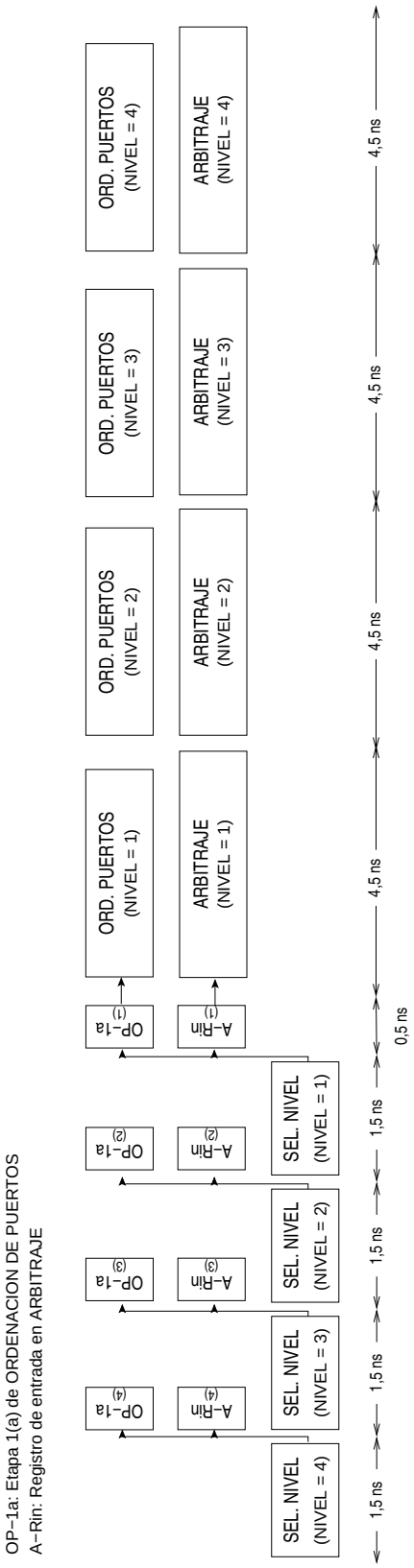


Figura 5.29: Implementación del algoritmo CCA: Solapamiento de la ejecución de los distintos módulos.

Capítulo 6

Evaluación de Prestaciones

En Capítulos anteriores han quedado expuestos los problemas en el soporte a la transmisión de datos multimedia existentes en la actualidad. Asimismo, se ha propuesto una nueva arquitectura de Encaminador orientada a resolver dichos problemas, detallando los algoritmos de planificación del tráfico que se emplearán en dicha arquitectura. Es el turno de comprobar si las prestaciones de la nueva arquitectura, junto con los algoritmos de planificación propuestos, cumple con los objetivos perseguidos para su diseño.

6.1. Método de evaluación

Existen diferentes métodos para la evaluación de las prestaciones de un sistema [79]. Por un lado, se pueden efectuar **mediciones reales** sobre un sistema existente. En este caso es necesario disponer de al menos un prototipo del sistema. Los principales inconvenientes de este método son la necesidad de un esfuerzo previo de desarrollo, y la falta de versatilidad, pues para probar diferentes opciones de diseño hay que elaborar distintos prototipos. En el caso del Encaminador Multimedia, donde es necesario evaluar un buen puñado de opciones de diseño, resulta inviable este método.

Otra posibilidad es elaborar un **modelo analítico** del sistema. Se trata de una alternativa económica, que proporciona resultados de forma inmediata. Sin embargo, su utilización depende del nivel de detalle requerido para los resultados. Además, para que el modelo sea susceptible de una evaluación analítica, se debe simplificar notablemente el modelo del sistema, con la consiguiente pérdida de precisión de los resultados. Puesto que en la evaluación del Encaminador Multimedia es necesario considerar un gran número de detalles, una evaluación analítica no resulta viable.

Por último, la **simulación** es un método utilizado ampliamente en muchos campos de investigación. Mediante la simulación por ordenador se puede evaluar con precisión cualquier sistema que no pueda ser evaluado analíticamente debido a su complejidad. Además, es posible considerar distintos niveles de detalle en el sistema. Este método será el empleado para evaluar las prestaciones de la arquitectura de encaminador propuesta en esta Tesis Doctoral. El principal inconveniente de este método es el elevado tiempo de ejecución necesario para la simulación del sistema.

Una vez realizado el simulador, hay que introducir la carga de prueba con el objeto de obtener resultados. En concreto, cuando se trata de evaluar elementos de interconexión la carga consiste en el tráfico inyectado en la red. Existen diversas aproximaciones para modelar

la carga de un sistema. La aproximación más extendida es el empleo de **cargas sintéticas**, donde el tráfico inyectado se genera aleatoriamente, siguiendo determinados patrones tanto de distribución de los destinos para los mensajes, como de tamaño y frecuencia de los mismos. De esta manera, la carga se genera de forma sencilla y rápida, aunque no siempre modela perfectamente el tráfico real que circula por la red en el sistema real.

También existe la posibilidad de emplear **trazas** obtenidas a partir de sistemas reales. La idea es almacenar la información acerca del tráfico medido en un sistema real en los denominados *ficheros de traza*. En estos ficheros se almacenan datos tales como tamaño, origen, destino o tiempo de generación de los mensajes, aunque el formato concreto depende de la aplicación específica. El problema de este método es que el tráfico registrado en la traza depende de muchos factores del sistema donde se ha obtenido, y no siempre es extrapolable a otros sistemas donde varía alguno de dichos factores.

Por último, se puede emplear **carga real de aplicaciones**, en los denominados *simuladores dirigidos por ejecución*. Este tipo de simuladores lo que hacen es ejecutar realmente una aplicación sobre el sistema simulado, lo que hace su ejecución especialmente costosa en recursos computacionales (se suele simular la ejecución completa de una aplicación paralela de complejidad media-alta sobre el sistema simulado, y todo ello se ejecuta típicamente en un monoprocesador). Por tanto, cuando se trata de evaluar un conjunto de parámetros de tamaño moderado, esta aproximación resulta inviable.

La evaluación de prestaciones del Encaminador Multimedia MMR, y de los algoritmos de planificación desarrollados para él, se va a llevar a cabo mediante simulación, pues es la única técnica capaz de ofrecer tanto el nivel de detalle como la versatilidad necesaria para el estudio. Por otro lado, la carga con que se alimentará el simulador será mixta, pues para algunos tipos de tráfico se empleará una carga generada sintéticamente, mientras que en otros se emplearán trazas obtenidas de aplicaciones reales. Todo ello será descrito con detalle en las siguientes secciones.

6.2. Herramienta de simulación: `mmrsim`

Con el objeto de evaluar la arquitectura de encaminador propuesta, se ha desarrollado una herramienta de simulación que captura el funcionamiento del mismo con un elevado nivel de detalle. La herramienta se ha desarrollado en el lenguaje de programación C++. Se han aprovechado las capacidades de orientación a objetos que proporciona este lenguaje para implementar un simulador modular, que admite configurar de forma simple las diferentes opciones de diseño, en especial los algoritmos de planificación de enlaces y conmutador.

El objetivo fundamental es validar la arquitectura de encaminador propuesta, y comprobar si los algoritmos de planificación desarrollados son capaces de cumplir efectivamente con las prestaciones requeridas para los diversos tipos de tráfico. Por ello, el simulador implementa con gran nivel de detalle un único encaminador, junto con las tarjetas de interfaz de red conectadas a cada puerto de entrada. Esta configuración emplea el Encaminador como "encaminador de grupo de trabajo", donde a sus puertos se conectan directamente los *hosts* [53, 76].

Entre el encaminador y cada tarjeta de red se implementa el control de flujo basado en créditos. Hay que recordar que la elección de este esquema de control de flujo, frente al procedimiento *stop-and-go* comúnmente empleado en redes de multicomputadores, viene motivada por las siquientes razones:

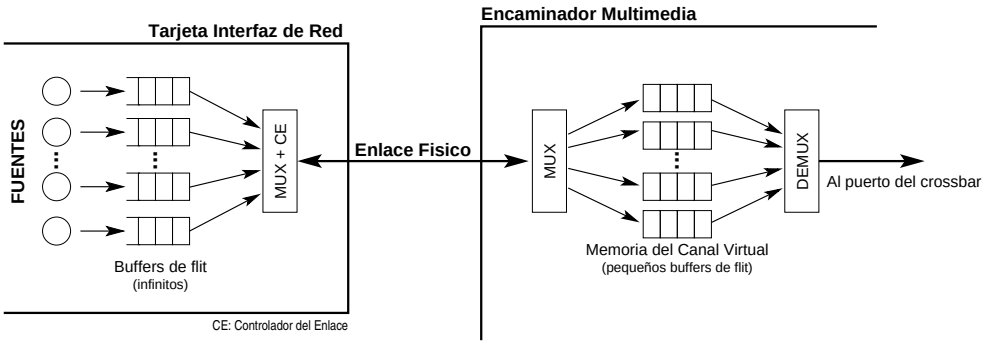


Figura 6.1: Representación conceptual de una entrada al MMR.

- En el caso de emplear control de flujo de tipo *stop-and-go*, el retardo de la información de control de flujo a través de los enlaces haría necesario el empleo de *buffers* de tamaño elevado por cada canal virtual. Esto, dado el elevado número de canales virtuales que soporta el MMR, elevaría de forma considerable su complejidad hardware y su tamaño en área de silicio.

- El control de flujo basado en créditos requiere de una mayor sobrecarga de control. Sin embargo, puesto que los flujos multimedia son muy largos, el tamaño de la unidad de control de flujo o flit se puede hacer lo suficientemente grande como para amortizarla.

- Se puede argumentar que hasta que la información de control de flujo (el crédito) llega al emisor, los enlaces permanecen inutilizados perjudicando la tasa de utilización de los mismos. Sin embargo, el control de flujo se establece por cada canal virtual, y no por cada enlace físico. Por tanto, al existir un gran número de ellos, la probabilidad de que haya al menos uno preparado para transmitir (es decir, con créditos disponibles) es muy elevada, especialmente con cargas medias-altas.

En la Figura 6.1 se representa la configuración conceptual del modelo simulado, en lo que respecta a las entradas del MMR. Las fuentes de tráfico cuando generan sus flits, los depositan en los *buffers* implementados en la tarjeta de red correspondiente. La tarjeta de red implementa tantos *buffers* como canales virtuales se definan en cada enlace físico del MMR. Estos *buffers* se consideran de tamaño infinito, pues en caso de necesidad se puede recurrir a la memoria del *host*. De esta manera, se facilita el empleo de *buffers* pequeños y compactos en el MMR.

El Controlador del Enlace situado en la tarjeta de red sigue un criterio cíclico entre los canales virtuales con flits listos para transmitir (es decir, con créditos disponibles), a la hora de hacer llegar un flit hasta el *buffer* correspondiente en el MMR. Este simple esquema se ha comprobado que es suficiente para enviar flits al MMR y garantizar la QoS, al menos cuando la red se compone de un único Encaminador. Esto se debe a que cuando el algoritmo de planificación garantiza QoS, el NIC simplemente adapta el envío de flits a las necesidades del encaminador, teniendo en cuenta la capacidad de los *buffers* del MMR (sólo para unos pocos flits) y el control de flujo empleado. Para el caso de una red compuesta por más de un Encaminador, posiblemente se deban analizar otras posibilidades más sofisticadas para el Controlador del Enlace.

La simulación está conducida por eventos. La transmisión de datos se simula al nivel de flit. El funcionamiento del encaminador se simula sobre la base de ciclos de flit, aunque la base

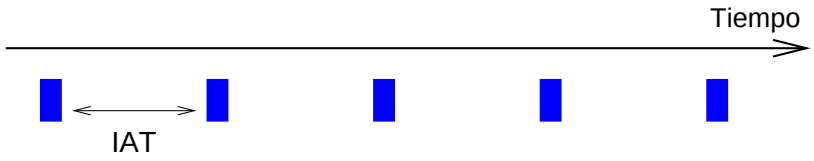


Figura 6.2: Fuente de tráfico CBR.

de tiempos está al nivel de phit. Esto es, la unidad de tiempo del simulador es el tiempo equivalente a transmitir un phit, es decir, un ciclo de reloj del encaminador. Tanto el tamaño de flit y de phit, como la frecuencia de funcionamiento del encaminador son parámetros de entrada del simulador.

Antes de describir los parámetros que configuran cada simulación, y puesto que algunos de ellos están relacionados con detalles de los generadores de tráfico utilizados, se van a describir dichos generadores de tráfico.

6.2.1. Descripción de los generadores de tráfico

El Encaminador Multimedia está ideado para soportar tanto conexiones de tráfico multimedia, que requieren de garantías de QoS, como otros tipos de tráfico no multimedia, en concreto mensajes de control, y tráfico de datos convencional *best-effort*. Por ello, ha sido necesario desarrollar generadores de tráfico para cada uno de estos tipos. En concreto, para el tráfico multimedia se han desarrollado generadores tanto de tráfico de tasa constante (CBR), como de tasa variable (VBR).

Fuentes CBR

Las fuentes de tráfico CBR han sido las más sencillas. Simplemente han de generar un flit cada cierto tiempo, estando éste determinado por el ancho de banda de la conexión. Al tiempo de separación entre flits, fijo para cada conexión de este tipo, se le denomina **IAT** (de *Inter-Arrival Time*, o tiempo entre llegadas), y se define como la inversa del ancho de banda de la conexión. En la Figura 6.2 se ilustra el comportamiento de una fuente de este tipo.

Para poder evaluar las prestaciones recibidas por este tipo de conexiones, y comprobar si realmente son acordes con las garantías necesarias para cada tipo, en las simulaciones que involucran tráfico CBR se han empleado mezclas aleatorias de 3 tipos de conexiones CBR: de 64 Kbps, de 1.5 Mbps y de 55 Mbps. De esta manera, se pueden analizar las prestaciones obtenidas por conexiones con requerimientos de ancho de banda bajos, medios y altos. Las primeras son representativas de conexiones telefónicas, las segundas pueden representar a conexiones de vídeo sin comprimir, mientras que las últimas son representativas de transmisiones de vídeo de alta definición.

Fuentes VBR

Las fuentes de tráfico variable se han implementado como fuentes de vídeo MPEG-2, por ser éste un ejemplo típico de tráfico multimedia.

El estándar de codificación de vídeo MPEG-2 [12] codifica los flujos de vídeo como una secuencia de distintos tipos de imágenes o *frames*, de tipo I, P y B, ordenadas según un patrón repetitivo y prefijado, llamado GOP (Group Of Pictures). El GOP empleado es IBBPBBPBBPBBPBB. Cada tipo de imagen tiene una finalidad diferente. Las imágenes de tipo I contienen toda la información de la imagen en ellas mismas, son autocontenidas Las imágenes de tipo P necesitan la información de las imágenes I para poder ser decodificadas. Por último, las imágenes de tipo B requieren la información de las imágenes de tipo I anteriores, y las de tipo P anteriores y posteriores para poder ser decodificadas. El ancho de banda requerido por cada tipo de imagen es diferente. Las imágenes de tipo I son las que consumen un mayor ancho de banda, y las de tipo B, las que menos.

Cada 33 milisegundos, se inyecta una imagen compuesta por un número determinado de flits. Este número se extrae de ficheros de traza correspondientes a películas de vídeo MPEG-2 reales. Dichos ficheros almacenan el tamaño en bits de cada imagen de la secuencia. Típicamente, cada imagen tiene que ser transmitida como una secuencia de flits. Estos flits, que componen cada imagen de vídeo, se pueden inyectar de distintas maneras a lo largo de los 33 milisegundos que transcurren entre la inyección de dos imágenes consecutivas.

Una posibilidad es distribuir la inyección de los flits de forma equiespaciada a lo largo de los 33 milisegundos de separación entre imágenes consecutivas. En la Figura 6.3 se muestra la inyección de dos tramas consecutivas, una compuesta de 6 flits, y otra de tres. Este método de inyección provoca que el tiempo de separación entre la inyección de dos flits consecutivos (el IAT) varíe de imagen a imagen. Por ejemplo, en la Figura 6.3 se puede comprobar como el IAT para la primera trama (IAT_1) es la mitad que el de la segunda (IAT_2), al tener el doble de flits. A este modelo se le ha denominado como **modelo de inyección SR** (por *Smooth Rate*, o tasa suavizada) [27].

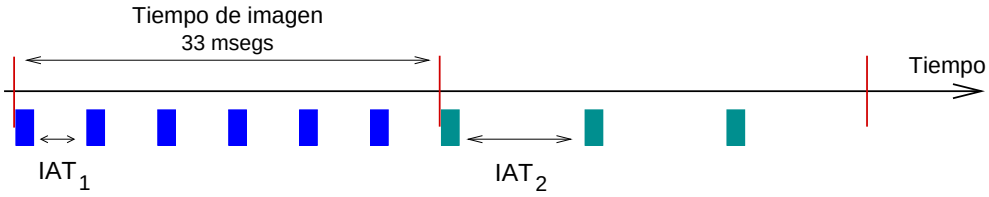


Figura 6.3: Fuente de tráfico VBR: modelo de inyección SR (flits equiespaciados).

Otra posibilidad es inyectar todos los flits que componen la imagen al principio de los 33 milisegundos, a una tasa pico prefijada. De esta forma, el tráfico generado presenta ráfagas, de forma similar a los trenes de paquetes [80]. En este caso, los flits de todas las tramas se inyectan con la misma separación, y se introducen periodos donde la fuente permanece inactiva entre el final de la inyección de los flits de una trama, y el inicio del siguiente tiempo de imagen. Esto se ilustra en la Figura 6.4. Las dos imágenes que se inyectan en este ejemplo tienen el mismo tamaño que las de la Figura 6.3. Obsérvese que los flits se inyectan con un IAT que es la mitad que el IAT_1 del caso anterior, es decir, al doble de velocidad. Así, en el caso de la primera trama, la fuente permanece ociosa durante la segunda mitad del tiempo de trama, mientras que en el caso de la segunda, la fuente permanece ociosa durante las últimas 3/4 partes del tiempo de trama. A este modelo se le ha denominado **modelo de inyección BB** (de *Back-to-Back*, o "uno tras de otro").

Las características de las películas empleadas en las simulaciones aparecen en la Tabla 6.1. Se muestran los datos relativos al tamaño de las imágenes: tamaño máximo, mínimo y medio, expresados en bits. De estos datos, se pueden extraer los requerimientos de ancho de banda

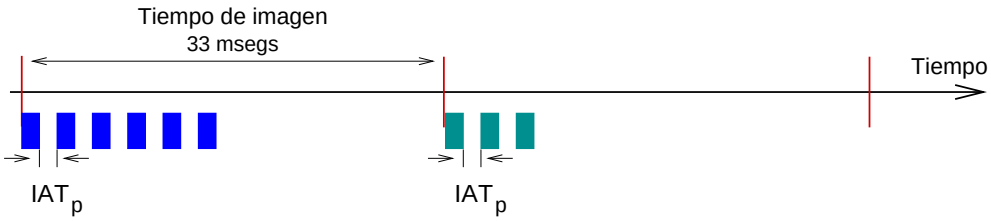


Figura 6.4: Fuente de tráfico VBR: modelo de inyección BB (flits a tasa pico).

	Tamaño de las Imágenes (bits)		
Secuencia	**Máximo**	**Mínimo**	**Medio**
Ayersroc	535030	148755	232976
Hook	454560	159622	272738
Martin	444588	116094	199880
Flower Garden	900139	308411	497126
Mobile Calendar	970205	412845	600742
Table Tennis	933043	260002	440547
Football	590532	340246	441459

Tabla 6.1: Estadísticas de las secuencias de vídeo MPEG-2 empleadas en las simulaciones.

tanto medios como máximos de las secuencias empleadas. En la Figura 6.5 se muestran los patrones de tráfico que presentan las secuencias empleadas. Se puede observar como los requerimientos de ancho de banda varían con el tiempo por el formato de la codificación MPEG2 (los picos son tramas I, etc.), pero también se introducen variaciones debidas al contenido de la información a codificar: imágenes estáticas o con mucho movimiento, cambios de escena, etc.

Cada secuencia tiene una longitud distinta. En caso de que durante la simulación se llegue al final de una secuencia, el generador de tráfico empieza de nuevo a transmitir imágenes desde el inicio de la secuencia para así mantener las mismas condiciones de carga durante todo el periodo de tiempo simulado.

Fuentes de control

Como se puso de manifiesto en la Sección 4.1, muchas aplicaciones multimedia requieren transmitir mensajes de control cortos además de los largos flujos de datos. Estos mensajes de control los puede generar el usuario (por ejemplo, comandos de control de una secuencia de vídeo: avance, parada, rebobinado...), o las tarjetas de interfaz de red (por ejemplo, cambio de los requerimientos de ancho de banda de una conexión, control de la congestión...). En el primer caso, el destino del mensaje de control será un *host*, mientras que en el segundo será un encaminador. En cualquier caso, estos mensajes se deben retransmitir con la mínima latencia, pues pueden afectar al comportamiento de los flujos de datos.

La tasa de generación de estos mensajes será normalmente baja. Por ejemplo, para los estudios realizados en [173], se genera un mensaje de control cada 33 milisegundos. El generador

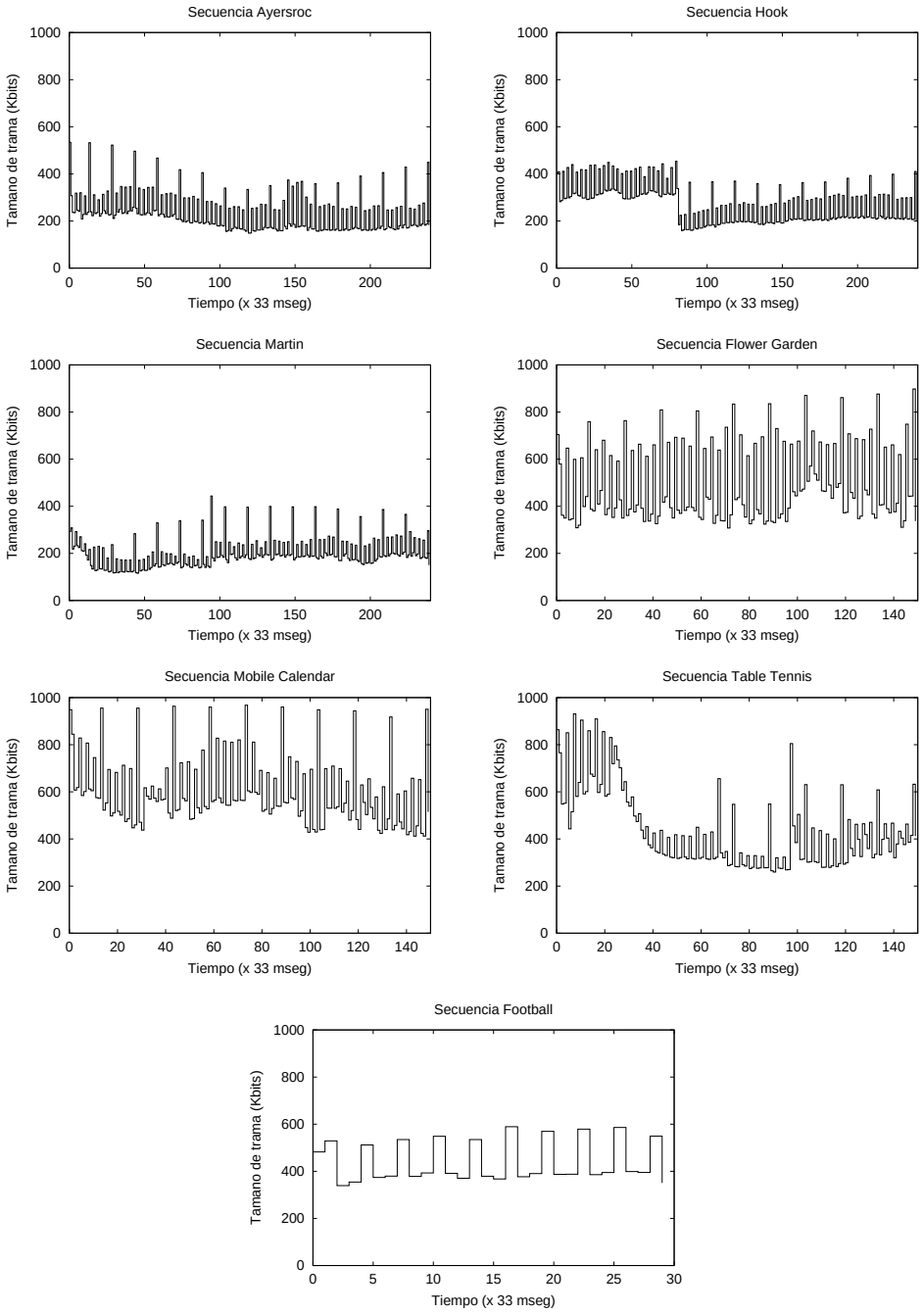


Figura 6.5: Patrones de tráfico de las secuencias MPEG-2 empleadas en las simulaciones.

de tráfico de control implementado en el simulador del MMR emite un mensaje de control también cada 33 milisegundos, pero con un margen de variación del 10 %. Es decir, la separación entre dos mensajes de control consecutivos será un valor aleatorio comprendido entre 29.7 y 36.3 milisegundos. En cuanto a la longitud, y dado que los flits tienen un tamaño elevado, se considera que cada mensaje de control ocupa un único flit.

Fuentes *best-effort*

Las fuentes de tráfico *best-effort* generan un tipo de **tráfico autosimilar**, que refleja fielmente el comportamiento del tráfico generado por aplicaciones de web [20].

Se dice que un patrón de tráfico es autosimilar si mantienen sus mismas propiedades independientemente de la escala a la que se observe. En el caso que se trata aquí, esto se traduce en que las fuentes de tráfico autosimilar exhiben ráfagas observables en diversas escalas de tiempos. Asimismo, pueden exhibir dependencias en el sentido de que los valores de cualquier instante están correlacionados con los valores de todos los instantes futuros.

Matemáticamente, la autosimilaridad se puede expresar de la forma siguiente [70]. Un proceso estocástico $X = (X_1, X_2, \ldots)$ se dice que es exactamente *autosimilar de segundo orden* con parámetro de Hurst:

$$H = 1 - \frac{\beta}{2} \quad (0 < \beta < 2)$$

si, para cualquier $M = 1, 2, 3, \ldots$,

$$Var(X^{(m)}) \propto m^{-\beta}$$

y

$$r^{(m)}(k) = r(k) \quad (k = 1, 2, 3, \ldots)$$

donde $r(k)$ es la función de autocorrelación, $X^{(m)} = (X_k^{(m)} : k : 1, 2, 3, \ldots)$ y

$$X_k^{(m)} = \frac{X_{km-m+1} + \cdots + X_{km}}{m}, \quad k \geq 1.$$

El generador de tráfico *best-effort* que se ha implementado está adaptado a partir de SURGE (*Scalable URL Reference Generator*) [20], un generador de carga de web realista. El tráfico generado de esta forma exhibe un comportamiento similar al de un conjunto de usuarios accediendo a un servidor.

El modelo de tráfico SURGE contiene una serie de parámetros para los cuales se han identificado las funciones de probabilidad que más se acercan al comportamiento real de las fuentes objeto de estudio. Para la finalidad de los estudios desarrollados en el MMR, sólo son necesarios los parámetros relacionados con el tamaño y temporización de las peticiones de servicio.

Más concretamente, el tráfico generado por una fuente *best-effort* autosimilar tiene una forma de ráfagas anidadas. Esto es debido a la forma en que se recupera una página web de un servidor. Según especifica el protocolo que gobierna la transferencia de páginas web HTTP (*HyperText Transfer Protocol*) [55], para recuperar una página web de un servidor, en

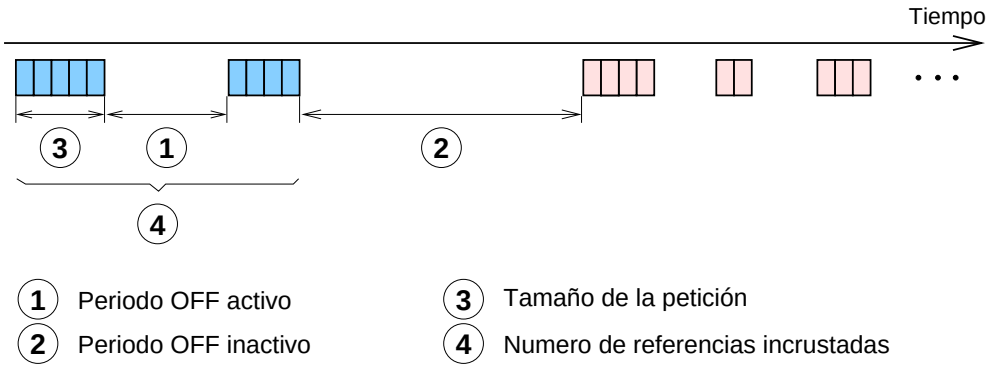


Figura 6.6: Fuente de tráfico *best-effort* autosimilar.

primer lugar se transmite la propia página (un fichero en formato HTML), y seguidamente, se transmiten los objetos incrustados en la misma (típicamente, imágenes, clips de audio, etc.). Tras la transmisión de cada objeto, la fuente entra en un periodo de inactividad (periodos "OFF"). Típicamente, el periodo de inactividad entre la transferencia de objetos de una misma página es bastante menor que entre el final y el inicio de la transferencia de una nueva página, debido a que en el segundo caso normalmente existe interacción con el usuario. Este comportamiento se modela en SURGE mediante los siguientes cuatro parámetros:

Número de referencias incrustadas Número de objetos que están referenciados en la página web, más la propia página web.

Tamaño de la petición Tamaño de los objetos a transmitir.

Periodo OFF activo Periodo de separación entre la transmisión de dos objetos de la misma petición.

Periodo OFF inactivo Periodo de separación entre la transmisión del último objeto de una petición y el primero de la siguiente.

Además, cada objeto (sea la página en sí, o cualquiera de los elementos incrustados en ella) se ha de transmitir como una secuencia de flits. En la implementación que se ha hecho, los flits que componen un mismo objeto se generan uno inmediatamente detrás del anterior. Todo esto se representa en la Figura 6.6. En ella se muestra la transmisión de dos páginas web. Los flits de cada una de ellas se muestran en diferente color. La primera contiene una referencia incrustada, mientras que la segunda contiene dos referencias.

SURGE modela cada uno de los cuatro parámetros comentados más arriba con una distribución estadística diferente. En la Tabla 6.2 se muestran las funciones de densidad de probabilidad, y los valores concretos para sus parámetros identificados por los desarrolladores de SURGE para generar el tráfico autosimilar.

Por último, señalar que el destino se elige de forma aleatoria para cada transacción, esto es, la información perteneciente a diferentes páginas web emitidas por una misma fuente puede ir dirigida a distintos destinos. Evidentemente, todos los flits que componen una página web, y sus referencias incrustadas, irán dirigidos al mismo destino.

Componente	Modelo	Función de probabilidad	Parámetros
Referencias incrustadas	Pareto	$p(x) = \alpha k^{\alpha} x^{-(\alpha+1)}$	$k = 1;\ \alpha = 2{,}43$
Tamaños de peticiones	Pareto	$p(x) = \alpha k^{\alpha} x^{-(\alpha+1)}$	$k = 1000;\ \alpha = 1{,}0$
Periodos OFF activos	Weibull	$p(x) = \frac{bx^{b-1}}{a^b} e^{-(\frac{x}{a})^b}$	$a = 1{,}46;\ b = 0{,}382$
Periodos OFF inactivos	Pareto	$p(x) = \alpha k^{\alpha} x^{-(\alpha+1)}$	$k = 1;\ \alpha = 1{,}5$

Tabla 6.2: Resumen de estadísticas empleadas en la generación de tráfico autosimilar.

6.2.2. Configuración de las simulaciones

Existe un número elevado de parámetros que controlan diversos aspectos de cada simulación. Estos parámetros se pueden clasificar en varias categorías. A continuación, se describirá el significado de cada uno de ellos.

1. Parámetros del encaminador Este conjunto de parámetros define las dimensiones del encaminador.

- **Número de puertos** Número de enlaces de entrada/salida que tendrá el encaminador.
- **Número de canales virtuales** Se trata del número de canales virtuales que implementará el encaminador por cada puerto de entrada/salida.
- **Valor de K** Este parámetro define el tamaño de la vuelta del planificador, que se emplea para reservar y adjudicar el ancho de banda a las distintas conexiones (ver Sección 4.3.2).
- **Tamaño del phit** Se trata del número de bits que pueden transmitirse en paralelo a través de cada enlace físico. Este valor coincide con el ancho del camino de datos interno del encaminador.
- **Tamaño del flit** Tamaño de la unidad de control de flujo del Encaminador. Esto es, el tamaño de flit para PCS (flujos multimedia) y el tamaño máximo de mensaje para *virtual cut-through* (tráfico no multimedia).
- **Ancho de banda de los enlaces** Capacidad de los enlaces físicos y del camino de datos interno del encaminador. Este parámetro, junto con el tamaño de phit, determina la duración del ciclo de reloj del encaminador.
- **Tamaño de los *buffers*** Tamaño en flits de los *buffers* asociados a cada canal virtual del MMR.

2. Configuración del planificador Estos parámetros deciden la configuración de los algoritmos de planificación que se emplean en el encaminador.

- **Tipo de planificador** Mediante este parámetro se seleccionan tanto el algoritmo planificador de enlaces, como el del conmutador. Existe un valor para cada combinación posible, de entre los algoritmos presentados en el Capítulo 5.
- **Número de candidatos** Número de canales virtuales seleccionados por los planificadores de enlaces, que serán la entrada al algoritmo de planificación del conmutador (ver Sección 5.1).

3. Configuración del control de admisión de conexiones (CAC) Estos parámetros configuran el criterio seguido para admitir o rechazar conexiones.

- **Tipo de CAC** Decide el tipo de algoritmo de CAC a emplear. Existen diversas posibilidades, pero en esta Tesis los estudios se han limitado a utilizar el algoritmo descrito en la Sección 4.3.3.
- **Factor de concurrencia** Parámetro que decide el grado de multiplexación estadística entre los picos de las conexiones VBR (ver Sección 4.3.3)

4. Configuración de la simulación Se trata de los parámetros relacionados intrínsecamente con el funcionamiento y control del simulador.

- **Semilla para números aleatorios** Es la semilla que se empleará para inicializar el generador de números aleatorios.
- **Fichero conexiones** Este fichero contiene los datos relativos a las conexiones que serán establecidas al inicio de la simulación. Dichos datos incluyen el tipo de la conexión (CBR, VBR), ancho de banda medio y pico (en su caso), puertos físicos origen y destino, y si es VBR, identificador de la traza de la secuencia de vídeo a transmitir.
- **¿Hay tráfico VBR?** Esta cuestión decide la forma de finalizar cada simulación.
- **Número de ciclos del periodo transitorio** Junto con el siguiente parámetro, define la duración de la simulación cuando no hay presente tráfico VBR. Se trata del número de ciclos del planificador a ejecutar al inicio de la simulación, durante los cuales no se recogen estadísticas. De esta forma, se persigue que el simulador alcance un estado estable, antes de proceder a medir resultados.
- **Número de ciclos del periodo permanente** Junto con el parámetro anterior, define la duración de la simulación cuando no hay presente tráfico VBR. Se trata del número de ciclos del planificador a ejecutar, durante los cuales las estadísticas pertinentes serán recogidas.
- **Número de GOPs a transmitir** Define la forma de parar la simulación cuando existen conexiones VBR. La simulación finaliza cuando por cada fuente VBR, han conseguido atravesar el encaminador tantos GOPs completos como se indique en este parámetro. Señalar que en este caso, se transmiten inicialmente dos tramas adicionales por cada conexión sobre las que no se recogen estadísticas, para compensar los efectos del desalineado en el inicio de las transmisiones. Esto es, se empiezan a medir resultados una vez que se todas las fuentes han alcanzado un estado estacionario, con todas transmitiendo.

5. Configuración de la carga Parámetros relativos al tipo y mezcla de cargas que se alimentarán al simulador.

- **¿Hay tráfico de control?** En caso afirmativo, se creará una fuente de tráfico de control por cada enlace físico.
- **¿Hay tráfico *best-effort*?** En caso afirmativo, se crearán fuentes de tráfico *best-effort* en todos los enlaces físicos.

- **Número de fuentes *best-effort*** por enlace físico del encaminador, en caso de que haya tráfico de este tipo. Definen la carga de tráfico *best-effort* que se introduce en el sistema simulado.
- **Tipo de fuentes VBR** Este parámetro define el modelo de inyección de los flits correspondientes a las fuentes de tipo VBR: modelo SR o BB.
- **Ancho de banda pico** Este parámetro define la tasa pico de generación de los flits para el modelo de inyección BB para las fuentes VBR.
- **Alineamiento de tramas** Este parámetro decide el alineamiento entre las distintas conexiones VBR. Las distintas conexiones se pueden alinear de tres formas diferentes [35]:
  - ***High Source Alignment* (HSA)**: Todas las fuentes transmiten sus imágenes de tipo I (las que más ancho de banda consumen) a la vez. Se trata por tanto del peor caso posible.
  - ***Low Source Alignment* (LSA)**: Las fuentes se alinean de manera escalonada, de forma que la imagen I de una secuencia coincida con la B de otra, y así sucesivamente. Este sería el mejor caso, pues los picos de unas conexiones se "encajan" en los valles de otra, aprovechando perfectamente el ancho de banda, y sin provocar picos de congestión.
  - **Aleatorio**: Las fuentes se alinean sin ningún patrón en particular. Se trata del caso más común.

  En la Figura 6.7 se muestran ejemplos de los tres casos. En los tres aparecen los perfiles de tráfico de 3 secuencias de vídeo, junto con el tamaño agregado de todas las tramas en cada momento. Se puede observar como para la alineación HSA (Figura 6.7-(a)) efectivamente todas las fuentes emiten sus tramas I a la vez, provocando que durante esos intervalos de tiempo el tráfico total generado crezca de forma importante.

  Por el contrario, en la alineación LSA (Figura 6.7-(b)), al desplazar cada secuencia un tiempo de trama respecto de la anterior, se consigue que los picos de una conexión (tramas I) coincidan con valles de otra (tramas B), de manera que no existen grandes variaciones en el perfil del tráfico agregado.

  Por último, con la alineación aleatoria (Figura 6.7-(c)) se puede observar que hay picos que coinciden y otros que no. Así, el perfil de tráfico exhibe picos más pronunciados que con el alineamiento LSA, pero menos que con alineamiento HSA. Este es el caso medio, el más habitual.
- **Desplazamiento interno de tramas** Decide si la trama inicial de cada conexión VBR se transmite desde su inicio, o desde algún punto aleatorio dentro de ella.

Cada simulación se desarrolla de la siguiente forma. El simulador, en primer lugar, crea e inicializa todas las estructuras de datos del encaminador, tarjetas de red, resultados, etc. Seguidamente, lee el fichero con la información de las conexiones multimedia que se van a constituir la carga del simulador. El fichero contiene los datos básicos de cada conexión: tipo (CBR o VBR), puerto de entrada, puerto de salida y ancho de banda medio, y además, para las conexiones de tipo VBR, el ancho de banda máximo o pico, y un identificador de la traza de vídeo MPEG-2 que se va a transmitir a través de esa conexión. Este fichero de conexiones se genera previamente, y con la idea de que la suma de ancho de banda medio de todas las conexiones que comparten el mismo enlace físico de entrada sea un tanto por

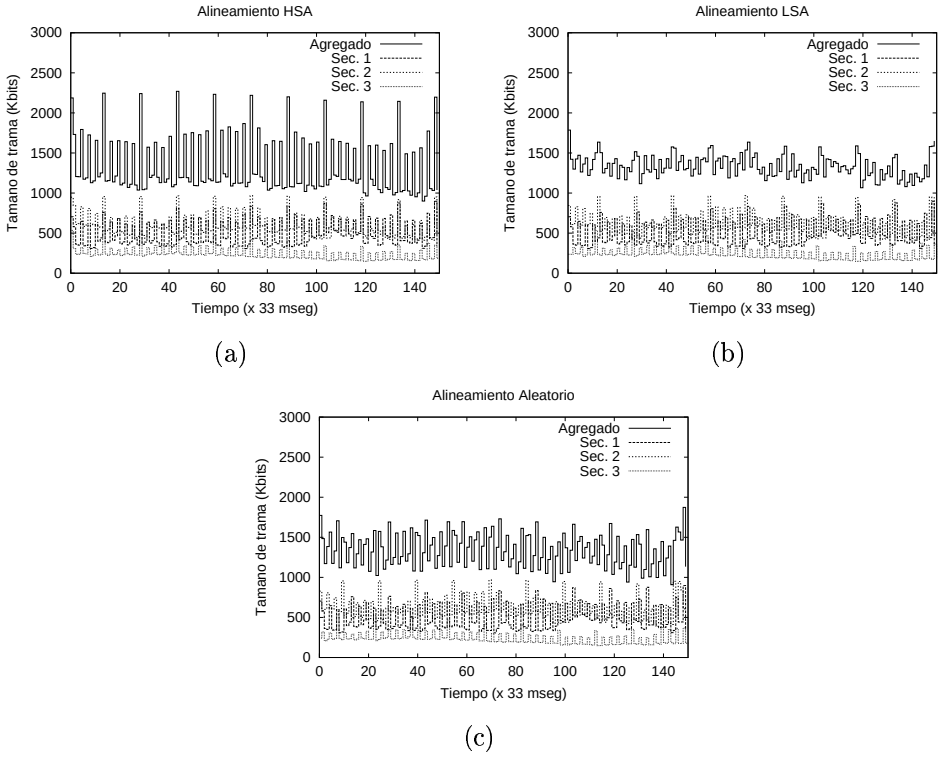


Figura 6.7: Alineación de las fuentes VBR.

ciento dado de la capacidad del enlace. Esto es, se genera una mezcla de conexiones de forma que la carga del sistema sea un porcentaje del ancho de banda del enlace. Estas conexiones están mezcladas de forma totalmente aleatoria. Los puertos de destino se generan de forma uniforme.

Cuando el simulador va leyendo el fichero de conexiones, interpreta cada línea como una petición de establecimiento de conexión. Y aplica sobre cada petición los criterios de admisión comentados en la Sección 4.3.3. Por tanto, puede que alguna conexión resulte rechazada, por falta de suficiente ancho de banda en el enlace de salida requerido para satisfacer sus necesidades. Hay que hacer notar que simplemente se implementa la verificación de los criterios de admisión de conexiones. No se simula en detalle la transferencia de la sonda de establecimiento de la misma. El efecto es que la recogida de datos, y la evaluación del encaminador, se hace sobre un periodo de tiempo donde las conexiones ya están establecidas, y ni se crean nuevas, ni se destruyen las existentes. Se trata por tanto de, dada una situación estacionaria, evaluar el comportamiento de la arquitectura así como de los algoritmos de reparto de recursos implementados en ella.

Una vez leído todo el fichero de conexiones, y reservados los recursos de las admitidas, se planifican los eventos iniciales que regulan el funcionamiento del encaminador y de las fuentes generadoras de tráfico, y la simulación empieza a ejecutarse.

Una vez que se alcanza la condición de finalización de la simulación, diferente según si el único tráfico multimedia presente es CBR, o si hay también tráfico VBR, se calculan las estadísticas de los resultados que se han ido recogiendo, y se muestran como salida del simulador.

6.2.3. Resultados ofrecidos por el simulador

La herramienta de simulación ofrece los siguientes resultados al terminar la simulación:

- **Tiempo de simulación**, medido en ciclos de reloj del encaminador y en milisegundos.
- **Porcentajes de uso de los candidatos** en los emparejamientos calculados por el planificador del conmutador.
- **Calidad del emparejamiento**, medido como el cociente entre emparejamientos conseguidos frente a emparejamientos posibles. Se dan el valor medio, mínimo, máximo, y la desviación típica.
- **Recuento de flits**: flits generados, recibidos en el encaminador, transmitidos por el encaminador, y en *buffers* del mismo, por cada canal virtual, por cada enlace físico, y totales.
- **Carga generada**, desglosada por enlace físico de entrada, y en media.
- **Utilización de los enlaces de entrada**, y media. Estos parámetros reflejan la cantidad de información que llega al encaminador.
- **Utilización de los enlaces de salida**, y media. La utilización media de los enlaces de salida se corresponde con la utilización media del *crossbar*.

En condiciones de carga normal, la carga media generada, la utilización media de los enlaces de entrada, y la utilización media del *crossbar* coinciden, puesto que todos los flits generados salen de sus respectivas tarjetas de interfaz, y atraviesan el encaminador. Cuando comienza la saturación, el encaminador no es capaz de trasegar todos los flits que le llegan, con lo que la utilización media de salida decrece respecto a la de entrada. Los *buffers* del MMR se empiezan a llenar, y finalmente, el control de flujo evitará que pasen flits desde los interfaces de red al MMR. Entonces, disminuirá la utilización media de los enlaces de entrada con respecto a la carga media generada.

Seguidamente, se proporcionan los siguientes resultados en cuanto a las prestaciones experimentadas por el tráfico que atraviesa el encaminador. Las medidas se desglosan por cada tipo de tráfico, y además para el tráfico CBR, por tipo de conexión (de bajos, medios y altos requerimientos de ancho de banda).

- **Retardo del encaminador**: tiempo transcurrido desde que un flit se almacena en su *buffer* de entrada del MMR correspondiente, hasta que es transmitido a través del *crossbar* hacia su enlace de salida. Es decir, se trata del tiempo de espera que experimenta el flit en los *buffers* del MMR, más el tiempo en atravesar el *crossbar*. Se da el valor máximo, medio, y desviación típica, medidos en ciclos del encaminador y en microsegundos.
- **Retardo desde la generación**: tiempo transcurrido desde que un flit se genera hasta que es transmitido a través del *crossbar* hacia su enlace de salida. Se da el valor máximo, medio, y desviación típica, medidos en ciclos del encaminador y en microsegundos. Este valor incluye el tiempo de espera experimentado por el flit en el interfaz de red donde se generó, más el retardo experimentado en atravesar el encaminador.

- **Distribución del retardo de los flits**: para cada tipo de conexión, se establecen un conjunto de *umbrales*. Dichos umbrales están relacionados con el IAT de la conexión en caso de las conexiones CBR, y con el tiempo de trama (33 msegs) en el caso de las conexiones VBR. Los umbrales son múltiplos y submúltiplos de estos valores. El simulador hace un recuento de los flits cuyo retardo es menor que cada umbral, calculando el porcentaje sobre el total de flits emitidos del mismo tipo. Así, se obtiene la distribución de los retardos del encaminador y desde la generación experimentados por los flits. Esta información da una idea muy ajustada de la QoS que reciben las conexiones.

  Además de los datos relativos a retardo de flits, para las conexiones VBR se facilitan los datos de retardo (del encaminador, desde generación, y distribución de los mismos) *relativos a las tramas de vídeo*, las unidades de datos de la aplicación en concreto considerada. El motivo es que para este tipo de aplicación, no son tan importantes las prestaciones experimentadas por los flits individuales, como por la unidad de datos completa. El destino necesita la imagen completa para poder decodificarla y visualizarla. Mientras la imagen esté disponible a su debido tiempo en el destino, el retardo sufrido por los flits individuales no es importante. Por tanto, la QoS de las conexiones VBR se medirá sobre tramas.

  Con el objeto de independizar las medidas de retardo de tramas del modelo de inyección empleado (una trama tarda más en ser inyectada con el modelo SR que con el BB), el retardo de una trama se asocia al retardo sufrido por el último flit que la compone. Esto es, el retardo desde generación de una trama se mide como el tiempo transcurrido desde el final de su generación, hasta el final de su recepción en el destino. Y análogamente, se define el retardo del encaminador de una trama: es el tiempo transcurrido desde que el último flit de la misma se almacena en los *buffers* del MMR, hasta que abandona el encaminador por el enlace de salida correspondiente.

- ***Jitter***: variación en el retardo de los flits de una conexión. Se calcula como la diferencia entre el retardo experimentado por un flit, y el experimentado por el flit anterior. Ambos retardos se consideran desde la generación. De nuevo, para conexiones VBR, el *jitter* se mide sobre las tramas de vídeo en lugar de sobre los flits que las componen.

6.3. Métricas para la evaluación de prestaciones

De entre todos los resultados proporcionados por el simulador, los que resultan de verdadero interés a la hora de evaluar al arquitectura y los algoritmos propuestos son los que se indican a continuación. El resto únicamente se emplean de forma auxiliar, bien para comprobar el correcto funcionamiento del simulador, bien como datos adicionales que en ciertos casos pueden arrojar más luz sobre el comportamiento indicado por la métricas de evaluación escogidas.

Los índices de prestaciones escogidos dependen del tipo de tráfico de que se trate. Se han escogido los siguientes:

- **Utilización media del *crossbar***
- **Retardo medio desde la generación**, medido sobre flits, para el tráfico CBR y el *best-effort*
- **Retardo medio desde la generación**, medido sobre tramas de vídeo, para el tráfico VBR

- **Retardo máximo desde la generación**, medido sobre flits, para el tráfico de control
- **Distribución de los retardos de flit desde la generación**, para el tráfico CBR
- **Distribución de los retardos de trama desde la generación**, para el tráfico VBR
- ***Jitter***, para flits CBR, y tramas VBR

El primero de los índices mide *cuánto* tráfico es capaz de trasegar el encaminador sin llegar a saturarse. En este sentido, la saturación se entiende en el sentido tradicional, comúnmente empleado en el análisis de redes de multicomputadores, esto es, cuando la utilización de salida cae por debajo de la carga generada, y los flits se acumulan en los *buffers* (del encaminador o de las tarjetas de red) sin llegar a ser transmitidos. La entrada en saturación implica un crecimiento no acotado de los retardos experimentados por los flits que consiguen atravesar el encaminador, lo que claramente contraviene las garantías de QoS que necesitan las aplicaciones multimedia. Ahora bien, cuando el nivel de carga está cercano a saturación, los retardos de los flits empiezan a ser elevados, y por tanto hay que analizar cuidadosamente cuál es el punto de carga límite más allá del cuál las garantías de QoS de las aplicaciones se ven comprometidas. Este punto límite no tiene por qué coincidir con el punto de saturación que se determinaría en una red convencional, si no que es posible que ocurra a niveles de carga algo menores. Para ayudar a determinar este punto límite, se emplean el resto de índices. Ellos proporcionan información acerca de *cómo (en el sentido de con qué QoS)* se transmite el tráfico.

Para los tráficos multimedia, las medidas de retardos medios dan una idea inicial del punto de saturación. Sin embargo, son las distribuciones de dichos retardos las que proporcionan el nivel de detalle necesario para determinar dicho punto límite de carga, más allá del cual, las garantías de QoS se ven comprometidas. Siempre se van a utilizar los retardos desde la generación, pues son los que dan la medida del tiempo total invertido por el tráfico en alcanzar su destino.

Para el tráfico de control, puesto que el objetivo es que llegue a su destino con el menor retardo posible, basta con observar el retardo máximo experimentado por sus flits desde su generación para saber si dicho objetivo se cumple.

Por último, para el tráfico *best-effort*, y puesto que el objetivo para este tipo de tráfico es ofrecerle buena latencia media y alta productividad, sin tener en cuenta ninguna clase de garantías de QoS, se adopta la medida tradicional empleada usualmente en los estudios desarrollados sobre redes convencionales: el retardo medio desde la generación (o latencia desde la generación).

6.4. Establecimiento de los parámetros básicos de diseño

Antes de llevar a cabo la evaluación exhaustiva de los algoritmos de planificación de enlaces y conmutador desarrollados para el Encaminador Multimedia, es necesario determinar los valores de algunos de los parámetros básicos involucrados en su diseño. Una vez hecho esto, el resto de pruebas se realizarán con la configuración determinada en este apartado.

En concreto, en este punto se trata de determinar los valores óptimos de los siguientes tres parámetros:

- Tamaño del flit

- Tamaño del ciclo del planificador (valor de K)
- Tamaño de los *buffers* en el MMR

El **tamaño de flit** influye en dos aspectos. Por un lado, cada flit transmitido, se acompaña de un phit con información del número de canal virtual al cual pertenece el flit en cuestión. Además, eventualmente implica la transmisión de un phit de control de flujo en sentido inverso. Entonces, conviene disponer de flits de tamaño elevado, para amortizar mejor la sobrecarga de control introducida por esos dos motivos. A esto hay que añadir que disponer de flits grandes facilita el trabajo del planificador de enlaces y conmutador, pues se dispondrá de más tiempo para ejecutarlos. Recordemos que el cálculo de la planificación se realiza en paralelo con la transmisión de flits, y por tanto, la duración del ciclo de flit es el tiempo del que se dispone para llevarla a cabo. Cuanto mayores sean los flits, de más tiempo se dispone para el cálculo de la planificación.

Por otro lado, se pueden aducir dos motivos contra el uso de flits grandes: que no se aprovechen eficientemente los flits con suficiente carga útil, y el incremento en la latencia. El primer inconveniente no es tal, debido a las características de las aplicaciones multimedia. Dichas aplicaciones generan grandes flujos de datos, con duraciones desde unos pocos segundos a varias horas, con lo que el porcentaje de flits que puedan no ser rellenados totalmente con carga útil es despreciable. Para el segundo inconveniente, hay que tener en cuenta que las aplicaciones multimedia no son excesivamente sensibles a la latencia, con lo que existe un cierto margen de tolerancia en ese punto. Este extremo hay que verificarlo en estas pruebas preliminares.

En cuanto al **tamaño del ciclo del planificador**, definido por el valor del parámetro K[1], está relacionado con la granularidad de reserva del ancho de banda. Con valores mayores de K, el ancho de banda del enlace se divide en trozos más pequeños, que son asignados a las distintas conexiones. Así, el ancho de banda reservado (una magnitud "discreta"), se aproxima más al ancho de banda realmente ocupado por las conexiones (una magnitud "continua"). Es decir, la reserva de ancho de banda se ajusta más a los requerimientos de las aplicaciones, ofreciendo un mejor aprovechamiento del ancho de banda de los enlaces.

Sin embargo, es posible que la mayor duración del ciclo de planificación influya negativamente en los valores de latencia o *jitter* de las conexiones. Esto es lo que hay que comprobar en este estudio preliminar.

Por último, el **tamaño de los *buffers*** en el Encaminador está directamente relacionado con la posibilidad de realizar un diseño compacto del mismo. En ese sentido, es conveniente el uso de *buffers* de tamaño mínimo, dada la elevada cantidad de canales virtuales que soporta el MMR. El uso de *buffers* pequeños no implica pérdida de flits, debido a que el MMR emplea control de flujo. Así, los flits serían eventualmente almacenados en los *buffers* de la tarjeta interfaz de red. Es decir, cuando se usen *buffers* pequeños, y debido al control de flujo, los flits de una conexión cuyo avance está detenido, se almacenarán en su mayor parte en la tarjeta de red, y una mínima parte en los encaminadores que atraviese la conexión. Esto facilita además que la fuente del tráfico ejerza un control sobre la conexión en presencia de congestión. Por ejemplo, la fuente puede decidir descartar flits de forma selectiva, o en un caso extremo, abortar la conexión. En tal caso, en la red quedaría una cantidad mínima de flits pertenecientes a dicha conexión, y los recursos desperdiciados en hacerlos llegar a su destino serían mínimos.

[1]Hay que recordar que el tamaño del ciclo del planificador es de $K \times CV$ ciclos de flit, siendo $K > 1$, y CV el número de canales virtuales por enlace físico en el MMR

Número de puertos	4
Número de canales virtuales	128
Tamaño de phit	16 bits
Ancho de banda de los enlaces	1.24 Gbps
Tipo de planificador	IABP + COA
Número de candidatos	4
Tipo de CAC	Sección 4.3.3
Factor de concurrencia	16
Semilla para números aleatorios	12345

Tabla 6.3: Evaluación preliminar: Parámetros comunes a todas las pruebas.

Por otro lado, el hecho de que los flits se almacenen básicamente en el interfaz de red en lugar de los encaminadores, puede dificultar la planificación de los recursos, provocando incrementos en el retardo de los flits, de manera que la QoS resulte perjudicada. El impacto puede ser mayor en presencia de tráfico VBR, debido a las ráfagas que presenta este patrón de tráfico. Tradicionalmente, los conmutadores han implementado *buffers* grandes, con el objeto de absorber dichas ráfagas, y evitar la pérdida de flits y la degradación de las prestaciones de los flujos multimedia [36]. Sin embargo, debido a que en el diseño del MMR se evita la pérdida de flits mediante el control de flujo, puede que el uso de *buffers* grandes ya no sea imprescindible. Por tanto, hay que determinar el tamaño mínimo de *buffers* en el Encaminador con el objeto de conseguir un diseño lo más compacto posible, pero sin comprometer la QoS requerida por las aplicaciones.

6.4.1. Configuración de las simulaciones

Todas las pruebas realizadas en este apartado se han configurado con los parámetros mostrados en la Tabla 6.3.

En cuanto a la carga, estas pruebas preliminares se han llevado a cabo únicamente con tráfico multimedia. Se ha analizado la franja de cargas medias-altas, entre el 50 % y el 100 % del ancho de banda de los enlaces.

Las pruebas con tráfico CBR se ejecutan durante 600 ciclos del planificador, más 10 ciclos de periodo transitorio.

Las pruebas con tráfico VBR se ejecutan hasta que cada conexión haya transmitido 6 GOPs. Al inicio de la simulación, existe un transitorio de 2 tiempos de trama antes de recoger estadísticas. Las tramas de las secuencias de vídeo se alinean de forma aleatoria, el caso más común, y cada secuencia empieza a transmitirse desde un punto aleatorio de su trama inicial (desplazamiento interno de tramas). Se presentan los resultados obtenidos empleando los dos tipos de fuentes VBR, según su modelo de inyección (SR o BB). El ancho de banda pico para el modelo de inyección BB se configura a 50 Mbps.

6.4.2. Tamaño de flit

En primer lugar, se va a determinar la influencia del tamaño de flit en las prestaciones del Encaminador. Para estas pruebas, se fija el tamaño de ciclo del planificador a un valor de K

de 16, y el tamaño de los *buffers* se fija a 1 flit.

Los tamaños de flit que se van a evaluar son de 128 y 1024 bits. Puesto que en ambos casos el tamaño de phit (el ancho de los enlaces y del camino de datos interno del conmutador) es idéntico e igual a 16 bits, ésto se traduce en un ciclo de flit de 9 y 65 ciclos del encaminador, respectivamente:

$$\frac{128\ \frac{bits}{flit}}{16\ \frac{bits}{phit}} = 8\ \frac{phits}{flit} \quad \rightarrow \quad 8\ ciclos + 1\ ciclo\ control = \mathbf{9\ ciclos}$$

$$\frac{1024\ \frac{bits}{flit}}{16\ \frac{bits}{phit}} = 64\ \frac{phits}{flit} \quad \rightarrow \quad 64\ ciclos + 1\ ciclo\ control = \mathbf{65\ ciclos}$$

Como se puede observar por las expresiones anteriores, en el caso de emplear flits de 128 bits, se introduce un ciclo de sobrecarga (esto es, que no sirve para transmitir carga útil) por cada 8 ciclos empleados en transmitir datos. Si el flit pasa a ser de 1024 bits, existe un ciclo de sobrecarga por cada 64 destinados a la transmisión de datos. Esto influye también en el consumo de ancho de banda efectuado para transmitir los datos de las aplicaciones. Cuando se establece cada conexión, se ha de tener en cuenta también la sobrecarga de control introducida a la hora de verificar las condiciones de admisión. Esto es, hay que tener en cuenta que para atender a un flujo multimedia que ocupa un ancho de banda B en términos de datos de usuario, hay que efectuar una reserva para soportar un ancho de banda de $B \times CF/TF$, siendo CF la duración del ciclo de flit expresada en ciclos de phit (o del encaminador, que es lo mismo), y TF el tamaño del flit expresado en phits.

Por ejemplo, para una conexión que genera un ancho de banda de 1.54 Mbps, con flits de 128 bits, el ancho de banda real necesario sería de:

$$1{,}54\ Mbps \times \frac{9}{8} \quad = \quad \mathbf{1{,}73\ Mbps}$$

Si el tamaño de flit es de 1024 bits, entonces el consumo real de ancho de banda sería sólo de:

$$1{,}54\ Mbps \times \frac{65}{64} \quad = \quad \mathbf{1{,}56\ Mbps}$$

El algoritmo de control de admisión de conexiones empleado tiene en cuenta esta sobrecarga a la hora de aceptar o rechazar las conexiones que componen la carga introducida al simulador. El efecto es que, a cargas moderadas-altas, al emplear flits más pequeños, la cantidad de ancho de banda rechazado se incrementa.

Hay que señalar también que en la Sección 5.2.4 se estimó que era necesario un tamaño de flit de 160-176 bits para poder ejecutar los algoritmos de planificación propuestos. Sin embargo, para realizar este análisis se ha considerado oportuno emplear un tamaño de flit menor para comprobar si las prestaciones obtenidas justifican la optimización de los diseños propuestos, o incluso de los propios algoritmos.

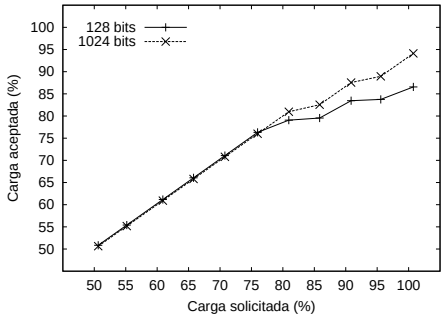


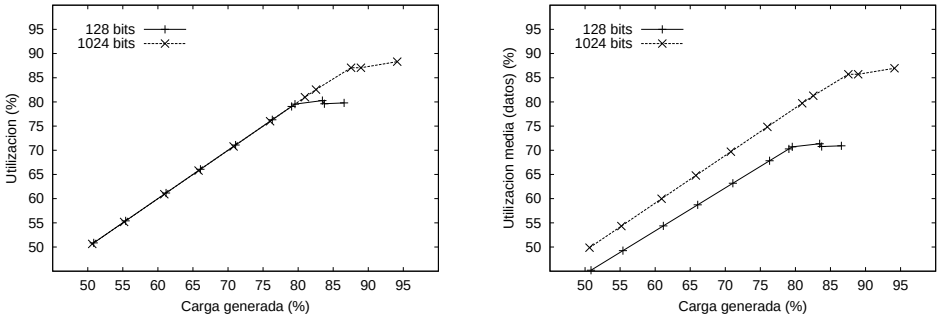
Figura 6.8: Impacto del tamaño de flit - Tráfico CBR: Carga aceptada frente a carga solicitada.

Se comenzará el análisis empleando tráfico de tipo CBR, compuesto por una mezcla aleatoria de conexiones de distintos requerimientos de ancho de banda, como se explicó en la Sección 6.2.1.

En la Figura 6.8 se puede observar lo comentado en párrafos anteriores respecto al rechazo de conexiones debido a la mayor sobrecarga de control. En dicha Figura, se muestra la carga aceptada, la que ha pasado con éxito el control de admisión, frente a la solicitada al inicio de la simulación. Por debajo del 76 % de carga, no existe diferencia en el ancho de banda admitido para los dos tamaños de flit. Sin embargo, a partir de ese punto, el empleo de flits grandes hace que se admita una mayor cantidad de ancho de banda. En concreto, si se emplean flits de 128 bits, la carga máxima que se admite está en torno al 86 %, mientras que con flits de 1024 bits, dicha carga máxima admitida se halla en torno al 95 %.

A continuación, hay que ver qué prestaciones se obtienen con cada tamaño de flit. En primer lugar, se analizará la utilización media del *crossbar*, para comprobar cuanta carga es capaz de trasegar el encaminador. La Figura 6.9-(a) muestra esta magnitud. En la franja de carga común para ambos tamaños de flit, esto es, hasta el 76 %, el encaminador es capaz de dar salida a toda la carga generada con ambas configuraciones. A partir de ese punto, el nivel de carga a que se somete el encaminador en cada caso es diferente, tal y como se señaló previamente. También se observa que la saturación llega en torno al 79 % de carga, cuando los flits son de 128 bits, mientras que si los flits ocupan 1024 bits, esto no ocurre hasta el 87 % de carga, aproximadamente. Estos resultados se refieren a carga total, es decir, incluyendo la sobrecarga introducida por el phit que se transmite con cada flit. En la Figura 6.9-(b) se muestra la utilización media del *crossbar*, pero en datos, es decir, descontando la parte de utilización debida a los phits de sobrecarga. En ella se observa claramente que para flits de 128 bits, el aprovechamiento del ancho de banda por parte de la información útil es menor.

Parece por tanto claro que el uso de flits más grandes es preferible, al menos en cuanto a la cantidad de carga útil que es capaz de trasegar el encaminador. Queda por comprobar si las prestaciones temporales se ven comprometidas por ello. Para ello, se va a emplear la distribución de retardos de flit desde la generación, desglosada por tipo de conexión. Hay que recordar que la carga CBR se compone de una mezcla aleatoria de conexiones de bajos, medios y altos requerimientos de ancho de banda. Estos resultados se muestran en las gráficas de la Figura 6.10, para los dos puntos de carga más altos, comunes a ambas configuraciones, esto es, para el 72 % y el 76 %. En ellas se aprecia como cuando se emplean los flits de mayor tamaño, hay algunos flits más que son capaces de cumplir con cotas de retardo más estrictas, en las conexiones de medios y altos requerimientos. La diferencia se acentúa al incrementar la carga. En las conexiones de bajo consumo de ancho de banda, no existen diferencias, puesto



(a) Utilización total (b) Utilización por datos

Figura 6.9: Impacto del tamaño de flit - Tráfico CBR: Utilización media del *crossbar*.

que en ambos casos todos los flits cumplen con todas las cotas impuestas.

Para determinar el punto máximo de carga alcanzable por cada uno de los casos analizados, es necesario verificar a qué nivel de carga la tasa de incumplimiento de cotas de retardo se hace intolerable. Tanto para la carga del 72 % como para la del 76 %, el 100 % de los flits sufre retardos menores que $IAT/16$ para las conexiones de 64 Kbps, que $IAT/2$ para las conexiones de 1.54 Mbps, y que $2 \times IAT$ para las de 55 Mbps[2]. Estos resultados entran dentro de lo aceptable. Nótese que los umbrales escogidos para las comparaciones dependen de los requerimientos de cada conexión, y son tanto más estrictos cuanto más requerimientos de ancho de banda tiene la conexión.

Para cargas superiores al 76 % ya no se pueden comparar directamente los resultados obtenidos para ambos tamaños de flit, puesto que los niveles de carga a los que se somete el encaminador son diferentes. En la Figura 6.11 se muestran los resultados obtenidos para el siguiente punto de carga obtenido con flits de 128 bits, que se sitúa en torno al 80 %. En este punto, las cotas anteriores ya no son cumplidas por todos los flits. Ya no se puede decir que se esté ofreciendo QoS a las aplicaciones, y por tanto, este nivel de carga no debería ser alcanzado.

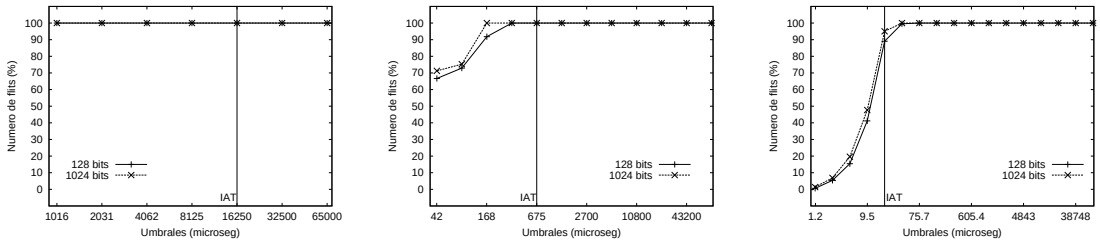
Por el contrario, para el siguiente punto de carga obtenido con flits de 1024 flits, un 83 % aproximadamente, los resultados son bastante similares a los obtenidos para una carga del 76 % con flits de 128 bits. Esto se muestra en las gráficas de la Figura 6.12.

Se puede concluir por tanto, que los flits de mayor tamaño son preferibles. No sólo mejoran el aprovechamiento del ancho de banda de los enlaces, sino que también se ha demostrado que con ellos el Encaminador es capaz de ofrecer un mayor rango de carga útil con garantías de QoS a las aplicaciones que generan tráfico CBR.

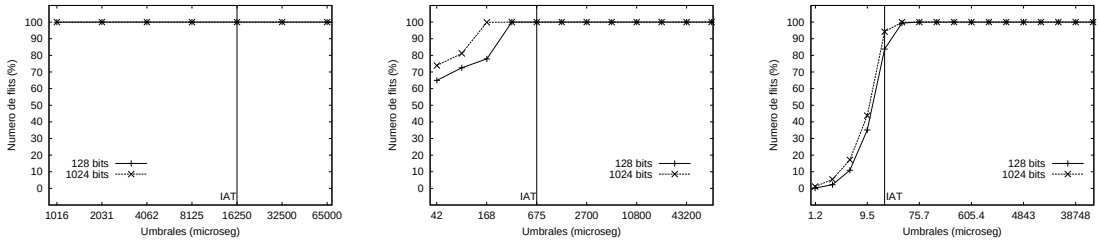
Falta verificar si se cumple lo mismo cuando la carga está compuesta por tráfico VBR. Se emplearán para ello las mismas métricas que para tráfico CBR, aunque esta vez los retardos estarán referidos a tramas en lugar de a flits individuales.

En primer lugar, se comprobará la carga aceptada frente a la solicitada (Figura 6.13). Esta gráfica es muy diferente a la análoga obtenida anteriormente para tráfico CBR, en el rango de cargas medias a moderadamente altas (50 %-80 %). Ahora, cuando los flits son pequeños, la carga introducida al sistema es mayor que cuando los flits son grandes, y en torno a un 10 % mayor de la solicitada. Esto se debe a que las fuentes VBR generan siempre la misma cantidad de carga útil, 6 GOPS de las distintas secuencias de vídeo MPEG-2. Por tanto, en

[2]IAT = *Inter-Arrival Time*, tiempo entre llegadas de los flits de una conexión

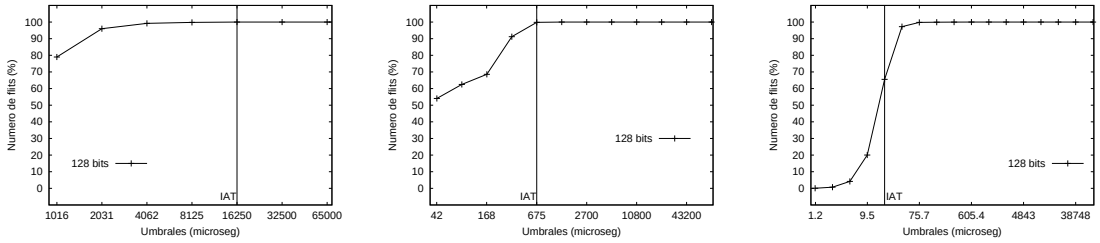


(a) 64 Kbps, carga = 72 % (b) 1.54 Mbps, carga = 72 % (b) 55 Mbps, carga = 72 %



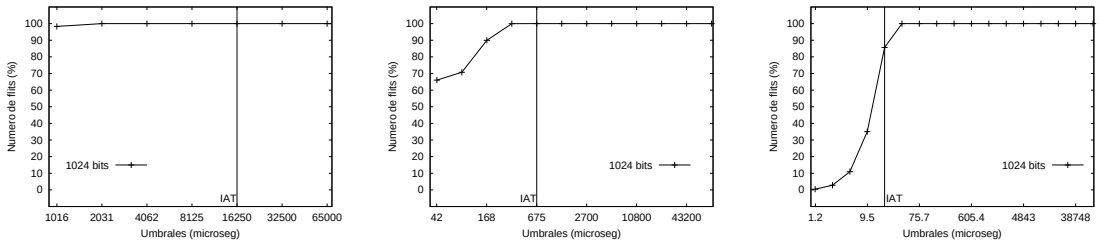
(a) 64 Kbps, carga = 76 % (b) 1.54 Mbps, carga = 76 % (b) 55 Mbps, carga = 76 %

Figura 6.10: Impacto del tamaño de flit - Tráfico CBR: Distribución del retardo medio de los flits desde su generación.



(a) 64 Kbps (b) 1.54 Mbps (b) 55 Mbps

Figura 6.11: Impacto del tamaño de flit - Tráfico CBR: Distribución del retardo medio de los flits desde su generación (flits de 128 bits, carga del 80 %).



(a) 64 Kbps (b) 1.54 Mbps (b) 55 Mbps

Figura 6.12: Impacto del tamaño de flit - Tráfico CBR: Distribución del retardo medio de los flits desde su generación (flits de 1024 bits, carga del 83 %).

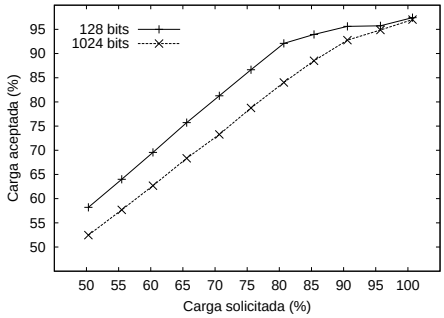
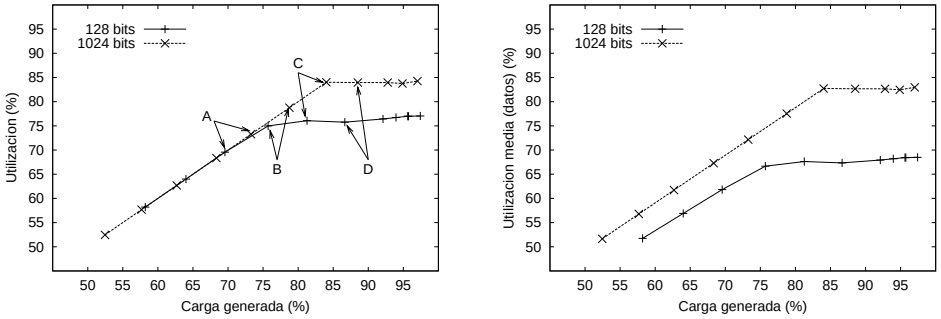


Figura 6.13: Impacto del tamaño de flit - Tráfico VBR: Carga aceptada frente a carga solicitada.



(a) Utilización total

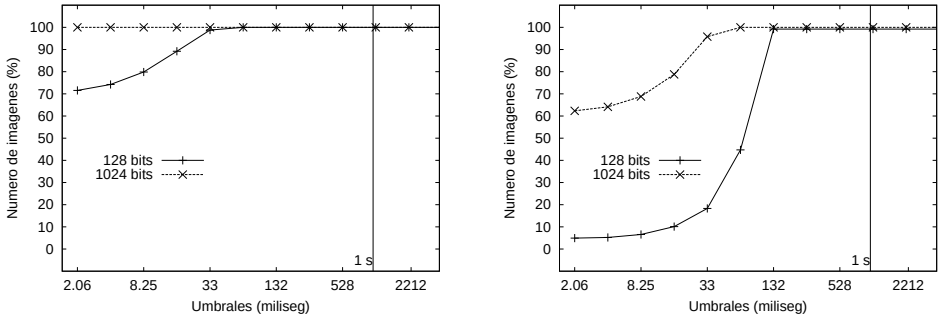
(b) Utilización por datos

Figura 6.14: Impacto del tamaño de flit - Tráfico VBR: Utilización media del *crossbar*.

ambos casos la cantidad de datos a transmitir es la misma. A esa carga se le ha de añadir la sobrecarga de control, que se haya justamente en torno al 10 % para flits de 128 bits, lo cual explica la diferencia de cargas reflejada en dicha gráfica. Dicha sobrecarga es de sólo el 1 % para flits de 1024 bits. A partir del 80 % de carga solicitada, la carga aceptada crece mucho más despacio para flits de 128 bits, debido al rechazo de conexiones. Cuando se emplean flits de 1024 bits, el efecto del rechazo de conexiones empieza a hacerse patente a partir de una carga solicitada del 90 %, aproximadamente. Esto es, al igual que sucedía con tráfico CBR, el uso de flits grandes permite aceptar un mayor número de conexiones cuando la carga es moderadamente elevada.

La cantidad de carga que el Encaminador es capaz de trasegar se refleja en las gráficas de la Figura 6.14 mediante la utilización media del *crossbar*. En la parte (a) de dicha gráfica, la utilización media total del *crossbar*, se puede observar como con flits pequeños el Encaminador entra en saturación mucho antes que con flits de mayor tamaño. En concreto, a partir de una carga total de entorno al 75 % (marcado como B en la Figura) el Encaminador ya no es capaz de dar salida a todos los datos que le llegan si se emplean flits pequeños. Por el contrario, si se emplean flits grandes, esto no ocurre hasta casi el 85 % de carga (marcado como C en la Figura). Si se descuenta la sobrecarga de control, esto se traduce en una utilización máxima neta de en torno al 65 % para flits de 128 bits, frente a algo más del 80 % obtenida con flits de 1024 bits (véase la parte (b) de la Figura).

Por último, queda por comprobar si las prestaciones temporales se mantienen con el uso de flits grandes. Para ello, se va a emplear la distribución de retardos de tramas desde la



(a) Puntos de carga A

(b) Puntos de carga B

(c) Puntos de carga C

(b) Puntos de carga D

Figura 6.15: Impacto del tamaño de flit - Tráfico VBR: Distribución de los retardos de imágenes desde la generación.

generación, medida en los puntos de carga marcados como A, B, C y D en la Figura 6.14-(a). Estos puntos se hayan en torno a los puntos de saturación. Las cotas de retardo consideradas son múltiplos y submúltiplos del tiempo de trama (TT), 33 milisegundos, y cubren un rango desde los 2.06 milisegundos ($TT/16$) hasta algo más de 2 segundos ($64 \times TT$). En las gráficas de la Figura 6.15 se muestran los resultados obtenidos para esta métrica. Hay que señalar que aunque para cada gráfica el nivel de carga no es el mismo en ambas configuraciones, siempre se ha empleado un punto de carga ligeramente inferior para el caso de flits pequeños. De esta forma, si se obtienen mejores prestaciones con flits grandes, se puede corroborar que efectivamente el Encaminador funciona mejor con ellos.

En el punto de carga A (Figura 6.15-(a)), el Encaminador se haya todavía lejos de saturarse para flits grandes, mientras que con flits pequeños se encuentra próximo a ella. Se puede apreciar como, en ese caso, existe un moderado porcentaje de tramas que no pueden cumplir con los umbrales más estrictos, mientras que todas las tramas cumplen la cota de retardo más estricta (2.06 milisegundos) cuando los flits son grandes. No obstante, la QoS recibida por las aplicaciones MPEG-2 con flits pequeños todavía es aceptable, pues todas las tramas obtienen retardos inferiores a 66 milisegundos, muy por debajo de la cota de 1 segundo, recomendada por el ATM Fórum para este tipo de servicios [143].

En el siguiente punto de carga considerado, el marcado como B (Figura 6.15-(b)), se aprecia una importante degradación en la cantidad de tramas que cumplen con las cotas más estrictas, para ambos tamaños de flit. La degradación es más importante para flits de 128 bits, pues se puede apreciar como incluso hay un pequeñísimo porcentaje de tramas que ni siquiera llegan a atravesar el Encaminador. Esto es señal de que éste ya está entrando en saturación en este nivel de carga (en torno al 76 %). Con flits grandes, por el contrario, todas

las tramas son retransmitidas a través del Encaminador, y todas ellas experimentan retardos inferiores a 66 milisegundos, con lo cual todavía se mantienen las garantías de QoS.

En el penúltimo punto de carga considerado (Figura 6.15-(c)), se aprecia claramente como el Encaminador ya está saturado para flits pequeños. Con flits grandes, aunque sólo una tercera parte de las tramas es capaz de obtener retardos inferiores a las cotas más estrictas (por debajo de 33 milisegundos), todavía se mantiene una cota medianamente razonable para la totalidad de las tramas, que es de 132 milisegundos. Esto ya no ocurre en el punto D (Figura 6.15-(d)), donde ya se aprecia como existen tramas incapaces de cruzar el Encaminador, incluso cuando se emplean flits grandes. Esto es, el máximo nivel de carga que el Encaminador es capaz de admitir sin comprometer las garantías de QoS de las aplicaciones de vídeo MPEG-2 se haya en torno al 80-84 % (entre los puntos B y C), siempre y cuando se empleen flits grandes.

Los resultados anteriores respecto al tráfico VBR se han obtenido empleando el modelo de inyección SR. Con el otro modelo de inyección implementado, el BB, también se han llevado a cabo las mismas pruebas, obteniendo resultados prácticamente idénticos, por lo que no se han incluido las gráficas correspondientes.

Se puede concluir por tanto, que **los flits de mayor tamaño son preferibles**. No sólo mejoran el aprovechamiento del ancho de banda de los enlaces, sino que también se ha demostrado que con ellos el Encaminador es capaz de ofrecer un mayor rango de carga útil con garantías de QoS tanto a las aplicaciones que generan tráfico CBR, como a las que generan tráfico MPEG-2 de tipo VBR.

Por otro lado, el uso de flits grandes da un mayor margen de funcionamiento a los algoritmos de planificación, pues así disponen del tiempo necesario para calcular los emparejamientos entre los canales de entrada y salida del Encaminador.

6.4.3. Tamaño del ciclo del planificador

El siguiente parámetro a evaluar, el tamaño de ciclo del planificador, determinado por el valor del parámetro K, está intrínsecamente relacionado con el mecanismo de reserva del ancho de banda en el MMR. Este valor determina la granularidad de la asignación de ancho de banda a las distintas conexiones.

Se van a analizar tres valores de este parámetro: 4, 16 y 64. El tamaño de flit se fija a 1024 bits, y los *buffers* serán de 1 flit. Con dichos valores de K, y dado que el número de canales virtuales por enlace físico es siempre de 128, se obtienen los datos mostrados en la Tabla 6.4. En esta Tabla se aprecia como al incrementar el valor de K, la unidad de asignación de ancho de banda se hace más fina. Esto posibilitará ajustar mejor la reserva de ancho de banda realizada por las conexiones a su consumo real. La consecuencia lógica es que al ajustar mejor la reserva al consumo real, cuando la carga sea moderada, será posible admitir un mayor número de conexiones, o mejor dicho, una mayor cantidad de ancho de banda real.

Para corroborar lo anterior, en la Figura 6.16 se muestra la carga aceptada en el simulador, una vez ejecutado el algoritmo de control de admisión de conexiones, frente a la carga solicitada, para simulaciones realizadas con tráfico CBR. Se puede observar como, cuando la carga solicitada sobrepasa el 75 % empiezan a haber diferencias entre el valor más bajo de K y los otros dos. Esto se debe al factor de granularidad introducido por K. Con un valor de 4, la alta granularidad hace que por ejemplo, para cada conexión de 64 Kbps, se reserven realmente 2.42 Mbps, casi 38 veces más de lo necesario. Esto puede suponer un gran desperdicio de ancho de banda. A cargas medias y bajas, el efecto de esta "super-reserva" no

K	Tamaño del ciclo del planificador	Unidad de ancho de banda asignable
4	$4 \times 128 = 512$ ciclos de flit	2.42 Mbps
16	$16 \times 128 = 2048$ ciclos de flit	0.60 Mbps
64	$64 \times 128 = 8192$ ciclos de flit	0.15 Mbps

Tabla 6.4: Relación entre K y la asignación de ancho de banda.

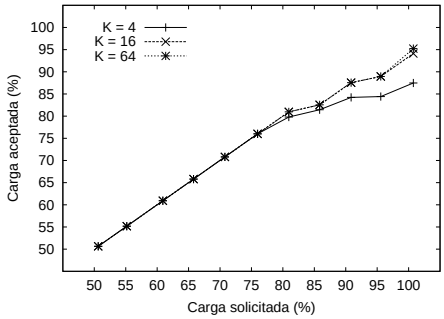


Figura 6.16: Impacto de K - Tráfico CBR: Carga aceptada frente a carga solicitada.

se nota, pues queda espacio para seguir adjudicando ancho de banda a las aplicaciones. Pero con cargas moderadas, este efecto se empieza a hacer patente, tal y como se confirma con la gráfica de la Figura 6.16. El efecto de la granularidad no se manifiesta tan drásticamente cuando el valor de K pasa de 16 a 64. Únicamente, cuando se solicita un 100 % de carga, se rechaza una mínima parte del ancho de banda debido a este efecto.

Por otro lado, en la franja de carga común a las tres configuraciones, el encaminador no se llega a saturar, y es capaz de trasegar toda la información que le llega. Esto se aprecia en la Figura 6.17, mediante la utilización media del *crossbar*.

Ahora es turno de comprobar que las prestaciones temporales no se ven comprometidas por incrementar el valor de K. Se van a utilizar para ello la distribución de los retardos de flit desde la generación. En la Figura 6.18 se muestran los valores de esta magnitud para los distintos tipos de conexiones en el punto de carga más elevado común a las tres configuraciones, esto es, en torno al 76 %. Se puede observar que no existen diferencias

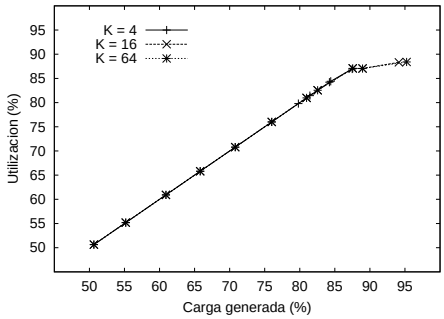
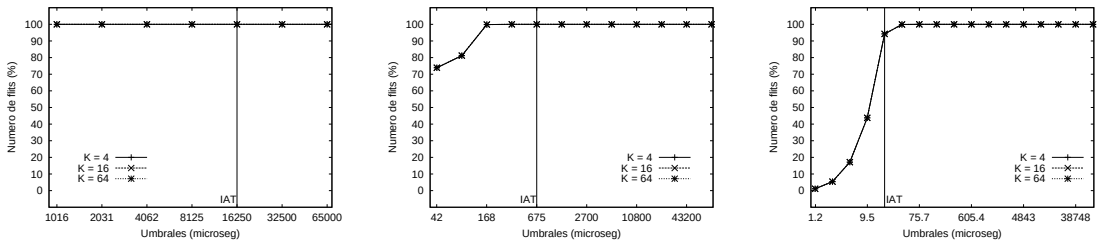
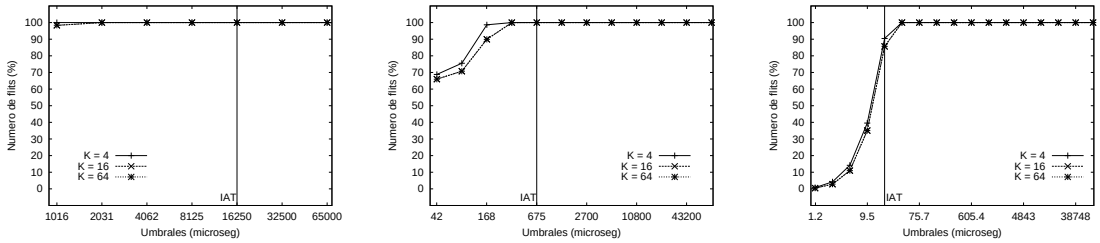


Figura 6.17: Impacto de K - Tráfico CBR: Utilización media del *crossbar*.



(a) 64 Kbps (b) 1.54 Mbps (b) 55 Mbps

Figura 6.18: Impacto de K - Tráfico CBR: Distribución del retardo medio de los flits desde su generación. Carga = 76 %.



(a) 64 Kbps (b) 1.54 Mbps (b) 55 Mbps

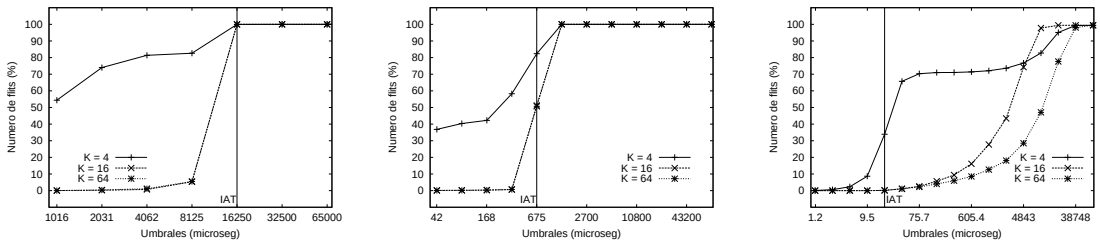
Figura 6.19: Impacto de K - Tráfico CBR: Distribución del retardo medio de los flits desde su generación. Carga = 82 %.

apreciables, y que en los tres casos se proporcionan suficientes garantías de QoS.

Cuando la carga se incrementa hasta niveles cercanos a saturación, los puntos de carga no son exactamente idénticos, pero existen puntos comunes alrededor del 82 % y del 87 %. Los resultados en cuanto a distribución de retardos de flit se muestran para estos dos puntos en las Figuras 6.19 y 6.20, respectivamente. Para el 82 % de carga (Figura 6.19), las diferencias entre los resultados obtenidos para los 3 valores de K son pequeñas. Parece que existe una ligera ventaja para un valor de K de 4. Esta diferencia se acentúa al aumentar la carga hasta el 87 % (Figura 6.20), aunque para este nivel de carga ya se observa que para todos los valores de K se ha entrado en saturación, pues la mayoría de los flits sólo son capaces de cumplir con las cotas más relajadas.

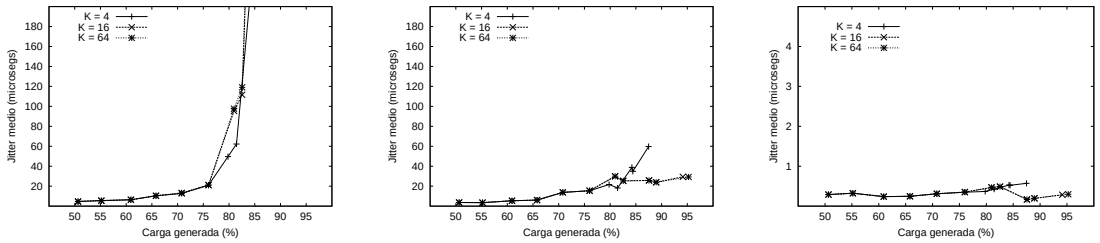
También es interesante comprobar el efecto que tiene el tamaño del ciclo del planificador sobre el *jitter*, o variación en el retardo. Esta magnitud se muestra en las gráficas de la Figura 6.21. Al igual que sucedía con los retardos, en los valores de *jitter* no existen diferencias apreciables en el rango de carga común para las tres configuraciones simuladas. Cuando la carga pasa del 76 %, empiezan a existir ligeras diferencias. En el caso de las conexiones con bajos requerimientos de ancho de banda, la diferencia es favorable para un valor de K de 4, mientras que sucede al contrario para las conexiones de medios y altos requerimientos.

Se puede concluir por tanto, que un valor del parámetro K reducido, impide la admisión de conexiones para las cuales todavía existe ancho de banda efectivo en los enlaces del encaminador. En el rango de carga donde este efecto no influye en el número de conexiones aceptadas, no existe diferencia alguna en las prestaciones temporales que experimentan los flits CBR.



(a) 64 Kbps (b) 1.54 Mbps (b) 55 Mbps

Figura 6.20: Impacto de K - Tráfico CBR: Distribución del retardo medio de los flits desde su generación. Carga = 87 %.



(a) 64 Kbps (b) 1.54 Mbps (b) 55 Mbps

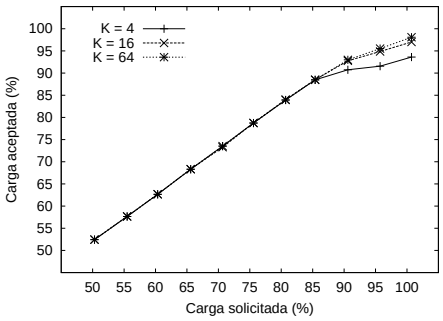
Figura 6.21: Impacto de K - Tráfico CBR: *Jitter* medio de los flits.

Por encima del 76 % de carga, se aprecia el efecto de la alta granularidad de asignación del ancho de banda de los enlaces sobre la cantidad de éste que resulta rechazado, junto con una ligera mejora de prestaciones para valores bajos de K. Señalar también que no existen diferencias apreciables en el comportamiento del encaminador cuando el valor de K pasa de 16 a 64, excepto cuando el encaminador ha entrado en saturación, y para los flits de conexiones de requerimientos más altos.

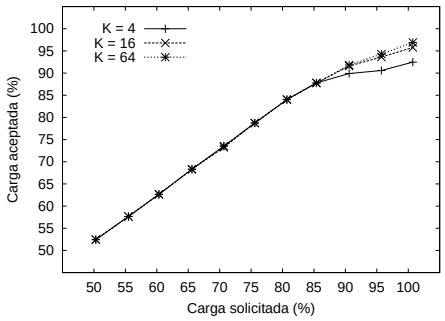
A continuación se analizará el efecto del valor de K sobre las prestaciones del tráfico VBR. Para ello, se seguirá el mismo procedimiento adoptado para el análisis con tráfico CBR, salvo que las medidas temporales (retardos y *jitter*) se tomarán sobre tramas de vídeo, en lugar de sobre flits individuales.

En primer lugar, se comprueba que se produce el mismo efecto que sucedía con tráfico CBR sobre la admisión de carga. En las gráficas de la Figura 6.22 se observa como efectivamente, a partir de un 84 % de carga solicitada, con $K = 4$ existe una mayor cantidad de carga rechazada, que con valores de K superiores. También existe una ligerísima diferencia en el mismo sentido entre el comportamiento obtenido para los valores de $K = 16$ y $K = 64$, pero es prácticamente despreciable. Esto sucede tanto para el modelo de inyección SR (parte (a) de la Figura), como para el BB (parte (b) de la Figura).

En cuanto a la utilización media del *crossbar* (Figura 6.23), no existen diferencias debidas al valor de K, en el rango de carga común. Para los tres valores analizados, el *crossbar* se satura en torno al 84 % de carga. Y justamente es a partir de ese punto de carga cuando se aprecia el efecto de K sobre el rechazo de conexiones. Esto es, en el rango de carga útil, previo a la saturación del *crossbar*, el valor de K no influye en la cantidad de carga que es capaz de trasegar el *crossbar*. Queda comprobar si existe alguna influencia sobre las garantías de QoS

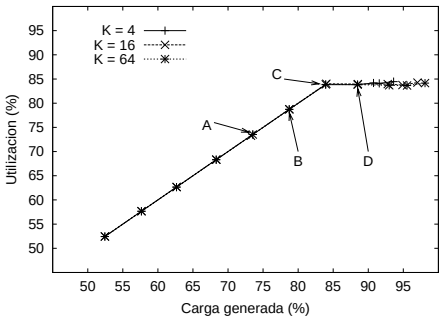


(a) Modelo de inyección SR

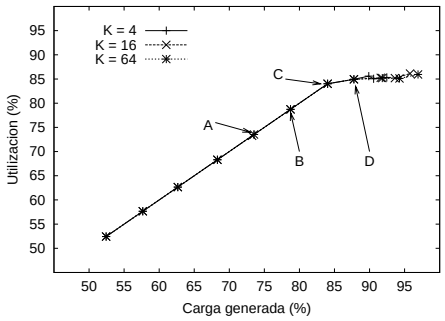


(a) Modelo de inyección BB

Figura 6.22: Impacto de K - Tráfico VBR: Carga aceptada frente a carga solicitada.

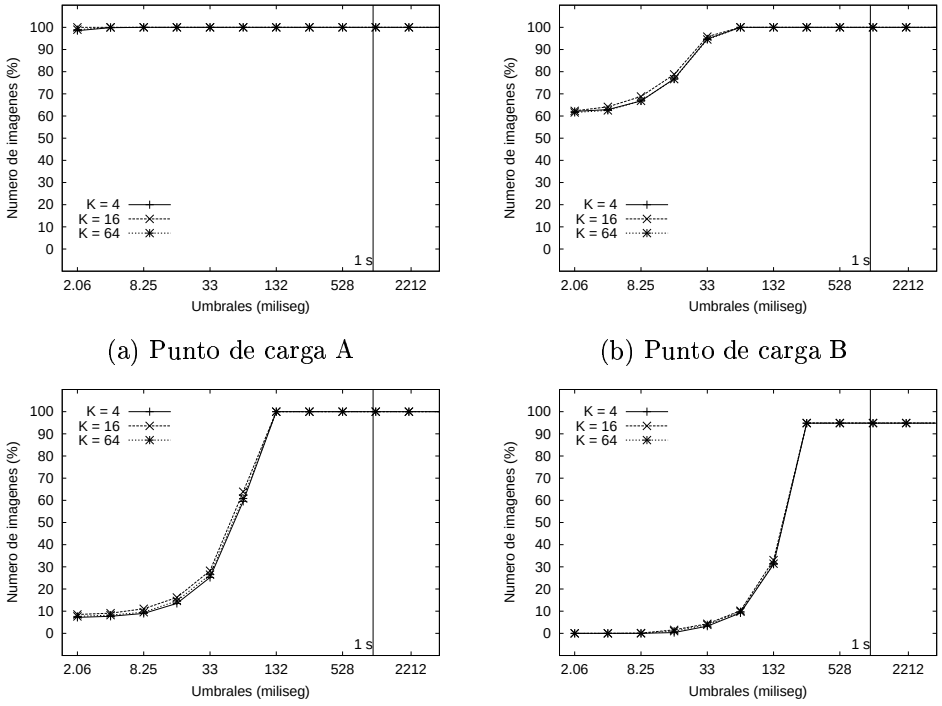


(a) Modelo de inyección SR



(a) Modelo de inyección BB

Figura 6.23: Impacto de K - Tráfico VBR: Utilización media del *crossbar*.



(a) Punto de carga A

(b) Punto de carga B

(c) Punto de carga C

(b) Punto de carga D

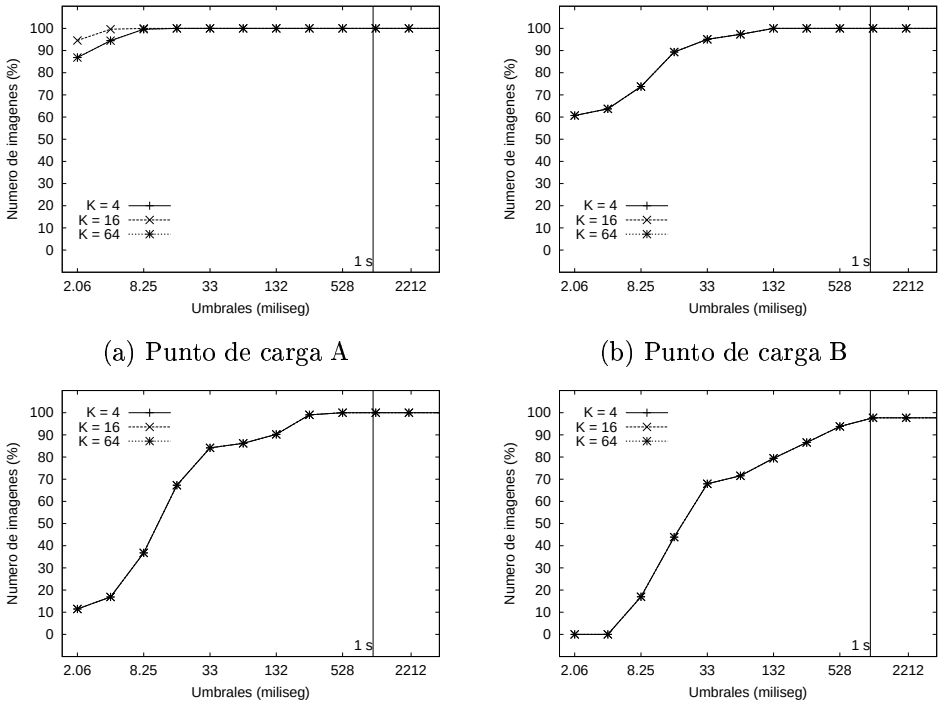
Figura 6.24: Impacto de K - Tráfico VBR (SR): Distribución de los retardos de imágenes desde la generación.

de las aplicaciones.

Para comprobar las prestaciones que reciben las aplicaciones, se va a utilizar la distribución del retardo de las tramas de vídeo desde su generación. Esta métrica se presenta en las Figuras 6.24 y 6.25, para los modelos de inyección SR y BB, respectivamente. Se muestran los resultados obtenidos para cuatro niveles de carga en torno al punto de saturación, marcados como A, B, C y D en la Figura 6.23. Las cotas de retardo consideradas son múltiplos y submúltiplos del tiempo de trama (TT), 33 milisegundos, y cubren un rango desde los 2.06 milisegundos ($TT/16$) hasta algo más de 2 segundos ($64 \times TT$).

Tanto para el modelo SR como para el BB, el resultado es similar: el valor de K apenas influye en los retardos experimentados por las tramas. Para ambos modelos de inyección, el comportamiento es similar. En el punto A (carga del 74 %, aproximadamente), el Encaminador está funcionando normalmente, y prácticamente todas las tramas cumplen con los umbrales más estrictos. En el punto B (carga del 79 %, aproximadamente), se empiezan a notar los efectos de la congestión, pues sólo las cotas de 66 y 132 milisegundos son válidas para todas las tramas, para cada modelo de inyección. El punto C (carga del 84 %, aproximadamente) se haya muy próximo a la saturación, aunque todas las tramas todavía cumplen con cotas varias veces inferiores al límite recomendado por el ATM Fórum de 1 segundo [143]. Finalmente, en el nivel de carga D (carga del 89 %, aproximadamente), ya se ha entrado en saturación, y existen tramas que no son capaces de atravesar el Encaminador. En conclusión, el Encaminador es capaz de ofrecer las garantías de QoS necesarias para las aplicaciones hasta un nivel de carga aproximado de 84 %, independientemente del tamaño del ciclo del planificador.

Por último, se analizará el *jitter* o variación en el retardo experimentado por las tramas



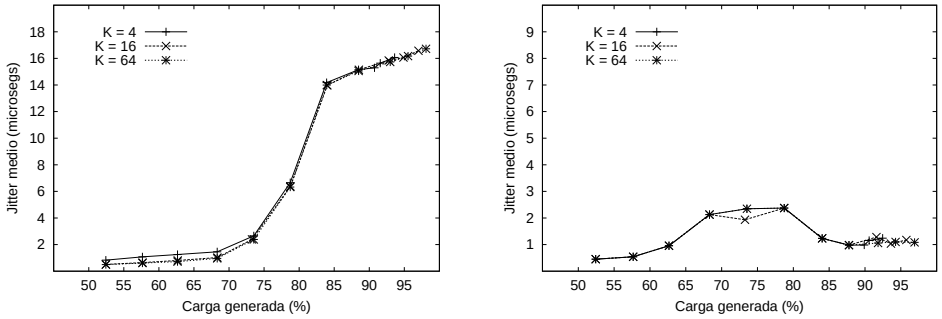
(a) Punto de carga A

(b) Punto de carga B

(c) Punto de carga C

(b) Punto de carga D

Figura 6.25: Impacto de K - Tráfico VBR (BB): Distribución de los retardos de imágenes desde la generación.



(a) Modelo de inyección SR (a) Modelo de inyección BB

Figura 6.26: Impacto de K - Tráfico VBR: *Jitter* medio de imágenes.

de vídeo. Los resultados se muestran en la Figura 6.26 para ambos modelos de inyección. De nuevo se observa como el valor de K apenas influye en las prestaciones temporales de las aplicaciones de vídeo. Además, en todo el rango de carga simulado, el *jitter* no crece de forma incontrolada, incluso cuando ya se ha entrado en saturación.

En conclusión, se ha comprobado que el uso de valores de K pequeños, produce una elevada granularidad en la reserva de ancho de banda que provoca el rechazo de un mayor número de conexiones a cargas moderadas. Este efecto es mayor en la pruebas realizadas con tráfico CBR.

En cuanto a las prestaciones temporales, apenas se ven afectadas por el valor de K. Sólo en el caso de tráfico CBR, se ha apreciado un ligero empeoramiento al hacer mayor el tamaño del ciclo de planificación. El motivo de esta mínima influencia es que realmente las decisiones de planificación se toman en cada ciclo de flit, y no a nivel global de ciclo de planificador. Luego el tamaño de éste sólo afecta a la reserva de ancho de banda por parte de las conexiones. Los resultados, por tanto, sugieren el empleo del valor intermedio simulado, esto es, **un valor de K igual a 16**.

6.4.4. Tamaño del *buffer*

Por último, queda por determinar el tamaño de los *buffers* asociados a cada canal virtual del MMR. Los *buffer* pequeños son preferibles, debido a la gran cantidad de canales virtuales, y al objetivo de realizar un diseño de encaminador compacto.

Las simulaciones de este apartado se han ejecutado con un valor para K de 16, y un tamaño de flit de 1024 bits. En cuanto a los tamaños de *buffer* empleados han sido tres: 1, 4 y 1024 flits. Los dos primeros sirven para acotar el tamaño mínimo de *buffer* necesario en el MMR para ofrecer la QoS requerida por las aplicaciones. El último tamaño se ha probado con la idea de que sea capaz de almacenar una imagen de vídeo MPEG-2 completa, por si eso tuviera alguna influencia en las prestaciones obtenidas. La imagen más grande empleada en las simulaciones contiene algo menos de 1000 flits, cuando éstos son de 1024 bits.

Las primeras pruebas se han realizado con tráfico CBR. Los resultados se muestran en las Figuras 6.27 (utilización media del *crossbar*), 6.28 (retardo medio de flits desde la generación) y 6.29 (*jitter* medio de flits). Como era de esperar, no existen diferencias apreciables al variar el tamaño de los *buffers* cuando sólo existe tráfico CBR. El motivo es que las conexiones sólo se admiten si hay suficientes recursos para atenderlas, y además sus requerimientos no varían

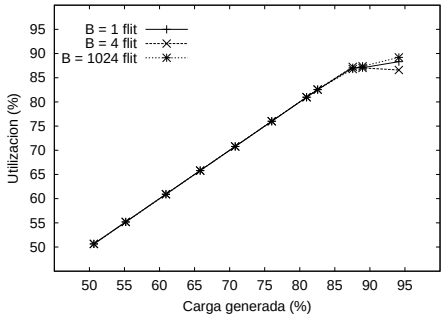
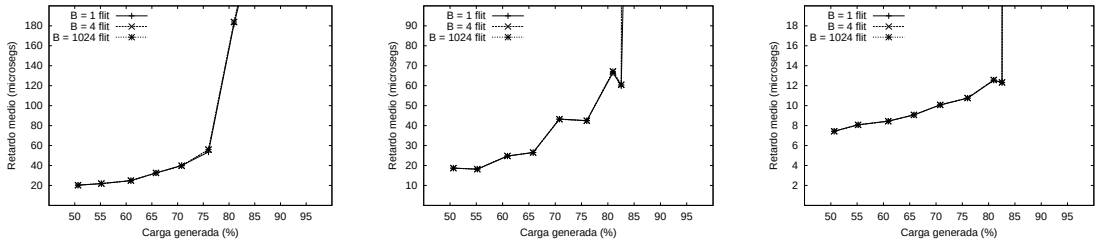


Figura 6.27: Impacto del tamaño de *buffer* - Tráfico CBR: Utilización media del *crossbar*.

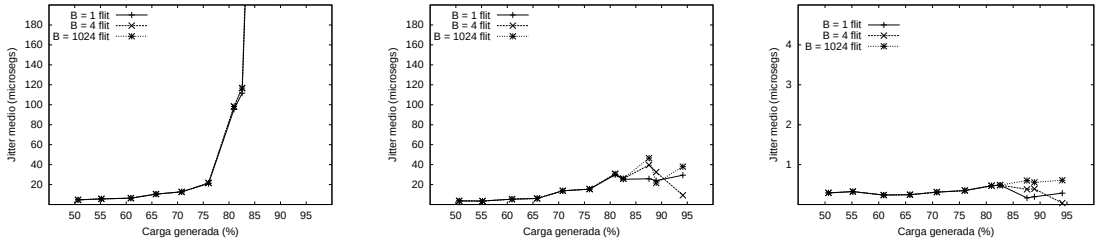


(a) 64 Kbps (b) 1.54 Mbps (b) 55 Mbps

Figura 6.28: Impacto del tamaño de *buffer* - Tráfico CBR: Retardo medio de los flits desde generación.

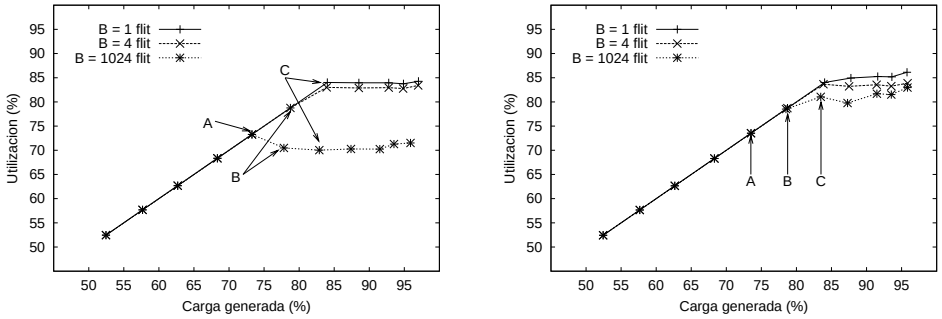
en el tiempo. Por tanto, mientras no exista saturación, los flits atraviesan el encaminador sin apenas formar colas. Por tanto, el incremento en tamaño de *buffer* no afecta a las prestaciones del tráfico CBR, y basta con tener *buffers* con capacidad para un flit.

El análisis de las prestaciones del Encaminador Multimedia al variar el tamaño de sus *buffers* resulta más interesante cuando se emplea tráfico VBR. Las simulaciones se han llevado a cabo empleando los dos modelos de inyección descritos en la Sección 6.2.1: el modelo SR, que realiza una inyección "suave" de los flits que componen cada trama de vídeo MPEG-2, y el modelo BB, que inyecta dichos flits a una tasa pico, al inicio del tiempo de trama. El primer modelo introduce una variación en el tiempo entre llegadas de los flits (el IAT) de la conexión por cada trama de vídeo, mientras que el segundo somete el encaminador a una



(a) 64 Kbps (b) 1.54 Mbps (b) 55 Mbps

Figura 6.29: Impacto del tamaño de *buffer* - Tráfico CBR: *Jitter* medio de los flits.



(a) Modelo de inyección SR (b) Modelo de inyección BB

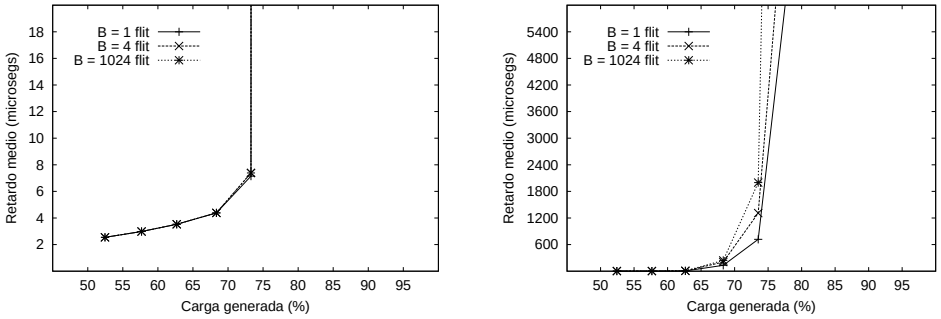
Figura 6.30: Impacto del tamaño de *buffer* - Tráfico VBR: Utilización media del *crossbar*.

carga compuesta de ráfagas, debido a que la inyección de flits se concentra en determinados instantes de tiempo. Sin embargo, el IAT para la conexión no varía a lo largo del tiempo.

En la Figura 6.30 se muestra la utilización media del *crossbar* para ambos modelos de inyección. El comportamiento para los tamaños de *buffer* de 1 y 4 es muy similar. En ambos casos, el encaminador entra en saturación a una carga del 84 %, aproximadamente. Además, en esos casos no existe degradación de prestaciones más allá del punto de saturación. Las fuentes llegan a generar hasta casi el 100 % de carga, pero más allá de una utilización del 84 %, el control de flujo hace que los flits que no caben en los *buffers* del MMR (que están llenos) se queden en los *buffers* de la tarjeta de red. Sorprendentemente, la utilización cae fuertemente cuando se emplean *buffers* de 1024 bits con el modelo de inyección SR. Un motivo es la degradación del mecanismo de prioridades cuando las colas en el MMR se hacen largas.

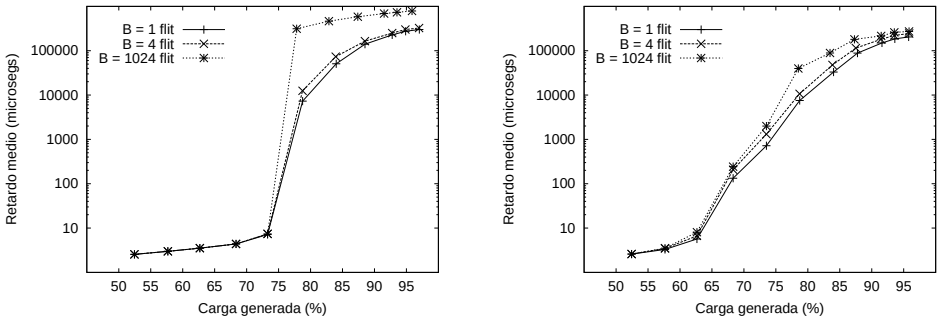
Hay que recordar que el mecanismo de cálculo de prioridad empleado en el algoritmo IABP tiene en cuenta el retardo en colas del MMR experimentado por cada flit para calcular la prioridad. Por tanto, existe un contador asociado a cada flit de cada *buffer* que almacena el valor de dicho retardo. Considérese el caso de que haya contención y los flits empiecen a formar colas. Cuando los *buffers* son grandes, la probabilidad de que los flits tengan que esperar a su turno para ser transmitidos dentro de las colas del MMR es mayor. Esto implica que su retardo en cola será más elevado que el que experimentarían si los *buffers* fuesen más pequeños. En ese caso, los flits pasarían más tiempo en los *buffers* de la tarjeta de red, y su retardo en cola dentro del encaminador no se incrementaría tanto. La consecuencia es que, cuando se emplean *buffers* grandes, los contadores se pueden desbordar con una probabilidad mayor, pues el retardo en cola puede alcanzar valores más elevados. De ahí que el mecanismo de prioridades pueda corromperse. Esta degradación también afecta ligeramente a las prestaciones obtenidas con el modelo BB.

Además, para el modelo SR, el tiempo entre llegadas de cada conexión (el IAT), también considerado por el algoritmo IABP para el cálculo de las prioridades de los flits, se modifica para cada imagen de vídeo. Esta modificación se transmite como un mensaje de control. Los mensajes de control se transmiten independientemente de los flits de la conexión, con la mayor prioridad, y por tanto, pueden llegar al encaminador antes de que todos los flits de la imagen anterior hayan atravesado el *crossbar*. El efecto es que, por ejemplo, cuando el encaminador está próximo a saturarse, y los flits permanecen más tiempo en las colas del MMR, los flits pertenecientes a una imagen de tipo I (las que más ancho de banda ocupan) son tratados como si perteneciesen a una imagen de tipo B (las que menos ancho



(a) Modelo de inyección SR (b) Modelo de inyección BB

Figura 6.31: Impacto del tamaño de *buffer* - Tráfico VBR: Retardo medio de imágenes desde su generación.



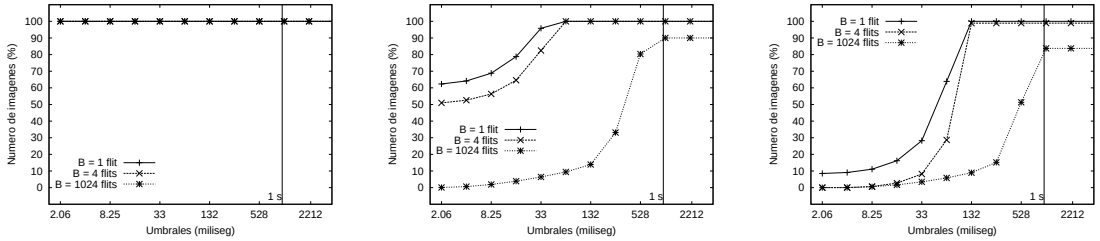
(a) Modelo de inyección SR (b) Modelo de inyección BB

Figura 6.32: Impacto del tamaño de *buffer* - Tráfico VBR: Retardo medio de imágenes desde su generación. Escala semi-logarítmica.

de banda consumen), y su valor de prioridad se incrementa más despacio. Por tanto, no llegan a transmitirse tan rápidamente como debieran. Este efecto, junto con la corrupción del mecanismo de cálculo de la prioridad, explica la degradación de prestaciones observada para *buffers* de 1024 flits, con el modelo de inyección SR.

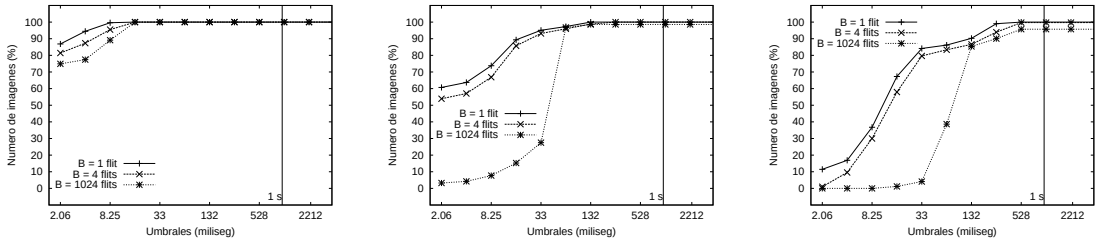
A continuación, hay que comprobar que las conexiones reciben la QoS que necesitan. Para ello, se emplearán medidas de retardo y *jitter*. Como se señaló en la Sección 6.3, para el tráfico VBR tanto el retardo como el *jitter* se miden sobre imágenes o tramas de vídeo, en lugar de sobre flits individuales.

En la Figura 6.31 se muestra el retardo medio de las tramas desde su generación, para los dos modelos de inyección. Nótense las diferentes escalas en los ejes verticales. Con el modelo de inyección SR, para cargas inferiores al 74 %, los retardos experimentados por las tramas son muy bajos. Para el punto de carga del 74 %, el incremento es ligeramente más acusado. Hay dos razones que explican este comportamiento. En primer lugar, existe mayor contención en el *crossbar* por el uso de los puertos de salida. En segundo lugar, y lo más importante, puesto que el tráfico no es uniforme, el encaminador entra en saturación cuando se transmiten tramas I, provocando retardos mayores. Más tarde, mientras se transmiten el resto de tramas, el planificador es capaz de transmitir todos los flits con retardos menores. Este efecto se hace patente a cargas algo más bajas cuando se emplea el modelo BB, puesto que además en ese



(a) Punto de carga A (74 %) (b) Punto de carga B (78 %) (c) Punto de carga C (84 %)

Figura 6.33: Impacto del tamaño de *buffer* - Tráfico VBR: Distribución del retardo medio de las imágenes desde su generación. Modelo de inyección SR.



(a) Punto de carga A (74 %) (b) Punto de carga B (78 %) (c) Punto de carga C (84 %)

Figura 6.34: Impacto del tamaño de *buffer* - Tráfico VBR: Distribución del retardo medio de las imágenes desde su generación. Modelo de inyección BB.

modelo la inyección de carga se concentra en los instantes iniciales del tiempo de trama. Para cargas superiores al 80 %, el encaminador entra claramente en saturación, y las tramas experimentan retardos superiores a cualquier cota válida.

Por otro lado, para el modelo de inyección SR, puede observarse que el retardo medio de las imágenes no se ve afectado por la variación en el tamaño de los *buffers* mientras la carga es media a moderada (hasta el 74 %). Por el contrario, cuando se emplea el modelo de inyección BB, los retardos medios experimentados por las tramas antes de la saturación son algo mayores, y empeoran cuando el tamaño del *buffer* se incrementa. El motivo es que el modelo de inyección BB concentra la inyección de flits al principio de cada tiempo de trama, provocando un periodo de saturación transitorio. Se ha mostrado previamente que los *buffers* pequeños proporcionan mejores prestaciones cuando se alcanza la saturación. Esto explica los mejores retardos medios obtenidos para *buffers* de 1 y 4 flits. Cuando las fuentes terminan de inyectar sus tramas, la saturación se difumina hasta el inicio del siguiente tiempo de trama. Pero los flits que ya se han inyectado sufrirán de esta congestión pasajera, como puede comprobarse por el incremento en el retardo medio de trama. Todo esto se aprecia mejor en las gráficas de la Figura 6.32, con el eje vertical de tiempos en escala logarítmica.

A continuación hay que comprobar cuantas tramas experimentan retardos inferiores a un conjunto de cotas razonables para el tipo de conexión que representan. Esto da una medida más exacta de la QoS que están recibiendo las conexiones MPEG-2, que la proporcionada por los retardos medios. Los resultados se muestran en la Figura 6.33, para el modelo de inyección SR, y en la Figura 6.34, para el modelo de inyección BB. Se muestran los niveles de carga más próximos al punto de saturación, marcados como A, B y C en la Figura 6.30. Las cotas de retardo consideradas son múltiplos y submúltiplos del tiempo de trama (TT),

33 milisegundos, y cubren un rango desde los 2.06 milisegundos ($TT/16$) hasta algo más de 2 segundos ($64 \times TT$).

En estas gráficas, y para ambos modelos de tráfico, puede comprobarse como efectivamente, las mejores prestaciones se obtienen siempre con *buffers* pequeños.

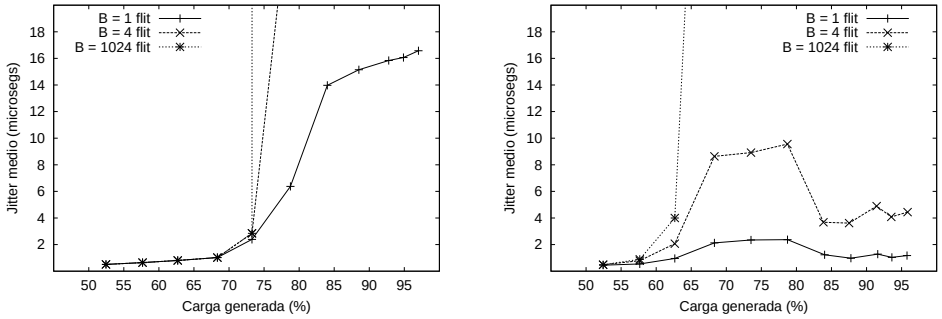
Cuando el encaminador todavía no se ha saturado (nivel de carga A), con el modelo de inyección SR casi todas las tramas son capaces de cumplir con la cota más estricta, de unos 2 milisegundos, sea cual sea el tamaño de *buffer* empleado. El modelo de inyección BB genera un tipo de carga más exigente, y sus prestaciones son algo peores. Además, en este caso sí que se aprecian diferencias entre los tres tamaños de *buffer*, ofreciendo mejores resultados los *buffers* pequeños. Concretamente, con *buffers* de 1 flit, todas las tramas experimentan retardos inferiores a 8.25 milisegundos. Para los otros dos tamaños considerados, la cota mínima cumplida para todas las tramas pasa a ser de 16.5 milisegundos, aunque para *buffers* de 4 flits, hay más tramas que cumplen las cotas más estrictas. En cualquier caso, estos valores son más que aceptables, pues el ATM Fórum recomienda un valor de 1 segundo como cota del retardo entre extremos para los servicios de transmisión de vídeo MPEG-2 [143].

Para el punto de carga marcado como B en la Figura 6.30, cuando se emplean *buffers* pequeños y el modelo de inyección SR, todas las tramas sufren retardos inferiores a 66 milisegundos, esto es, dos veces el tiempo de trama. Más aún, si los *buffers* son de 1 flit, el porcentaje de tramas que cumplen con cotas más estrictas se incrementa. Sin embargo, cuando los *buffers* son grandes, esto no se cumple. Es más, existen tramas generadas que ni siquiera llegan a atravesar el encaminador, lo cuál corrobora la degradación en la utilización media del *crossbar* observada para el modelo SR, con este tamaño de *buffers*. Esto es, para este nivel de carga, cuando se emplea el modelo de inyección SR y *buffers* grandes, el encaminador está claramente saturado. Por tanto, el empleo de *buffers* grandes en este caso impide el cumplimiento de las garantías de QoS que necesitan las conexiones MPEG-2. Para el modelo de inyección BB, y *buffers* de 1 flit todas las tramas sufren retardos inferiores a 132 milisegundos. Cuando los *buffers* son de 4 flits, el comportamiento es ligeramente peor. Y con *buffers* grandes, se aprecia como no todas las tramas han sido retransmitidas, indicando que el encaminador está entrando en saturación.

Para el último punto de carga considerado, marcado como C en la Figura 6.30, el encaminador está prácticamente saturado, aunque el uso de *buffers* pequeños hace que hayan más tramas capaces de cumplir con cotas de retardo más ajustadas. Es más, el encaminador es capaz de transmitir todas las tramas generadas con el modelo SR únicamente si se emplean *buffers* de 1 flit. En ese caso, todas las tramas son capaces de cumplir con cotas de 132 milisegundos, mientras que con el modelo BB todas las tramas experimentan retardos inferiores a medio segundo cuando se emplean *buffers* pequeños. Sin embargo, el porcentaje de tramas capaces de cumplir con cotas más estrictas disminuye significativamente con respecto al del punto de carga anterior.

En cualquier caso, estos valores son más que aceptables, pues el ATM Fórum recomienda un valor de 1 segundo como cota del retardo entre extremos para los servicios de transmisión de vídeo MPEG-2 [143]. Por tanto, se podría delimitar el rango de carga útil hasta el 80 % de carga, aproximadamente, si se usan *buffers* pequeños.

Otra medida interesante para comprobar la QoS recibida por las conexiones multimedia es el *jitter* o variación en el retardo experimentado por las tramas de vídeo. El valor medio para esta medida se muestra en la Figura 6.35, para los dos modelos de inyección considerados. En el rango de carga útil determinado en párrafos anteriores, esto, cargas de hasta el 80 %, aproximadamente, se puede observar como en los dos casos el *jitter* se mantiene acotado cuando se emplean *buffers* de 1 flit. En ese caso, el valor del *jitter* se sitúa por debajo



(a) Modelo de inyección SR (b) Modelo de inyección BB

Figura 6.35: Impacto del tamaño de *buffer* - Tráfico VBR: *Jitter* medio de imágenes.

de los 8 microsegundos para el modelo SR, y por debajo de los 10 microsegundos, en el modelo BB. Más aún, para cargas inferiores al 70 %, el *jitter* es inferior a 1 microsegundo para todos los tamaños de *buffer*, cuando se emplea el modelo SR. Para el modelo BB, cuando la carga es menor del 60 %, el valor del *jitter* también se mantiene por debajo del microsegundo, independientemente del tamaño de *buffer*. Pero a partir de ese punto, el *jitter* aumenta con el tamaño de *buffer*, creciendo de forma incontrolada cuando se emplean *buffers* grandes. Este comportamiento es consecuente con lo observado en el retardo experimentado por las tramas de vídeo. Con *buffers* pequeños, el mecanismo de cálculo de la prioridad funciona correctamente, y por tanto, todas las tramas sufren retardos similares, con lo que el *jitter* se mantiene acotado. Estos resultados son muy aceptables, pues el *jitter* permitido en transmisiones de vídeo MPEG-2 es de varios milisegundos, esto es, la variación en el retardo debe ser lo suficientemente baja como para que una persona pueda ver la secuencia de forma continua, a un ritmo regular. Además, existen técnicas de absorción del *jitter* [133], que pueden emplearse en el destino para compensar valores de *jitter* de hasta varios milisegundos.

En conclusión, **el tamaño de *buffer* se fijará a 1 flit** para los estudios siguientes, pues ha quedado demostrado que, para la configuración simulada, es el que mejor satisface las necesidades de QoS impuestas por las aplicaciones.

6.5. Evaluación de los algoritmos de planificación de enlaces

En el Capítulo 5 se introdujeron los algoritmos de planificación de recursos empleados en el Encaminador Multimedia MMR. Estos algoritmos se dividen en dos partes claramente diferenciadas, el algoritmo de planificación de enlaces y el algoritmo de planificación del conmutador, con misiones distintas y complementarias. Asimismo, se describieron varias propuestas de algoritmos, tanto para la planificación de los enlaces, como para la planificación del conmutador.

Una vez determinado el valor de los parámetros básicos de diseño en la Sección previa, llega el turno de analizar detalladamente las prestaciones de los algoritmos de planificación propuestos. En primer lugar, el análisis se centrará sobre los algoritmos de planificación de enlaces. Estos algoritmos consisten en la selección de un conjunto de canales virtuales por cada enlace físico, que serán los que tendrá en cuenta posteriormente el algoritmo de planificación del conmutador. El criterio para efectuar la selección se basa en la asignación de unos valores de prioridad dinámica, que varían en el tiempo según las necesidades de QoS

de cada flit.

En la Sección 5.1 se introdujeron tres criterios distintos para calcular la prioridad de los flits, que dan lugar a tres algoritmos de planificación distintos:

IABP (*Inter-Arrival Biased Priority*) La prioridad se relaciona con el ancho de banda de cada conexión, expresado como su IAT o tiempo entre llegadas de dos flits consecutivos.

JBP (*Jitter Biased Priority*) Se emplea una estimación de la variación en el retardo o *jitter* de la conexión para calcular la prioridad.

SIABP (*Simple IABP*) Aproximación a IABP, mucho más simple de implementar en hardware.

En esta Sección se van a analizar las prestaciones obtenidas con cada uno de estos algoritmos, manteniendo un algoritmo de planificación del *crossbar* común. En concreto, el algoritmo de emparejamiento empleado será COA, basado en una matriz de selección por niveles (Sección 5.2.1). El objetivo es determinar cuál es el algoritmo de planificación de enlaces que mejor garantiza la QoS requerida por las aplicaciones. Además, también se trata de comprobar qué grado de mejora se obtiene al incrementar el número de canales virtuales seleccionados por el planificador de enlaces (esto es, el número de candidatos elegidos por canal físico).

6.5.1. Comparativa IABP vs. JBP

Análisis con tráfico CBR

En primer lugar, se compararán las prestaciones de los dos primeros algoritmos, IABP y JBP, con tráfico CBR. Los parámetros con los que se han configurado las simulaciones se muestran en la Tabla 6.5. Se observa como sólo se ha variado el algoritmo de planificación de enlaces, y el número de candidatos obtenidos por el mismo. Las simulaciones se han ejecutado durante 600 ciclos del planificador, más 10 ciclos de periodo transitorio.

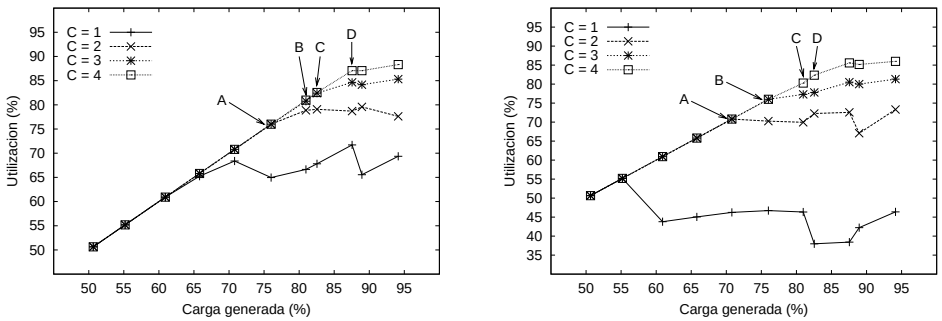
La utilización media del *crossbar* se muestra en la Figura 6.36, para los dos algoritmos de planificación, y todos los niveles de candidatos posibles (de 1 a 4). Se puede apreciar como para ambos algoritmos, al aumentar el número de candidatos, la utilización del *crossbar* es mayor. Sin embargo, según se incrementa este número de candidatos, la mejora va siendo cada vez menos acentuada.

En concreto, para IABP se alcanza en torno al 66 % de utilización antes de entrar en saturación, cuando se emplea un único candidato. Esta utilización máxima crece hasta el 77 % del ancho de banda de los enlaces, cuando se consideran dos niveles de candidatos. Al pasar a tres candidatos, se llega hasta casi el 85 % de saturación, e incluso el encaminador puede alcanzar casi el 90 % de utilización si se consideran los 4 niveles de candidatos posibles.

Por otro lado, el algoritmo JBP entra en saturación en general a niveles de carga más bajos que IABP. El caso más evidente es para un candidato, donde se alcanza la saturación con una carga de aproximadamente el 55 % del ancho de banda de los enlaces, y donde se produce una fuerte degradación al incrementar la carga generada (obsérvese la diferente escala del eje vertical en la Figura 6.36-(b)). Cuando el número de candidatos es mayor, la utilización máxima que es capaz de soportar el encaminador antes de saturarse, alcanza niveles algo más parecidos a los de IABP: un 70 % para dos candidatos, un 75 % para tres candidatos, y en torno al 83 % para cuatro candidatos.

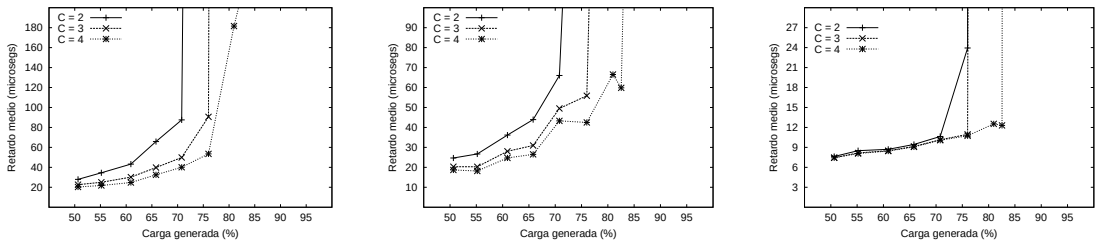
Número de puertos	4
Número de canales virtuales	128
Valor de K	16
Tamaño de phit	16 bits
Tamaño de flit	1024 bits
Ancho de banda de los enlaces	1.24 Gbps
Tamaño de los *buffers* del MMR	1 flit
Tipo de planificador	{IABP, JBP} + COA
Número de candidatos	{1, 2, 3, 4}
Tipo de CAC	Sección 4.3.3
Factor de concurrencia	16
Semilla para números aleatorios	12345

Tabla 6.5: Evaluación de los algoritmos de planificación de enlaces: Parámetros comunes a todas las pruebas.



(a) Algoritmo IABP (b) Algoritmo JBP

Figura 6.36: Planificación de enlaces - Algoritmos IABP y JBP: Tráfico CBR. Utilización media del *crossbar*.



(a) 64 Kbps (b) 1.54 Mbps (c) 55 Mbps

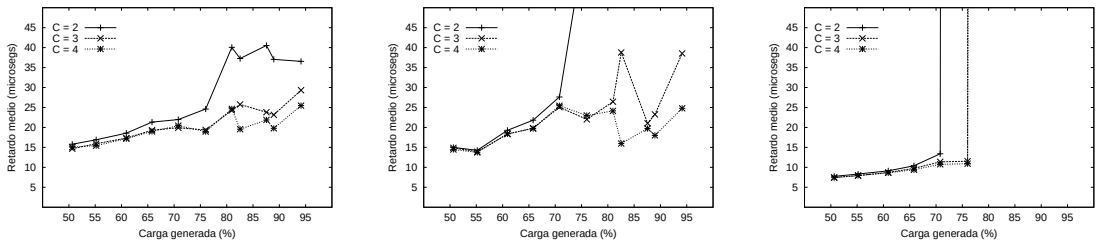
Figura 6.37: Planificación de enlaces - Algoritmo IABP: Tráfico CBR. Retardo medio de los flits desde su generación.

En términos de utilización, por tanto, queda claro que es necesario emplear más de un nivel de candidatos por enlace físico para obtener unas prestaciones adecuadas. Esto es lógico, pues con un sólo candidato por enlace, el algoritmo de emparejamiento tiene pocas opciones para realizar emparejamientos válidos, lo que da como consecuencia una pobre utilización. Esto es válido para ambos algoritmos, aunque el efecto sea bastante más acusado para JBP. Por ello, en los resultados que se presentan a continuación, se han considerado 2 o más niveles de candidatos.

Para medir la QoS que reciben los flits, se ha medido el retardo medio de los flits desde su generación, la distribución del mismo, y el *jitter* medio de los flits. Todas estas medidas se desglosan por tipo de conexión: bajos, medios y altos requerimientos de ancho de banda.

En la Figura 6.37 se muestran los retardos medios de flit desde la generación obtenidos con el algoritmo IABP. Puede apreciarse como el aumento del número de candidatos se traduce en menores retardos medios para los tres tipos de conexiones, así como en un incremento de la productividad (se llega hasta casi el 85 % con cuatro candidatos). Por debajo de saturación, la mejora en retardo es más notable en las conexiones con requerimientos bajos y medios. Esto se debe a que, al incrementar el número de candidatos, se pueden obtener más emparejamientos entre entradas y salidas. Debido al funcionamiento del mecanismo de las prioridades, los emparejamientos adicionales probablemente involucrarán a flits de las conexiones menos prioritarias, mejorando así sus prestaciones. Por otro lado, la mejora en productividad al emplear cuatro candidatos, se aprecia sobre todo en las conexiones de medios y altos requerimientos de ancho de banda. Esto quiere decir que esta mejora se consigue a costa de las conexiones de pocos requerimientos de ancho de banda, que experimentan retardos mucho más elevados a esos niveles de carga. De todas formas, hay que recordar que estas conexiones tienen requerimientos de QoS más relajados, y habrá que analizar detalladamente cuantos flits cumplen con cotas de retardo razonables para ese tipo de conexión. Por otro lado, hay que resaltar también las diferentes escalas en el eje vertical de las tres gráficas. Los retardos de las conexiones de 55 Mbps se hayan en un rango entre los 6 y los 12 microsegundos, mientras que para las conexiones de 64 Kbps, los retardos son de varias decenas de microsegundos. Esto refleja el diferente servicio recibido por los distintos tipos de conexiones, de acuerdo con sus requerimientos.

El retardo medio desde la generación experimentado por los flits cuando se emplea el algoritmo JBP se muestra en la Figura 6.38, para los tres tipos de conexiones considerados. En este caso, la influencia del número de candidatos empleados sobre el retardo de los flits es menor que con IABP, especialmente cuando se pasa de 3 a 4 candidatos. Esto es muy interesante, porque el algoritmo se podría implementar de forma mucho más simple, con menos candidatos, sin perder QoS. Hay que resaltar también que, en este caso, todas las conexiones reciben



(a) 64 Kbps (b) 1.54 Mbps (c) 55 Mbps

Figura 6.38: Planificación de enlaces - Algoritmo JBP: Tráfico CBR. Retardo medio de los flits desde su generación.

un tratamiento similar (las escalas de los ejes verticales son todas iguales). Esto podría no ser deseable, pues las conexiones que requieren menos ancho de banda tienen requerimientos temporales menos estrictos. Así, el algoritmo les está proporcionando más QoS de la requerida. A pesar de esto, las conexiones de mayores requerimientos todavía reciben la QoS que necesitan y transmiten sus flits a tiempo, con retardos similares a los alcanzados con el algoritmo IABP. Sin embargo, JBP se satura a cargas más bajas que IABP. De esta forma se hacen patentes las consecuencias que produce el hecho de proporcionar QoS en exceso a las conexiones de menos requerimientos.

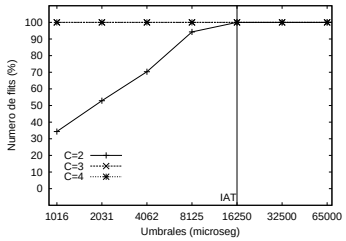
Para obtener más detalle acerca de qué QoS se ofrece a cada conexión, se ha calculado la distribución de los retardos sufridos por los flits. Se han considerado un conjunto de cotas de retardo distintas para cada tipo de conexión, todas relativas al IAT de las mismas, y se ha llevado el recuento de los flits cuyo retardo es inferior a dichas cotas o umbrales. Los resultados se expresan como porcentajes sobre los flits generados, y se han calculado para los puntos de carga situados en torno al punto de saturación (marcados como A, B, C y D en las gráficas de la Figura 6.36).

En la Figura 6.39 se muestra la distribución de los retardos de los flits cuando se emplea el algoritmo IABP. Para el punto de carga más baja considerado, en torno al 76 %, ya se observa como con dos candidatos hay muchos flits que sólo cumplen con los umbrales más relajados. Por ejemplo, sólo poco más del 40 % de los flits generados pertenecientes a conexiones de 55 Mbps sufren retardos inferiores al IAT de esa conexión (19 microsegundos). Esto indica que en ese punto el encaminador está casi saturado. Sin embargo, cuando se emplean 3 o 4 candidatos, dicho porcentaje sube hasta casi el 95 %, que ya es algo tolerable.

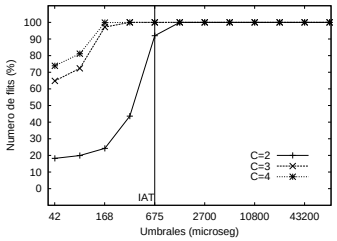
Para el siguiente punto de carga, un 81 % aproximadamente, el uso de sólo dos niveles de candidatos es inviable, pues apenas se cumplen las cotas más relajadas, y además hay flits que ni siquiera llegan a poder cruzar el encaminador, como se aprecia para las conexiones de 55 Mbps. El uso de 3 candidatos tampoco ofrece demasiada mejora respecto al uso de dos, por lo que si se desea alcanzar estos niveles de carga ofreciendo garantías de QoS, sólo es posible empleando los 4 candidatos. Para ese caso, los flits de todas las conexiones todavía cumplen con cotas de retardo razonables: casi el 100 % de los flits de las conexiones de 55 Mbps tienen retardos inferiores al doble de su IAT (38 microsegs.), el 100 % de los flits de las conexiones de 1.54 Mbps sufren retardos inferiores a la mitad de su IAT (337.5 microsegs.), y la totalidad de los flits de las conexiones de 64 Kbps cumplen con el umbral más estricto considerado para ellas, establecido en $IAT/16$ (1016 microsegs.).

En el punto de carga C (el 84 %), no existen grandes diferencias respecto al anterior. Sólo algunos flits de las conexiones de 64 Kbps no cumplen con la cota más estricta. Para el

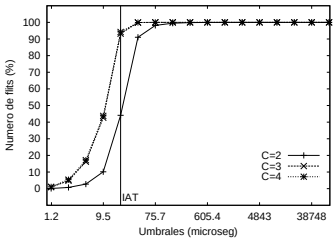
Punto de carga A (carga = 76 %)



(a) 64 Kbps

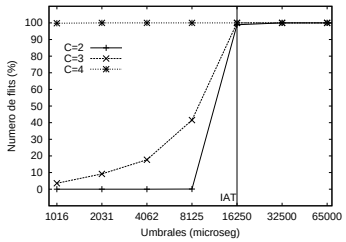


(b) 1.54 Mbps

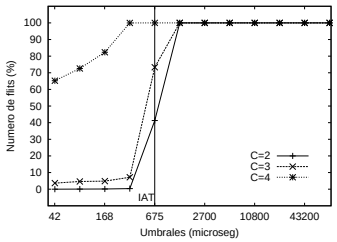


(c) 55 Mbps

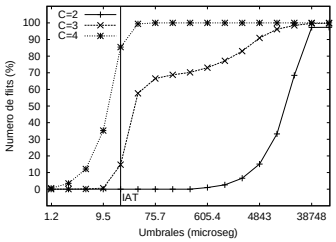
Punto de carga B (carga = 81 %)



(a) 64 Kbps

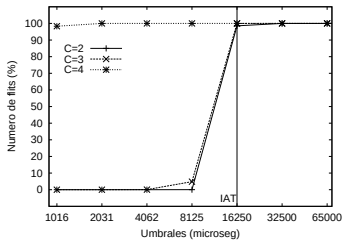


(b) 1.54 Mbps

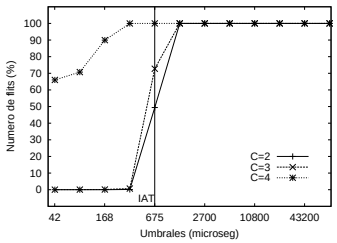


(c) 55 Mbps

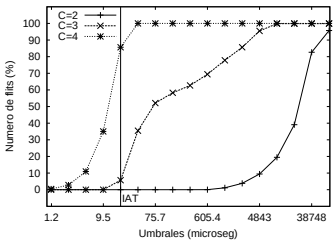
Punto de carga C (carga = 84 %)



(a) 64 Kbps

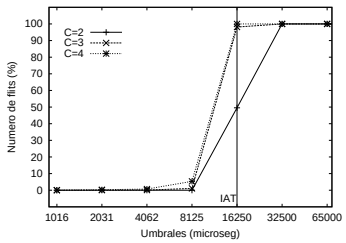


(b) 1.54 Mbps

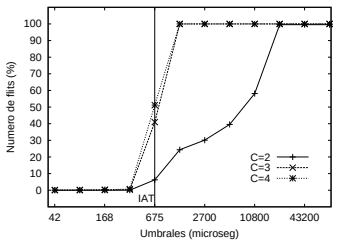


(c) 55 Mbps

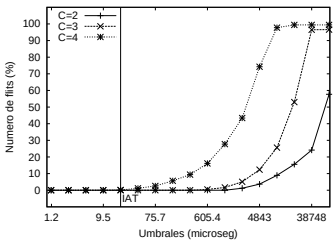
Punto de carga D (carga = 88 %)



(a) 64 Kbps



(b) 1.54 Mbps



(c) 55 Mbps

Figura 6.39: Planificación de enlaces - Algoritmo IABP: Tráfico CBR. Distribución del retardo medio de los flits desde su generación.

último punto de carga, en torno al 88 %, el encaminador está claramente saturado, independientemente del número de candidatos empleado.

Para JBP, los puntos de carga considerados para este análisis de los retardos de flit son ligeramente inferiores que para IABP, pues la saturación se produce a menores cargas. Los resultados se muestran en las gráficas de la Figura 6.40. El punto de carga más baja considerada, un 71 % aproximadamente, no hay diferencias apreciables entre el uso de 2 o más candidatos. Todos los flits de las conexiones de 64 Kbps y 1.54 Mbps obtienen retardos inferiores a las cotas más estrictas consideradas. Para las conexiones de mayor ancho de banda, se cumplen cotas todavía razonables. Por ejemplo, más del 90 % de los flits sufren retardos inferiores al IAT (19 microsegs.), y el 100 % sufen retardos inferiores a $8 \times IAT$ (152 microsegs.) para todos los números de candidatos considerados.

Cuando la carga aumenta hasta el 76 % (punto B), lo anterior sólo se mantiene cuando se emplean 3 o más candidatos. Existe una importante degradación cuando se emplean sólo 2 candidatos, que afecta sobre todo a las conexiones más exigentes, algunos de cuyos flits ni siquiera llegan a ser transmitidos.

Al seguir aumentando la carga hasta el 81 % (punto C), se empiezan a apreciar diferencias entre el uso de 3 o 4 candidatos, sobre las conexiones de 55 Mbps. Únicamente cuando se emplean los 4 candidatos, el Encaminador es capaz de retransmitir todos los flits de las conexiones de 55 Mbps, aunque existe en torno a un 10 % de flits que sólo son capaces de cumplir con las cotas más relajadas (mayores de $32 \times IAT$, unos 600 microsegs.). Al incrementar la carga hasta el 84 %, no existen cambios significativos.

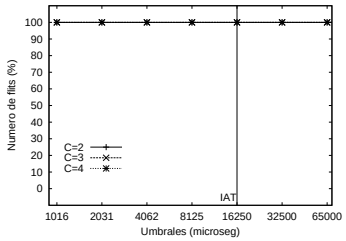
Hay que resaltar que, cuando se emplea el algoritmo JBP, las conexiones que sufren todo el impacto de la degradación de servicio producida por la saturación, son las que más recursos requieren, mientras que las conexiones de requerimientos medios y bajos apenas sufren degradación alguna. Por el contrario, con IABP, la degradación en el servicio afecta en mayor o menor medida a todos los tipos de conexiones. Eso en términos de retardo experimentado por los flits. Pero el objetivo del algoritmo JBP es minimizar el *jitter* experimentado por los flits de las conexiones. Falta pues por comprobar si esto se cumple.

En las Figuras 6.41 y 6.42 se muestra el *jitter* medio experimentado por los flits, al emplear cada algoritmo, y según el número de candidatos empleado. Para IABP, se observan valores de *jitter* en el mismo orden de magnitud que los correspondientes valores de retardo. Así, las conexiones de 64 Kbps, cuyos flits experimentaban retardos de varias decenas de microsegundos, experimentan *jitters* también en ese orden de magnitud. Por el contrario, las conexiones de mayores requerimientos, obtienen valores para el *jitter* por debajo del microsegundo, incluso cuando el Encaminador ya se ha saturado.

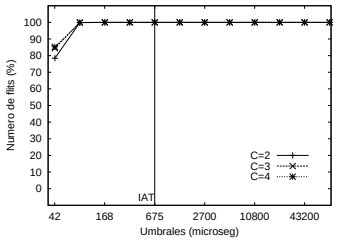
En cuanto al comportamiento de JBP, se puede observar que el *jitter* que proporciona a todas las conexiones es similar, siendo solamente de unos pocos microsegundos para cargas inferiores a saturación. Incluso para las conexiones de mayores requerimientos, el *jitter* en dicha franja de carga es inferior a 1 microsegundo y comparable por tanto con los resultados obtenidos con IABP para esas mismas conexiones.

En resumen, los resultados presentados sugieren que el algoritmo IABP es capaz de proporcionar la QoS necesaria para las distintas aplicaciones hasta niveles de carga del 75 % al 85 %, dependiendo del número de niveles de candidatos empleados. Si se emplea el algoritmo JBP, la carga máxima aceptada sin hacer peligrar las garantías de QoS de las conexiones pasa a estar entre el 70 % y el 75 %, según el número de candidatos. Esto es atendiendo a la productividad máxima alcanzable, respetando las garantías de QoS de las aplicaciones en términos de retardo. Sin embargo, es JBP quien proporciona mejores prestaciones a todas las conexiones, si la QoS se mide en términos de *jitter*.

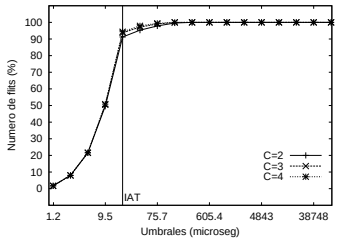
Punto de carga A (carga = 71 %)



(a) 64 Kbps

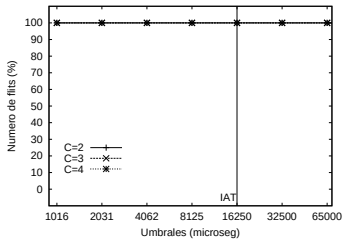


(b) 1.54 Mbps

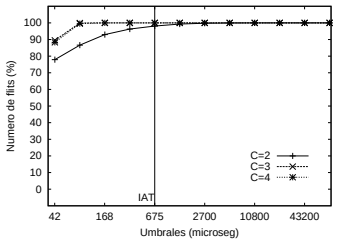


(c) 55 Mbps

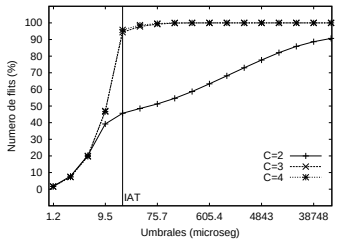
Punto de carga B (carga = 76 %)



(a) 64 Kbps

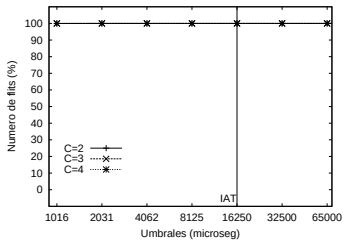


(b) 1.54 Mbps

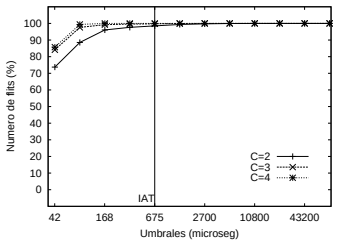


(c) 55 Mbps

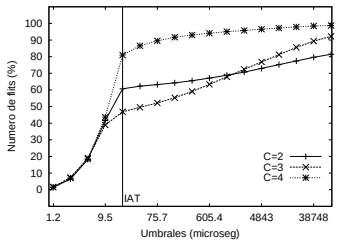
Punto de carga C (carga = 81 %)



(a) 64 Kbps

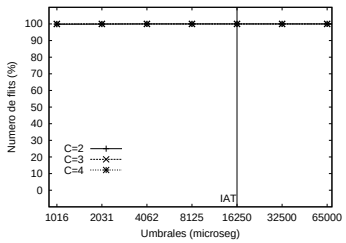


(b) 1.54 Mbps

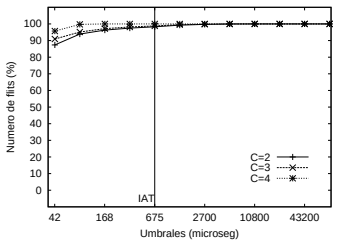


(c) 55 Mbps

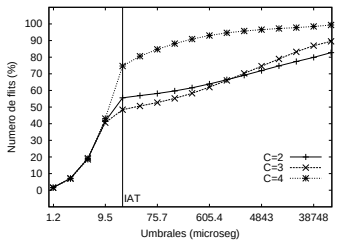
Punto de carga D (carga = 84 %)



(a) 64 Kbps

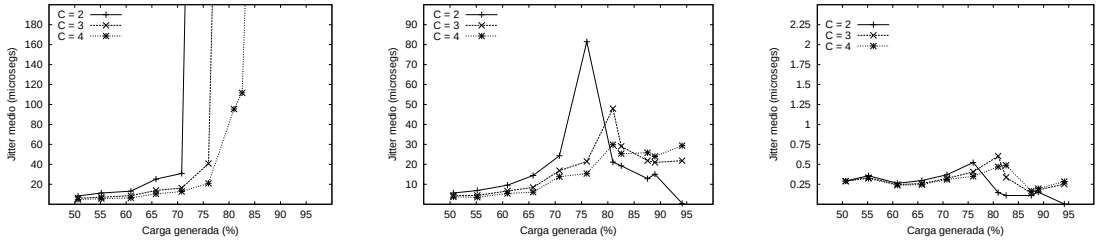


(b) 1.54 Mbps



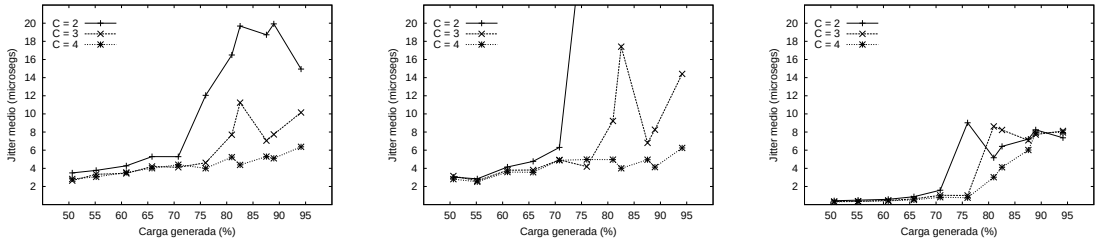
(c) 55 Mbps

Figura 6.40: Planificación de enlaces - Algoritmo JBP: Tráfico CBR. Distribución del retardo medio de los flits desde su generación.



(a) 64 Kbps (b) 1.54 Mbps (c) 55 Mbps

Figura 6.41: Planificación de enlaces - Algoritmo IABP: *Jitter* medio de los flits.



(a) 64 Kbps (b) 1.54 Mbps (c) 55 Mbps

Figura 6.42: Planificación de enlaces - Algoritmo JBP: Tráfico CBR. *Jitter* medio de los flits.

Es decir, mediante IABP se aprovecha mejor el ancho de banda de los enlaces y del *crossbar*, a costa de provocar un *jitter* relativamente elevado a las conexiones de menores requerimientos de ancho de banda. Mientras que mediante JBP se proporcionan valores de *jitter* mucho más reducidos para todas las conexiones, a costa de alcanzar una menor productividad.

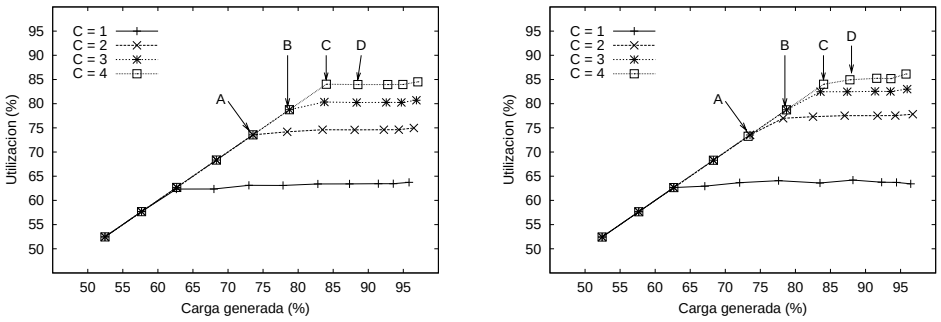
Análisis con tráfico VBR

El análisis anterior se ha llevado a cabo con tráfico CBR exclusivamente. La adaptación del algoritmo IABP a tráfico VBR es directa: se emplea el tiempo entre llegadas de los flits, que varía a lo largo del tiempo de vida de la conexión si se emplea el modelo SR, y permanece constante en el modelo BB. Por contra, el algoritmo JBP emplea una estimación del *jitter* acumulado sufrido por los flits de la conexión. Esta medida tiene pleno sentido cuando se trata de conexiones CBR, pero el *jitter* de flit no tiene tanto sentido para las conexiones de vídeo MPEG-2 que se emplean como tráfico VBR.

De todas formas, se van a evaluar también las prestaciones ofrecidas por el Encaminador Multimedia con el algoritmo JBP cuando el tráfico es de tipo VBR, para comprobar hasta que punto el empleo del *jitter* de flits en el cálculo de la prioridad beneficia o perjudica a las conexiones de este tipo de tráfico.

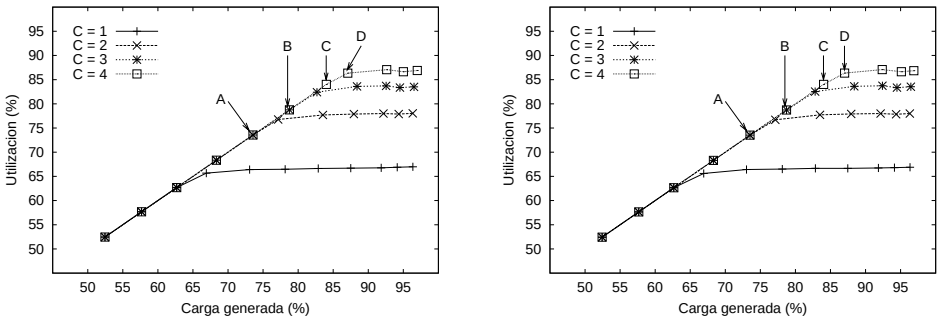
Las simulaciones se han ejecutado hasta que cada conexión establecida ha transmitido 6 GOPs completos a través del Encaminador. Se ha dejado un tiempo equivalente a dos tiempos de trama al inicio de la simulación como transitorio, antes de recoger datos para las estadísticas. Las conexiones se alinean de forma aleatoria dentro de un GOP. El resto de parámetros son idénticos a los de la Tabla 6.5.

La utilización media del *crossbar* con el algoritmo IABP se muestra en la Figura 6.43, para



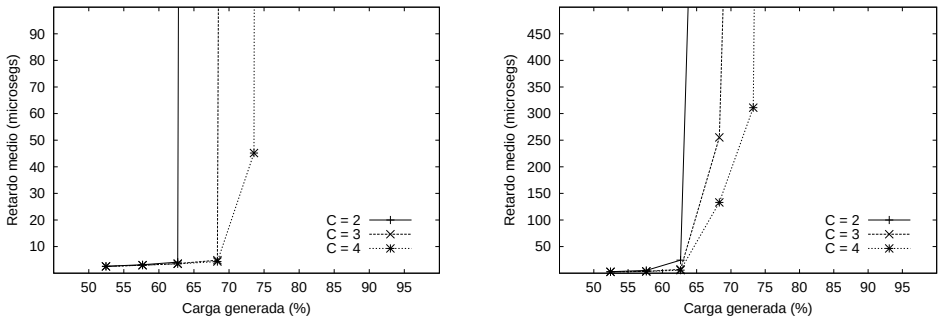
(a) Modelo SR (b) Modelo BB

Figura 6.43: Planificación de enlaces - Algoritmo IABP: Tráfico VBR. Utilización media del *crossbar*.



(a) Modelo SR (b) Modelo BB

Figura 6.44: Planificación de enlaces - Algoritmo JBP: Tráfico VBR. Utilización media del *crossbar*.



(a) Modelo SR (b) Modelo BB

Figura 6.45: Planificación de enlaces - Algoritmo IABP: Tráfico VBR. Retardo medio de las imágenes desde su generación.

los dos modelos de inyección, y todos los niveles de candidatos posibles (de 1 a 4). Al igual que sucedía con tráfico CBR, y con ambos modelos de inyección, cuando se aumenta el número de candidatos, la utilización del *crossbar* crece. Sin embargo, el grado de mejora va siendo cada vez menor, al ir creciendo el número de candidatos.

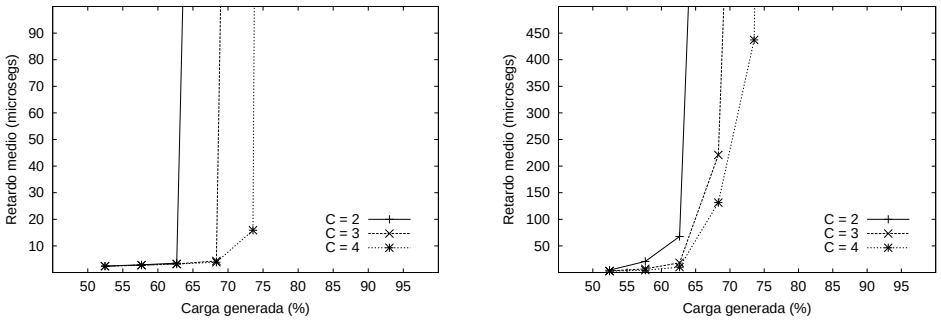
En concreto, con el modelo de inyección SR, los niveles de utilización alcanzados antes de saturación se hayan en torno al 63 %, 74 %, 79 % y 84 % del ancho de banda de los enlaces, para 1, 2, 3 y 4 niveles de candidatos, respectivamente. Con el modelo de inyección BB, y cuando se emplean 2 o más candidatos, las cargas a las cuales se alcanza la saturación son prácticamente idénticas que con el modelo SR. En todos los casos, una vez se ha entrado en saturación, no existe degradación, es decir, aunque se sigue aumentando la carga generada, la utilización media del *crossbar* se mantiene más o menos constante.

En cuanto a la utilización media del *crossbar* experimentada con JBP (Figura 6.44), apenas se observan diferencias respecto a lo obtenido para IABP. Únicamente se aprecia un levísimo incremento en la utilización media máxima alcanzada por el *crossbar*, cuando se emplea el modelo SR.

Para ambos algoritmos, se ha podido comprobar como, la utilización alcanzada con un sólo nivel de candidatos es muy pobre. Esto pone de manifiesto de nuevo la necesidad de emplear más de un candidato por enlace físico para obtener un rendimiento adecuado del Encaminador. Por tanto, los resultados siguientes sólo se presentan para 2 o más candidatos por enlace físico.

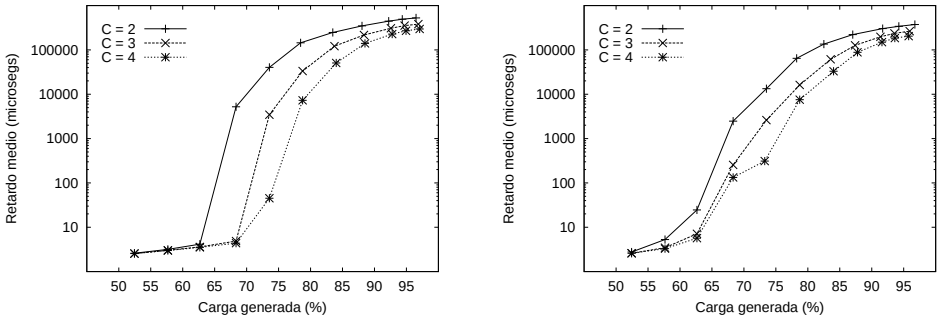
Al igual que se hizo para CBR, es el turno de medir la QoS recibida por la información que atraviesa el Encaminador. Se hará de nuevo empleando medidas de retardo y *jitter*, pero con alguna variación. Puesto que el tráfico VBR empleado se compone de secuencias de vídeo MPEG-2, las prestaciones se van a medir respecto a la unidad de datos de esa aplicación, esto es, las tramas de vídeo, en lugar de sobre flits individuales. De esta forma, el análisis se acerca más a la QoS recibida realmente por la aplicación. Por tanto, los resultados que se presentan a continuación serán el retardo desde la generación y el *jitter* medios experimentados por las tramas, y la distribución de dichos retardos desde la generación.

En las Figuras 6.45 y 6.46 se muestra el retardo medio desde la generación medido sobre las tramas de vídeo, para ambos modelos de inyección, y para ambos algoritmos, respectivamente. El rango completo de retardos obtenidos para todos los niveles de carga simulados se aprecia mejor en las gráficas de las Figuras 6.47 y 6.48, donde el eje vertical se expresa en escala logarítmica.



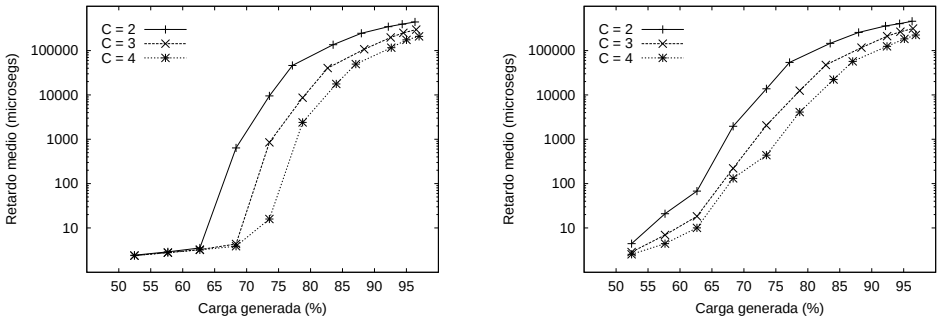
(a) Modelo SR (b) Modelo BB

Figura 6.46: Planificación de enlaces - Algoritmo JBP: Tráfico VBR. Retardo medio de las imágenes desde su generación.



(a) Modelo SR (b) Modelo BB

Figura 6.47: Planificación de enlaces - Algoritmo IABP: Tráfico VBR. Retardo medio de las imágenes desde su generación (escala semilogarítmica).



(a) Modelo SR (b) Modelo BB

Figura 6.48: Planificación de enlaces - Algoritmo JBP: Tráfico VBR. Retardo medio de las imágenes desde su generación (escala semilogarítmica).

Respecto a los resultados obtenidos con el algoritmo IABP, para el modelo de inyección SR (parte (a) de la Figura 6.45) se aprecia un incremento en la productividad de aproximadamente un 5 % al pasar de 2 a 3 candidatos, y de casi otro 5 % al pasar de 3 a 4 candidatos. Para este último caso, 4 candidatos, cuando la carga sobrepasa el 75 %, los retardos experimentados por las tramas se incrementan notablemente. Ese drástico incremento en el retardo se produce para cargas mayores del 70 % y del 65 %, al emplear 3 y 2 niveles de candidatos, respectivamente.

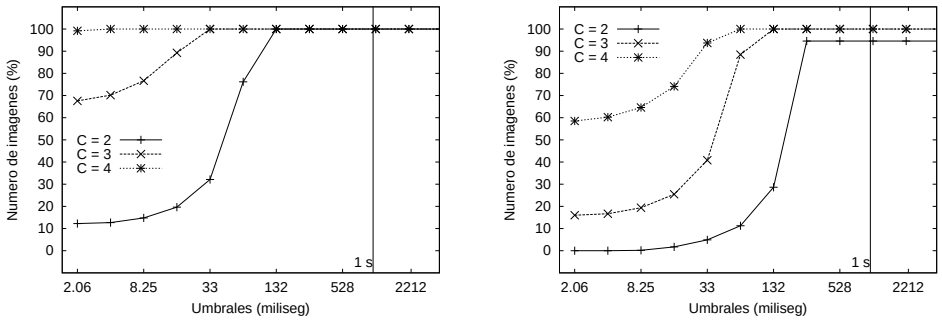
Para el modelo de inyección BB (parte (b) de la misma Figura) el comportamiento es similar, aunque en este caso los retardos experimentados son mayores que con el modelo SR (obsérvese la distinta escala en el eje vertical de las gráficas de la Figura 6.45). Esto se debe a que con el modelo BB la generación de flits se concentra al inicio de cada trama. Por tanto, cuando la carga es moderada, se pueden llegar a introducir periodos de saturación momentáneos en esos instantes, que elevan el retardo experimentado por las tramas. Una vez generada la trama, la fuente permanece ociosa hasta el inicio del siguiente periodo de trama, permitiendo que el Encaminador vaya dando salida a los flits generados. De esta manera, esa saturación más o menos puntual se va difuminando gradualmente.

Los resultados obtenidos con el algoritmo JBP (Figuras 6.46 y 6.48) son muy similares a los obtenidos con IABP. Se puede observar como los retardos medios se disparan a los mismos valores de carga que cuando se emplea el algoritmo IABP, para los tres niveles de candidatos y los dos modelos de inyección considerados. De nuevo, cuando las imágenes de vídeo se inyectan según el modelo BB, los retardos que experimentan son más elevados que si se inyectan según el modelo SR. Para el modelo BB, a cargas moderadamente bajas, por debajo del 65 %, se puede apreciar un ligerísimo incremento en el retardo experimentado por las tramas de vídeo, respecto a lo observado para IABP.

La distribución de los retardos de las tramas ofrece más información acerca de la QoS ofrecida por el Encaminador a las conexiones de vídeo. Esta distribución se ha obtenido llevando el recuento de tramas cuyos retardos desde la generación son inferiores a un conjunto de cotas, relativas al tiempo de trama ($TT = 33$ milisegundos). Las cotas oscilan entre los 2.06 milisegundos ($TT/16$), y los 2212 milisegundos ($TT \times 64$). Los resultados se expresan como porcentajes sobre el número de tramas generadas por las fuentes. Se han analizado los puntos de carga situados alrededor de saturación, señalados como A, B, C y D en la Figura 6.43.

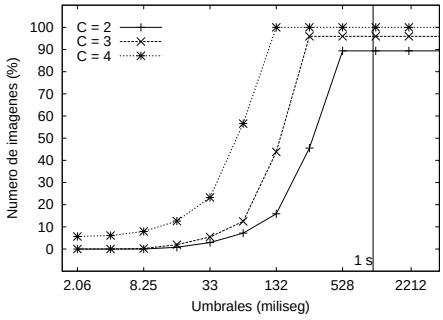
En las gráficas de la Figura 6.49 se muestra la distribución de retardos cuando se emplea el modelo de inyección SR, para el algoritmo IABP. En todos los puntos de carga considerados se observa como al incrementar el número de candidatos empleados, el porcentaje de tramas capaces de cumplir con cotas de retardo mucho más estrictas se incrementa notablemente, sobre todo para los puntos de carga A, B y C. Asimismo, según aumenta la carga, se observa como el Encaminador va entrando en saturación. En el punto de carga B (81 %), si se emplean 2 candidatos, ya hay tramas que no son capaces de cruzar el Encaminador, y de las que se transmiten, casi ninguna es capaz de cumplir con las cotas más estrictas. Esto mismo sucede con una carga del 84 % (punto C) para 3 candidatos, y con una carga del 88 % (punto D) cuando se emplean 4 candidatos. Para cualquier número de candidatos, justo antes de entrar en saturación, todas las tramas obtienen retardos inferiores a 132 milisegundos ($TT \times 4$), e incluso cumplen con cotas mucho más ajustadas cuando la carga está por debajo de la saturación.

En cuanto al comportamiento con el algoritmo JBP, reflejado en las gráficas de la Figura 6.50, se observa en general como hay más tramas que pueden cumplir con cotas más estrictas de retardo que con IABP. Ahora bien, la cota mínima que cumplen la totalidad de las tramas se incrementa. Por ejemplo, en el punto de carga A (74 %), cuando se emplean los 4 candidatos, no hay diferencias: todas las tramas sufren retardos inferiores a la cota más estricta con ambos

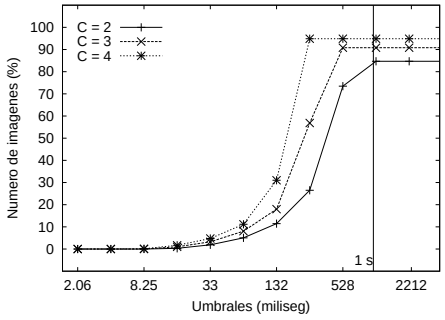


Punto de carga A (carga = 74 %)

Punto de carga B (carga = 79 %)

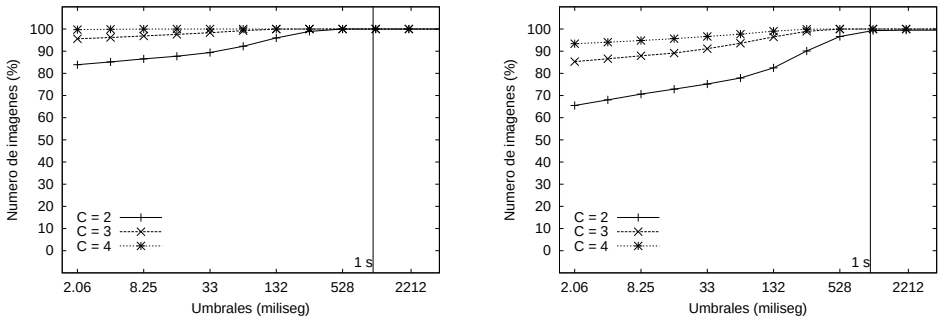


Punto de carga C (carga = 84 %)



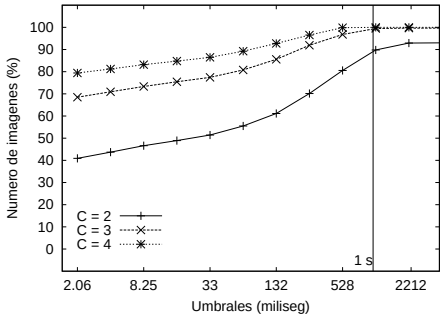
Punto de carga D (carga = 88 %)

Figura 6.49: Planificación de enlaces - Algoritmo IABP: Tráfico VBR - modelo SR. Distribución del retardo medio de las imágenes desde su generación.

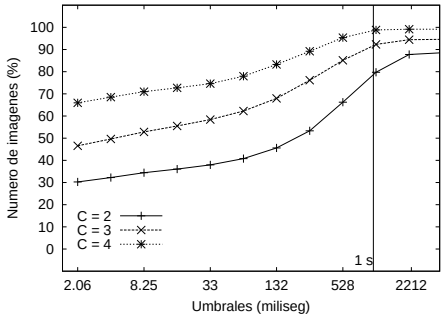


Punto de carga A (carga = 74 %)

Punto de carga B (carga = 79 %)

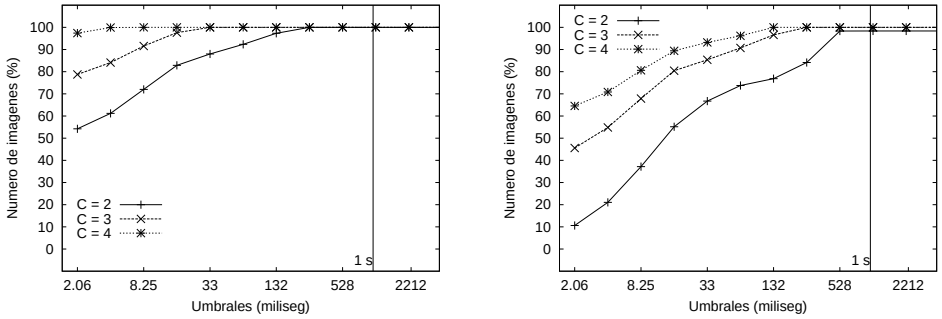


Punto de carga C (carga = 84 %)

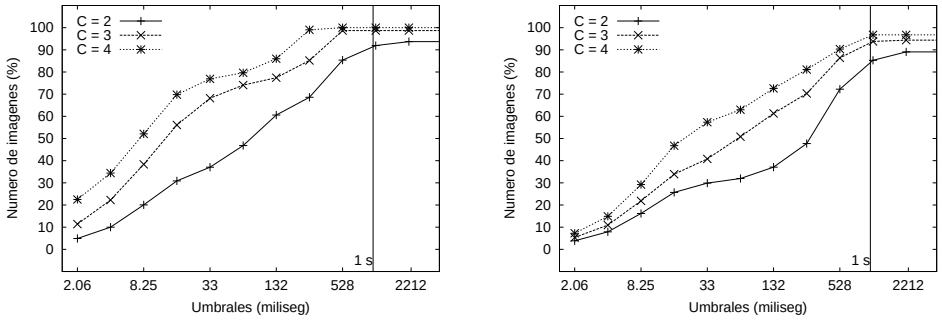


Punto de carga D (carga = 88 %)

Figura 6.50: Planificación de enlaces - Algoritmo JBP: Tráfico VBR - modelo SR. Distribución del retardo medio de las imágenes desde su generación.



Punto de carga A (carga = 74 %) Punto de carga B (carga = 79 %)



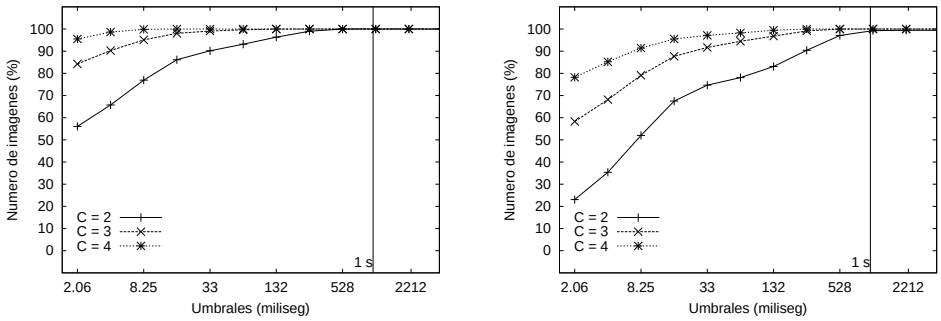
Punto de carga C (carga = 84 %) Punto de carga D (carga = 88 %)

Figura 6.51: Planificación de enlaces - Algoritmo IABP: Tráfico VBR - modelo BB. Distribución del retardo medio de las imágenes desde su generación.

algoritmos. Por el contrario, si se emplean 3 candidatos, con IABP todas las tramas sufren retardos inferiores a 33 milisegundos, mientras que con JBP esta cota mínima se duplica. En el siguiente punto de carga (B, 79 %), si se emplea el algoritmo IABP con 4 candidatos, la mínima cota que cumplen todas las tramas es de 66 milisegundos. Para JBP con el mismo número de candidatos, dicha cota mínima aumenta hasta los 264 milisegundos. Por último, para el punto de carga C (84 %), la cota mínima que cumplen todas las tramas con IABP y 4 candidatos es de 132 milisegundos, mientras que con JBP pasa a ser de medio segundo, esto es, 4 veces mayor. La conclusión es que, cuando se emplea JBP, el rango de los valores de retardo experimentados por las tramas es más amplio: hay más tramas que cumplen con cotas más estrictas, pero sin embargo, el retardo máximo se incrementa.

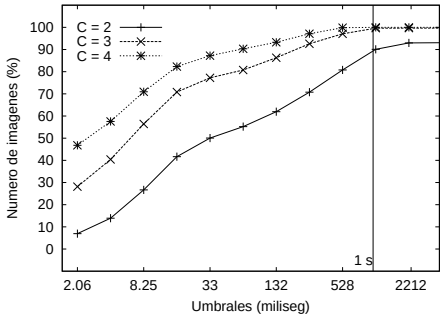
En cuanto a los resultados obtenidos para el modelo BB (Figuras 6.51 y 6.52), de nuevo son muy similares a los observados para el modelo SR. Tanto con IABP como con JBP, al incrementar el número de candidatos, aumenta también el número de tramas que cumplen con las cotas establecidas. En este caso, la mejora es algo más acusada al pasar de 2 a 3 niveles de candidatos, que al pasar de 3 a 4 niveles. Al igual que sucedía con el modelo SR, se aprecia como el nivel de carga al que se produce la saturación aumenta con el número de candidatos.

Más concretamente, para el algoritmo IABP, en el nivel de carga B (79 %) ya se aprecia como si emplean solo dos candidatos, hay tramas que se quedan almacenadas en los *buffers* del MMR y de la tarjeta de red, y no llegan a atravesar el Encaminador. Para tres niveles de candidatos, este efecto se hace patente para una carga del 84 % (punto C), mientras que cuando se consideran los cuatro niveles de candidatos, el descenso en productividad se presenta cuando la carga llega al 88 % (punto D).

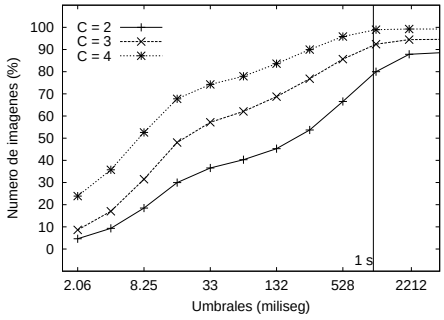


Punto de carga A (carga = 74 %)

Punto de carga B (carga = 79 %)



Punto de carga C (carga = 84 %)



Punto de carga D (carga = 88 %)

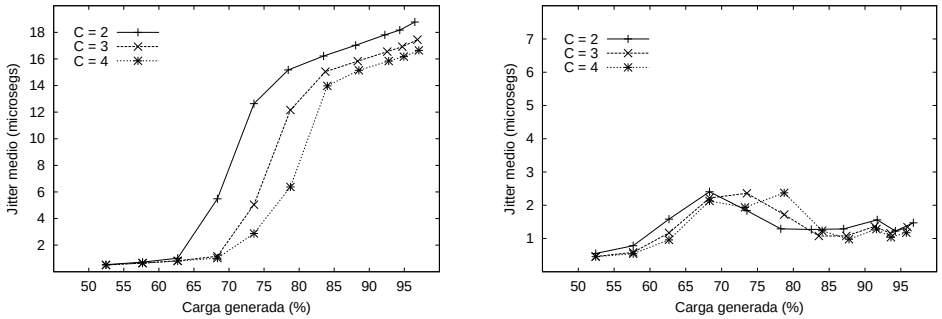
Figura 6.52: Planificación de enlaces - Algoritmo JBP: Tráfico VBR - modelo BB. Distribución del retardo medio de las imágenes desde su generación.

Por otro lado, también se aprecia como el rango de retardos experimentados por las tramas es mayor que con el modelo SR: para los mismos niveles de carga, hay más tramas que cumplen con umbrales más estrictos, mientras que también se incrementa ligeramente el número de tramas que sólo puede cumplir con las cotas más relajadas. Este efecto se hace más patente cuando se acerca la saturación, por ejemplo, para una carga del 84 % (punto C), y 4 niveles de candidatos. El motivo de este comportamiento es la forma de inyección de las tramas, y el hecho de que el tráfico MPEG-2 presenta una gran variabilidad en el tamaño de las mismas. En efecto, cuando se inyecta una trama de tipo I (de gran tamaño), al generar todos los flits que la componen al inicio del tiempo de trama, la carga a la que se somete al Encaminador es mayor que si se genera una trama de tipo B (las de menor tamaño). Por tanto, cuando coincidan varias tramas de tipo I, el retardo obtenido será mayor que en períodos donde coincidan más tramas de tipo P o B, más pequeñas. Y viceversa, cuando coincidan varias tramas de tipo B, la carga será menor, y el retardo también será menor. Es decir, el nivel de carga "instantáneo" presenta mayores oscilaciones que cuando se emplea el modelo SR, provocando una mayor variación en el rango de valores de retardo experimentado por las tramas. Debido a la menor frecuencia de las tramas de tipo I, y al alineamiento aleatorio de las secuencias que se emiten por las distintas conexiones, los periodos de carga más alta son pocos, pero provocan que un pequeño porcentaje de tramas sólo puedan cumplir con cotas algo más relajadas que las obtenidas con el modelo SR. En concreto, para cualquier número de candidatos, justo antes de entrar en saturación, todas las tramas obtienen retardos inferiores a 264 milisegundos ($TT \times 8$), el doble de la cota obtenida para el modelo SR.

En los resultados obtenidos con el algoritmo JBP (Figura 6.52), y al contrario de lo observado para el algoritmo IABP, se observa como hay menos imágenes que pueden cumplir las cotas más estrictas de retardo, con respecto a lo observado con el modelo SR. Además, los retardos máximos experimentados por las tramas no se incrementan. Es decir, el rango de valores de los retardos experimentados por las tramas es menos amplio. Esto se debe a que en este modelo de inyección, la tasa de inyección de los flits es siempre la misma: la tasa pico definida en el modelo. Por tanto, todos los flits se tratan por igual, y esto da unos valores e retardo de las tramas algo más homogéneos que cuando se emplea el modelo SR. En ese caso, la tasa de inyección de flits varía de trama a trama, esto es, las tramas de tipo I (más grandes) se transmiten a mayor tasa que por ejemplo, las de tipo B (las más pequeñas). Ya se comprobó en el análisis sobre tráfico CBR como JBP perjudica en general a las conexiones de mayores requerimientos de ancho de banda, en favor de las menos exigentes. Pues esto mismo sucede cuando se emplea el modelo SR: puesto que la tasa de inyección varía de trama en trama, las tramas que más ancho de banda consumen se ven perjudicadas en favor de las que menos consumen, dando como resultado el mayor rango de retardos observado para este modelo de inyección en la Figura 6.51. Como en una secuencia de vídeo MPEG-2, hay menos tramas I que B o P, el porcentaje de tramas cuyo retardo se dispara es más bajo, mientras que aumenta el número de tramas con retardos bajos.

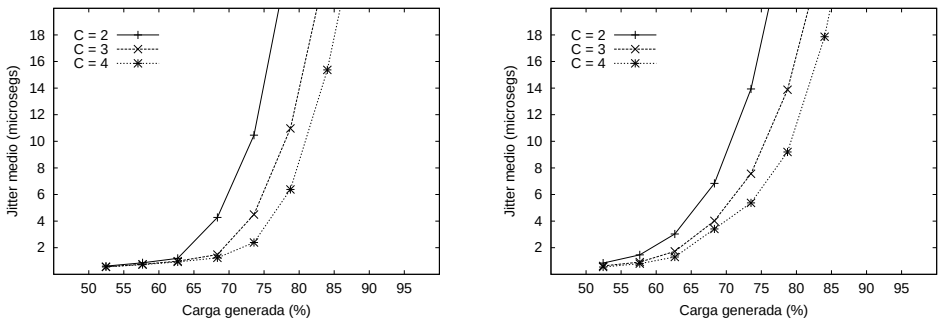
Por ultimo, queda por analizar el *jitter* o variación en el retardo experimentado por las tramas de vídeo. Hay que recordar que la transmisión de secuencias de vídeo es bastante sensible a este parámetro, pues un valor de *jitter* excesivo afecta a la continuidad de visualización de la secuencia, disminuyendo la QoS de la aplicación.

En la Figura 6.53 se muestran los valores de *jitter* medios de trama obtenidos para ambos modelos de inyección, con el algoritmo IABP. Para el modelo SR (parte (a) de la Figura), se observa como cuando la carga está por debajo del nivel de saturación, el valor del *jitter* se mantiene muy bajo, por debajo del microsegundo cuando la carga es inferior al 70 %. Cuando la carga se aproxima a la saturación, los valores de *jitter* se incrementan notablemente, pero siempre se mantienen por debajo de los 8 microsegundos. Incluso una vez se ha entrado en



(a) Modelo SR (b) Modelo BB

Figura 6.53: Planificación de enlaces - Algoritmo IABP: Tráfico VBR. *Jitter* medio de las imágenes desde su generación.



(a) Modelo SR (b) Modelo BB

Figura 6.54: Planificación de enlaces - Algoritmo JBP: Tráfico VBR. *Jitter* medio de las imágenes desde su generación.

saturación, el valor del *jitter* no crece incontroladamente, sino que se mantiene por debajo de los 20 microsegundos.

En cuanto al modelo BB (parte (b) de la Figura anterior), los valores medios de *jitter* permanecen siempre por debajo de los 3 microsegundos. Teniendo en cuenta la distribución de los valores de retardo de las tramas observada anteriormente, estos pequeños valores para el *jitter* indican que las tramas van aumentando o disminuyendo su retardo progresivamente, de manera que la variación del mismo, esto es, el *jitter*, se mantiene baja. Hay que recordar que el *jitter* se calcula entre tramas consecutivas de la misma conexión.

El *jitter* medio de las tramas cuando de emplea el algoritmo JBP se muestra en la Figura 6.54. Lo primero que llama la atención es que en este caso los valores de *jitter* sí que llegan a elevarse incontroladamente al ir entrando en saturación. No obstante, cuando la carga está por debajo de saturación, los valores obtenidos son de unos pocos microsegundos. Concretamente, para el modelo de inyección SR, los valores obtenidos antes de saturación son muy similares a los experimentados con IABP. Por el contrario, con el modelo de inyección BB, el *jitter* medio se eleva considerablemente cuando la carga sobrepasa el 60 %, aproximadamente. También se aprecia una mayor influencia del número de candidatos empleados en el valor del *jitter*.

Los valores obtenidos para ambos modelos de inyección, y para ambos algoritmos cuando la carga está por debajo del nivel al cual se produce la saturación, son muy razonables, y más aún teniendo en cuenta que existen técnicas de absorción del *jitter* en el destino, mediante las cuales se pueden tolerar valores de *jitter* de hasta varios milisegundos [133].

De este estudio realizado para tráfico VBR, se desprende que los dos algoritmos de planificación, IABP y JBP, son capaces de ofrecer garantías de QoS al tráfico VBR. El rango de carga útil obtenido para ambos algoritmos es similar, y aumenta con el número de candidatos, hasta un máximo del 80 %, aproximadamente, cuando se emplean 4 niveles de candidatos por enlace físico. Ambos ofrecen buenas prestaciones en términos de retardo: con IABP las tramas obtienen retardos máximos menores que con JBP, y con JBP el porcentaje de tramas que obtiene retardos más ajustados es mayor.

En cuanto al *jitter* o variación del retardo experimentado por las tramas, por debajo de saturación, y para el modelo de inyección SR, ambos algoritmos también ofrecen resultados similares. Para el modelo de inyección BB, y a cargas moderadas, el *jitter* medio obtenido con JBP se eleva considerablemente con respecto al obtenido con IABP. Aún así, todavía se mantiene dentro de límites perfectamente asumibles por las aplicaciones de vídeo. Cuando se ha entrado en saturación, el valor del *jitter* permanece acotado cuando se emplea el algoritmo IABP, pero se dispara incontroladamente al emplear el algoritmo JBP. Esto sugiere que la métrica de *jitter* empleada en JBP (hay que recordar que se emplea una estimación del *jitter* acumulado sufrido por los *flits* de la conexión en el cálculo de la prioridad) no es del todo adecuada para el tipo de aplicación VBR que se emplea como carga.

Otra posibilidad sería emplear una estimación del *jitter* de las tramas de vídeo para el cálculo de la prioridad. Sin embargo, además de complicar el diseño del planificador de enlaces, esta modificación implicaría ajustar el diseño del Encaminador a una aplicación muy concreta. Esto no es en absoluto deseable, pues limitaría la aplicabilidad del Encaminador a otros tipos de aplicaciones multimedia que generan tráfico VBR, pero no MPEG-2.

A la vista tanto de los resultados obtenidos con tráfico VBR que se acaban de presentar y discutir, como de los obtenidos con tráfico CBR, se puede concluir que el algoritmo IABP es preferible sobre JBP. El motivo es que aunque con VBR el rango de carga útil es el mismo para ambos algoritmos, con tráfico CBR existía una importante mejora de IABP sobre JBP

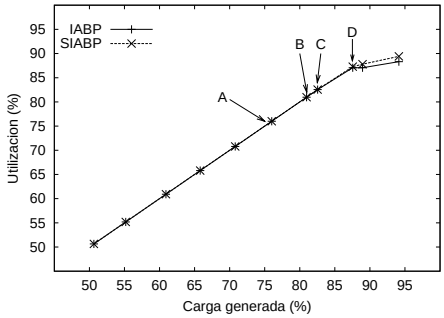


Figura 6.55: Planificación de enlaces - Algoritmo SIABP: Tráfico CBR. Utilización media del *crossbar*.

en términos de productividad máxima alcanzada, a la vez que proporcionando las garantías de QoS que necesitan las aplicaciones multimedia.

6.5.2. Comparativa IABP vs. SIABP

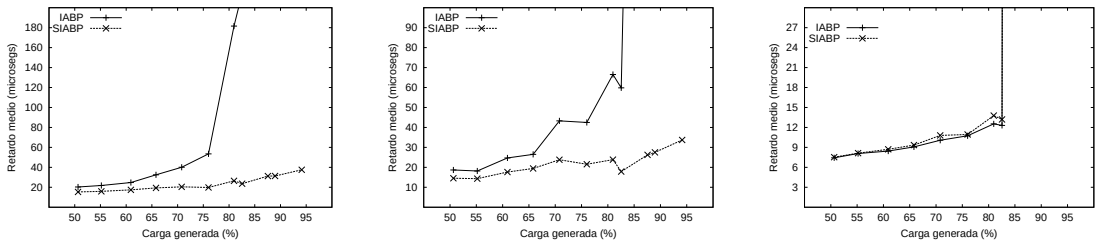
En los resultados anteriores se ha demostrado como el algoritmo de planificación de enlaces IABP permite ofrecer garantías de QoS tanto a tráfico CBR como VBR cuando se emplean 2 o más candidatos. Además, el rango de carga útil que ofrece es mayor que el obtenido con el algoritmo JBP cuando la carga se compone de tráfico CBR. La mejor productividad, manteniendo además las garantías de QoS que necesitan las aplicaciones, se obtiene seleccionando 4 canales virtuales candidatos por cada enlace físico.

Sin embargo, ya se demostró en la Sección 5.1.3 que se puede realizar una aproximación al algoritmo IABP, que supone un importante ahorro en hardware: el algoritmo denominado **SIABP**. Queda por comprobar que este ahorro en hardware no compromete las garantías de QoS ofrecidas a las aplicaciones. Para ello, se emplearán las mismas métricas que en estudios anteriores: utilización media del *crossbar*, retardo y *jitter* medio de flits (tráfico CBR) o de tramas de vídeo (tráfico VBR), y distribución de dichos retardos.

Análisis con tráfico CBR

En primer lugar, se van a comparar las prestaciones obtenidas para ambos algoritmos con tráfico CBR. La utilización media del *crossbar* (Figura 6.55) es prácticamente idéntica para ambos algoritmos. Únicamente cuando la carga sobrepasa la saturación, se puede apreciar como el algoritmo SIABP ofrece una degradación más leve.

En cuanto al retardo medio de los flits (Figura 6.56), se aprecia como las prestaciones se mantienen muy similares para las conexiones de mayores requerimientos. Por el contrario, las conexiones de medios y bajos requerimientos, existe una significativa mejora. Esto se debe a que con SIABP, las conexiones de menos requerimientos ven su prioridad incrementada de forma más rápida que con IABP, especialmente mientras el retardo en cola no es demasiado grande. De ahí que el retardo medio experimentado por sus flits sea menor, pues son considerados antes en la planificación. De todas formas, esto no afecta de forma significativa a las prestaciones obtenidas por las conexiones de mayores requerimientos. Su retardo medio únicamente llega a experimentar un aumento máximo de en torno al 10 % cuando la carga



(a) 64 Kbps (b) 1.54 Mbps (c) 55 Mbps

Figura 6.56: Planificación de enlaces - Algoritmo SIABP: Tráfico CBR. Retardo medio de los flits desde su generación.

llega al 80 %. Hay que recordar que cada vez que se actualiza un valor de prioridad, se multiplica por dos,y que los valores de prioridad se actualizan de manera menos frecuente según aumenta el retardo en cola. Con IABP, la prioridad se incrementa de forma más continua pero también más lenta.

La distribución de los retardos de los flits (Figura 6.57) para los puntos de carga en torno a la saturación confirma lo anterior: existe una mejora de prestaciones para las conexiones de menores requerimientos, manteniéndose las prestaciones para las conexiones que más ancho de banda consumen. Es más, la degradación de prestaciones una vez se ha entrado en saturación para estas conexiones, es menor con el algoritmo SIABP.

Por último, en lo que se refiere al *jitter* (Figura 6.58), de nuevo las conexiones de requerimientos medios y bajos ven drásticamente reducidos los valores de *jitter* experimentados por sus flits. Para las conexiones de 55 Mbps, el *jitter* sufre un ligero incremento, pero aún así los valores se encuentran siempre por debajo de los 2 microsegundos.

De los resultados anteriores, se desprende que, al menos con tráfico CBR, la reducción en complejidad de hardware introducida por SIABP no compromete las prestaciones recibidas por las aplicaciones. Más aún, las prestaciones de las conexiones menos exigentes incluso se ven mejoradas con este algoritmo. Queda por comprobar si sucede lo mismo con tráfico VBR.

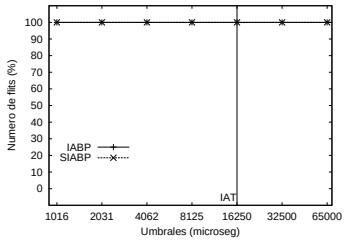
Análisis con tráfico VBR

Las pruebas con tráfico VBR se han ejecutado durante el tiempo suficiente como para transmitir 6 GOPs completos por cada conexión. Además, se deja el tiempo equivalente a dos tiempos de trama como periodo transitorio antes de medir estadísticas. Las conexiones se alinean de forma aleatoria dentro de un tiempo de GOP.

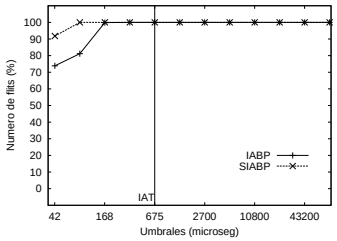
En la Figura 6.59 se muestra la utilización media del *crossbar* obtenida para los dos modelos de inyección simulados (SR y BB). En ambos casos, el Encaminador se satura en torno al 84 % de utilización de los enlaces, aunque la degradación de prestaciones tras la saturación es ligeramente menor con el nuevo algoritmo.

Ahora hay que comprobar si las tramas que son retransmitidas por el Encaminador obtienen las garantías de QoS que necesitan. En la Figura 6.60 se muestra el retardo medio de las tramas de vídeo medido desde su generación. Nótese que el eje vertical está en escala logarítmica. Se puede apreciar como, para ambos modelos de inyección, los dos algoritmos se comportan de manera muy parecida antes de entrar en saturación. Cuando el nivel de carga

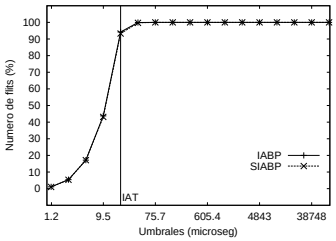
Punto de carga A (carga = 76 %)



(a) 64 Kbps

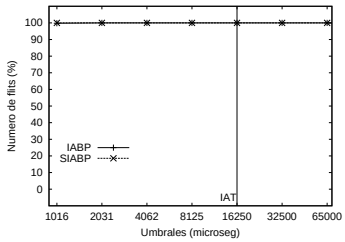


(b) 1.54 Mbps

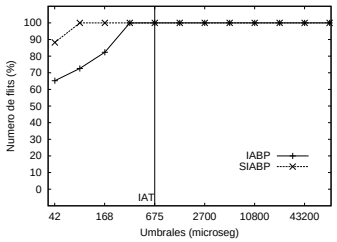


(c) 55 Mbps

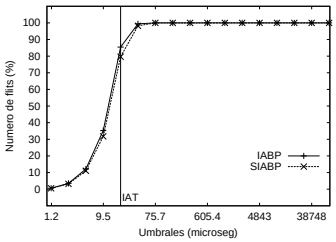
Punto de carga B (carga = 81 %)



(a) 64 Kbps

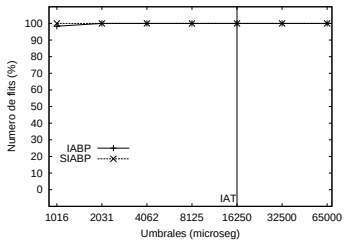


(b) 1.54 Mbps

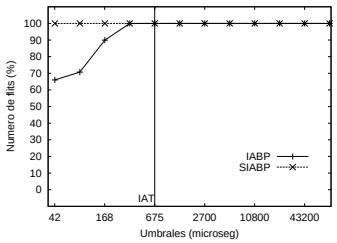


(c) 55 Mbps

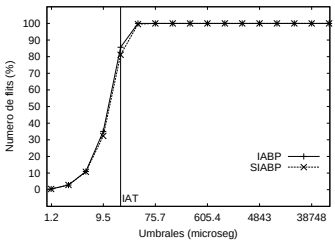
Punto de carga C (carga = 84 %)



(a) 64 Kbps

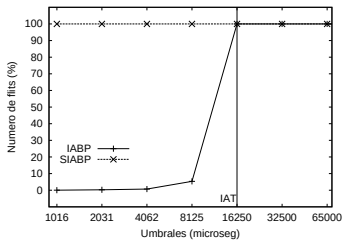


(b) 1.54 Mbps

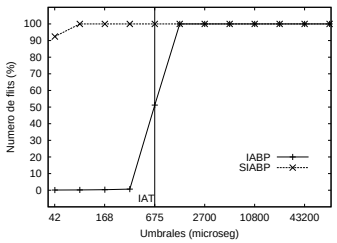


(c) 55 Mbps

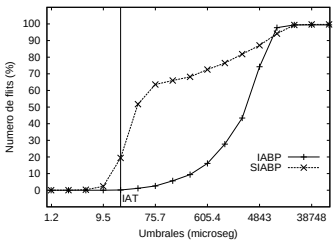
Punto de carga D (carga = 88 %)



(a) 64 Kbps

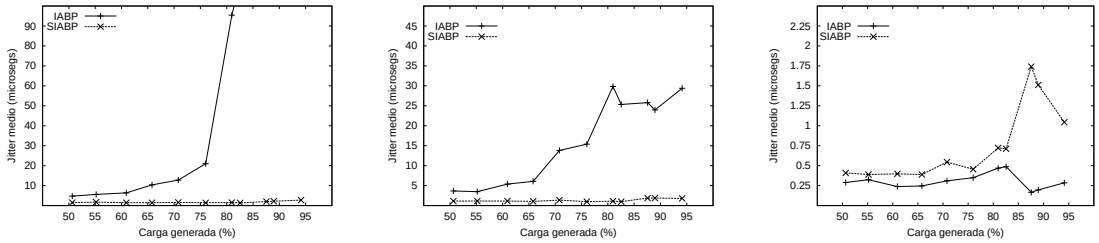


(b) 1.54 Mbps



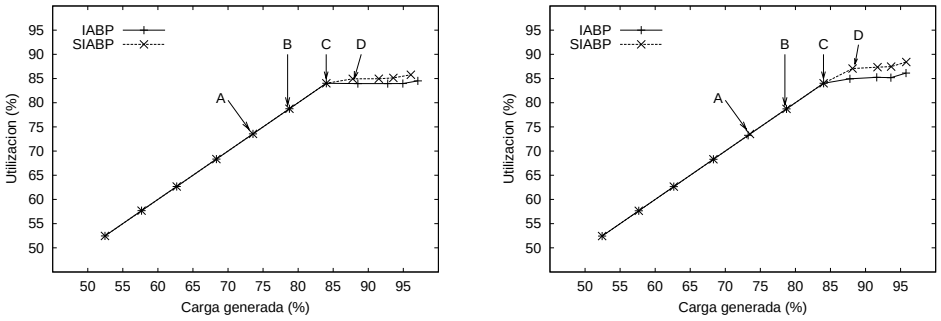
(c) 55 Mbps

Figura 6.57: Planificación de enlaces - Algoritmo SIABP: Tráfico CBR. Distribución del retardo medio de los flits desde su generación.



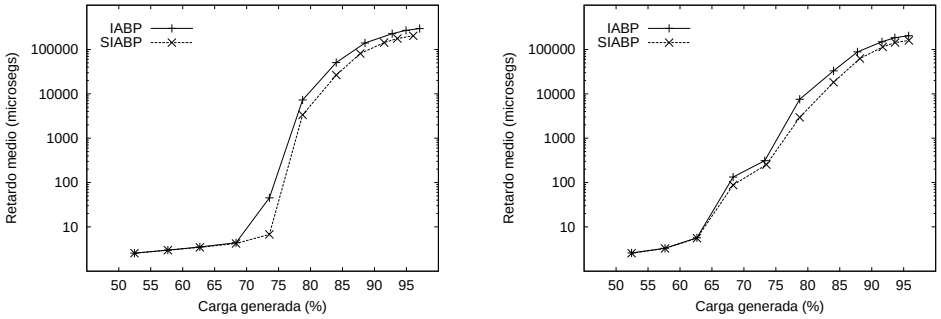
(a) 64 Kbps (b) 1.54 Mbps (c) 55 Mbps

Figura 6.58: Planificación de enlaces - Algoritmo SIABP: Tráfico CBR. *Jitter* medio de los flits desde su generación.



(a) Modelo SR (b) Modelo BB

Figura 6.59: Planificación de enlaces - Algoritmo SIABP: Tráfico VBR. Utilización media del *crossbar*.



(a) Modelo SR (b) Modelo BB

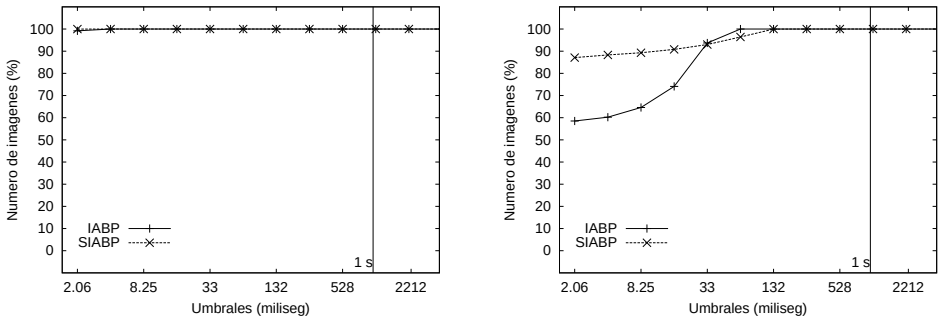
Figura 6.60: Planificación de enlaces - Algoritmo SIABP: Tráfico VBR. Retardo medio de las imágenes desde su generación (escala semilogarítmica).

se aproxima a saturación (en torno al 80 %), SIABP ofrece resultados ligeramente mejores que IABP. Esto es cierto para ambos modelos de inyección, aunque el efecto se aprecia a cargas ligeramente inferiores (por encima del 70 %) con el modelo SR.

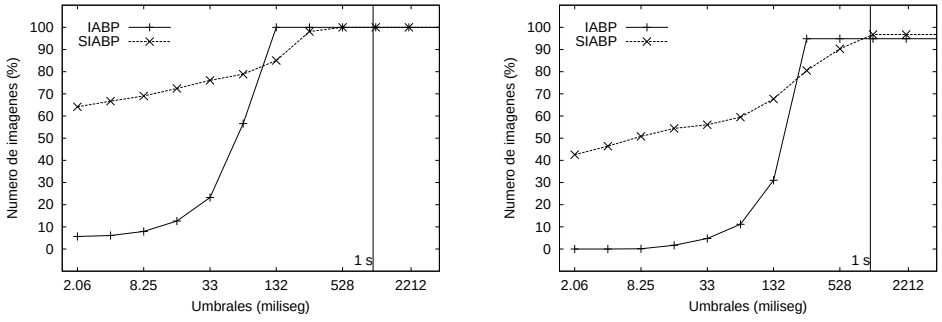
Para obtener más detalles acerca del retardo que experimentan las tramas de vídeo, se va a analizar la distribución de los mismos. Esta distribución se ha obtenido comparando el retardo de cada trama con un conjunto de cotas de retardo. Los valores de estas cotas son relativos al tiempo de trama ($TT = 33$ milisegundos), y son múltiplos y submúltiplos del mismo. El rango de valores considerado oscila entre los 2.06 milisegundos ($TT/16$), y los 2212 milisegundos ($TT \times 64$).

En las Figuras 6.61 y 6.62 se muestran los resultados obtenidos para los puntos de carga en torno a la saturación, marcados como A, B, C y D en la Figura 6.59, para el modelo de inyección SR y el modelo de inyección BB, respectivamente.

Cuando se emplea el modelo de inyección SR, todas las tramas experimentan retardos inferiores a la más estricta de las cotas consideradas cuando la carga está todavía algo lejos de la saturación (punto de carga A, en torno al 74 %). Cuando la carga se va incrementando (punto B, un 79 %), ya hay tramas que no pueden cumplir con las cotas más estrictas, siendo mayor la degradación cuando se emplea el algoritmo IABP. No obstante, las prestaciones todavía son buenas, pues todas las tramas sufren retardos inferiores a 4 veces el tiempo de trama ($TT \times 4$), con ambos algoritmos. Curiosamente, con IABP todas las tramas son capaces de cumplir con la cota del doble del tiempo de trama, mientras que al emplear SIABP el porcentaje de cumplimiento para esa cota es de algo más del 95 %. Cuando la carga aumenta hasta el 84 % (punto C), el Encaminador está ya entrando en saturación, y las tramas apenas son capaces de cumplir con las cotas más estrictas cuando se emplea IABP, mientras que con SIABP, en torno al 65 % de tramas cumplen con la cota más estricta. Por contra, con IABP el 100 % de las tramas tienen retardos inferiores a $TT \times 4$ (132 milisegundos), mientras que con SIABP sólo el 85 % de las tramas cumplen con esa cota. Sin embargo, prácticamente todas las tramas cumplen con el siguiente valor de cota considerado ($TT \times 8$, 264 milisegundos). Ya para el siguiente punto de carga analizado (punto D, en torno al 88 %), el Encaminador está claramente saturado, y se aprecia como hay tramas que no llegan a alcanzar la salida del mismo. De nuevo, hay más tramas que cumplen con las cotas más estrictas cuando se emplea SIABP, e incluso el porcentaje de tramas que logran atravesar el Encaminador es ligeramente mayor con este algoritmo que con IABP. Esto está relacionado con la menor degradación en términos de utilización observada en la Figura 6.59-(a).

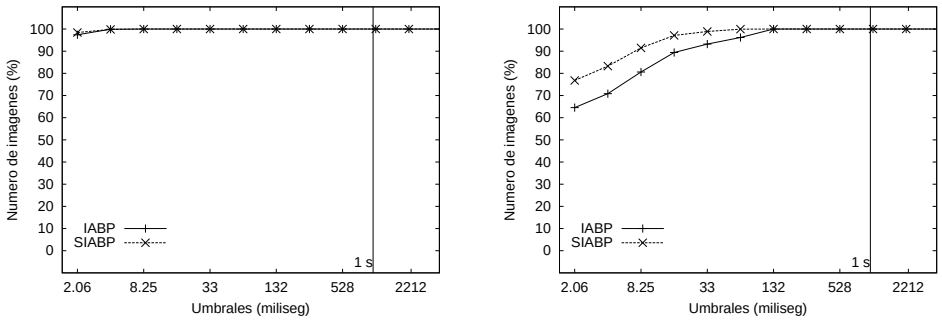


Punto de carga A (carga = 74 %) Punto de carga B (carga = 79 %)

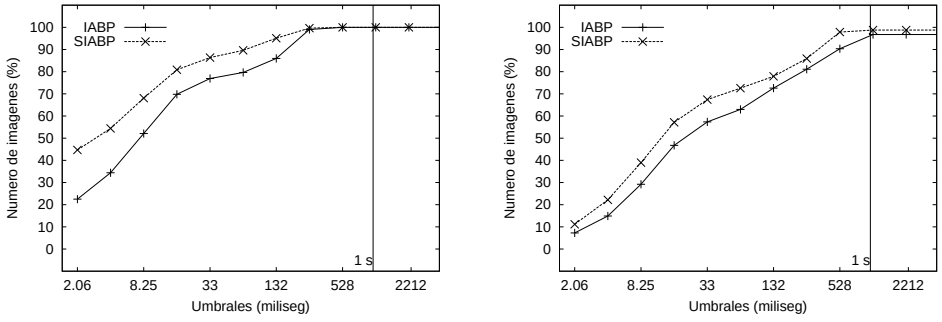


Punto de carga C (carga = 84 %) Punto de carga D (carga = 88 %)

Figura 6.61: Planificación de enlaces - Algoritmo SIABP: Tráfico VBR - modelo SR. Distribución del retardo medio de las imágenes desde su generación.

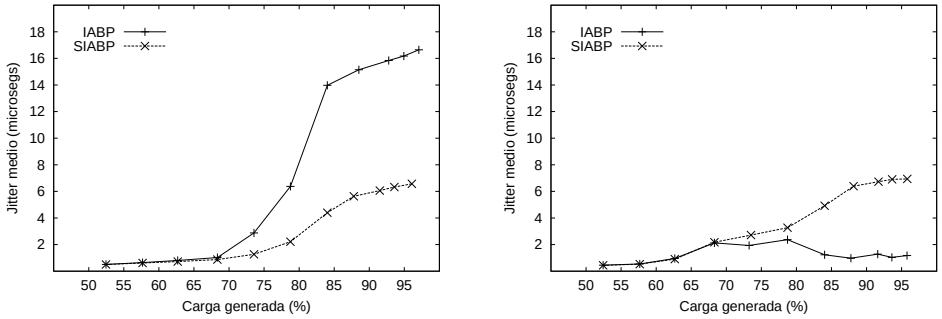


Punto de carga A (carga = 74 %) Punto de carga B (carga = 79 %)



Punto de carga C (carga = 84 %) Punto de carga D (carga = 88 %)

Figura 6.62: Planificación de enlaces - Algoritmo SIABP: Tráfico VBR - modelo BB. Distribución del retardo medio de las imágenes desde su generación.



(a) Modelo SR (b) Modelo BB

Figura 6.63: Planificación de enlaces - Algoritmo SIABP: Tráfico VBR. *Jitter* medio de las imágenes.

Cuando el modelo de inyección empleado es el BB, existen menos diferencias en los resultados obtenidos para ambos algoritmos. Cuando el nivel de carga es moderado, pero todavía lejano a la saturación (punto A, 74 %), no existen diferencias apreciables entre los dos algoritmos, para el rango de umbrales considerado. En ambos casos, todas las tramas cumplen con la cota establecida en torno a los 4 milisegundos ($TT/8$), y más del 95 % de las mismas cumple con la cota más estricta considerada ($TT/16 = 2.06$ milisegundos). Cuando la carga aumenta, acercándose a saturación (punto B, 79 %), existe una mejora a favor del algoritmo SIABP. Todas las tramas sufren retardos inferiores a 132 milisegundos ($TT \times 4$) si se emplea IABP, mientras que con SIABP todas las tramas cumplen con la cota situada en $TT \times 2$ (66 miliseg.), e incluso más del 95 % de las tramas cumplen con la cota situada en $TT/2$ (16.5 miliseg.). En el siguiente punto considerado, en torno al 84 % de carga (punto C), es necesario emplear una cota de $TT \times 8$ (264 milisegundos) para que sea cumplida por prácticamente todas las tramas con ambos algoritmos. Sin embargo, con SIABP hay una mayor cantidad de tramas capaces de cumplir con cotas más estrictas. Una vez que la saturación se ha alcanzado (punto D, 88 %), existen tramas generadas que no llegan a atravesar el Encaminador. Sin embargo, este porcentaje es algo superior con el algoritmo IABP.

Los valores observados para el retardo son bastante razonables, pues una cota típica del retardo para transmisiones de vídeo MPEG-2 es de un segundo entre extremos [143].

El motivo de este comportamiento es que los flits inyectados por fuentes SR normalmente pasan menos tiempo en los *buffers*, pues se inyectan a una tasa menor que los flits generados por fuentes BB. Dentro de ese periodo de tiempo, su prioridad se incrementa más rápidamente. Hay que señalar que con SIABP, cuando el retardo en cola sigue creciendo, la prioridad se dobla con menor frecuencia. Por otro lado, las fuentes BB inyectan sus flits como ráfagas, a una tasa pico fija. Por tanto, sus flits normalmente tienen que esperar más, y sus valores de prioridad se van incrementando cada vez más despacio, de manera más similar a como funciona IABP. Este efecto se hace más patente cuando la carga sobrepasa el 70 %.

Por último, en lo que respecta al *jitter* sufrido por las tramas de vídeo (Figura 6.63), con SIABP se obtienen claramente mejores resultados que con IABP para el modelo de inyección SR, especialmente para cargas superiores al 70 %. Con el modelo de inyección BB, por el contrario, SIABP ofrece resultados ligeramente peores. Aún así, el valor del *jitter* medio se haya siempre por debajo de los 8 microsegundos, que es una cota bastante razonable. Hay que tener en cuenta que existen técnicas de absorción del *jitter* en el destino, con las cuales se pueden tolerar valores de *jitter* de hasta varios milisegundos [133].

De todo lo anterior se puede afirmar que el rango de carga útil, esto es, sin comprometer las

garantías de calidad de servicio de las aplicaciones, obtenido para SIABP es prácticamente idéntico al observado para IABP. Más aún, en algunos casos las prestaciones son incluso mejores, a pesar de la importante reducción en hardware obtenida con SIABP respecto a IABP.

Anteriormente se puso de manifiesto que el número de candidatos influye notablemente en la productividad alcanzada por el Encaminador. Por tanto, en las pruebas realizadas para los análisis reflejados en las secciones siguientes, **el algoritmo empleado como planificador de enlaces será SIABP, con 4 niveles de candidatos**.

6.5.3. Comparativa con Virtual-Clock (VC)

Con el objeto de ponderar el comportamiento del algoritmo SIABP, este apartado se dedica a evaluar su comportamiento con respecto a un conocido algoritmo de planificación del tráfico: el algoritmo ***Virtual-Clock* (VC)** [178]. Este algoritmo, junto con algunos otros más, fue descrito en la Sección 3.5.

El algoritmo VC fue originalmente propuesto para su uso en conmutadores con *buffers* a la salida. Se trata de un algoritmo que asigna prioridades a los paquetes, según sus requerimientos de ancho de banda. La idea es asociar a cada conexión un "reloj virtual", que debe avanzar cada vez que llega un paquete de esa conexión. Este reloj determina en cada momento el tiempo esperado de llegada de un paquete. Cada paquete se marca entonces con el valor del reloj virtual en el momento en que llega a la cola. Los paquetes de transmiten en orden creciente del valor de ese tiempo virtual. Este esquema es fácilmente adaptable al Encaminador Multimedia, como algoritmo de planificación de enlaces.

El algoritmo de cálculo de la prioridad originalmente propuesto para el algoritmo VC es el siguiente:

1. Al recibir el primer paquete del flujo i:

$$VirtualClock_i \leftarrow auxVC_i \leftarrow tiempo\ real$$

2. Al recibir cada paquete del flujo i:

   a) $auxVC_i \leftarrow max(tiempo\ real, auxVC_i)$

   b) $VirtualClock_i \leftarrow (VirtualClock_i + Vtick_i)$;
   $auxVC_i \leftarrow (auxVC_i + Vtick_i)$
   (Para paquetes de tamaño constante, $Vtick_i = 1/AR_i$ paquetes/seg, donde AR_i es la tasa de llegada de paquetes)

   c) Marcar el paquete con el valor de $auxVC_i$

3. Insertar el paquete en su cola de salida. Los paquetes se insertan y se sirven según el orden creciente de sus valores marcados.

Se puede observar como hay dos variables asociadas a cada flujo: $VirtualClock_i$ y $auxVC_i$. Estas dos variables son necesarias para realizar una monitorización del consumo de ancho de banda por parte de cada flujo, y controlar que ninguna conexión exceda la cuota de ancho de banda que se le haya asignado. En el caso del Encaminador Multimedia, se asume que las conexiones son fiables, y por tanto, no sería necesario efectuar dicha monitorización. Por tanto, sólo es necesaria una variable asociada a cada flujo: $auxVC_i$.

Por otro lado, se observa como el reloj virtual avanza con paso igual a $Vtick_i = 1/AR_i$, cuando los paquetes son de tamaño constante. Este es precisamente el caso del MMR, donde la conmutación se realiza sobre flits de tamaño fijo. Además, $1/AR_i$ es precisamente el tiempo entre llegadas especificado para los flits de una conexión. Esto implica que el reloj virtual avanzará con paso igual al IAT (*Inter Arrival Time*) de la conexión.

Con todo lo anterior, el cálculo de las prioridades de los flits según el algoritmo *Virtual Clock* adaptado al MMR quedaría de la siguiente forma:

1. Al recibir el primer flit de la conexión i:

$$auxVC_i \leftarrow tiempo\ real$$

2. Al recibir cada flit de la conexión i:

   a) $auxVC_i \leftarrow max(tiempo\ real, auxVC_i)$

   b) $auxVC_i \leftarrow (auxVC_i + IAT_i)$

   c) Marcar el flit con el valor de $auxVC_i$

3. Los flits más prioritarios son los de menor valor marcado.

En cuanto a la complejidad de implementación, con VC la prioridad de los flits no necesita ser recalculada en cada ciclo de flit. Al llegar el flit se le asigna el valor adecuado y éste ya no se modifica mientras el flit permanece en el Encaminador. Esto es una ventaja de VC sobre los algoritmos propuestos para el Encaminador Multimedia. Ahora bien, el valor de las prioridades en VC, que se toma de un reloj de tiempo real, es siempre creciente, incluso entre flits consecutivos y distintos. Por tanto, tarde o temprano el registro que almacena este reloj llegará a su valor máximo, y empezará a contar de nuevo desde 0. La consecuencia de esto es que se asignarán menores tiempos de reloj virtual a los flits que han llegado más tarde, y por tanto tendrán mayor prioridad que flits que llevan más tiempo esperando a ser transmitidos.

En los siguientes apartados se van a comparar las prestaciones obtenidas con el algoritmo VC como planificador de enlaces en el MMR, frente al algoritmo SIABP, que es el algoritmo propuesto que admite una implementación más eficiente, ofreciendo también las mejores prestaciones.

Siguiendo el mismo esquema empleado hasta ahora, se van a presentar en primer lugar los resultados obtenidos con una carga compuesta únicamente por tráfico CBR, y posteriormente, se presentarán los mismos datos, pero con tráfico VBR. Todas las simulaciones se configuran con los parámetros de la Tabla 6.5. Los planificadores de los enlaces generan 4 niveles de candidatos.

Análisis con tráfico CBR

Al igual que en el resto de pruebas llevadas a cabo con carga CBR, estas simulaciones se han ejecutando durante 600 ciclos del planificador, más 10 ciclos adicionales que conforman un periodo transitorio antes de recoger los datos estadísticos.

La Figura 6.64 muestra la utilización media del *crossbar* para el algoritmo SIABP y para VC. Se puede observar como apenas existen diferencias entre ambos, sólo una ligerísima mejora

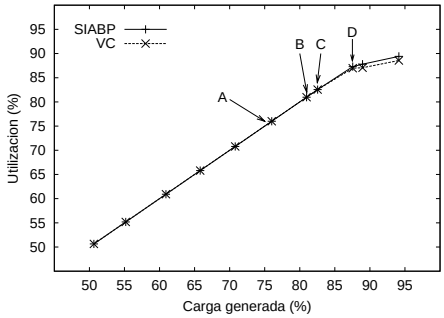


Figura 6.64: Planificación de enlaces - Algoritmo SIABP vs. VC: Tráfico CBR. Utilización media del *crossbar*.

a favor de SIABP una vez se ha entrado en saturación. Esto indica que el Encaminador Multimedia es capaz de trasegar la misma cantidad de flits con ambos algoritmos. Hay que comprobar qué garantías de QoS se ofrecen a cada flujo de datos.

La distribución del retardo experimentado desde la generación por los flits de cada tipo de conexión ofrece una idea clara de la QoS que reciben las conexiones multimedia. En las gráficas de la Figura 6.65 se muestra esta métrica para los niveles de carga en torno a saturación, marcados como A, B, C y D en la Figura 6.64.

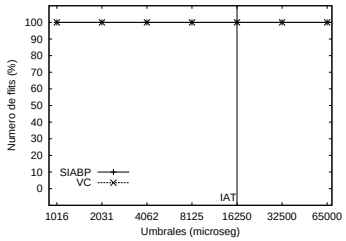
Para las conexiones de menos requerimientos (64 Kbps), se observa como cuando el nivel de carga está próximo a saturación, el porcentaje de flits que no pueden cumplir con las cotas más estrictas aumenta progresivamente cuando se usa VC. Por el contrario, todos los flits emitidos obtienen retardos inferiores a la cota más estricta cuando se emplea SIABP, incluso cuando el Encaminador ya se ha saturado (punto D, 87 %). El retardo máximo experimentado por los flits es también bastante más elevado cuando se emplea VC. Por ejemplo, para el punto de carga B (81 %), la cota más estricta que cumplen todos los flits se sitúa en los 8125 microsegundos con VC, mientras que con SIABP está por debajo de los 1016 microsegundos.

Las conexiones de requerimientos medios (1.54 Mbps) exhiben un comportamiento similar al anteriormente comentado para las conexiones de 64 Kbps. Al aumentar la carga, el porcentaje de flits que no cumplen las cotas más estrictas se incrementa progresivamente cuando se usa VC, mientras que con SIABP, el porcentaje de flits que no cumple la cota más estricta no pasa del 10 %. En todos los puntos de carga analizados, incluso cuando ya se ha entrado en saturación, todos los flits experimentan retardos inferiores a 84 microsegundos.

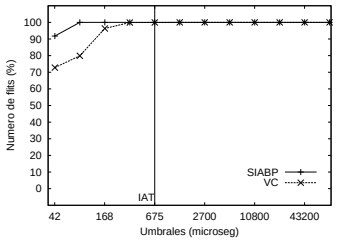
Por último, es en las conexiones de mayores requerimientos (55 Mbps) donde se hace patente más el impacto causado por la saturación. En el nivel de carga más alto considerado (punto D, 87 %), se observa como los flits son incapaces de cumplir con las cotas más estrictas, independientemente de si se usa VC o SIABP. No obstante, con SIABP la degradación de QoS es menor, pues hay mayor cantidad de flits que obtienen retardos inferiores a cotas más estrictas. Por ejemplo, el 50 % de los flits sufren retardos inferiores a 36 microsegundos ($2 \times IAT$) con SIABP, mientras que sólo apenas un 20 % de los flits sufren retardos inferiores a la misma cota, cuando se emplea VC. Para el resto de niveles de carga, el comportamiento obtenido para ambos algoritmos es muy similar.

La conclusión que se obtiene a la vista de los resultados anteriores es que el Encaminador Multimedia puede garantizar la QoS en términos de retardo a todas las conexiones CBR hasta mayores niveles de carga cuando se emplea SIABP que cuando se emplea VC. Antes de producirse la saturación, no existe gran diferencia en las prestaciones obtenidas por las

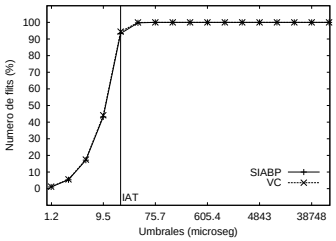
Punto de carga A (carga = 76 %)



(a) 64 Kbps

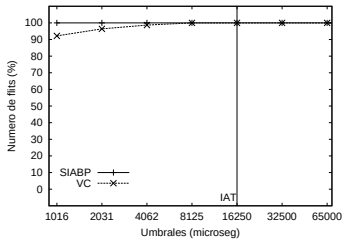


(b) 1.54 Mbps

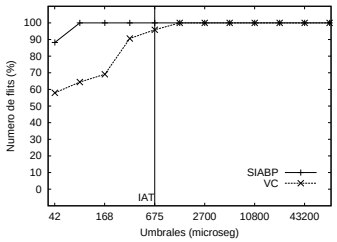


(c) 55 Mbps

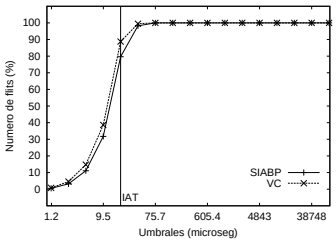
Punto de carga B (carga = 81 %)



(a) 64 Kbps

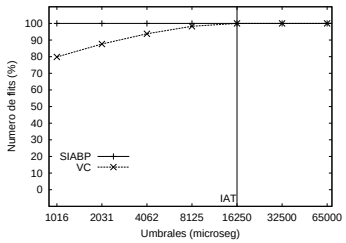


(b) 1.54 Mbps

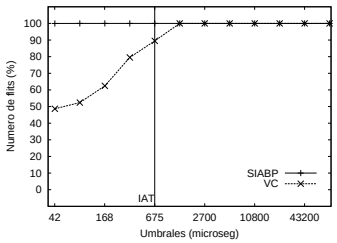


(c) 55 Mbps

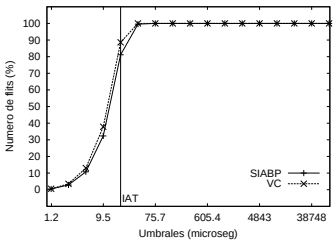
Punto de carga C (carga = 84 %)



(a) 64 Kbps

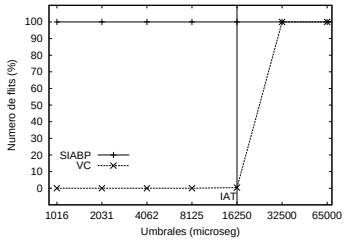


(b) 1.54 Mbps

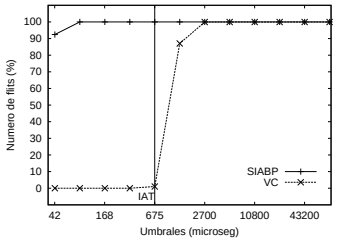


(c) 55 Mbps

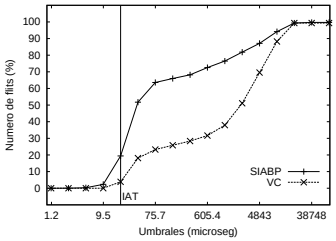
Punto de carga D (carga = 87 %)



(a) 64 Kbps

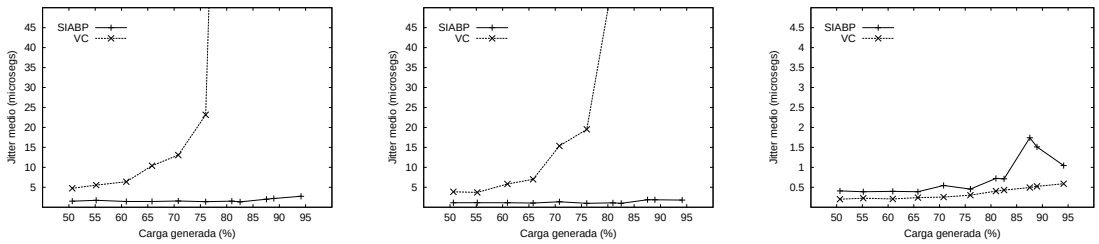


(b) 1.54 Mbps



(c) 55 Mbps

Figura 6.65: Planificación de enlaces - Algoritmo SIABP vs. VC: Tráfico CBR. Distribución del retardo medio de los flits desde su generación.



(a) 64 Kbps (b) 1.54 Mbps (c) 55 Mbps

Figura 6.66: Planificación de enlaces - Algoritmo SIABP vs. VC: Tráfico CBR. *Jitter* medio de los flits.

conexiones de más requerimientos. Sin embargo, las conexiones de requerimientos menores, ven degradada fuertemente la QoS que reciben.

Por último, queda por analizar la variación del retardo o *jitter* experimentada por los flits de las conexiones. El valor medio de esta magnitud se muestra en la Figura 6.66, para los tres tipos de tráfico considerados. En esta Figura se observa de nuevo como con VC las conexiones de menos requerimientos resultan perjudicadas también en términos de *jitter* en todo el rango de carga simulado. El valor del *jitter* incluso crece desmesuradamente cuando la carga se acerca al 80 %. Por el contrario, las conexiones de más requerimientos obtienen valores de *jitter* inferiores cuando se emplea VC, aunque la diferencia con los valores obtenidos con SIABP es mínima.

En conclusión, se puede decir que con el algoritmo SIABP el Encaminador Multimedia ofrece un mejor soporte a las garantías de QoS de los distintos tipos de conexiones CBR. Con el algoritmo VC, por el contrario, las conexiones de menores requerimientos ven comprometidas sus garantías de QoS cuando la carga sobrepasa el 76 % del ancho de banda de los enlaces, aproximadamente. Cuando se emplea SIABP, esto no sucede hasta niveles de carga superiores al 85 %.

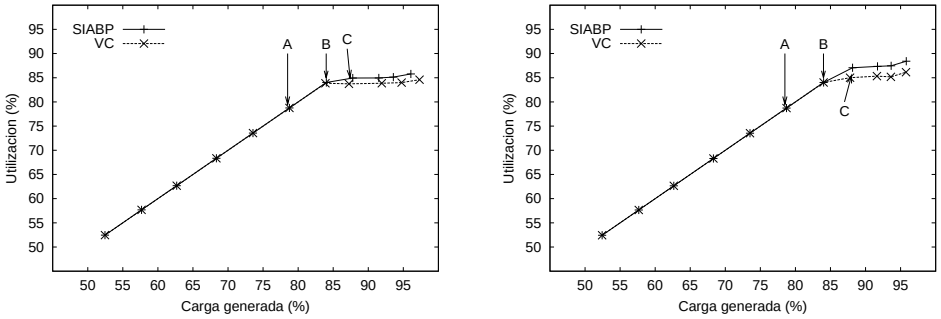
Seguidamente se llevará a cabo el mismo estudio comparativo entre SIABP y VC, pero esta vez empleando tráfico VBR.

Análisis con tráfico VBR

En el estudio presentado a continuación, se han utilizado los dos modelos de inyección de tráfico VBR disponibles: SR y BB. Las simulaciones se han ejecutado hasta que cada conexión ha podido transmitir a través del Encaminador al menos 6 GOPs. Como es habitual, se ha dejado el tiempo equivalente a dos tiempos de trama como periodo transitorio.

La Figura 6.67 muestra la utilización media del *crossbar* para ambos algoritmos objeto de estudio, y para los dos modelos de inyección considerados. Las gráficas tienen un aspecto similar a la obtenida con tráfico CBR. De nuevo, apenas hay diferencias en los resultados obtenidos con los dos algoritmos. Únicamente se aprecia una ligera mejor utilización del *crossbar* cuando ya se ha entrado en saturación, con el algoritmo SIABP. Hay que analizar por tanto la QoS ofrecida a las conexiones. Puesto que los flujos empleados son conexiones de vídeo MPEG-2, las métricas de la QoS se van a medir sobre imágenes en lugar de sobre flits individuales.

En las gráficas de la Figura 6.68 se muestra la distribución de los retardos de las tramas de vídeo, para los niveles de carga situados en torno a la saturación, para el modelo de



(a) Modelo SR (b) Modelo BB

Figura 6.67: Planificación de enlaces - Algoritmo SIABP vs. VC: Tráfico VBR. Utilización media del *crossbar*.

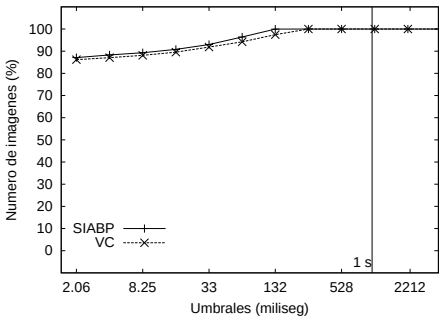
inyección SR. En los puntos de carga A y B (79 % y 84 %, respectivamente), el Encaminador todavía no se ha saturado. Se puede observar como el comportamiento de las tramas con ambos algoritmos es muy similar. En el punto de carga C (88 %) el Encaminador ya ha entrado en saturación, independientemente del algoritmo empleado. Esto se aprecia porque hay tramas que no llegan a ser retransmitidas a través del *crossbar*. Aquí se aprecia una ligera ventaja en favor de VC, aunque en ese caso el retardo máximo experimentado por las tramas transmitidas es el doble del experimentado con SIABP.

Las gráficas de la Figura 6.69 muestran también la distribución de los retardos de las tramas en torno a saturación, para el modelo de inyección BB. Para este caso, en todos los puntos de carga analizados, SIABP ofrece mejores resultados que VC. En general, las tramas son capaces de cumplir con cotas más estrictas cuando se emplea SIABP. Al contrario de lo que sucedía con el modelo SR, esta tendencia se mantiene incluso cuando ya se ha entrado en saturación (punto C, 89 %). Además, para el punto de carga A (79 %), todas las tramas experimentan retardos inferiores a 66 milisegundos ($2 \times$ *tiempo de trama*) cuando se emplea SIABP, siendo el doble ($4 \times$ *tiempo de trama*) cuando se emplea VC.

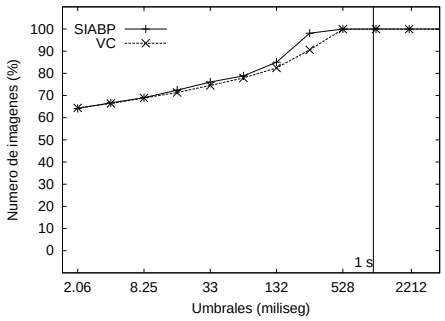
Se podría decir por tanto que, al menos en lo que respecta al retardo de las tramas, las prestaciones de VC y SIABP antes de saturación son comparables cuando las tramas se inyectan con el modelo SR. Cuando es el modelo BB el que se utiliza para la inyección de las tramas, por el contrario, SIABP es capaz de ofrecer cotas más estrictas a un mayor porcentaje de tramas que VC. Queda por comprobar qué sucede con el *jitter* experimentado por las tramas.

La Figura 6.70 muestra el valor medio del *jitter* obtenido, para los dos modelos de inyección considerados. En el caso del modelo SR, no existe gran diferencia entre los dos algoritmos. Sólo cuando la carga aumenta por encima del 80 % existe una mejora más notable a favor de SIABP. El caso del modelo BB es justo al contrario. Hasta una carga de entre el 70 % y el 80 %, los valores de *jitter* son similares, y a partir del 80 % de carga, existe un mejora, pero esta vez a favor de VC. No obstante, en todos los casos se trata de valores de unos pocos microsegundos, que apenas influyen en la percepción de la secuencia de vídeo. Es más, hay que recordar que existen técnicas de absorción del *jitter* capaces de tolerar *jitters* de varios milisegundos [133].

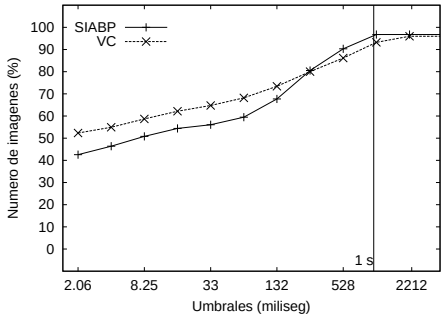
Como conclusión del estudio llevado a cabo con el algoritmo *Virtual Clock* en el MMR, se puede decir que para ciertos tipos de tráfico ofrece prestaciones comparables a SIABP. No obstante, SIABP es capaz de obtener un mayor rango de carga útil para todos los tipos



Punto de carga A (carga = 79 %)

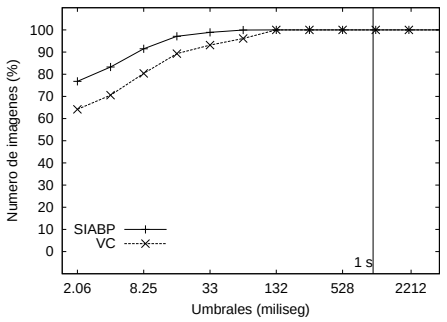


Punto de carga B (carga = 84 %)

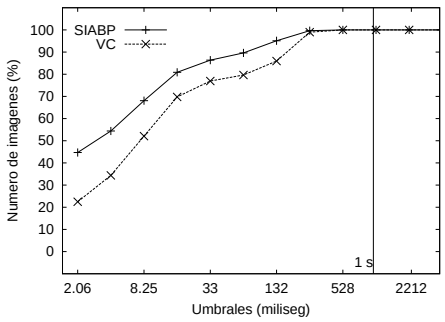


Punto de carga C (carga = 88 %)

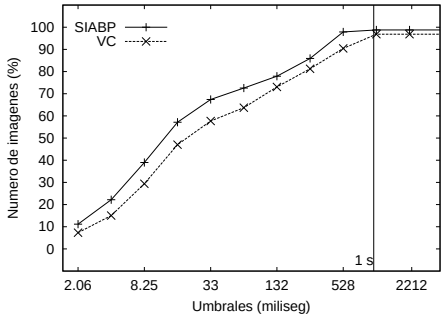
Figura 6.68: Planificación de enlaces - Algoritmo SIABP vs. VC: Tráfico VBR - modelo SR. Distribución del retardo medio de las imágenes desde su generación.



Punto de carga A (carga = 79 %)

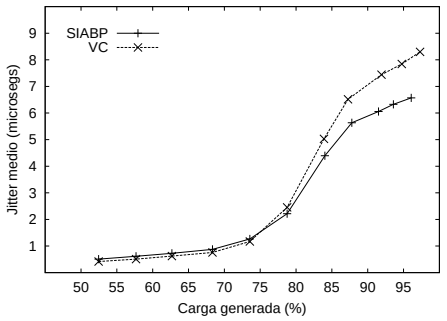


Punto de carga B (carga = 84 %)

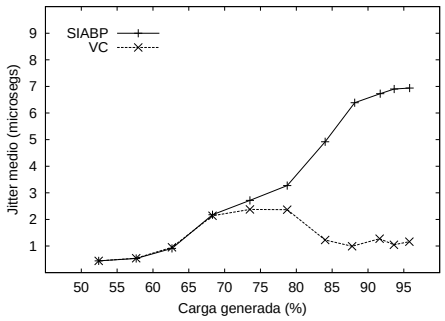


Punto de carga C (carga = 89 %)

Figura 6.69: Planificación de enlaces - Algoritmo SIABP vs. VC: Tráfico VBR - modelo BB. Distribución del retardo medio de las imágenes desde su generación.



(a) Modelo SR



(b) Modelo BB

Figura 6.70: Planificación de enlaces - Algoritmo SIABP vs. VC: Tráfico VBR. *Jitter* medio de las imágenes.

de conexiones. Esto se hace especialmente patente en casos como la mezcla de tráfico CBR compuesta por conexiones con requerimientos bastante dispares, donde las conexiones de menos requerimientos veían reducido el nivel de QoS recibido.

6.6. Evaluación de los algoritmos de planificación del conmutador

Una vez se han analizado las prestaciones ofrecidas por los distintos algoritmos de planificación de enlaces propuestos, llega el turno de evaluar el comportamiento del otro componente clave en el cálculo de la planificación del tráfico: el *algoritmo de planificación del conmutador*. Hay que recordar que el objetivo de este algoritmo es obtener un emparejamiento libre de conflictos entre los canales virtuales de entrada y de salida del Encaminador, tomando como punto de partida los vectores de candidatos facilitados por los planificadores de los enlaces físicos.

En el Capítulo 5 se presentaron dos algoritmos de planificación del conmutador diseñados especialmente para el MMR. Ambos están basados en el uso de una matriz de selección y de un vector de conflictos para resolver la contención por el uso de los recursos que pueda ocurrir entre los distintos canales virtuales candidatos. Sin embargo, los algoritmos difieren en la organización de estas estructuras de datos.

El primero de los algoritmos, **COA (*Conflict Order Arbiter*)**, organiza la matriz de selección y el vector de conflictos en niveles de candidatos, de manera que los emparejamientos se llevan a cabo entre candidatos de nivel uno (los más prioritarios de cada enlace físico) en primer lugar, luego entre candidatos de nivel dos, y así sucesivamente (Sección 5.2.1).

El segundo algoritmo propuesto, **CCA (*Candidate Conflict Arbiter*)**, difiere respecto del anterior en que los candidatos se consideran todos a la vez, sin distinción entre niveles (Sección 5.2.2).

En esta Sección se van a analizar las prestaciones ofrecidas por ambos algoritmos de planificación del *crossbar*. Para ello, se va a proceder de forma similar al método de análisis seguido hasta ahora.

Se va a emplear como referencia un algoritmo clásico de emparejamiento como es el ***Wave Front Arbiter* (WFA)** [162]. Diversos estudios han demostrado que WFA ofrece prestaciones similares a las de otros algoritmos como PIM [17] o iSLIP [110], o incluso superiores cuando el tráfico no es uniforme [115], mientras que su complejidad de implementación es menor. WFA por tanto, es un algoritmo apropiado para una implementación práctica, que ofrece prestaciones comparables e incluso a veces superiores a las de otras aproximaciones, con un menor coste hardware.

Además, tal y como se describió en la Sección 3.4.1, el funcionamiento del algoritmo WFA se basa en el empleo de una matriz de puntos de decisión. La organización empleada es en cierto modo similar a la utilizada por los algoritmos desarrollados para el MMR, basados en una matriz de selección. Esta es otra de las razones que han motivado la elección del algoritmo WFA como punto de referencia.

En la Tabla 6.6 se muestran los valores establecidos para los parámetros comunes a todas las pruebas. Tal y como se indicó en la conclusión de la Sección anterior, el algoritmo planificador de enlaces será fijo, y consistirá en el algoritmo SIABP con 4 niveles de candidatos.

Número de puertos	4
Número de canales virtuales	128
Valor de K	16
Tamaño de phit	16 bits
Tamaño de flit	1024 bits
Ancho de banda de los enlaces	1.24 Gbps
Tamaño de los *buffers* del MMR	1 flit
Tipo de planificador	SIABP + {COA, CCA, WFA}
Número de candidatos	4
Tipo de CAC	Sección 4.3.3
Factor de concurrencia	16
Semilla para números aleatorios	12345

Tabla 6.6: Evaluación de los algoritmos de planificación del conmutador: Parámetros comunes a todas las pruebas.

En primer lugar, se analizarán las prestaciones obtenidas con los distintos algoritmos con tráfico CBR, para continuar el análisis con tráfico VBR, considerando ambos modelos de inyección de tramas.

6.6.1. Evaluación con tráfico CBR

Al igual que en casos anteriores, las pruebas llevadas a cabo con tráfico CBR se ejecutan durante 600 ciclos del planificador, empleando 10 ciclos más al inicio de la simulación como periodo transitorio. Los resultados que se van a presentar seguirán la misma estructura que en Secciones anteriores: se comienza obteniendo la cantidad de información que es capaz de trasegar el Encaminador con cada configuración simulada, para continuar analizando los distintos índices de la QoS recibida por los flujos de datos.

La Figura 6.71 muestra la utilización media del *crossbar* que se obtiene para los algoritmos de planificación considerados. Se puede comprobar como la única diferencia entre los dos nuevos algoritmos de planificación propuestos para el MMR está en la menor pérdida de prestaciones exhibida cuando se emplea el algoritmo de planificación CCA (sin niveles), una vez se ha entrado en saturación. Es decir, el hecho de considerar todos los candidatos a la vez a la hora de calcular la planificación mejora las prestaciones del Encaminador, al menos en términos de cantidad de datos retransmitidos. Con respecto al comportamiento de WFA, se aprecia como el Encaminador entra en saturación cuando la carga generada excede el 75 % de utilización de los enlaces. Este nivel es bastante inferior al alcanzado por los nuevos algoritmos (entre un 10 % y un 15 % menor), lo que sugiere que el empleo de un algoritmo de emparejamiento tradicional no es adecuado para su empleo en el entorno concreto del MMR. Seguidamente, hay que comprobar las garantías de QoS ofrecidas a las aplicaciones en cada caso.

El retardo medio de los flits desde su generación se muestra en las gráficas de la Figura 6.72 para los tres tipos de conexiones CBR consideradas. Para las conexiones de requerimientos medios y bajos (gráficas (a) y (b) de la Figura), la diferencia existente entre ambas aproximaciones es muy pequeña, pero se decanta ligeramente en favor del algoritmo CCA, que

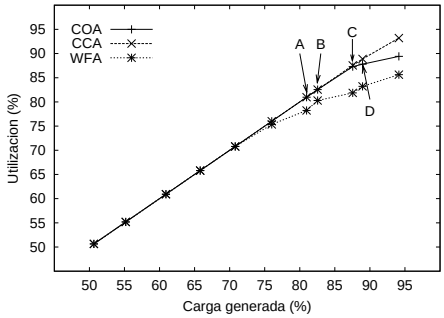
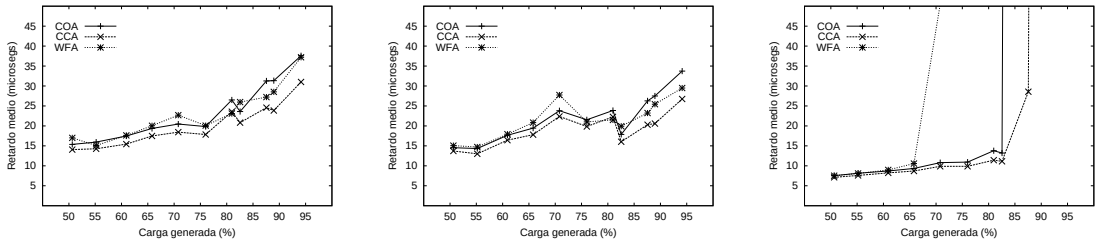


Figura 6.71: Planificación del conmutador: Tráfico CBR. Utilización media del *crossbar*.



(a) 64 Kbps (b) 1.54 Mbps (c) 55 Mbps

Figura 6.72: Planificación del conmutador: Tráfico CBR. Retardo medio de los flits desde su generación.

considera todos los candidatos a la vez. Esta diferencia a favor del algoritmo CCA es mucho más acusada en las conexiones de mayores requerimientos (gráfica (c) de la Figura), pues existe un incremento en la productividad máxima de en torno al 5 % del ancho de banda de los enlaces. La conclusión es que el aumento en productividad obtenido al considerar todos los candidatos a la vez se traduce en que, en general, los flits tardan menos en ser retransmitidos. Puesto que las conexiones de mayores requerimientos son las que más flits generan, son las más beneficiadas por esta mejora.

En lo que respecta a las prestaciones obtenidas con el algoritmo WFA, las conexiones de requerimientos bajos y medios obtienen retardos medios similares a los obtenidos con los otros dos algoritmos. Sin embargo, las conexiones de mayores requerimientos (Figura 6.72-(c)), sufren un drástico incremento en sus retardos cuando la carga supera el 65 %. Esta sería precisamente la productividad máxima alcanzable por el MMR con el algoritmo WFA como planificador del conmutador, cuando se emplea tráfico CBR.

La mejora en productividad del algoritmo CCA sobre el algoritmo COA que se ha observado, se puede confirmar analizando la tasa de emparejamientos conseguidos por ambos algoritmos, que da una medida de la calidad del emparejamiento (ver la Sección 6.2.3). Esta tasa se calcula como el cociente entre el número de puertos finalmente emparejados frente a número de puertos con algún candidato para emparejar, oscilando su valor entre 0 (no se ha conseguido ningún emparejamiento) y 1 (se han conseguido emparejar todos los puertos con algún candidato).

En la Figura 6.73 se muestra el valor medio obtenido para esta tasa en función de la carga generada, para los tres algoritmos de planificación. Efectivamente, la tasa de emparejamientos

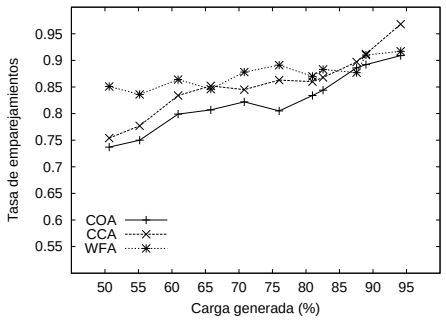


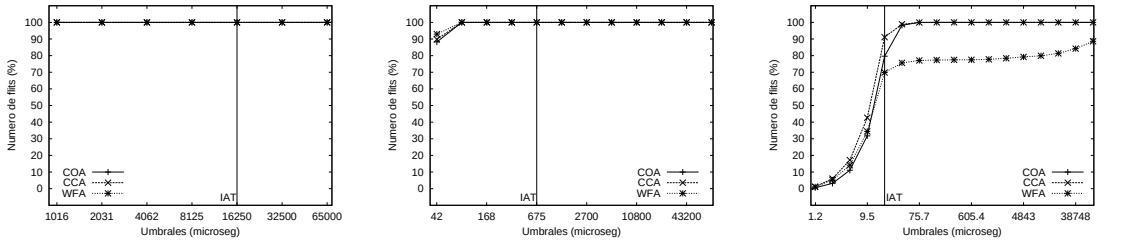
Figura 6.73: Planificación del conmutador: Tráfico CBR. Calidad del emparejamiento.

obtenida para el algoritmo CCA es siempre algo superior a la obtenida con el algoritmo COA. A cargas medias y moderadas, la mejor calidad en los emparejamientos no se ve reflejada en la utilización media del *crossbar*. El motivo es que todavía quedan suficientes "huecos" libres en el ciclo de planificación para poder "ir encajando" los flits. Así, aunque los emparejamientos conseguidos en cada ciclo de flit no sean tan eficientes con el algoritmo COA como con CCA, la utilización media no se resiente, porque los flits acaban siendo retransmitidos. Sin embargo, el retardo experimentado por los flits sí que se ve afectado, siendo ligeramente superior cuando se emplea el algoritmo COA. Ahora bien, cuando el Encaminador está acercándose a la saturación, no siempre será posible encajar los flits en huecos libres, por lo que el mejor aprovechamiento del *crossbar* que realiza el algoritmo CCA en cada ciclo de flit se traduce en una mayor utilización media del mismo, y en una mayor productividad máxima.

El caso del algoritmo WFA es diferente. Al contrario de lo que sucede con los otros dos algoritmos, el valor de la tasa de emparejamientos obtenidos por WFA se mantiene más o menos constante en todo el rango de carga. En gran parte de los puntos de carga generados, es superior a los valores obtenidos con COA y CCA. Esto indica que el grado de emparejamientos conseguidos es bastante bueno. Sin embargo, puesto que en ese proceso no se tienen en cuenta los requerimientos de las aplicaciones, no siempre los flits seleccionados para transmitir se corresponden con los que más urgencia tienen en ser transmitidos. Esto afecta en mayor medida a los flits de las conexiones más exigentes. Hay que tener en cuenta que un retardo excesivo en la planificación de los flits de estas conexiones provoca la formación de colas en los *buffers* según se van generando. Así, el retardo sufrido por los mismos se incrementa drásticamente. En última instancia se provoca la degradación de la utilización media del *crossbar*, pues los flits se quedan almacenados en los *buffers* esperando casi indefinidamente a que llegue su turno para ser retransmitidos. En pocas palabras, la idea se podría expresar diciendo que, a cargas moderadas, la tasa de planificación de los flits de mayores requerimientos es menor que su tasa de generación, provocando el crecimiento incontrolado de las colas de flits. Evidentemente, esta situación también se podría llegar a producir para el resto de conexiones, pero puesto que su tasa de generación es mucho menor, el nivel de carga necesario todavía no se llega a alcanzar. Y para el resto de algoritmos este fenómeno también sucede, pero a niveles de carga significativamente superiores.

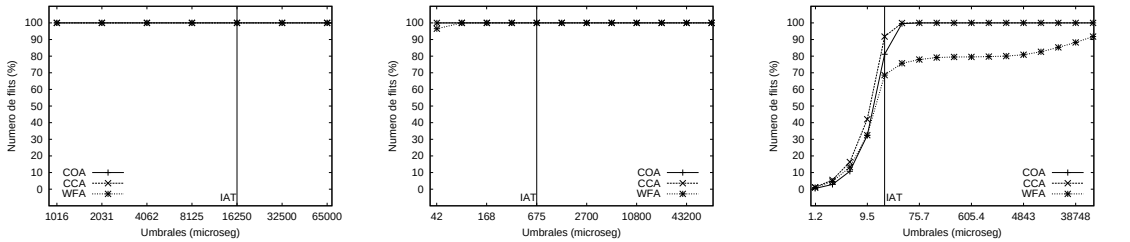
La distribución del retardo experimentado por los flits de las distintas conexiones ofrece una información más detallada acerca de dichos retardos. Esta distribución se obtiene calculando el porcentaje de los flits generados que obtienen retardos inferiores a una serie de cotas. Dichas cotas son múltiplos y submúltiplos del IAT (tiempo entre llegadas de dos flits consecutivos) de cada conexión, con lo que serán tanto más ajustadas cuanto mayores sean los requerimientos

Punto de carga A (carga = 81 %)



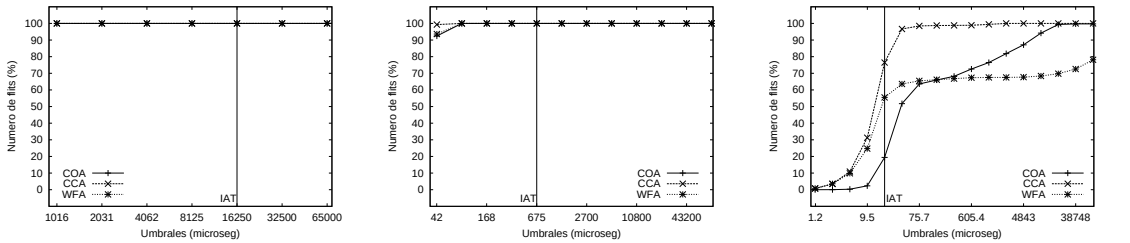
(a) 64 Kbps (b) 1.54 Mbps (c) 55 Mbps

Punto de carga B (carga = 83 %)



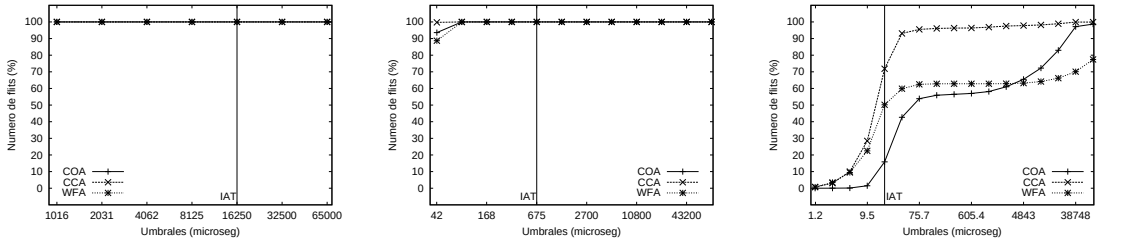
(a) 64 Kbps (b) 1.54 Mbps (c) 55 Mbps

Punto de carga C (carga = 87 %)



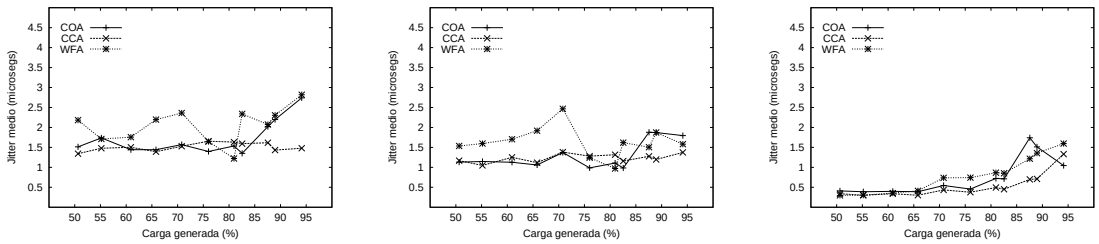
(a) 64 Kbps (b) 1.54 Mbps (c) 55 Mbps

Punto de carga D (carga = 89 %)



(a) 64 Kbps (b) 1.54 Mbps (c) 55 Mbps

Figura 6.74: Planificación del conmutador: Tráfico CBR. Distribución del retardo medio de los flits desde su generación.



(a) 64 Kbps (b) 1.54 Mbps (c) 55 Mbps

Figura 6.75: Planificación del conmutador: Tráfico CBR. *Jitter* medio de los flits.

de ancho de banda de la conexión. Las gráficas de las Figura 6.74 muestran los resultados obtenidos para los puntos de carga marcados como A, B, C y D en la Figura 6.71, situados en torno a la saturación.

Para las conexiones de requerimientos bajos y medios, todos los flits son capaces de cumplir con cotas muy estrictas incluso con las cargas más elevadas. Sólo en el caso de las conexiones de 1.54 Mbps existe una pequeña diferencia a favor del algoritmo CCA en el porcentaje de flits que cumplen con la cota más estricta. Cuando se emplea WFA, las conexiones de requerimientos medios ven reducido el porcentaje de sus flits que es capaz de cumplir con los umbrales más estrictos.

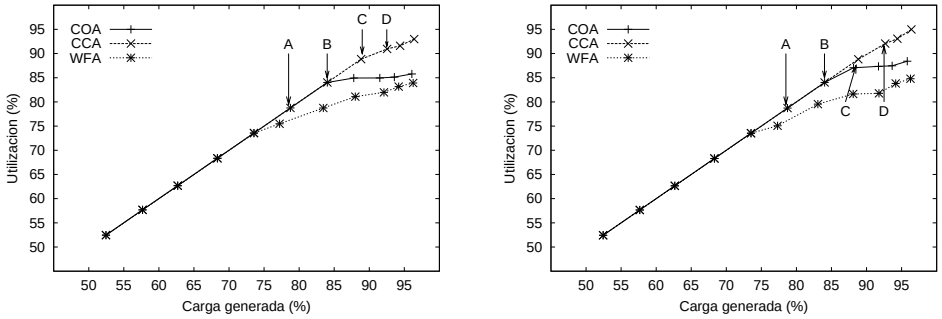
De nuevo, las diferencias entre los tres algoritmos de planificación se hacen más patentes sobre las conexiones de 55 Mbps. Lo más sobresaliente que se puede observar en las gráficas de la Figura 6.74 correspondientes a estas conexiones es que, para todos los niveles de carga considerados, el algoritmo WFA se encuentra completamente saturado, y hay flits incapaces de atravesar el Encaminador. En cambio, los algoritmos COA y CCA son capaces de mantener buenas prestaciones a estas conexiones hasta cargas del 83 % y 87 %, respectivamente.

Efectivamente, la diferencia entre los algoritmos CCA y COA se acentúa a partir del punto de carga C (87 %). En dicho punto, se aprecia como el Encaminador ya se ha saturado cuando se emplea el algoritmo COA, mientras que con el algoritmo CCA la saturación está cercana, pero todavía no ha ocurrido del todo. Concretamente, para ese punto de carga, con el algoritmo COA la mayoría de los flits sólo puede cumplir las cotas más relajadas (mayores de 4000 microsegundos), mientras que con el algoritmo CCA, prácticamente todos los flits obtienen retardos inferiores a $IAT \times 4$, (75.7 microsegundos). Más aún, cuando la carga crece hasta el 89 %, si se emplea el algoritmo COA incluso hay flits que no llegan a cruzar el Encaminador. Por el contrario, con el algoritmo CCA, todavía hay más del 90 % de flits que sufren retardos inferiores a $IAT \times 4$.

Por último, en la Figura 6.75 se muestran los valores de *jitter* medio experimentados por los flits de las distintas conexiones. Se observa como el valor del *jitter* es muy similar para los tres algoritmos, y en ningún caso llega a crecer de forma desmesurada, incluso cuando el Encaminador ya se ha saturado.

A la vista de los resultados presentados, se puede concluir que tanto con el algoritmo de planificación COA, como con el algoritmo CCA, el Encaminador Multimedia es capaz de garantizar QoS al tráfico CBR en un amplio rango de cargas. Sin embargo, el uso del algoritmo CCA permite ampliar este rango de carga útil en un 5 % aproximadamente, gracias a su mejor aprovechamiento del ancho de banda del *crossbar*.

También se ha demostrado como el uso de un algoritmo de emparejamiento general, que no tiene en cuenta los requerimientos de las aplicaciones, como es WFA, impide el cumplimiento



(a) Modelo de inyección SR

(b) Modelo de inyección BB

Figura 6.76: Planificación del conmutador: Tráfico VBR. Utilización media del *crossbar*.

de las garantías de QoS a las aplicaciones multimedia más exigentes, cuando la carga generada es superior al 65 % de utilización de los enlaces. Cuando se emplean los algoritmos especialmente desarrollados para el MMR, el rango de carga útil llega hasta el 83 % o el 87 %, según se emplee el algoritmo COA o el algoritmo CCA.

Seguidamente, se realizará el mismo análisis sobre tráfico VBR, para ver si estas conclusiones se mantienen para ese tipo de tráfico.

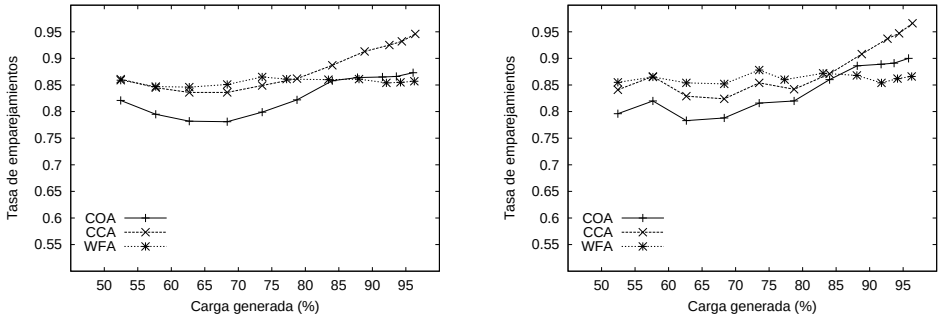
6.6.2. Evaluación con tráfico VBR

Al igual que en casos anteriores, las pruebas llevadas a cabo con tráfico VBR se han ejecutado hasta que han atravesado el Encaminador 6 GOPs de cada conexión. Se ha dejado el tiempo equivalente a dos tiempos de trama como periodo transitorio al inicio de la simulación. Los resultados temporales (retardos y *jitter*) se miden sobre tramas de vídeo en lugar de sobre flits individuales. Se presentan los resultados obtenidos considerando los dos modelos de inyección empleados para el tráfico VBR (SR y BB).

La Figura 6.76 muestra la utilización media del *crossbar* obtenida para los dos algoritmos de planificación considerados, y para ambos modelos de inyección. También se incluyen las curvas correspondientes al uso de WFA como algoritmo de planificación del conmutador en el MMR.

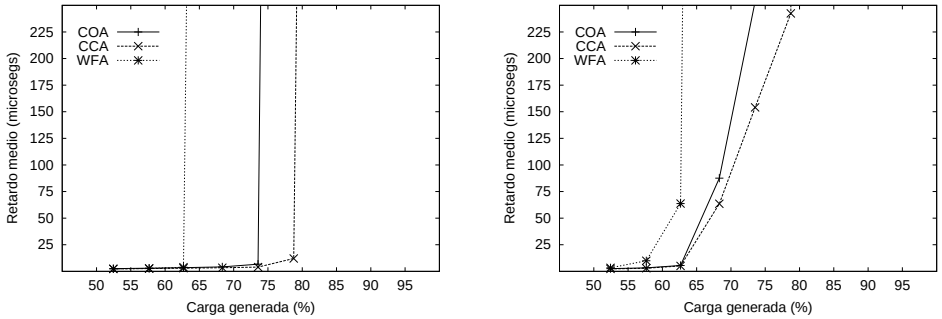
Al igual que sucedía con tráfico CBR, para cargas altas existe una mejora a favor del algoritmo CCA en términos de utilización. Esta mejora es más acusada que con tráfico CBR, y se hace notable a partir de un 84 % de carga generada, aproximadamente (punto marcado como B en la Figura). Esta mejora es ligeramente inferior en el modelo BB que en el modelo SR, especialmente en el punto de carga marcado como C en la Figura. También se aprecia como la utilización del *crossbar* empieza a decaer a cargas cercanas al 75 %, cuando se emplea el algoritmo WFA.

La razón para la mejora en utilización observada entre los algoritmos CCA y COA se encuentra de nuevo en la mejor calidad de los emparejamientos conseguidos por CCA. En la Figura 6.77 se representa la tasa de emparejamientos obtenida al emplear cada algoritmo, para ambos modelos de inyección. El algoritmo CCA obtiene emparejamientos más provechosos que el algoritmo COA, en término medio. La mejora es mayor con el modelo de inyección SR, lo que concuerda con la mayor ganancia en utilización observada previamente para este modelo.



(a) Modelo de inyección SR (b) Modelo de inyección BB

Figura 6.77: Planificación del conmutador: Tráfico VBR. Calidad del emparejamiento.



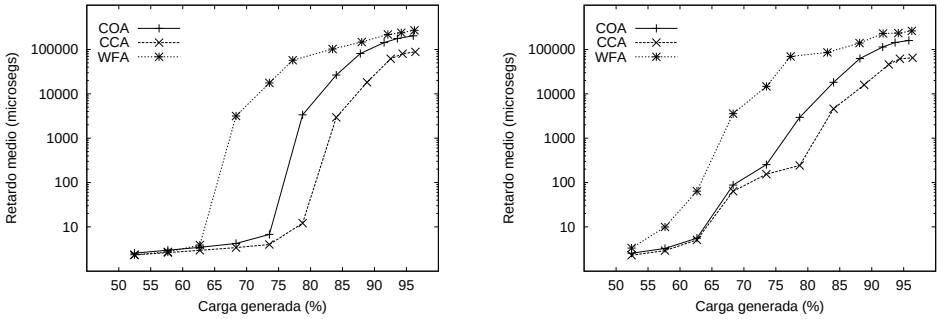
(a) Modelo de inyección SR (b) Modelo de inyección BB

Figura 6.78: Planificación del conmutador: Tráfico VBR. Retardo medio de las imágenes desde su generación.

Respecto a la tasa de emparejamientos conseguida por el algoritmo WFA, de nuevo se mantiene aproximadamente constante en todo el rango de carga. A niveles de carga inferiores al 80 %, la tasa de emparejamientos obtenida por WFA es comparable a la obtenida por CCA para ambos modelos de inyección. Cuando la carga generada sobrepasa ese nivel, es el algoritmo CCA quien obtiene claramente un mejor aprovechamiento del *crossbar*. Sin embargo, esta mejora no se traduce en una mayor productividad debido de nuevo a que los emparejamientos no se escogen teniendo en cuenta los requerimientos de las distintas conexiones.

El retardo medio de las tramas desde su generación se muestra en las gráficas de la Figura 6.78 para los dos modelos de inyección considerados. La mejor calidad de los emparejamientos obtenidos con el algoritmo CCA es la causa de la mayor productividad máxima alcanzada por éste. Concretamente, se observa que, de forma similar a como sucedía con tráfico CBR, la productividad máxima se incrementa en torno a un 5 % en ambos modelos de inyección si se emplea el algoritmo CCA. Por otro lado, los resultados obtenidos con WFA muestran una productividad máxima inferior al 65 %, nivel similar al alcanzado cuando el tráfico está compuesto exclusivamente por conexiones CBR.

El rango completo de valores para este retardo medio se muestra en la Figura 6.79, con el eje vertical en escala logarítmica. En estas gráficas se puede apreciar como para cargas medias a moderadas, las tramas experimentan en general retardos ligeramente inferiores si se usa



(a) Modelo de inyección SR (b) Modelo de inyección BB

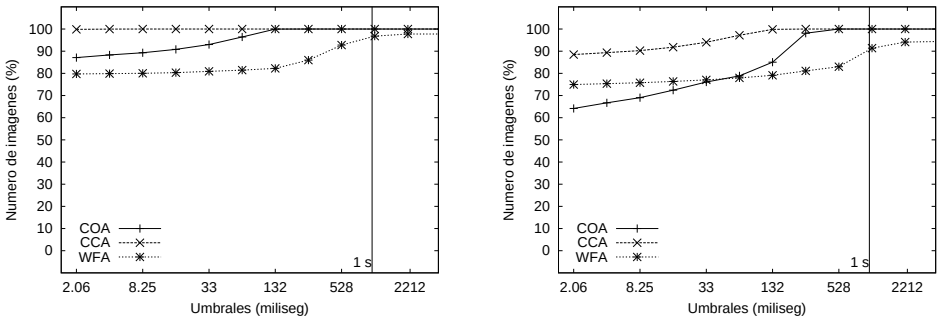
Figura 6.79: Planificación del conmutador: Tráfico VBR. Retardo medio de las imágenes desde su generación (escala semilogarítmica).

del algoritmo CCA. Cuando la carga crece hasta valores cercanos al 80 %, la diferencia entre los resultados obtenidos para ambos algoritmos se acentúa. Es más, a esos niveles de carga es cuando el Encaminador con el algoritmo COA está saturado, mientras que cuando se emplea el algoritmo CCA la saturación todavía no se ha producido. En cuanto al algoritmo WFA, se aprecia de nuevo claramente como para cargas cercanas al 70 %, el retardo medio experimentado por las tramas de vídeo es muy superior al proporcionado por los algoritmos COA y CCA.

Todo lo anterior se ve con más detalle analizando la distribución de los retardos experimentados por las tramas de vídeo desde su generación. Esta distribución se obtiene calculando el porcentaje de las tramas generadas que obtienen retardos inferiores a una serie de cotas. Dichas cotas son múltiplos y submúltiplos del tiempo de trama (TT = 33 milisegundos). En las Figuras 6.80 y 6.81 se muestran los resultados obtenidos para los puntos de carga marcados como A, B, C y D en la Figura 6.76, situados en torno a la saturación, para cada modelo de inyección, respectivamente.

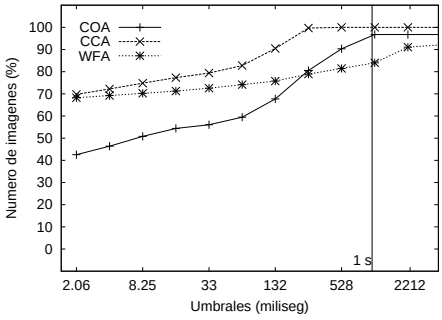
Observando primero los resultados obtenidos para el modelo de inyección SR, se observa como en todos los casos, el empleo del algoritmo CCA permite a las tramas cumplir con cotas más estrictas. Por ejemplo, para el punto de carga A (79 %), mientras que con el algorimo COA la cota más estricta que cumplen todas las tramas es de 132 milisegundos, en el caso del algoritmo CCA la totalidad de las tramas obtiene retardos inferiores al umbral más estricto considerado. Cuando la carga se incrementa hasta alcanzar el 89 % (punto C), se aprecia como cuando se emplea el algoritmo COA, el Encaminador ya se ha saturado e incluso hay tramas que no llegan a atravesar el Encaminador. Por el contrario, en ese mismo nivel de carga, el empleo del algoritmo CCA permite que todas las tramas experimenten retardos inferiores a 264 milisegundos ($TT \times 8$), un valor apurado, pero todavía aceptable. Téngase en cuenta que el ATM Fórum recomienda una cota para el retardo de un segundo entre extremos, para servicios de transmisión de vídeo MPEG-2 [143]. En cambio, cuando se emplea el algoritmo WFA, e incluso para el menor nivel de carga considerado (79 %), se observa como hay tramas que ni siquiera llegan a cruzar el Encaminador, indicando la saturación del mismo.

En cuanto a la distribución de los retardos de las tramas cuando se emplea el modelo de inyección BB, se puede decir prácticamente lo mismo que para el modelo SR. Hay un mayor porcentaje de tramas que experimentan retardos inferiores a cotas más estrictas cuando se emplea el algoritmo CCA que con el algoritmo COA. Y la saturación cuando se emplea el

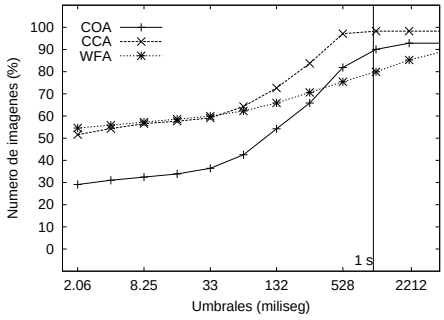


Punto de carga A (carga = 79 %)

Punto de carga B (carga = 84 %)

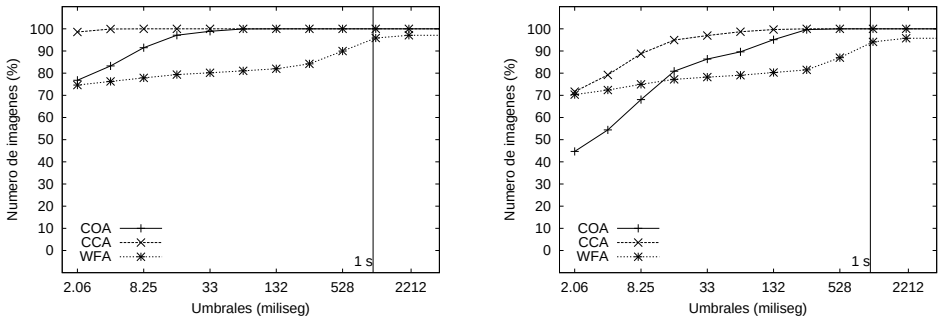


Punto de carga C (carga = 89 %)

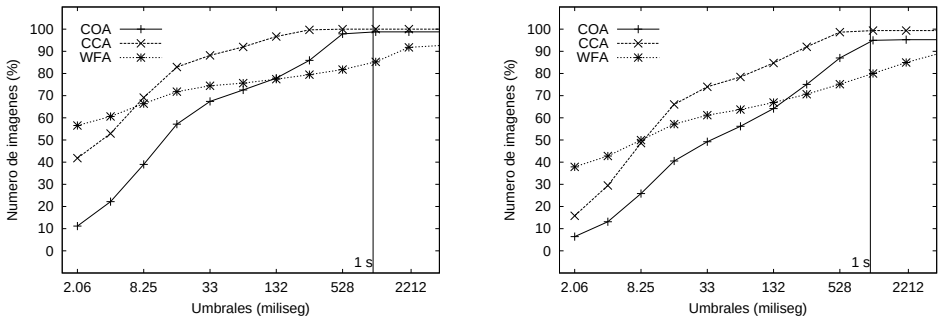


Punto de carga D (carga = 92 %)

Figura 6.80: Planificación del conmutador: Tráfico VBR - modelo SR. Distribución del retardo medio de las imágenes desde su generación.

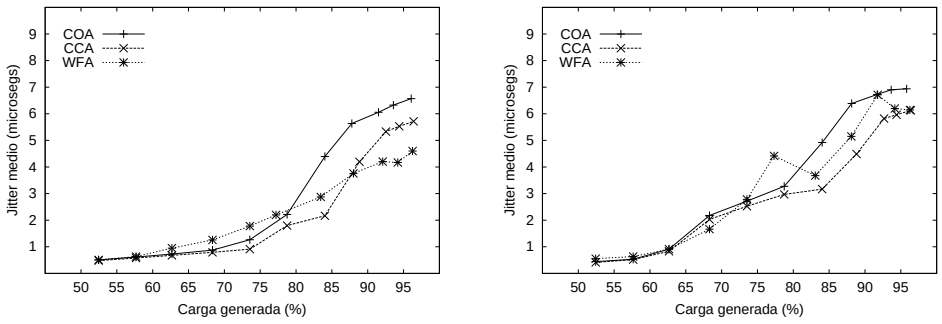


Punto de carga A (carga = 79 %) Punto de carga B (carga = 84 %)



Punto de carga C (carga = 89 %) Punto de carga D (carga = 92 %)

Figura 6.81: Planificación del conmutador: Tráfico VBR - modelo BB. Distribución del retardo medio de las imágenes desde su generación.



(a) Modelo de inyección SR (b) Modelo de inyección BB

Figura 6.82: Planificación del conmutador: Tráfico VBR. *Jitter* medio de las imágenes.

algoritmo CCA se produce a cargas mayores del 89 %, en torno a un 5 % más que con el algoritmo COA. Cuando se emplea el algoritmo WFA, el Encaminador está ya saturado a los niveles de cargo mostrados en la Figura 6.81, tal y como lo demuestra el hecho de que hay tramas que ni siquiera llegan a alcanzar los puertos de salida. Antes de entrar en saturación, todas las tramas experimentan retardos inferiores a 268 milisegundos ($TT \times 8$), cuando se emplean los algoritmos de planificación COA y CCA.

Por último, en la Figura 6.82 se muestran los valores de *jitter* medio experimentados por las tramas de vídeo, para ambos modelos de inyección. Cuando la carga está por debajo del 70 %, apenas hay diferencias entre los resultados ofrecidos para los dos algoritmos. Incluso al aumentar la carga por encima de ese valor, las diferencias, aunque son mayores, siguen siendo poco significativas. Sin embargo, en todos casos se observa como el valor de esta variación en el retardo permanece siempre por debajo de los 7 microsegundos. Hay que recordar que mediante el empleo de técnicas de absorción del *jitter* en el destino, se pueden tolerar valores de *jitter* de hasta varios milisegundos [133].

De todo este análisis se desprende que el uso de un algoritmo de planificación que no contemple las garantías de QoS de las aplicaciones, como WFA, limita la carga máxima que puede soportar el Encaminador Multimedia sin saturarse y respetando la QoS de las conexiones, a valores menores del 70 % del ancho de banda de los enlaces. En cambio, cuando los algoritmos de planificación tienen en cuenta dichas garantías de QoS, como es el caso de COA y de CCA, el Encaminador Multimedia es capaz de ofrecer las garantías de QoS que necesitan los flujos multimedia, tanto los que generan tráfico CBR, como los que generan tráfico VBR de tipo MPEG-2, en un rango de cargas significativamente mayor. Sin embargo, el rango de carga útil alcanzado con el algoritmo COA, esto es, los niveles de carga a los cuales las garantías de QoS se mantienen, es aproximadamente un 5 % inferior al obtenido cuando se emplea el algoritmo CCA.

Ya se demostró en la Sección 5.2.4 que, para la configuración y el tamaño de flit empleados, ambos algoritmos pueden ser implementados en hardware de forma que calculan una planificación válida dentro del tiempo designado para ello (un ciclo de flit). Por tanto, puesto que ambos algoritmos cumplen con los requisitos temporales para su implementación en hardware dentro del MMR, y dado que la diferencia en las productividades máximas alcanzadas no es demasiado acusada, el análisis que se llevará a cabo en el siguiente apartado se realizará teniendo en cuenta ambos algoritmos de planificación.

6.7. Evaluación final

En las Secciones previas se han analizado en detalle las prestaciones ofrecidas por los distintos algoritmos de planificación tanto de enlaces como del conmutador, cuando el tráfico está compuesto por tráfico multimedia exclusivamente. Sin embargo, el Encaminador Multimedia también debe de soportar otros tipos de tráfico, en concreto mensajes cortos de control, y tráfico tradicional de datos tipo *best-effort*.

Los objetivos en cuanto a las prestaciones que debe alcanzar cada uno de estos tipos de tráfico no multimedia son bien distintas. Por un lado, los mensajes de control ocurren con una frecuencia muy baja, y deben ser retransmitidos con un retardo mínimo, pues pueden afectar al comportamiento de las conexiones multimedia. Por otro lado, el tráfico *best-effort* sólo debe de ocupar el ancho de banda no adjudicado a las conexiones multimedia, sin interferir en las prestaciones de éstas. El objetivo es que este tráfico obtenga una elevada productividad y una latencia media mínima, pero sin comprometer las garantías de QoS ofrecidas a los flujos multimedia.

Número de puertos	4
Número de canales virtuales	128
Valor de K	16
Tamaño de phit	16 bits
Tamaño de flit	1024 bits
Ancho de banda de los enlaces	1.24 Gbps
Tamaño de los *buffers* del MMR	1 flit
Tipo de planificador	SIABP + {COA, CCA}
Número de candidatos	4
Tipo de CAC	Sección 4.3.3
Factor de concurrencia	16
Semilla para números aleatorios	12345

Tabla 6.7: Evaluación final: Parámetros comunes a todas las pruebas.

Para evaluar las prestaciones obtenidas por estos tipos de tráfico, se procederá de la forma siguiente:

Tráfico de control Se va a analizar el *retardo máximo* experimentado por los flits de control. En primer lugar, el análisis se realizará con carga compuesta sólo de tráfico multimedia, y posteriormente, con cargas compuestas de tráfico multimedia más *best-effort*.

Tráfico *best-effort* Se va a analizar la productividad máxima que se puede alcanzar con este tipo de tráfico. La carga se compondrá de tráfico multimedia más tráfico de control. Por tanto, otro objetivo del análisis será comprobar que la presencia del tráfico *best-effort* no compromete las prestaciones de las conexiones multimedia ni las de los mensajes de control.

En todos los casos, existe una componente básica de la carga, que serán las conexiones multimedia. Esta fracción de carga estará compuesta por una mezcla de conexiones tanto CBR como VBR, al 50 %. Por tanto, en primer lugar se analizará el comportamiento de esta componente base, para tenerlo como referencia al ir añadiendo los otros tipos de tráfico.

Los parámetros de las simulaciones se han configurado en todos los casos con los valores de la Tabla 6.7. Para cada punto de carga, la mitad de la misma está compuesta de conexiones CBR, y la otra mitad de conexiones VBR, aproximadamente. Hay que recordar que las conexiones generadas se escogen aleatoriamente de entre un conjunto de conexiones con consumos de ancho de banda prefijados. Por tanto, ajustar la carga generada a un porcentaje exacto manteniendo la aleatoriedad resulta altamente complicado. Lo que se hace es generar conjuntos de conexiones cuya carga global esté dentro de un estrecho margen alrededor de la carga solicitada.

El tráfico VBR se ha generado según el modelo de inyección SR, y las distintas fuentes están alineadas de forma aleatoria. El tiempo de simulación ha sido el suficiente como para que cada conexión VBR haya transmitido completamente 6 GOPs, más un periodo transitorio inicial equivalente a dos tiempos de trama.

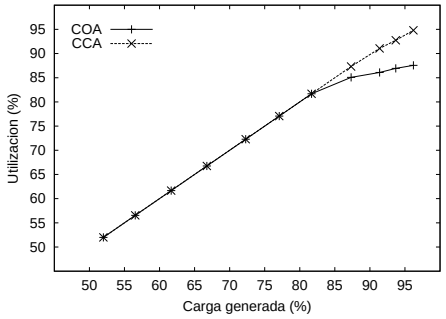
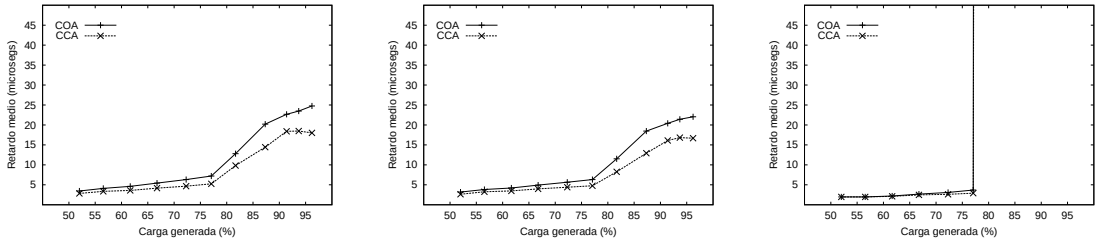


Figura 6.83: Evaluación final: Tráfico multimedia mixto. Utilización media del *crossbar*.



(a) 64 Kbps (b) 1.54 Mbps (c) 55 Mbps

Figura 6.84: Evaluación final: Tráfico multimedia mixto. Retardo medio de los flits CBR desde su generación.

6.7.1. Evaluación con tráfico multimedia mixto

Con el doble objetivo de evaluar la interacción entre los distintos tipos de flujos multimedia, y de tener una referencia para las pruebas posteriores, en esta Sección se analizan las prestaciones que ofrece el MMR a las distintas conexiones cuando la carga generada se compone de conexiones CBR y VBR mezcladas.

La Figura 6.83 muestra la utilización media del *crossbar* obtenida para los dos algoritmos de planificación del conmutador propuestos, COA y CCA. Los resultados son similares a los obtenidos cuando la carga se componía únicamente de tráfico VBR, esto es, a partir del 82 % de carga existe una mayor pérdida de utilización cuando se emplea el algoritmo COA.

A continuación se evalúan las prestaciones temporales obtenidas por ambos tipos de tráfico. Las gráficas de la Figura 6.84 muestran el retardo medio desde la generación experimentado por los flits de las conexiones CBR. En general, los valores obtenidos son ligeramente inferiores para el algoritmo CCA. Pero lo más destacable es la saturación que experimentan las conexiones de 55 Mbps cuando la carga media generada supera el 77 % de carga. Esto supone un decremento en la productividad máxima de en torno al 10 % con respecto a los resultados obtenidos con tráfico exclusivamente de tipo CBR (ver la gráfica correspondiente de la Figura 6.72 en la página 202). Los flits pertenecientes al resto de conexiones también ven incrementado significativamente su retardo a partir de ese nivel de carga, pero los valores se mantienen acotados.

En cuanto a las prestaciones obtenidas por las conexiones VBR, en la Figura 6.85 se muestra el retardo medio experimentado por las imágenes de vídeo desde su generación. De nuevo,

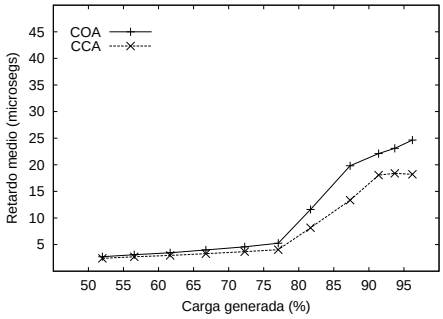
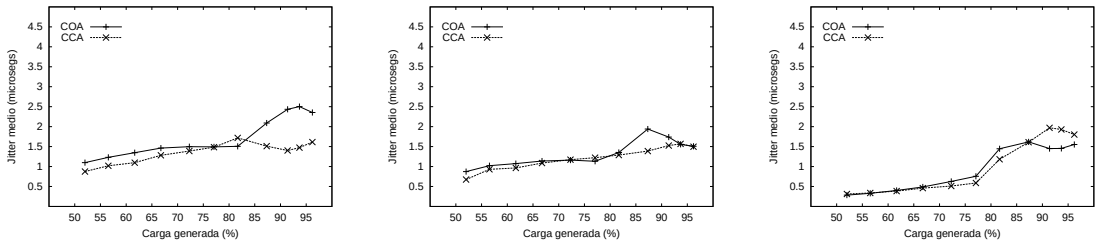


Figura 6.85: Evaluación final: Tráfico multimedia mixto. Retardo medio de las imágenes VBR desde la generación.



(a) 64 Kbps (b) 1.54 Mbps (c) 55 Mbps

Figura 6.86: Evaluación final: Tráfico multimedia mixto. *Jitter* medio de los flits CBR.

el algoritmo CCA es capaz de ofrecer mejores resultados. Y también se aprecia un sensible incremento de los valores de retardo cuando la carga media generada sobrepasa el 77 % de utilización de los enlaces. Sin embargo, los valores de retardo permanecen acotados, al igual que sucedía con las conexiones CBR de requerimientos bajos y medios.

La explicación a este comportamiento es que en presencia de tráfico VBR, el nivel de carga ya no permanece constante durante todo el tiempo simulado, sino que fluctúa según van evolucionando las conexiones de vídeo. Cuando la carga es moderada a alta, esta fluctuación provoca que en ciertos periodos de tiempo, el Encaminador se sature, resultando perjudicados los flits pertenecientes a las conexiones más exigentes, que son precisamente las conexiones CBR de 55 Mbps. Esto concuerda con lo observado en las pruebas efectuadas sólo con tráfico CBR, donde las conexiones perjudicadas por la saturación eran precisamente las más exigentes.

En lo que respecta al *jitter* experimentado por los flits CBR (Figura 6.86), y por las tramas de vídeo VBR (Figura 6.87), simplemente señalar que se mantiene acotado en todo el rango de carga simulado. Los valores medios están en todos los casos en torno a unos pocos microsegundos.

La distribución de los retardos experimentados por los flits CBR y por las imágenes VBR desde la generación ofrece una información más detallada del comportamiento del Encaminador. Para las conexiones CBR de requerimientos medios y bajos (Figura 6.88), y para las conexiones VBR (Figura 6.89), se muestran los resultados obtenidos en el punto de mayor carga considerado, en torno al 96 %. En todos los casos se observa como todos los flits y todas las tramas son capaces de cumplir con las cotas más estrictas consideradas, con ambos

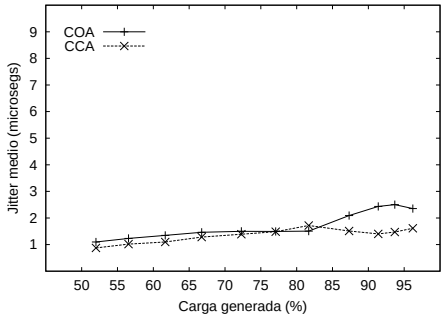
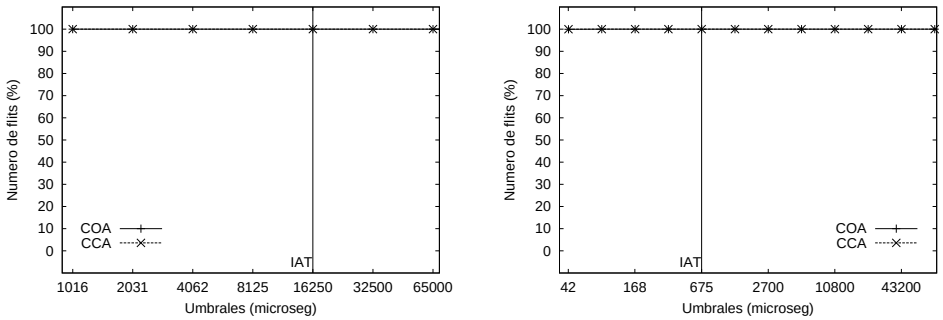


Figura 6.87: Evaluación final: Tráfico multimedia mixto. *Jitter* medio de las imágenes VBR.



(a) 64 Kbps (b) 1.54 Mbps

Figura 6.88: Evaluación final: Tráfico multimedia mixto. Distribución del retardo medio de los flits CBR a la carga máxima generada (96 % aprox.).

algoritmos de planificación.

En el caso de las conexiones CBR de mayores requerimientos de ancho de banda (Figura 6.90), se muestran los resultados obtenidos para los puntos de carga justo anterior y posterior a la saturación, correspondientes al 77.5 % y al 82 % de carga, respectivamente. Antes de entrar en saturación, todos los flits sufren retardos inferiores al IAT de la conexión (19 microsegundos), mientras que al aumentar la carga media hasta el siguiente punto simulado, existe una importante pérdida de prestaciones. También es interesante comprobar como cerca del 90 % de los flits en el caso de CCA obtienen retardos inferiores al IAT, y casi el 10 % restante obtiene retardos muy elevados, de más de 19 milisegundos. Esto se explica porque en los periodos donde la carga es más baja (debidos a que coinciden más cantidad de tramas de tipo B de las conexiones de vídeo), los flits sufren retardos bajos. Sin embargo, cuando coinciden un mayor número de tramas de tipo I, la carga instantánea se eleva por encima de la saturación, elevando drásticamente el retardo sufrido por los flits transmitidos durante esos instantes.

De los resultados presentados se deduce que, en presencia de tráfico mixto, el rango de carga útil ofrecido por el Encaminador Multimedia es algo inferior al obtenido cuando el tráfico se compone de sólo un tipo de tráfico multimedia, sea éste CBR o VBR. La disminución es algo más acusada frente a una carga compuesta únicamente por tráfico CBR. Esto es lógico, pues en ese caso el nivel de carga generada no fluctúa con el tiempo, y por tanto no existen picos de carga temporales, que provoquen periodos de saturación transitoria en el Encaminador.

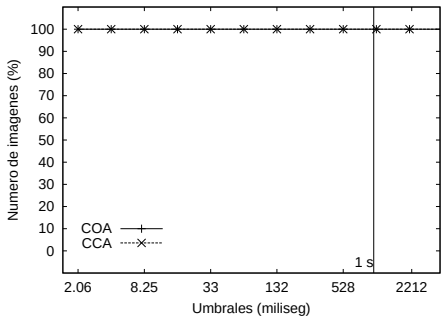
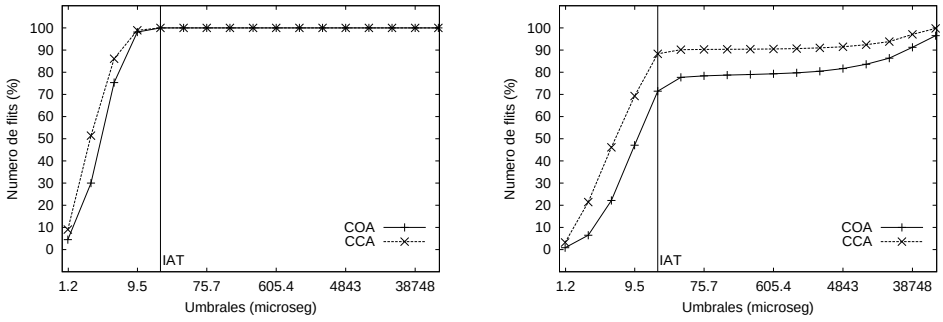


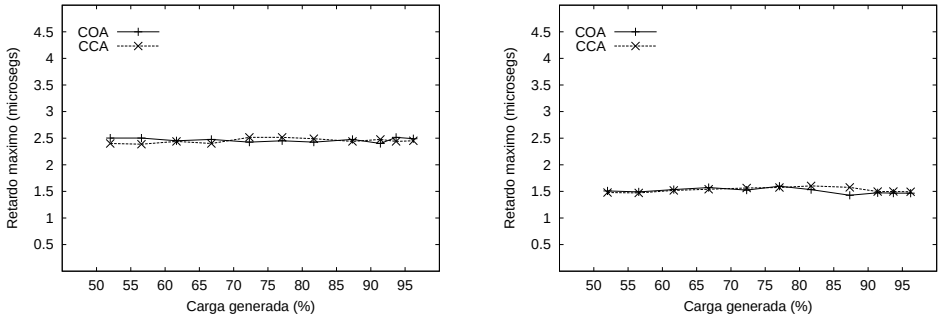
Figura 6.89: Evaluación final: Tráfico multimedia mixto. Distribución del retardo medio de las imágenes VBR a la carga máxima generada (96 % aprox.).



(a) Carga = 77.5 %

(b) Carga = 82 %

Figura 6.90: Evaluación final: Tráfico multimedia mixto. Distribución del retardo medio de los flits de las conexiones CBR más exigentes (55 Mbps).



(a) Retardo máximo (b) Retardo medio

Figura 6.91: Evaluación final: Tráfico multimedia mixto + control. Retardo experimentado por los flits de control desde su generación.

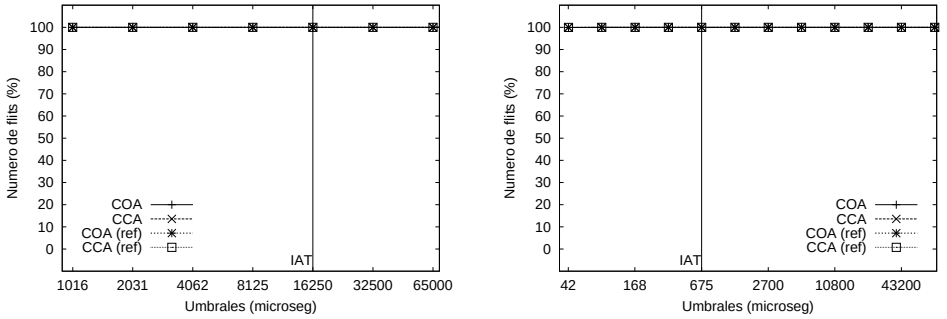
6.7.2. Evaluación con tráfico de control

Una vez analizada la interacción entre los flujos multimedia de distintos tipos, es el turno de evaluar el comportamiento del Encaminador cuando a la carga generada contribuyen fuentes de tráfico no multimedia. En primer lugar, se analizarán las prestaciones obtenidas cuando se incluye tráfico de control.

Los resultados presentados a continuación se han obtenido considerando la misma carga multimedia empleada en el apartado anterior, a la cuál se le han añadido fuentes de tráfico de control. En concreto, se ha establecido una fuente de control (ver Sección 6.2.1) por cada enlace físico, que siempre genera sus flits a través de los canales virtuales 0 de cada enlace.

El principal objetivo del Encaminador con respecto a los flits de control es ofrecerles una latencia mínima, con el objeto de que alcancen su destino cuanto antes. Para conseguir este objetivo, los algoritmos de planificación de enlaces asignan a estos flits el valor máximo de prioridad posible, para que siempre resulten elegidos como candidatos. Además, los algoritmos de planificación del *crossbar*, emparejan siempre en primer lugar a los canales virtuales 0 (dedicados al tráfico de control), en caso de que tengan flits listos para transmitir, y créditos disponibles. Por ello, se ha analizado el *retardo máximo* experimentado por estos flits en presencia de tráfico multimedia mixto. El valor para este parámetro se muestra en la Figura 6.91-(a). En esta gráfica se puede observar como, para ambos algoritmos de planificación del conmutador considerados, el retardo máximo se mantiene prácticamente constante y en torno a los 2'5 microsegundos independientemente de la carga multimedia generada. El retardo medio también se muestra en la parte (b) de la misma Figura, observándose como se mantiene igualmente constante y en torno a 1,5 microsegundos. En términos del Encaminador el valor máximo equivale a aproximadamente 3 ciclos de flit, mientras que el valor medio es algo menor de 2 ciclos de flit. Puesto que atravesar el *crossbar* consume un ciclo de flit, el tiempo adicional se debe a la posible contención por algún puerto de salida con otros flits de control procedentes de otros enlaces, y a que los flits no tienen porqué llegar al Encaminador justo al tiempo de iniciar el proceso de la planificación, lo que introduce un tiempo de espera inicial que siempre será menor que un ciclo de flit.

En cuanto al impacto causado por la presencia del tráfico de control sobre las conexiones multimedia, en las gráficas de las Figuras 6.92, 6.93 y 6.94 se muestra la distribución de retardos para los puntos de máxima carga en el caso de las conexiones CBR menos exigentes, y de las VBR, y para los puntos antes y después de la saturación en el caso de las conexiones



(a) 64 Kbps (b) 1.54 Mbps

Figura 6.92: Evaluación final: Tráfico multimedia mixto + control. Distribución del retardo medio de los flits CBR a la carga máxima generada (96 % aprox.).

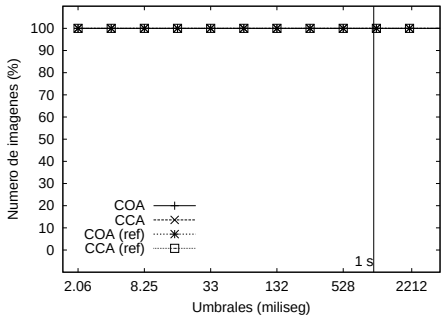


Figura 6.93: Evaluación final: Tráfico multimedia mixto + control. Distribución del retardo medio de las imágenes VBR a la carga máxima generada (96 % aprox.).

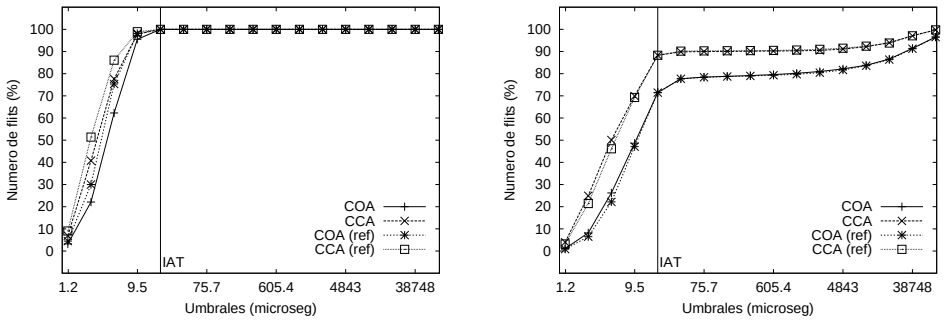
CBR más exigentes. Las curvas denotadas con "(ref)" se refieren a la carga de referencia, compuesta únicamente por conexiones multimedia. En todos los casos se puede observar como el impacto de este tráfico sobre las prestaciones de las conexiones multimedia es nulo. Únicamente los flits de las conexiones CBR más exigentes se ven mínimamente afectados, aunque la cota más estricta que cumplen todos los flits no se ve modificada.

A la vista de estos resultados, se puede concluir que el Encaminador Multimedia es capaz de garantizar el retardo máximo sufrido por los flits de control, a cualquier nivel de carga. Además, la presencia de este tipo de tráfico no compromete las prestaciones obtenidas por las conexiones multimedia.

6.7.3. Evaluación con tráfico *best-effort*

En las Secciones previas se ha analizado el efecto de la interacción entre los distintos flujos multimedia sobre las garantías de QoS que recibe cada clase de tráfico. También se han evaluado las prestaciones ofrecidas por el Encaminador Multimedia cuando sobre la mezcla de tráfico anterior se añade tráfico de control. Por tanto, queda por comprobar qué sucede cuando a los dos tipos de tráfico anteriores, se añaden fuentes de tráfico convencional de tipo *best-effort*.

En este punto no hay que olvidar que el Encaminador Multimedia, como su propio nombre



(a) Carga = 77.5 %

(b) Carga = 82 %

Figura 6.94: Evaluación final: Tráfico multimedia mixto + control. Distribución del retardo medio de los flits de las conexiones CBR más exigentes (55 Mbps).

indica, está diseñado para soportar de manera adecuada el tráfico generado por aplicaciones multimedia. Puesto que el tráfico tradicional de tipo *best-effort* coexistirá normalmente con los flujos multimedia, es necesario que el Encaminador los considere de alguna forma a la hora de retransmitir sus flits.

Tal y como se indicó en la Sección 5.1, la política que aplica el Encaminador Multimedia a los flits *best-effort* es que deben ocupar el ancho de banda de los enlaces que no haya sido adjudicado a conexiones multimedia. La idea es que la presencia de este tráfico no afecte las garantías de QoS que reciben los flujos multimedia. Este es el principal objetivo a cumplir. Adicionalmente, se debe procurar que el tráfico *best-effort* alcance buena productividad y latencia media.

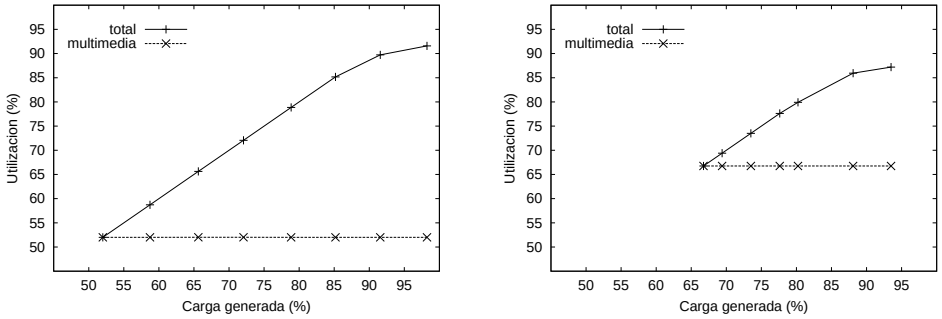
Este apartado está dedicado a comprobar si efectivamente se cumplen las dos premisas señaladas en el párrafo anterior. Para ello, se han llevado a cabo una serie de pruebas, donde el Encaminador se ha configurado de la forma habitual (ver la Tabla 6.7), pero la carga generada varía con respecto a la manera empleada hasta ahora.

En todas las pruebas realizadas hasta ahora, se han generado una serie de puntos de carga en el rango comprendido entre el 50 % y el 100 % de la utilización del ancho de banda de los enlaces. Cada punto de carga consistía en un conjunto de conexiones multimedia[3]. Ahora bien, en este apartado parte de la carga será generada por fuentes de tráfico *best-effort*, y parte estará compuesta por conexiones multimedia.

El procedimiento que se ha seguido es realizar dos tandas de prueba. En cada una de ellas se ha fijado un cierto porcentaje de carga compuesta por flujos multimedia, y se ha variado la cantidad de carga generada por fuentes *best-effort*. Los porcentajes de carga multimedia "base" fijados son el 52 % y el 67 % del ancho de banda de los enlaces. La mezcla de conexiones es la misma empleada para los mismos puntos en la Sección 6.7.1. La carga *best-effort* va aumentando progresivamente hasta completar casi la totalidad del ancho de banda de los enlaces.

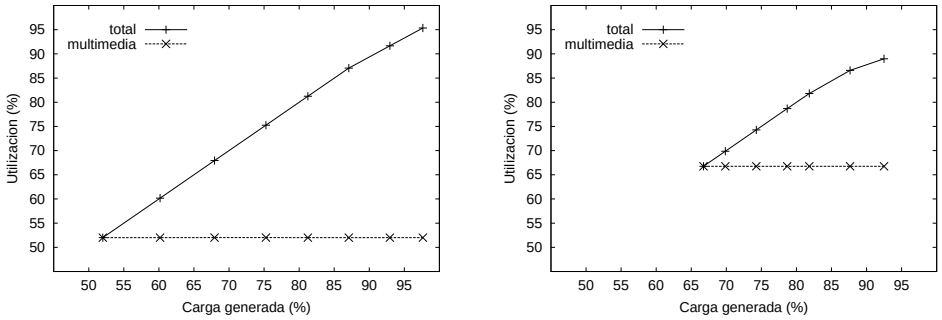
La idea se aprecia con mayor claridad en las gráficas de la Figura 6.95, que presentan la utilización media del *crossbar*. La línea horizontal representa el porcentaje de utilización debido a las conexiones multimedia, y la diferencia con el total supone la cantidad de carga generada por fuentes *best-effort*. Obsérvese que en todos los casos, el punto de menor carga

[3]En el caso del apartado 6.7.2, la carga adicional generada por las fuentes de mensajes de control es mínima.



(a) COA, carga multimedia = 52 %

(b) COA, carga multimedia = 67 %



(c) CCA, carga multimedia = 52 %

(d) CCA, carga multimedia = 67 %

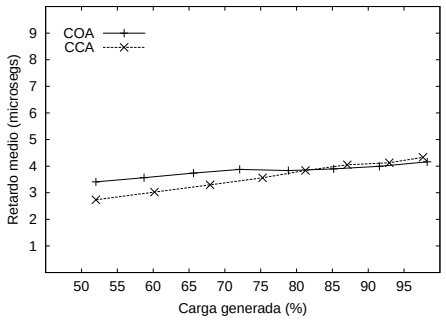
Figura 6.95: Evaluación final: Tráfico multimedia mixto + control + *best effort*. Utilización media del *crossbar*.

corresponde a una carga compuesta totalmente por conexiones multimedia. Este punto será la referencia a tener en cuenta a la hora de verificar el impacto de la presencia del tráfico *best-effort* sobre las prestaciones de las conexiones multimedia.

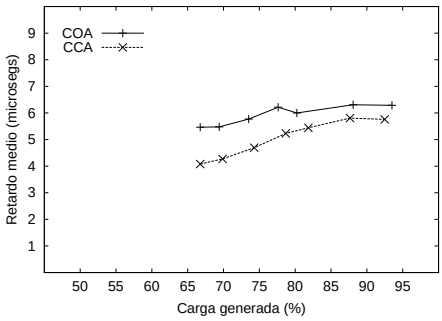
En dichas gráficas se observa como cuando se emplea el algoritmo COA de planificación del conmutador, el *crossbar* se empieza a saturar a partir de cargas de en torno al 85 %. La saturación se presenta a cargas algo superiores cuando el planificador es CCA. Hay que recordar que esto también sucedía cuando la carga estaba compuesta exclusivamente por conexiones multimedia.

Seguidamente, hay que evaluar las prestaciones temporales ofrecidas a cada tipo de tráfico. En la Figura 6.96 se muestra el retardo medio de los flits de las distintas conexiones CBR, medido desde su generación. Los puntos correspondientes a los niveles de carga más bajos corresponden a una carga exenta de tráfico *best-effort*. Para una carga multimedia base del 52 %,, y para ambos algoritmos de planificación, el incremento en el retardo experimentado por los flits es muy pequeño, de en torno a 1 microsegundo, como máximo (equivalentes a 1'2 ciclos de flit, aproximadamente). Cuando la carga multimedia base es del 67 %, el incremento llega en algunos casos a los 2 microsegundos, esto es, en torno a 2'4 ciclos de flit. En general, se puede apreciar como el algoritmo CCA se ve algo más afectado por la presencia del tráfico *best-effort* que el algoritmo COA.

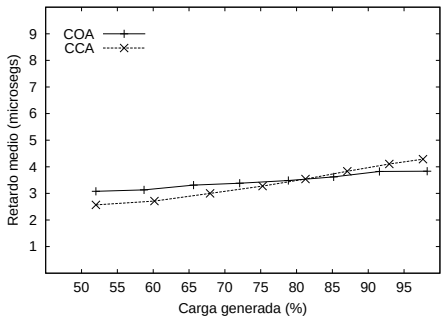
Respecto a las imágenes de vídeo VBR, cuyo retardo medio desde la generación se muestra en la Figura 6.97, la conclusión que se puede obtener es similar a lo anterior. De nuevo, existe un ligero incremento en el retardo de las tramas, que es algo superior en el caso del algoritmo CCA. Este incremento es inferior al microsegundo para el algoritmo COA, y menor que 2 microsegundos en el caso de CCA.



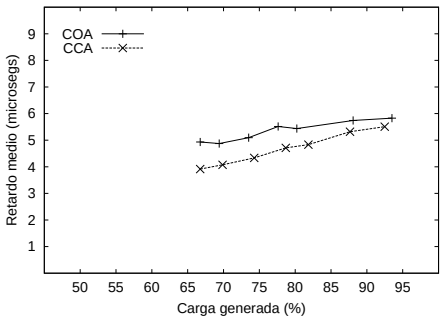
(a) 64 Kbps, carga multimedia = 52 %



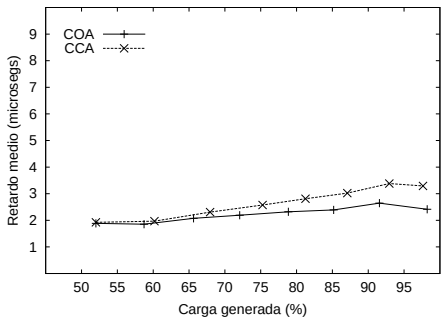
(b) 64 Kbps, carga multimedia = 67 %



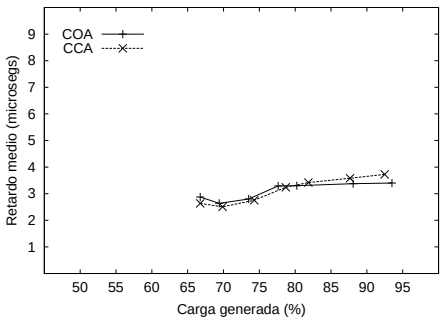
(c) 1.54 Mbps, carga multimedia = 52 %



(d) 1.54 Mbps, carga multimedia = 67 %

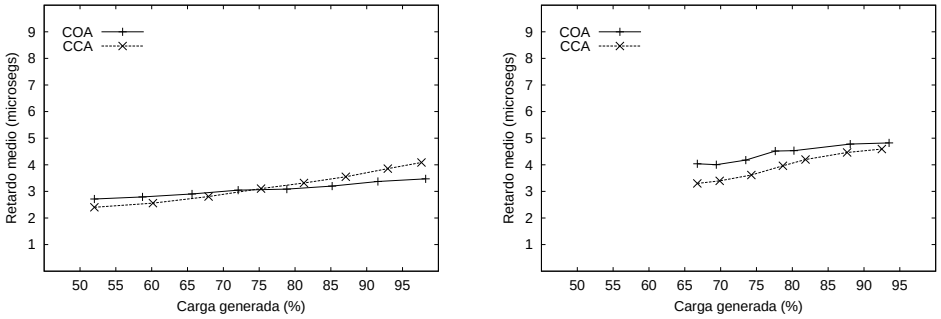


(e) 55 Mbps, carga multimedia = 52 %



(f) 55 Mbps, carga multimedia = 67 %

Figura 6.96: Evaluación final: Tráfico multimedia mixto + control + *best effort*. Retardo medio de los flits CBR desde su generación.



(a) Carga multimedia = 52 %

(b) Carga multimedia = 67 %

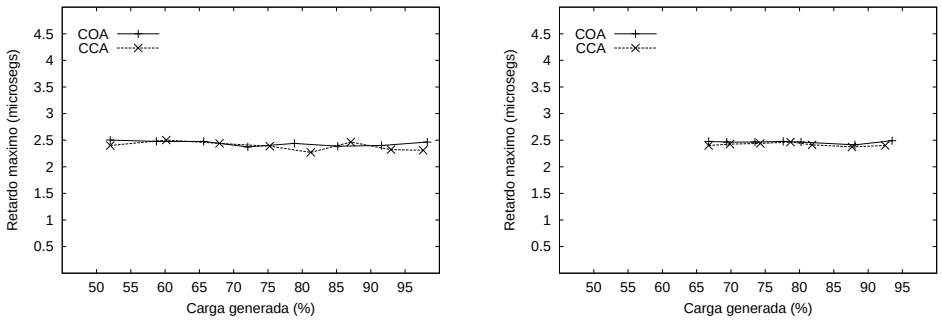
Figura 6.97: Evaluación final: Tráfico multimedia mixto + control + *best effort*. Retardo medio de las tramas de vídeo VBR desde su generación.

Estos incrementos están introducidos evidentemente por la presencia del tráfico *best-effort*. El hecho de que a los flits de este tipo de tráfico se les asigne la prioridad mínima, que además no se modifica, impide que sean seleccionados como candidatos por delante de flits pertenecientes a conexiones multimedia. Sin embargo, hay que recordar que, en ambos algoritmos de planificación, el criterio de la prioridad de los flits se emplea en la fase de *arbitraje* (ver la Sección 5.2). Esta fase se ejecuta tras la fase de *ordenación de puertos*, la cual se basa en el número de conflictos existentes por cada puerto de salida. La presencia de flits *best-effort* entre los candidatos seleccionados puede influir en el número de conflictos por puerto, modificando el orden de emparejamiento de los puertos, y afectando al retardo experimentado por los flits de las conexiones multimedia. El criterio del número de conflictos tiene más peso en el algoritmo CCA que en el COA (hay que recordar que en COA se va aplicando sucesivamente sobre las peticiones de los candidatos de cada nivel), lo que explica que el incremento en el retardo sea algo superior para dicho algoritmo.

En lo que respecta al tráfico de control, su retardo máximo no sufre apenas alguna variación. En la Figura 6.98 se puede apreciar como su valor está siempre en torno a los 2'5 microsegundos. Esto es debido a que a los flits de control se les asigna siempre la prioridad máxima, con lo que siempre resultan elegidos como candidatos, y además, a la hora de planificar el conmutador, tanto COA como CCA emparejan en primer lugar a los flits de control. Así, la presencia del tráfico *best-effort* no les afecta en lo más mínimo, pues el emparejamiento de estos flits de control no sigue el procedimiento habitual establecido para cada algoritmo.

Una información más detallada acerca de la QoS que reciben los flujos multimedia la ofrecen las distribuciones de retardos desde la generación. Las Figuras 6.99 y 6.100 muestran los valores obtenidos para esta magnitud, para los flits de las conexiones CBR de 64 Kbps y 1.54 Mbps y para las tramas de vídeo, respectivamente. Las curvas corresponden al nivel de carga generada más alto en cada caso. Como referencia, en cada gráfica se muestra la curva correspondiente a la misma carga multimedia indicada, pero sin tráfico *best-effort*. En todos estos casos, la presencia de tráfico *best-effort* no afecta el cumplimiento de las cotas establecidas, ni siquiera las más estrictas.

Sin embargo, para las conexiones CBR más exigentes (55 Mbps) se puede observar en la Figura 6.101 como el porcentaje de flits que no cumple las cotas más estrictas disminuye con respecto a la referencia. Aún así, las garantías de QoS no se ven comprometidas, pues en el peor de los casos (algoritmo CCA, carga base multimedia del 67 %), todos los flits son capaces de obtener retardos inferiores al IAT (tiempo entre llegadas) de la conexión (19



(a) Carga multimedia = 52 %

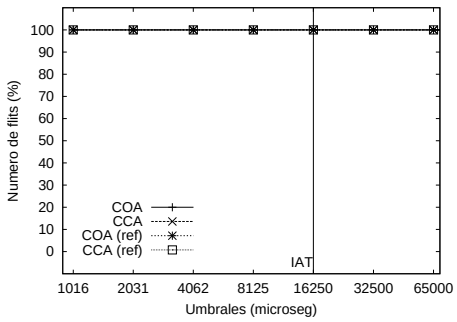
(b) Carga multimedia = 67 %

Figura 6.98: Evaluación final: Tráfico multimedia mixto + control + *best effort*. Retardo máximo de los flits de control desde su generación.

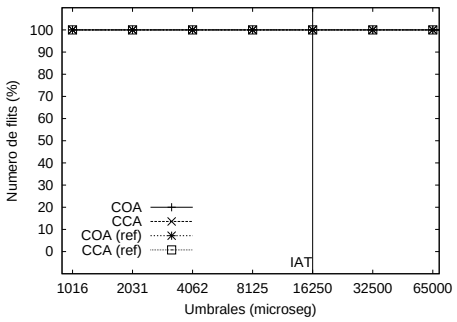
microsegundos).

Por último, queda por comprobar qué prestaciones recibe el tráfico *best-effort*. El retardo medio experimentado por sus flits se muestra en la Figura 6.102. En ella se puede apreciar como la saturación ocurre en torno al 85 % de carga *total* generada cuando la base multimedia es del 52 %, y a una carga ligeramente inferior, cuando la carga base multimedia aumenta hasta el 67 %. Cuando el nivel de carga es inferior a estos valores, el retardo medio de los flits *best-effort* se halla por debajo del milisegundo. Los valores son ligeramente superiores cuando la carga base multimedia es del 67 % . Esto es lógico, pues en ese caso, para el mismo punto de carga total, hay menos flits *best-effort* que han de competir con más flits multimedia. Y los flits *best-effort* llevan las de perder en la planificación, por lo que su retardo medio se incrementa.

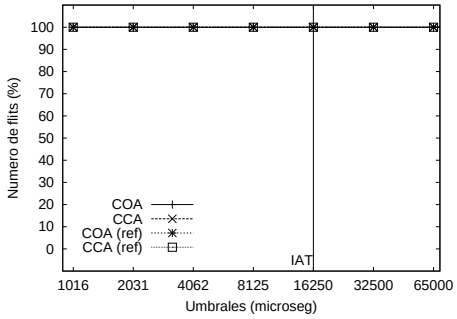
De este estudio se desprende que el Encaminador Multimedia efectivamente es capaz de trasegar con garantías una carga compuesta por diferentes clases de tráfico, tanto multimedia como convencional. Esta situación es la más aproximada a la que se presenta en entornos reales, donde las aplicaciones multimedia conviven con aplicaciones tradicionales. Se ha demostrado como la presencia de tráfico *best-effort* no afecta a las garantías de QoS que reciben las aplicaciones multimedia, ni a las prestaciones obtenidas por los mensajes de control. Además, el tráfico *best-effort* es capaz de aprovechar el ancho de banda restante hasta la saturación del *crossbar*.



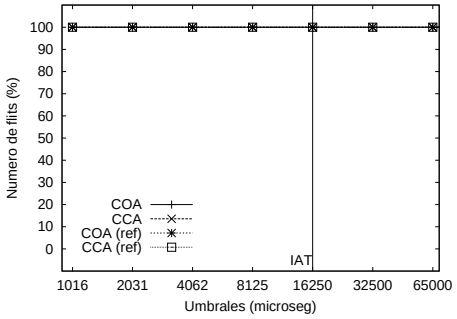
(a) 64 Kbps, carga multimedia = 52 %



(b) 64 Kbps, carga multimedia = 67 %

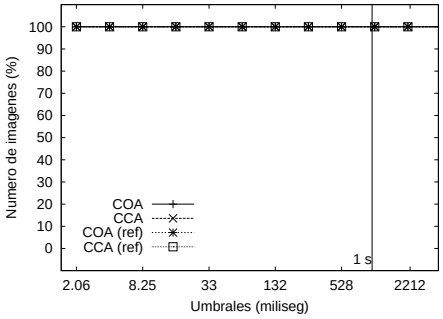


(a) 1.54 Mbps, carga multimedia = 52 %

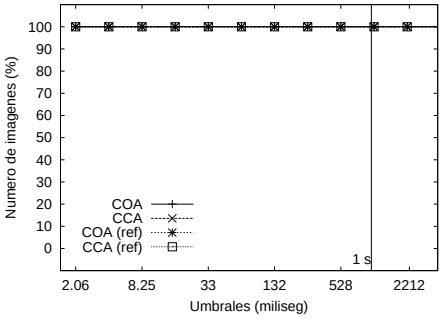


(b) 1.54 Mbps, carga multimedia = 67 %

Figura 6.99: Evaluación final: Tráfico multimedia mixto + control + *best effort*. Distribución del retardo medio de los flits CBR a las cargas máximas generadas.

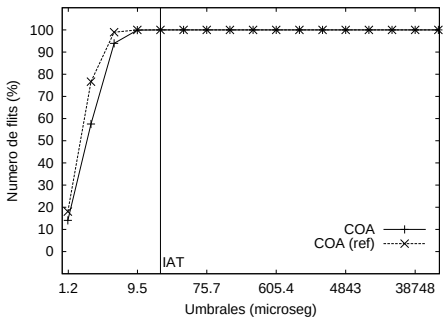


(a) Carga multimedia = 52 %

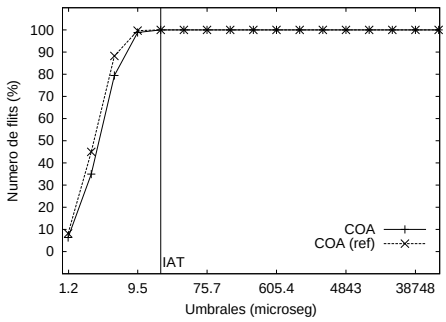


(b) Carga multimedia = 67 %

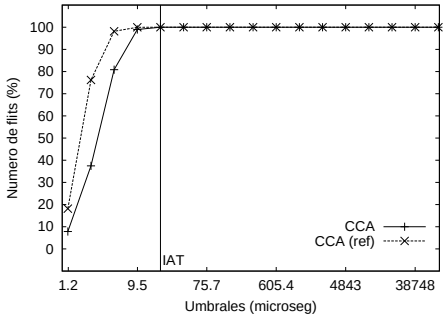
Figura 6.100: Evaluación final: Tráfico multimedia mixto + control + *best effort*. Distribución del retardo medio de las imágenes VBR a las cargas máximas generadas.



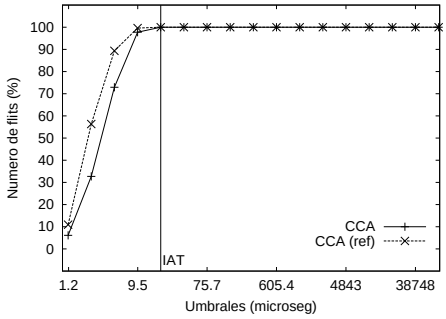
(a) COA, carga multimedia = 52 %



(b) COA, carga multimedia = 67 %

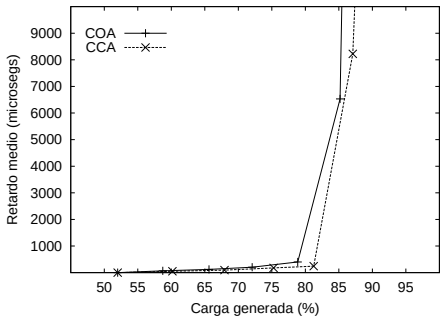


(a) CCA, carga multimedia = 52 %

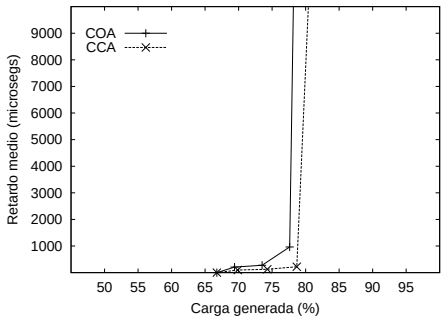


(b) CCA, carga multimedia = 67 %

Figura 6.101: Evaluación final: Tráfico multimedia mixto + control + *best effort*. Distribución del retardo medio de los flits de las conexiones CBR más exigentes a las cargas máximas generadas.



(a) Carga multimedia = 52 %



(b) Carga multimedia = 67 %

Figura 6.102: Evaluación final: Tráfico multimedia mixto + control + *best effort*. Retardo medio de los flits *best effort* desde su generación.

Capítulo 7

Conclusiones y Trabajo Futuro

Una vez se ha expuesto exhaustivamente todo el trabajo realizado, este Capítulo está dedicado a resumir las principales conclusiones obtenidas del trabajo reflejado en la presente memoria. Asimismo, se indican las contribuciones a diversos Congresos a las que ha dado lugar. Por último, se plantearán algunas posibles líneas de investigación que se pueden llevar a cabo como continuación del trabajo presentado en esta memoria.

7.1. Conclusiones y aportaciones

La principal aportación que se ha realizado en esta Tesis ha sido el diseño y evaluación de una **nueva arquitectura de Encaminador**, orientada específicamente a soportar el tráfico multimedia cada vez más presente en entornos de área local. El Encaminador Multimedia toma ideas y soluciones ya empleadas en conmutadores diseñados para tráfico convencional en redes de multicomputadores y LAN/SAN, pero las combina de una manera novedosa para dar el soporte adecuado a los diferentes tipos de tráfico presentes en entornos COW/LAN actuales. Estos tipos de tráfico son flujos multimedia, tanto de tasa constante (CBR) como variable (VBR), mensajes de control cortos y tráfico convencional de tipo *best-effort*.

Las características clave del Encaminador Multimedia pueden resumirse en los siguientes puntos:

Organización con *buffers* a la entrada Las últimas tendencias en el diseño de encaminadores y conmutadores para redes de altas prestaciones emplean esta organización, pues otras alternativas resultan demasiado costosas para su adecuación a la elevada frecuencia de funcionamiento de los enlaces actuales.

Técnica de conmutación híbrida Los flujos multimedia necesitan de una técnica orientada a la conexión, que les garantice recursos en toda la ruta a seguir. Por otro lado, el tráfico *best-effort* y los mensajes de control se beneficiarían de una técnica sin conexiones. Por tanto, el Encaminador Multimedia MMR trata a cada tipo de tráfico con la técnica más adecuada. Se emplea un esquema orientado a la conexión para el tráfico multimedia, y un esquema *virtual cut-through* para las otras clases de tráfico.

Elevado número de canales virtuales En un entorno local, el número de flujos multimedia es moderadamente alto, en torno a unos pocos cientos de conexiones multimedia. Puesto que el Encaminador Multimedia adjudica un canal virtual a cada conexión, el

número de éstos que debe implementarse es relativamente elevado. Para facilitar su implementación, se ha propuesto una organización distinta a la convencional del espacio dedicado a *buffers*. basada en el empleo de módulos de memoria RAM convenientemente entrelazados.

***Crossbar* multiplexado** El elevado número de canales virtuales desaconseja la realización práctica del Encaminador alrededor de un *crossbar* completamente demultiplexado, pues su coste sería prohibitivo. Sin embargo, la elección de un *crossbar* multiplexado complica el problema de la planificación del tráfico, pues es necesario realizar el arbitraje tanto en los puertos de entrada como dentro del *crossbar*.

Transmisión síncrona de flits El hecho de tener que respetar las garantías de QoS de cada flujo puede obligar a retrasar la transmisión de un flit en favor de otro más prioritario. Esta idea es difícil de implementar si los flits se transmiten asíncronamente. Por ello, en el MMR la transmisión de flits se realiza de forma síncrona. Esto es, el avance de flits se organiza según *ciclos de flit*, esto es, en cada ciclo de flit, todos los puertos de entrada transmitirán como máximo un flit a la vez.

Control de flujo basado en créditos La pérdida de información por desbordamiento de los *buffers* se evita empleando un esquema de control de flujo basado en créditos sobre cada canal virtual. Este esquema resulta adecuado pues el tamaño de las unidades de control de flujo, los flits, debe ser grande para amortizar la reconfiguración del *crossbar* en cada ciclo de flit.

Flits grandes El empleo de flits grandes resulta conveniente por varios motivos. En primer lugar, de esta forma la reconfiguración del *crossbar* se amortiza sobre una mayor cantidad de datos. Además, puesto que el cálculo de la planificación se realiza en paralelo con la transmisión de flits, se dispone de más tiempo para llevar a cabo dicho cálculo. Por último, de esta forma se amortiza también mejor la sobrecarga debida la información de control y de control de flujo asociada a cada canal virtual.

Para validar los parámetros de diseño se efectuaron una serie de pruebas preliminares, reflejadas en la Sección 6.4. Las conclusiones que se pueden obtener de este estudio son las siguientes:

1. En cuanto al tamaño de los flits:
   - Los flits grandes efectivamente mejoran el aprovechamiento del ancho de banda de los enlaces
   - Adicionalmente, amplían el rango de carga útil del Encaminador, donde es capaz de ofrecer las garantías de QoS necesarias para las aplicaciones multimedia
   - Las pruebas realizadas aconsejan *un tamaño de flit de 1024 bits*
2. En cuanto al tamaño del ciclo de planificación (determinado por el valor del parámetro K):
   - Su valor no influye sustancialmente en las prestaciones temporales ofrecidas a las aplicaciones multimedia
   - En cambio, sí que afecta al número de conexiones que son admitidas o rechazadas, pues está relacionado directamente con la granularidad de asignación del ancho de banda. Cuando aumenta el tamaño del ciclo de planificación, el ancho de banda

se asigna con una granularidad más fina, lo que permite ajustar mejor la reserva efectuada por las conexiones a sus requerimientos reales. Esto permite la admisión de un mayor número de conexiones, especialmente cuando la carga es moderada

- En las simulaciones llevadas a cabo, se ha mostrado como *un valor de K igual a 16* permite obtener una granularidad adecuada del ancho de banda. Valores mayores no redundan en un mayor número de conexiones aceptadas

3. En cuanto al tamaño de los *buffers* del MMR:

   - En presencia de tráfico VBR, el uso de *buffers* de gran capacidad no sólo no ofrece ninguna ventaja con respecto al uso de *buffers* más pequeños, sino que empeora notablemente las prestaciones ofrecidas a este tipo de tráfico
   - Se ha demostrado como las mejores prestaciones se obtienen además con el tamaño de *buffers* mínimo, esto es, *1 flit por cada canal virtual.*

Una vez definida la arquitectura, el trabajo se ha centrado sobre el problema de la planificación del tráfico. En esta línea se han propuesto varios **algoritmos de planificación del tráfico** adecuados para la nueva arquitectura de conmutador.

Por el hecho de usar un *crossbar* multiplexado, la planificación ha tenido que resolver dos problemas: i) el arbitraje entre canales virtuales que comparten un mismo enlace físico, y ii) el arbitraje entre los puertos de entrada por el uso de los puertos de salida. El primer problema consiste en la *planificación de enlaces*, y debe abordar la resolución de la contención por el uso de cada puerto de entrada al *crossbar*. El segundo problema es la *planificación del conmutador*, y trata del cálculo de emparejamientos entre los puertos de entrada y salida del *crossbar*, de manera que no existan conflictos por el uso de los mismos. Al plantear ambas soluciones de planificación, se ha tenido en cuenta, por encima de la maximización de la utilización del *crossbar* y de los enlaces, la provisión de las garantías de QoS que necesitan los flujos generados por aplicaciones multimedia.

Concretamente, se han propuesto **tres nuevos algoritmos de planificación de enlaces**, basados en el concepto de prioridades dinámicas. Dichas prioridades se calculan en base a los requerimientos de QoS de las conexiones, y de alguna medida de la QoS que realmente están recibiendo.

La evaluación de estos algoritmos se ha llevado a cabo en la Sección 6.5. Las conclusiones obtenidas son:

- El empleo del número máximo de candidatos por enlace físico aumenta sensiblemente la productividad alcanzada por el Encaminador
- El algoritmo IABP ofrece un buen aprovechamiento del ancho de banda de los enlaces, y es capaz de proporcionar buenas prestaciones en términos de retardo a las conexiones multimedia
- El algoritmo JBP optimiza el valor del *jitter* de los flits CBR, a costa de alcanzar una menor productividad. Sin embargo, cuando se aplica a conexiones VBR, no es capaz de optimizar el *jitter* sufrido por las unidades de datos de las aplicaciones
- El algoritmo SIABP ofrece mejores prestaciones que el algoritmo IABP a las conexiones de menores requerimientos sin perjudicar a las más exigentes
- Además, el algoritmo SIABP reduce drásticamente la complejidad hardware del mecanismo de cálculo de la prioridad

- El uso del algoritmo SIABP ofrece un mayor rango de carga útil que otros algoritmos tradicionalmente empleados para la planificación de enlaces. SIABP permite satisfacer las necesidades de QoS de aplicaciones con diferentes requerimientos a cargas superiores que otros algoritmos, que fueron diseñados para su uso en conmutadores con *buffers* a la salida

Para completar la solución al problema de la planificación, se han propuesto **dos algoritmos de planificación del conmutador**. Estos algoritmos se basan en el grado de conflictos por puerto de salida, y en la prioridad de los flits a transmitir, para calcular los emparejamientos entre entradas y salidas del *crossbar*. De esta forma se consigue no sólo obtener una buena utilización de los enlaces y del *crossbar*, sino que las garantías de QoS de las conexiones multimedia sean satisfechas.

Estos algoritmos se han evaluado en la Sección 6.6, junto con otro algoritmo convencional, que no tiene en cuenta los requisitos de QoS de los flits. Las conclusiones que se pueden obtener son las siguientes:

- El empleo de un algoritmo de planificación del conmutador que no tiene en cuenta las garantías de QoS de las conexiones multimedia limita severamente el rango de carga útil alcanzable por el Encaminador

- Los dos algoritmos propuestos son capaces de ofrecer las garantías de QoS que necesitan las aplicaciones multimedia, incluso a cargas moderadas a altas.

- El rango de carga útil es ligeramente superior para el algoritmo CCA que para el algoritmo COA, aunque ambos ofrecen mejores prestaciones que otras propuestas simples que no tienen en cuenta las necesidades de QoS de las aplicaciones multimedia a la hora de realizar la planificación del *crossbar*

Por último, la Sección 6.7 se ha dedicado a analizar las prestaciones del Encaminador Multimedia MMR con todos los tipos de tráfico considerados: conexiones multimedia, mensajes de control y tráfico *best-effort*. En concreto, se han demostrado los siguientes puntos:

- El Encaminador propuesto, junto con los algoritmos de planificación desarrollados, efectivamente es capaz de ofrecer las garantías de QoS a las conexiones multimedia

- Esta garantía es independiente de la presencia o no de tráfico convencional *best-effort*

- Los mensajes de control llegan a su destino con una latencia mínima, que es independiente de la carga generada, y de la presencia de los distintos tipos de tráfico

- El ancho de banda no ocupado por las conexiones multimedia es aprovechado de forma eficiente por el tráfico *best-effort*

- El rango de carga útil que ofrece el Encaminador alcanza el 80 %-90 % de utilización de los enlaces, según la configuración del Encaminador, y la composición de la carga empleada

7.2. Publicaciones relacionadas con la Tesis

Durante el desarrollo de la presente Tesis Doctoral, se han publicado partes y versiones previas del trabajo realizado en diversos Congresos y Simposios tanto nacionales como internacionales. En orden cronológico, se trata de las siguientes publicaciones:

- J. Duato, S. Yalamanchili, **M.B. Caminero**, D. Love, F.J. Quiles, *"MMR: A high-performance multimedia router. Architecture and design trade-offs"*. 5th Int. Symposium on High Performance Computer Architecture (HPCA-5). Orlando, Florida (EE.UU.). Enero, 1999. Proceedings publicados por IEEE Computer Society con I.S.B.N. 0-7695-0004-8/99, p.p. 300-309.

  En este trabajo se presenta por primera vez la arquitectura global del Encaminador Multimedia MMR, desglosando sus principales características, ponderando las alternativas de diseño, y justificando las decisiones tomadas al definir su arquitectura.

- **M.B. Caminero**, F.J. Quiles, J. Duato, D. Love, S. Yalamanchili, *"Performance Evaluation of the Multimedia Router with MPEG-2 Video Traffic"*. Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing (CANPC'99). Orlando, Florida (EE.UU.). Enero, 1999. Proceedings publicados como Lecture Notes in Computer Science, Vol. 1602, pp. 62-76. Ed. Springer-Verlag. I.S.B.N.: 3-540-65915-3.

  Este trabajo contiene una primera evaluación de prestaciones, empleando fuentes VBR generadas sintéticamente, y una versión muy simple de la herramienta de simulación.

- **M.B. Caminero**, C. Carrión, F.J. Quiles, S. Yalamanchili, J. Duato, *"MMR: Un encaminador para Tráfico Multimedia"*. X Jornadas de Paralelismo. La Manga del Mar Menor (Murcia). Septiembre, 1999. Actas publicadas con Depósito Legal MU. 1549-1999, pp. 219-224.

  Este artículo presenta el algoritmo de planificación de enlaces IABP, y el algoritmo de planificación del conmutador posteriormente bautizado como COA (Conflict Order Arbiter). Sus prestaciones se evalúan con tráfico CBR. Se trata de una versión preliminar del artículo presentado el año siguiente en el IPDPS'00 [104].

- D. Love , S. Yalamanchili, J. Duato, **M.B. Caminero**, F.J. Quiles, *"Switch Scheduling in the Multimedia Router (MMR)"*. International Parallel and Distributed Processing Symposium IPDPS'2000. Cancún (Méjico). Mayo, 2000. Proceedings publicados por IEEE Computer Society con I.S.B.N. 0-7695-0574-0, p.p. 5-11.

  En este artículo se presentan y se evalúan los algoritmos de planificación de enlaces IABP y JBP, junto con el algoritmo de planificación del conmutador basado en una matriz de selección por niveles, denominado posteriormente COA. La evaluación se lleva a cabo con tráfico compuesto de conexiones CBR con diferentes requerimientos de ancho de banda.

- **M.B. Caminero**, C. Carrión, F.J. Quiles, J. Duato, *"El Router Multimedia (MMR): Influencia de los Recursos de Almacenamiento"*. XI Jornadas de Paralelismo. Granada. Septiembre, 2000. Actas publicadas con I.S.B.N. 84-699-3003-6, pp. 155-160.

  Este artículo presenta una evaluación de la influencia del tamaño de los *buffers* situados en el MMR sobre la QoS que reciben las conexiones multimedia CBR y VBR, estas últimas con el modelo de inyección SR. Se demuestra como el empleo de *buffers* pequeños mejora notablemente las prestaciones alcanzadas por este tipo de tráfico. Se trata de una versión preliminar del artículo presentado el año siguiente en el CAC'01 [27].

- **M.B. Caminero**, C. Carrión, F.J. Quiles, J. Duato, S. Yalamanchili, *"Tuning Buffer Size in the Multimedia Router (MMR)"*. Workshop on Communication Architectures for Clusters (CAC'01). San Francisco (EE.UU.). Abril, 2001. Proceedings publicados por IEEE Computer Society con I.S.B.N. 0-7695-0990-8/01 (CD-ROM).

  En este artículo se presenta una evaluación más detallada de la influencia del tamaño de los *buffers* del MMR sobre las prestaciones ofrecidas al tráfico multimedia. El análisis se centra en el tráfico VBR, empleando los dos modelos de inyección de los flits que componen las tramas (SR y BB). Se llega a la conclusión de que los *buffers* con capacidad de un flit por canal virtual bastan para proveer a las conexiones de la QoS que necesitan.

- **M.B. Caminero**, C. Carrión, F.J. Quiles, J. Duato, S. Yalamanchili, *"A Cost-effective Hardware Link Scheduling Algorithm for the Multimedia Router (MMR)"*. IEEE International Conference on Networking (ICN'01), Colmar (Francia). Julio, 2001. Publicado como Lecture Notes in Computer Science, Vol. 2094, pp. 358-369, Ed. Springer-Verlag. I.S.B.N.: 3-540-42303-6.

  En este trabajo se presenta el algoritmo de planificación de enlaces SIABP, y se compara con la propuesta previa (IABP [104]) en términos de complejidad de implementación, y de prestaciones ofrecidas al tráfico CBR y VBR. Se demuestra como el nuevo algoritmo no sólo simplifica drásticamente el hardware necesario para su implementación, sino que además mantiene, e incluso a veces mejora, las prestaciones obtenidas por los flujos multimedia.

- **M.B. Caminero**, C. Carrión, F.J. Quiles, J. Duato, *"Sobre la Planificación de Enlaces en el Encaminador Multimedia (MMR)"*. XII Jornadas de Paralelismo. Valencia. Septiembre, 2001. Actas publicadas con I.S.B.N. 84-9705-043-6, pp. 155-160.

  Este artículo consiste en una versión reducida del trabajo presentado en Julio del mismo año en el ICN'01 [26].

- **M.B. Caminero**, C. Carrión, F.J. Quiles, J. Duato, S. Yalamanchili, *"Investigating Switch Scheduling Algorithms to Support QoS in the MultiMedia Router MMR"*. Workshop on Communication Architecture for Clusters (CAC'02). Fort Lauderdale, Florida (EE.UU.). Abril, 2002. Proceedings publicados por IEEE Computer Society, con I.S.B.N. 0-7695-1573-8 (CD-ROM).

  En este artículo se comparan las prestaciones del algoritmo de planificación de conmutadores COA, con las proporcionadas por el algoritmo WFA, propuesto por Tamir y Chi [162]. Se demuestra como el algoritmo COA es capaz de proporcionar garantías de QoS a las aplicaciones en un rango de cargas más amplio que el algoritmo WFA. Esto se debe a que, al contrario que WFA y otros algoritmos similares a él, COA tiene en cuenta los requerimientos de QoS de las conexiones a la hora de realizar las decisiones de emparejamiento.

- **M.B. Caminero**, C. Carrión, F.J. Quiles, J. Duato, S. Yalamanchili, *"A Multimedia Router Architecture to Provide High Performance and QoS Guarantees to Mixed Traffic"* . IEEE International Conference on Multimedia and Expo (ICME'2002). Lausana (Suiza). Agosto, 2002. Proceedings publicados por IEEE Computer Society, con I.S.B.N. por determinar.

  Este artículo presenta la evaluación completa del algoritmo de planificación COA, con carga compuesta por todos los tipos de tráfico considerados: conexiones multimedia CBR y VBR, mensajes de control y tráfico *best-effort*. Se demuestra como el Encaminador Multimedia, con el algoritmo COA como planificador del conmutador, es capaz

de satisfacer adecuadamente las necesidades de cada tipo de tráfico. El estudio toma de nuevo como referencia al algoritmo WFA [162].

Además, los siguientes trabajos han sido enviados a diferentes Congresos y Revistas internacionales, y se encuentran en proceso de revisión:

- **M.B. Caminero**, C. Carrión, F.J. Quiles, J. Duato, S. Yalamanchili, *"On the Minimum Buffer Size Requirements of the Multimedia Router"*.

  Se trata de una versión ampliada del artículo presentado en el CAC'01 [27]. En ella se amplía el análisis sobre el tamaño de *buffers* del MMR, considerando el algoritmo de planificación de enlaces SIABP. Se demuestra como los resultados obtenidos respecto al tamaño de *buffers* en el MMR para el algoritmo IABP se mantienen para la versión simplificada del mismo, SIABP. Esto es, cuando se emplea SIABP como planificador de enlaces, también bastan *buffers* de un flit por canal virtual del MMR para proporcionar la QoS que necesitan las aplicaciones multimedia.

- **M.B. Caminero**, C. Carrión, F.J. Quiles, J. Duato, S. Yalamanchili, *"A New Switch Scheduling Algorithm to improve QoS in the MultiMedia Router (MMR)"*.

  En este artículo se presenta el algoritmo de planificación del conmutador CCA (Candidate Conflict Arbiter). Sus prestaciones se comparan con las ofrecidas por el algoritmo WFA [162] en el entorno objeto de estudio. La evaluación se lleva a cabo con tráfico multimedia, tanto CBR como VBR.

7.3. Trabajo futuro

Como suele suceder en todo trabajo de investigación, nunca se puede poner un punto y final, pues siempre quedan aspectos por tratar y por considerar. Una vez alcanzados los objetivos a cumplir en esta Tesis, quedan todavía abiertos los siguientes temas:

- La continuación más evidente al trabajo presentado consiste en ampliar el estudio a una red compuesta de Encaminadores Multimedia. Esta es una línea de trabajo compleja, que involucra varias tareas:

  - En primer lugar, habría que estudiar la capacidad de garantizar la QoS cuando se acopla el efecto de los distintos planificadores. Los algoritmos diseñados y presentados en el presente trabajo ofrecen buenas prestaciones, y un rango de carga útil muy amplio. El estudio se ha limitado a un sólo encaminador, más los correspondientes interfaces de red, para acotar el problema y poder llevar a cabo un estudio exhaustivo del mismo. Pero cuando se traslade el problema a una red, es posible que haya que ajustar el funcionamiento de los distintos planificadores para que su acoplamiento no comprometa la consecución de los objetivos de diseño planteados.
  - También sería necesario efectuar un exhaustivo estudio de los algoritmos de encaminamiento, pues estos algoritmos se emplearán a la hora de establecer las conexiones multimedia, y a la hora de transmitir los mensajes de control y los flits *best-effort*. Las rutas que decidan para cada tipo de tráfico pueden afectar a las garantías de QoS que reciban las aplicaciones, pues un esquema simple podría llevar a cargar unas rutas más que otras.

  - Relacionado en cierta forma con el punto anterior, está el estudio de los algoritmos de Control de Admisión de Conexiones. Este control deberá de efectuarse en cada encaminador que pertenezca a una ruta entre un origen y un destino, cuando existe una petición de establecimiento de conexión. El objetivo de este control es que no se sobrepase el nivel de carga máxima en la red, a partir del cual las garantías de QoS quedan comprometidas.

- Todos los estudios realizados en esta Tesis se han realizado considerando una tarjeta interfaz de red sencilla. Su funcionalidad se limita a mantener los *buffers* donde las fuentes de tráfico depositan la información que generan, realizar el control de flujo basado en créditos con el puerto de entrada al MMR correspondiente, y transmitir los flits a través del enlace físico según un esquema cíclico bajo demanda. Una vía de trabajo podría consistir en hacer las tarjetas de red más sofisticadas, y analizar qué influencia tienen sobre las prestaciones ofrecidas al tráfico. Quizás esto permita simplificar el diseño del Encaminador.

- Siguiendo la tendencia marcada por Infiniband [132], otro tema a analizar consiste en la reducción del número de canales virtuales presentes en el MMR. Esto obligaría a romper la correspondencia biunívoca que existe en el diseño actual entre conexiones multimedia y canales virtuales. Por tanto, habría que plantear nuevas estrategias de compartición de recursos entre conexiones, de manera que todas puedan recibir sus garantías de QoS sin comprometer las garantías de otras conexiones.

Señalar por último que el trabajo presentado en esta Tesis Doctoral, así como las líneas expuestas en esta Sección dedicada a Trabajos Futuros, se ha desarrollado y/o se desarrollarán en el marco de dos proyectos concedidos por la Comisión Interministerial de Ciencia y Tecnología (C.I.C.Y.T.):

TIC97-0897-C04-02 *"Desarrollo de una Red de Estaciones de Trabajo de Altas Prestaciones y Bajo Coste"*, proyecto coordinado entre la Universidad de Castilla-La Mancha, la Universidad Politécnica de Valencia y la Universidad de Murcia, estando su periodo de vigencia comprendido entre Junio de 1997 y Diciembre de 2000.

TIC2000-1151-C07-02 *"Redes de Estaciones de Trabajo de Altas Prestaciones para Aplicaciones Multimedia"*, proyecto coordinado entre la Universidad de Castilla - La Mancha, la Universidad Politécnica de Valencia, la Universidad de Murcia, la Universidad de Valencia y la Universidad Jaime I, y siendo su periodo de vigencia desde Enero de 2001 hasta Diciembre de 2003.

Asimismo, se ha contado con diversas ayudas del Ministerio de Educación y Cultura, de la Universidad de Castilla-La Mancha, y de la Consejería de Ciencia y Tecnología de la Junta de Comunidades de Castilla-La Mancha.

Bibliografía

[1] Página web de "InfiniBand^{TM} Trade Association", en `http://infinibandta.com`.

[2] Página web de Inter-Fone, en `http://www.inter-fone.com/`.

[3] Página web de la Cadena 100, en `http://www.cadena100.es`.

[4] Página web de la Cadena Ser, en `http://www.cadenaser.es`.

[5] Página web de Microsoft NetMeeting, en `http://www.microsoft.com/windows/netmeeting/`.

[6] Página web de Microsoft Windows Media, en `http://www.microsoft.com/windows/windowsmedia/`.

[7] Página web de RealNetworks, en `http://www.realnetworks.com`.

[8] Página web del ATM Forum, en `http://www.atmforum.com/`.

[9] Synopsys DesignWare Guide, version 1998.02.

[10] Synopsys Reference Manual, version 1998.02.

[11] IEEE Standard VHDL; Language Reference Manual. IEEE STD1076 (1993).

[12] "Generic coding of moving pictures and associated audio". Recomendation H.262. Draft International Standard ISO/IEC 13818-2 (1994).

[13] M. Ajtai, J. Komlos y S. Szemeredi. "An O(N log N) sorting network". En *Proceedings de 25th ACM Symposium on Theory of Computing* (1983).

[14] F.J. Alfaro, J.L Sánchez y J. Duato. "A strategy to manage time sensitive traffic in InfiniBand". En *Proceedings del Workshop on Communication Architecture for Clusters (CAC'02)*. IEEE Computer Society (Abril 2002).

[15] F.J. Alfaro, J.L Sánchez, J. Duato y C.R. Das. "A strategy to compute the InfiniBand arbitration tables". En *Proceedings del International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society (Abril 2002).

[16] M. Allman, V. Paxson y W. Stevens. "TCP congestion control", Internet RFC 2581, Proposed Standard (Abril 1999).

[17] T. E. Anderson et al.. "High speed switch scheduling for local area networks". *ACM Transactions on Computer Systems* (1993).

[18] T. E. Anderson et al.. "A case for NOW (Networks of Workstations)". *IEEE Micro* (Febrero 1995).

[19] INFINIBAND TRADE ASSOCIATION. "InfiniBand Architecture specification, volume 1, release 1.0" (Octubre 2000).

[20] P. BARFORD Y M. CROVELLA. "Generating representative web workloads for network and server performance evaluation". En *Proceedings de ACM Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'98)*.

[21] K. BATCHER. "Sorting networks and their applications". En *Proceedings de AFIPS Spring Joint Computer Conference*.

[22] B. BENSAOU, K. T. CHAN Y D. H. K. TSANG. "Credit-Based Fair Queuing (CBFQ): A simple and feasible scheduling algorithm for packet networks". En *Proceedings de IEEE ATM Workshop* (1997).

[23] R. BHAGWAN Y B. LIN. "Fast and scalable priority queue architecture for high-speed network switches". En *Proceedings de IEEE INFOCOM* (Marzo 2000).

[24] U. BLACK. "QOS in Wide Area Networks". Prentice Hall (2000).

[25] N. BODEN ET AL.. "Myrinet: A gigabit per second LAN". *IEEE Micro* (Febrero 1995).

[26] M.B. CAMINERO, C. CARRIÓN, F.J. QUILES, J. DUATO Y S. YALAMANCHILI. "A cost-effective hardware link scheduler algorithm for the Multimedia Router (MMR)". En *Proceedings de International Conference on Networking (ICN'01) - Lecture Notes on Computer Science*, vol. 2094. Springer-Verlag (Julio 2001).

[27] M.B. CAMINERO, C. CARRIÓN, F.J. QUILES, J. DUATO Y S. YALAMANCHILI. "Tuning buffer size in the Multimedia Router (MMR)". En *Proceedings de Workshop on Communication Architecture for Clusters (CAC'01) - International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society (Abril 2001).

[28] M.B. CAMINERO, C. CARRIÓN, F.J. QUILES, J. DUATO Y S. YALAMANCHILI. "Investigating switch scheduling algorithms to support QoS in the Multimedia Router MMR". En *Proceedings del Workshop on Communication Architecture for Clusters (CAC'02)*. IEEE Computer Society (Abril 2002).

[29] E. CAVALLERO, M. BUTTO Y A. TONIETTI. "Effectiveness of the leaky bucket policing mechanism in ATM networks". *IEEE Journal on Selected Areas of Communications* **9**(3) (1991).

[30] J. CHAO. "Saturn: A terabit packet switch using dual round-robin". *IEEE Communications Magazine* (Diciembre 2000).

[31] T. M. CHEN Y S. S. LIU. "ATM Switching Systems". Artech House Publishers (1995).

[32] A. A. CHIEN Y J. H. KIM. "Approaches to quality of service in high-performance networks". En *Proceedings de Parallel Computer Routing and Communications Workshop (PCRCW)* (1997).

[33] S.T. CHUANG, A. GOEL, N. MCKEOWN Y B. PRABHAKAR. "Matching output queuing with a combined input output queued switch". *IEEE Journal on Selected Areas in Communications* **17**(6) (Junio 1999).

[34] D.E. COMER. "Internetworking with TCP/IP: Principles, Protocols, and Architectures", vol. 1. Prentice-Hall, 4 edición (2000).

[35] P. CUENCA. "Codificación y Transmisión Robusta de Señales de Vídeo MPEG-2 de Caudal Variable sobre Redes de Transmisión Asíncrona ATM". Tesis Doctoral, Universidad Politécnica de Valencia (1998).

[36] P. CUENCA, L. OROZCO-BARBOSA, F. J. QUILES Y A. GARRIDO. "Loss-resilient ATM protocol architecture for MPEG-2 video communications". *IEEE Journal on Selected Areas in Communications* **18**(6) (Junio 2000).

[37] P. CUENCA, L. OROZCO-BARBOSA, L. WANG, A. GARRIDO Y F. QUILES. "Packing scheme for layered coding MPEG-2 video transmission over ATM based networks". En *Proceedings de IEEE ATM Workshop* (Mayo 1997).

[38] J.G. DAI Y B. PRABHAKAR. "The throughput of data switches with and without speedup". En *IEEE INFOCOM* (2000).

[39] I. DALGIC. "Performance of Ethernet and ATM Networks Carrying Video Traffic Based on Accurate Characteristics of Video Sources". Tesis Doctoral, Universidad de Stanford (Agosto 1996).

[40] W. J. DALLY. "Virtual-channel flow control". *IEEE Transactions on Parallel and Distributed Systems* **3**(2) (Marzo 1992).

[41] W. J. DALLY ET AL.. "The J-machine: A fine grain concurrent computer". En *Proceedings de IFIP World Computer Congress (IFIP WCC'89)* (1989).

[42] W. J. DALLY Y C. L. SEITZ. "The Torus routing chip". *Journal of Distributed Computing* **1**(3) (1986).

[43] W.J. DALLY ET AL.. "Architecture and implementation of the Reliable Router". En *Proceedings de Hot Interconnects II* (1994).

[44] B. V. DAO, J. DUATO Y S. YALAMANCHILI. "Configurable flow control mechanisms for fault-tolerant routing". En *Proceedings de 22nd International Symposium on Computer Architecture (ISCA)* (Junio 1995).

[45] A. DEMERS, S. KESHAV Y S. SHENKER. "Analysis and simulations of a fair queuing algorithm". En *Proceedings de ACM SIGCOMM* (1989).

[46] M. DOWD, Y. PERL, L. RUDOLPH Y M. SAKS. "The periodic balanced sorting network". *Journal of the ACM* **36**(4) (Octubre 1989).

[47] J. DUATO, P. LÓPEZ, F. SILLA Y S. YALAMANCHILI. "A high performance router architecture for interconnection networks". En *Proceedings de International Conference on Parallel Processing (ICPP)* (Agosto 1996).

[48] J. DUATO, A. ROBLES, F. SILLA Y R. BEIVIDE. "A comparison of router architectures for virtual cut-through and wormhole switching in a NOW environment". En *Proceedings de 13th International Parallel Processing Symposium, IEEE Computer Society* (1998).

[49] J. DUATO, A. ROBLES, F. SILLA Y R. BEIVIDE. "A comparison of router architectures for Virtual Cut-Through and Wormhole switching in a NOW environment". *Journal of Parallel and Distributed Computing* (61) (2001).

[50] J. DUATO, S. YALAMANCHILI, M. B. CAMINERO, D. LOVE Y F. J. QUILES. "MMR: A high-performance multimedia router - Architecture and design trade-offs". En *Proceedings de International Symposium on High Performance Computer Architecture (HPCA-5)*. IEEE Computer Society (1999).

[51] J. DUATO, S. YALAMANCHILI Y L. NI. "Interconnection Networks: An Engineering Approach". IEEE Computer Society (1997).

[52] H. EBERLE Y E. OERTLI. "Switcherland: A QoS communication architecture for workstation clusters". (Junio 1998).

[53] M. ESTEVE, J. C. GUERRI Y C. PALAU. "Redes de Áreal Local y su Interconexión". Servicio de Publicaciones de la Universidad Politécnica de Valencia (1999).

[54] D. FERRARI Y D. VERMA. "A scheme for real-time channel establishment in wide-area networks". *IEEE Journal on Selected Areas in Communications* (Abril 1990).

[55] R. FIELDING ET AL.. "Hypertext Transfer Protocol – HTTP/1.1", Internet Request for Comments (RFC) 2616 (Junio 1999).

[56] D. GALE Y L.S. SHAPLEY. "College admissions and the stability of marriage". *American Mathematical Monthly* **69** (1962).

[57] M. GALLES. "Spider: A high-speed network interconnect". *IEEE Micro* **17**(1) (Enero/Febrero 1997).

[58] D. GARCIA Y D. WATSON. "Servernet II". En *Proceedings del Workshop on Parallel Computer Routing and Communication* (Junio 1997).

[59] M. W. GARRET. "Contributions Toward Real-Time Services on Packet-Switched Networks". Tesis Doctoral, CU/CTR/TR 340-93-20, Universidad de Columbia (Mayo 1993).

[60] P. T. GAUGHAN Y S. YALAMANCHILI. "Pipelined Circuit-Switching: A fault-tolerant variant of wormhole routing". En *Proceedings del 4th IEEE International Symposium on Parallel and Distributed Processing* (Diciembre 1992).

[61] P. T. GAUGHAN Y S. YALAMANCHILI. "Adaptive routing protocols for hypercube interconnection networks". *IEEE Computer* (Mayo 1993).

[62] P. T. GAUGHAN Y S. YALAMANCHILI. "A family of fault-tolerant routing protocols for direct multiprocessor networks". *IEEE Transactions on Parallel and Distributed Systems* (Mayo 1995).

[63] M. GERLA ET AL.. "Quality of Service support in high-speed, wormhole routing networks". En *Proceedings del International Conference on Network Protocols* (Octubre 1996).

[64] J. GHOSH, A. HUKKOO Y A. VARMA. "Neural networks for fast arbitration and switching noise reduction in large crossbars". *IEEE Transactions on Circuits and Systems* **38**(8) (Agosto 1991).

[65] S. J. GOLESTANI. "A framing strategy for congestion management". *IEEE Journal on Selected Areas in Communications* (Septiembre 1991).

[66] S. J. GOLESTANI. "Congestion-free communication in high-speed packet networks". *IEEE Transactions on Communications* (Diciembre 1991).

[67] S. J. GOLESTANI. "Duration-limited statistical multiplexing of delay-sensitive traffic in packet networks". En *Proceedings de IEEE INFOCOM* (1991).

[68] S. J. GOLESTANI. "A self-clocked fair queuing scheme for broadband applications". En *Proceedings de IEEE INFOCOM* (1994).

[69] M. W. GOUDREAU, S. G. KOLLIOPOULOS Y S. B. RAO. "Scheduling algorithms for input-queued switches: Randomized techniques and experimental evaluation". En *Proceedings de IEEE INFOCOM* (2000).

[70] S.D. GRIBBLE ET AL.. "Self-similarity in file systems". En *Proceedings de Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'98).*

[71] D. GRUNWALD Y D. A. REED. "Analysis of backtracking routing in binary hypercube computers". Informe técnico UIUC-DCS-R-89-1486, Department of Computer Science, University of Illinois at Urbana-Champaign (Febrero 1989).

[72] J. L. HENNESSY Y D. A. PATTERSON. "Computer Architecture: A quantitative approach". Morgan-Kaufmann (1996).

[73] R. HORST. "TNet: A reliable System Area Network". *IEEE Micro* (Febrero 1995).

[74] A. HUNG, G. KESIDIS Y N. MCKEOWN. "ATM input-buffered switches with guaranteed-rate propiety". En *Proceedings de IEEE ISCC'98* (Julio 1998).

[75] K. HWANG Y F. A. BRIGGS. "Arquitectura de Computadora y Procesamiento Paralelo". McGraw-Hill (1987).

[76] CISCO SYSTEMS INC. "Network Design and Case Studies". Cisco Press, 2 edición (2000).

[77] Intel Corporation. "iPSC/2 and iPSC/860 User's guide" (1991).

[78] R. D. ISAAC. "The future of CMOS technology". *IBM Journal of Research and Development* **44**(3) (Mayo 2000).

[79] R. JAIN. "The art of computer systems performance analysis". Ed. John Wiley & Sons (1991).

[80] R. JAIN Y S. ROUTHIER. "Packet trains – Measurement and a new model for computer network traffic". *IEEE Journal on Selected Areas in Comunications* (Septiembre 1986).

[81] J. JONSSON Y J. VASELL. "A comparative study of methods for time-deterministic message delivery in a multiprocessor architecture". En *Proceedings de IEEE International Parallel Processing Symposium (IPPS)* (1996).

[82] C. KALMANEK, H. KANAKIA Y S. KESHAV. "Rate controlled servers for very high-speed networks". En *Proceedings de IEEE Global Telecommunications Conference* (1990).

[83] S. KALYANARAMAN. Página web "Congestion control literature", en `http://www.ecse.rpi.edu/Homepages/shivkuma/research/congpapers.html`. (2001).

[84] A.C. KAM Y K.Y. SIU. "Linear complexity algorithms for QoS support in input-queued switches with no speedup". *IEEE Journal on Selected Areas in Communications* **17**(6) (Junio 1999).

[85] G. KARLSSON. "Asynchronous transfer of video". *IEEE Communications Magazine.*

[86] M. J. KAROL, K. Y. ENG Y H. OBARA. "Improving the performance of input-queued ATM packet switches". En *Proceedings de IEEE INFOCOM* (1992).

[87] M. J. KAROL, M. G. HLUCHYJ Y S. P. MORGAN. "Input versus output queuing on a space division packet switch". *IEEE Transactions on Communications* (Diciembre 1987).

[88] R. KARP, U. VAZIRANI Y V. VAZIRANI. "An optimal algorithm for on-line bipartite matching". En *Proceedings del 22 ACM Symposium on Theory of Computing* (1990).

[89] M. KATEVENIS ET AL.. "ATLAS I: A single-chip ATM switch for NOWs". En *Proceedings de Workshop on Communications and Architectural Support for Network-based Parallel Computing (CANPC)* (1997).

[90] M. KATEVENIS, S. SIDIROPOULOS Y C. CORCOUBETIS. "Weighted round-robin cell multiplexing in a general-purpose ATM switch". *IEEE Journal on Selected Areas in Communications* (Octubre 1991).

[91] P. KERMANI Y L. KLEINROCK. "Virtual Cut-Through: A new computer communication switching technique". *Computer Networks* **3** (1979).

[92] J. H. KIM. "Bandwidth and latency guarantees in low-cost, high-performance networks". Tesis Doctoral, University of Illinois at Urbana-Champaign (1997).

[93] J. H. KIM Y CHIEN A. A. "Rotating Combined Queuing (RCQ): Bandwidth and latency guarantees in low-cost, high-performance networks". En *Proceedings de Int. Symposium on Computer Architecture (ISCA)* (1996).

[94] D.E. KNUTH. "The Art of Computer Programming, Vol. 3: Sorting and Searching". Addison-Wesley (1973).

[95] S. KONSTANTINIDOU Y L. SNYDER. "The Chaos router". *IEEE Transactions on Computers* **43**(12) (Diciembre 1994).

[96] P. KRISHNA, N. PATEL, A. CHARNY Y R. SIMCOE. "On the speedup required for work-conserving crossbar switches". *IEEE Journal on Selected Areas in Communications* **17**(6) (1999).

[97] J. F. KUROSE Y K. W. ROSS. "Computer Networking: A top-down approach featuring the Internet". Addison-Wesley (2000).

[98] B. KUSZMAUL. "The RACE network architecture". En *Proceedings de IEEE International Parallel Processing Symposium (IPPS)* (1995).

[99] B. KWAN ET AL.. "Best-effort bandwidth reservation in high-speed LAN's using wormhole routing". En *Proceedings de Int. Conf. on Computer Communications and Networks (ICCCN)* (1996).

[100] S. S. LAM Y G. G. XIE. "Burst scheduling networks". Informe técnico TR-94-20, Dept. of Computer Science, Univ. of Texas at Austin. (Octubre 1996).

[101] S. S. LAM y G. G. XIE. "Group priority scheduling". Informe técnico TR-95-28, Dept. of Computer Science, Univ. of Texas at Austin. (Enero 1997).

[102] J. LI y N. ANSARI. "Practical scheduling algorithms for input-queued switches". En *Proceedings de of the Conference on Information Sciences and Systems (CISS)* (2000).

[103] S. LI y N. ANSARI. "Input-queued switching with QoS guarantees". En *Proceedings de IEEE INFOCOM* (1999).

[104] D. LOVE, S. YALAMANCHILI, J. DUATO, M.B. CAMINERO y F.J. QUILES. "Switch scheduling in the Multimedia Router (MMR)". En *Proceedings de International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society (Mayo 2000).

[105] N. R. MALIK. "Circuitos electrónicos : Análisis, simulación y diseño". Prentice-Hall (1996).

[106] M. MANO. "Digital Design". Prentice-Hall International, 2 edición (1991).

[107] M. A. MARSAN, A. BIANCO, P. GIACCONE, E. LEONARDI y F. NERI. "Router architectures exploiting input-queued cell-based switching fabrics". En *nternational Workshop on QoS in Multiservice IP Networks (QoS-IP)* (2001).

[108] P. MCKENNEY. "Stochastic fairness queuing". En *Proceedings de IEEE INFOCOM* (1990).

[109] N. MCKEOWN. "Scheduling Algorithms for Input-Queued Cell Switches". Tesis Doctoral, Universidad de California en Berkeley (1995).

[110] N. MCKEOWN. "iSLIP: A scheduling algorithm for input-queued switches". *IEEE Transactions on Networking* **7**(2) (Abril 1999).

[111] N. MCKEOWN, V. ANANTHARAM y J. WALRAND. "Achieving 100 % throughput in an input-queued switch". En *Proceedings de IEEE INFOCOM*, vol. 1 (Marzo 1996).

[112] N. MCKEOWN y T.E. ANDERSON. "A quantitative comparison of iterative scheduling algorithms for input-queued switches". *Computer Networks and ISDN Systems* **30**(24) (Diciembre 1998).

[113] A. MEKKITTIKUL. "Scheduling Non-uniform Traffic in High Speed Packet Switches and Routers". Tesis Doctoral, Universidad de Stanford (1998).

[114] A. MEKKITTIKUL y N. MCKEOWN. "A starvation-free algorithm for achieving 100 % throughput in an input-queued switch". En *Proceedings de Int. Conf. on Computer Communications and Networks (ICCCN)* (1996).

[115] A. MEKKITTIKUL y N. MCKEOWN. "A practical scheduling algorithm to achieve 100 % throughput in input-queued switches". En *Proceedings de IEEE INFOCOM* (1998).

[116] C. MINKENBERG y T. ENGBERSEN. "A combined input and output queued packet-switched system based on PRIZMA switch-on-a-chip technology". *IEEE Communications Magazine* (Diciembre 2000).

[117] S. W. MOON, J. REXFORD y K. G. SHIN. "Scalable hardware priority queue architectures for high-speed packet switches". *IEEE Transactions on Computers* **49**(11) (2000).

[118] J. B. NAGLE. "On packet switches with infinite storage". *IEEE Transactions on Communications* (Abril 1997).

[119] M. NANDAKUMAR ET AL.. "A 0.25 μm gate length CMOS technology for 1V low power applications - Device design and power/performance considerations". En *Proceedings del 1996 Symposium on VLSI Technology* (1996).

[120] G. NONG Y M. HAMDI. "On the provision of Quality-of-Service guarantees for input queued switches". *IEEE Communications Magazine* (Diciembre 2000).

[121] G. NONG, J. K. MUPPALA Y M. HAMDI. "Analysis of non-blocking ATM switches with multiple input queues". *IEEE/ACM Transactions on Networking* **7**(1) (1999).

[122] A. G. NOWATZYK ET AL.. "S-Connect: from Network of Workstations to supercomputer performance". En *Proceedings del 22nd International Symposium on Computer Architecture* (1995).

[123] H. OBARA Y T. YASUSHI. "An efficient contention resolution algorithm for input queuing ATM cross-connect switches". *International Journal of Digital and Analog Cabled Systems* **2**(4) (1989).

[124] H. OHNISHI, T. OKADA Y K. NOGUCHI. "Flow control schemes and delay/loss tradeoff in ATM networks". *IEEE Journal on Selected Areas in Communications* **6**(9) (Diciembre 1988).

[125] S. OLARIU, M. C. PINOTTI Y S. Q. ZHENG. "How to sort N items using a sorting network of fixed I/O size". *IEEE Transactions on Parallel and Distributed Systems* **10**(5) (Mayo 1999).

[126] T. OLIVARES, P. CUENCA, F. QUILES, A. GARRIDO, J. SANCHEZ Y J. DUATO. "Interconnection network behavior on a multicomputer in the parallelization of the MPEG coding algorithm. Worm-hole vs Packet-Switching routing". En *Proceedings de IEEE High Performance Computing (HiPC'97)* (Diciembre 1997).

[127] S. PAKIN, M. LAURIA Y A. CHIEN. "High performance messaging on workstations: Illinois Fast Messages on Myrinet". En *Proceedings de Supercomputing* (1995).

[128] A. PAREKH Y R. GALLAGER. "A generalized processor sharing approach to flow control in integrated services networks – the single node case". En *Proceedings de IEEE INFOCOM* (1992).

[129] Y. K. PARK Y G. LEE. "Applications of neural networks in high-speed communication networks". *IEEE Communications Magazine* (Octubre 1995).

[130] Y. K. PARK Y G. LEE. "NN based ATM cell scheduling with queue length-based priority scheme". *IEEE Journal on Selected Areas in Communications* **15**(2) (Febrero 1997).

[131] J. PELISSIER. "Providing quality of service over Infiniband^{TM} architecture fabrics". En *Proceedings de Hot Interconnects* (2000).

[132] E. PENDERY Y J. EUNICE. "InfiniBand Architecture: Bridge over troubled waters", `http://www.infinibandta.org/press/`. (2000).

[133] M. PERKINS Y P. SKELLY. "A hardware MPEG clock recovery experiment for variable bit rate video transmission". Informe técnico ATM94-0434, ATM Forum (1994).

[134] L. L. Peterson y B. S. Davie. "Computer Networks: A Systems Approach, 2nd edition". Morgan Kaufmann (2000).

[135] G. Pfister. "High Performance Mass Storage and Parallel I/O", capítulo 42: "An Introduction to the InfiniBand Architecture". IEEE Press and Wiley Press (2001).

[136] B. Prabhakar y N. McKeown. "On the speedup required for combined input and output switching". Informe técnico CSL-TR-97-738, Universidad de Stanford (Noviembre 1997).

[137] M. Prycker. "Asynchronous transfer mode: solution for broadband ISDN". Ellis Horwood Limited (1991).

[138] J. Rexford, A. Greenberg y F. Bonomi. "Hardware-efficient fair queueing architectures for high-speed networks". En *Proceedings de IEEE INFOCOM*.

[139] J. Rexford, J. Hall y K. G. Shin. "A router architecture for real-time point-to-point networks". En *Proceedings de International Symposium on Computer Architecture (ISCA)* (1996).

[140] R. Schoenen, G. Post y G. Sander. "Weighted arbitration algorithms with priorities for input-queued switches with 100 % throughput". En *Proceedings de International Workshop on Broadband Switching Systems (BSS'99)* (Junio 1999).

[141] M. Schroeder. Página web "SRC Research: Networks", en `http://www.research.compaq.com/SRC/org/networks.html`. (2000).

[142] 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* (1991).

[143] M. Schwartz y D. Beaumont. "Quality of Service requirements for audio-visual multimedia services". Informe técnico ATM94-0640, ATM Forum (1994).

[144] S. L. Scott y G. M. Thorson. "Optimized routing in the Cray T3D". En *Proceedings del Parallel Computer Routing and Communications Workshop (PCRCW).* Springer Verlag Lecture Notes in Computer Science (1994).

[145] S. L. Scott y G. M. Thorson. "The Cray T3E network: Adaptive routing in a high performance 3D torus". En *Proceedings del Symposium on High Performance Interconnects (Hot Interconnects 4)* (1996).

[146] C. L. Seitz. "The Cosmic Cube". *Communications of the ACM* **28**(1) (1985).

[147] M. Shreedhar y G. Varghese. "Efficient fair queuing using deficit round-robin". *IEEE/ACM Transactions on Networking* (Junio 1996).

[148] F. Silla. "Routing and flow control in networks of workstations". Tesis Doctoral, Universidad Politécnica de Valencia (1998).

[149] F. Silla y J. Duato. "Improving the efficiency of adaptive routing in networks with irregular topology". En *Proceedings de Conference on High Performance Computing (HiPC)* (1997).

[150] F. Silla y J. Duato. "On the use of virtual channels in networks of workstations with irregular topology". En *Proccedings del Parallel Computer Routing and Communications Workshop (PCRCW)* (Junio 1997).

[151] F. SILLA Y J. DUATO. "High-performance routing in networks of workstations with irregular topology". *IEEE Transactions on Parallel and Distributed Systems* **11**(7) (2000).

[152] F. SILLA ET AL.. "Efficient adaptive routing in networks of workstations with irregular topology". En *Proceedings de Workshop on Communications and Architectural Support for Network-based Parallel Computing (CANPC)* (1997).

[153] R. SIVARAM, C. B. STUNKEL Y D. K. PANDA. "HIPIQS: A high-performance switch architecture using input queuing". Informe técnico OSU-CISRC-10/97-TR45, Ohio State University (1998).

[154] SCOTT S.L. Y J.R GOODMAN. "The impact of pipelined channels on k-ary n-cube networks". *IEEE Transactions on Parallel and Distributed Systems* **5**(1) (1994).

[155] W. STALLINGS. "Data and Computer Communications". Prentice-Hall, 5 edición (1997).

[156] R. STEINMETZ Y L. C. WOLF. "Quality of Service: Where are we?". En *Proceedings de IFIP Fifth International Workshop on Quality of Service (IWQOS'97)* (Mayo 1997).

[157] D. STILIADIS. "Traffic scheduling in packet-switched networks: Analysis, design and implementation". Tesis Doctoral, University at California at Santa Cruz (1996).

[158] D. STILIADIS Y A. VARMA. "Providing bandwidth guarantees in an input-buffered crossbar switch". En *Proceedings de IEEE INFOCOM* (1995).

[159] D. STILIADIS Y A. VARMA. "Design and analysis of frame-based queuing: A new traffic scheduling algorithm for packet-switched networks". En *Proceedings de SIGMETRICS* (1996).

[160] I. STOICA Y H. ZHANG. "Exact emulation of an output queuing switch by a combined input and output queuing switch". En *Proceedings IEEE/IFIP International Workshop on QoS (IWQoS'98)* (Mayo 1998).

[161] C. B. STUNKEL ET AL.. "The SP-2 high-performance switch". *IBM Systems Journal: Scalable Parallel Computing* **34**(2) (1995).

[162] Y. TAMIR Y H.C. CHI. "Symmetric crossbar arbiters for VLSI communication switches". *IEEE Transactions on Parallel and Distributed Systems* **4**(1) (1993).

[163] R. E. TARJAN. "Data structures and networks algorithms". *Society for Industrial and Applied Mathematics* (1983).

[164] K. TODA ET AL.. "Design and implementation of a priority forwarding router chip for real-time interconnection networks". *International Journal of Mini and Microcomputers* (Enero 1995).

[165] D. TOWSLEY. "Providing Quality of Service in packet switched networks". *Lecture Notes in Computer Science* **729**.

[166] T.P. TROUDET Y S.M. WALTERS. "Hopfield neural network architecture for crossbar switch control". *IEEE Transactions on Circuit and Systems* **CAS-38** (Enero 1991).

[167] J.S. TURNER. "New directions in communications (or Which way to the information age?)". *IEEE Communications Magazine* **25** (1986).

[168] D. VERMA, H. ZHANG y D. FERRARI. "Delay jitter control for real-time communication in a packet switching network". En *Proceedings de Tricomm* (1991).

[169] T. VON EICKEN, D. E. CULLER, S. C. GOLDSTEIN y K. E. SCHAUSER. "Active Messages: A mechanism for integrated communication and computation". En *Proceedings de International Symposium on Computer Architecture (ISCA)* (1992).

[170] W.D. WEBER ET AL.. "The Mercury interconnect architecture: A cost-effective infraestructure for high-performance servers". En *Proceedings del 24th International Symposium on Computer Architecture* (1997).

[171] T. WELLER y B. HAJEK. "Scheduling nonuniform traffic in a packet-switching system with small propagation delay". *IEEE/ACM Transactions on Networking* **5**(6) (Diciembre 1997).

[172] P.N. WERAHERA y A.P. JAYASUMANA. "Fiber Distributed Data Interface: Throughput evaluation with multiple classes of traffic". *IEEE Transactions on Communications* **42**(2/3/4) (Febrero 1994).

[173] K. H. YUM, E. J. KIM y C. R. DAS. "QoS provisioning in clusters: An investigation of router and NIC design". En *Proceedings de* 28^{th} *International Symposium on Computer Architecture (ISCA)* (Junio 2001).

[174] K. H. YUM, A. VAIDYA, C. R. DAS y A. SIVASUBRAMANIAN. "Investigating QoS support for traffic mixes with the MediaWorm router". En *Proceedings de High-Performance Computer Architecture (HPCA-6)* (2000).

[175] K. Y. YUN. "A terabit multi-service switch with Quality of Service support". En *Proceedings de Hot Interconnects* (2000).

[176] K.Y. YUN ET AL.. "A self-timed real-time sorting network". En *Proceedings de IEEE International Conference on Computer Design* (1998).

[177] H. ZHANG y D. FERRARI. "Rate-controlled static-priority queuing". En *Proceedings de IEEE INFOCOM* (1993).

[178] L. ZHANG. "Virtual Clock: A new traffic control algorithm for packet switching networks". *ACM Transaction on Computer Systems* (Mayo 1991).