University of Castilla-La Mancha

A publication of the

Department of Computer Science

Técnicas de Planificación de Conmutadores Orientadas a Garantizar QoS a Tráfico Multimedia

by

M. Blanca Caminero Herráez, Francisco J. Quiles Flor

Technical Report **#DIAB-00-02-09** Febrero 2000

DEPARTAMENTO DE INFORMÁTICA
ESCUELA POLITÉCNICA SUPERIOR
UNIVERSIDAD DE CASTILLA-LA MANCHA
Campus Universitario s/n
Albacete - 02071 - Spain
Phone +34.967.599200, Fax +34.967.599224

2

Técnicas de Planificación de Conmutadores Orientadas a Garantizar QoS a Tráfico Multimedia

M. Blanca Caminero Herráez, Francisco J. Quiles Flor

2

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

Durante los últimos treinta 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. Dos décadas después, 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. Durante este periodo, los arquitectos de ambos tipos de redes se han centrado 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.

En los últimos años, ha habido un crecimiento explosivo de las comunicaciones de datos contínuos (audio/vídeo) sobre redes de computadores. 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. 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. Estas aplicaciones incluyen vídeoconferencia, vídeo (o audio) bajo demanda, aprendizaje remoto, librerías de vídeo, televisión de alta definición, telecompra, y trabajo corporativo soportado por computador (CSCW, *Computer Supported Cooperative Work*).

Además de los mayores requerimientos de ancho de banda, las aplicaciones multimedia también necesitan un tipo completamente distinto de servicios de red, reflejado en lo que se denomina *Calidad de Servicio* (QoS).

Otro aspecto a considerar en las redes de computadores es el *control o la evitación de la congestión en la red*. Éste ha sido un tema de intenso debate durante varios años. Existen

varias propuestas en la literatura, a veces incompatibles entre sí. 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 las funciones básicas de un algoritmo de planificación del tráfico, y 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 contínuo 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.

Al emplear conmutación de paquetes, las aplicaciones pueden transmitir ráfagas de datos, con un caudal pico mucho mayor que su caudal medio. Pero los recursos de la red se reservan en términos de su caudal de transmisión medio. Por tanto, es necesario disponer de espacio de almacenamiento en los nodos intermedios de la red, para absorber las ráfagas y evitar las pérdidas de paquetes. Esta técnica se basa en el principio clave del *multiplexado estadístico*.

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 por un enlace de salida. Esto es precisamente lo que hace el *algoritmo de planificación del tráfico*.

1.1 Conceptos previos

En este apartado, estableceremos un conjunto de conceptos que serán referidos con frecuencia a lo largo del trabajo.

1.1.1 Calidad de servicio

La *calidad de servicio* (*QoS*) se refiere al conjunto de parámetros relacionados con las incidencias experimentadas por el tráfico transportado, como por ejemplo, retardos sufridos por los paquetes, variación en los retardos y tasa de pérdida de paquetes (en su caso). 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. Para tratar con la gran cantidad de tipos de tráfico que circula sobre una red de servicios integrados, es necesario definir un número limitado de *clases de servicio*. Cada clase se define mediante unos requisitos específicos de QoS. Se suelen considerar tres clases básicas en las redes de servicios integrados [29]:

Caudal constante (CBR: *Constant Bit Rate*) Esta clase se emplea para emular la conmutación de circuitos. Los paquetes se generan a un caudal constante. Para este tipo de tráfico, la reserva de recursos de la red es simple porque el tráfico llega periódicamente a ritmo constante. 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.

Caudal variable (VBR: *Variable Bit Rate*) Esta clase permite a los usuarios enviar un caudal variable de información. El tráfico se compone de ráfagas, que empiezan a intervalos fijos, pero el tamaño de cada ráfaga varía. Esta variación impredecible del caudal generado hace que soportar el tráfico VBR de forma eficiente sea difícil. Si se reservan recursos de red según los caudales pico es muy posible que se estén desperdiciando recursos, pero si se hace la reserva en base a caudales medios, puede ocurrir que no se puedan soportar variaciones en el tráfico. Si se emplea multiplexación estadística,

pueden producirse pérdidas de paquetes. Los flujos de audio y vídeo comprimidos, como por ejemplo el tráfico MPEG, pertenecen a esta categoría de tráfico. 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.

Caudal disponible (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*).

Ésta es una simplificación de las categorías establecidas por el ATM Forum (CBR, *Real Time*-VBR, *Non Real Time*-VBR, ABR, UBR) [7, 5].

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. En principio, la diferenciación del tráfico beneficia al proveedor de servicios de red, puesto que se permite relajar los requerimientos de una QoS adecuada.

Con relación a la calidad de servicio, las prestaciones de la red se refieren al conjunto de parámetros que miden la habilidad de la red para proporcionar servicios entre usuarios. Al nivel de establecimiento de la conexión, los parámetros de prestaciones de la red incluyen el retraso en el establecimiento de la conexión, el retraso en su liberación, y la probabilidad de bloqueo.

Al nivel de célula o paquete, hablamos de tasa de error, tasa de pérdida de células o paquetes (en el caso de que se permitan descartes, como en ATM), tasa de inserción errónea de células o paquetes, retardo entre extremos, y variación en el retardo (*jitter*).

1.1.2 Compartición estadística frente a aislamiento

El *aislamiento de los flujos de tráfico* facilita la protección de la calidad de servicio requerida para cada conexión. Esto se consigue reservando para cada conexión su caudal máximo o caudal 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 caudales pico mayor que la capacidad física del enlace (esto siempre y cuando la suma de sus caudales medios sea inferior al caudal del enlace). Si

los flujos de tráfico son numerosos e independientes, puede darse el caso de que sus caudales instantáneos totales sobrepasen la capacidad del enlace (esto es, cuando muchas fuentes transmiten simultáneamente con sus caudales pico), pero esto sucede con una probabilidad muy pequeña. Esto se expresa mediente 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 caudal pico total y el caudal 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 las células o paquetes de las colas se planifican para su transmisión por un enlace compartido.

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

Por ejemplo, las células con prioridad de retardo alta tendrán preferencia en la transmisión sobre las células 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 proteger 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.

1.1.3 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. 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.

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. A nivel de conexión, cuando llega una nueva petición de conexión, se comprueban en cada conmutador una serie de condiciones para su admisión. La nueva conexión se admite únicamente si hay suficientes recursos para atender tanto la nueva, como las conexiones existentes, al nivel de QoS requerido. 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. Puesto que ambos niveles de control 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

Instantáneo	Descarte selectivo de paquetes Planificación de paquetes Suavizado del tráfico
Retardo de propagación entre extremos	Indicación explícita de congestión hacia adelante (EFCI)
Retardo de propagación de ida y vuelta entre extremos	Control de admisión de conexiones Encaminamiento
Minutos/horas...	...

Tabla 1.1: Mecanismos de control del tráfico, según sus escalas de tiempo

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 de servicios integrados.

1.1.4 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 células o paquetes consecutivos)
- Células o 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 tabla 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 las células 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.

El presente trabajo se centra sobre el estudio de los algoritmos de planificación (que actúan a nivel de paquetes). Todos los algoritmos de planificación existentes necesitan conocer información acerca de los parámetros de QoS solicitada por los flujos de datos para poder llevar a cabo la planificación adecuadamente. Esta información es facilitada durante la fase de establecimiento de una conexión, por lo que también habrá que prestar atención a los mecanismos de admisión de conexiones.

1.2 Características de una disciplina de servicio

Como hemos dicho previamente, 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 variablidad: 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.

1.3 Planteamiento del trabajo

A continuacin efectuamos una exposición del estado del arte actual en lo que se refiere a algoritmos de planificación de conmutadores. Describimos las diferencias en la planificación según tengamos una organización con *buffers* situados a la entrada o a la salida. Establecemos una clasificación de los mismos, y exponemos en detalle un ejemplo de cada clase. Para acabar el trabajo, describimos un par de algoritmos de planificación empleados en encaminadores para SAN y redes de multicomputadores.

Capítulo 2

Algoritmos de Planificación

Los algoritmos de planificación propuestos hasta la fecha pueden dividirse en dos categorías principales, *basados en prioridades* y *basados en tramas*, según la granularidad de la planificación [29].

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.

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

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

Basados en Prioridades

Orientados al retardo	Orientados al ancho de banda
CODA [49]	FQ [11]
RACE [30]	**VC** [51]
TDC [23]	PGPS [39]
DEDD [14]	SCFQ [21]
JEDD [50]	FFQ [48]
RCSD [52]	

Basados en Tramas

Conservan el trabajo	No conservan el trabajo
SG [20, 19]	**WRR** [26]
HRR [24]	DRR [43]

Tabla 2.1: Clasificación de los algoritmos existentes

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 2.1 se muestra la clasificación que hemos expuesto en los párrafos anteriores, junto con algunos algoritmos de ejemplo para cada grupo. En el resto del capítulo, repasaremos sus características más distintivas, aunque veremos con más detalle los que aparecen marcados en negrita en la tabla.

Por otro lado, existen muchas arquitecturas para conmutadores propuestas en la literatura (un buen repaso aparece en [5]).

Los tipos más comunes de interconexión entre entradas y salidas (denominadas en la literatura como *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 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

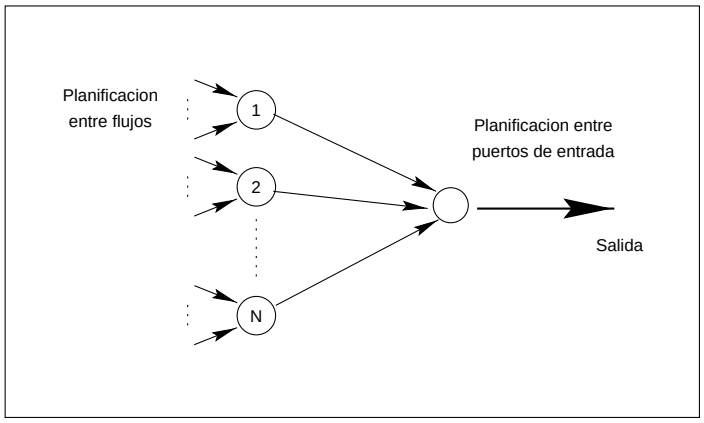


Figura 2.1: Los dos niveles de planificación en un conmutador con *buffers* a la entrada

ser enviado inmediatamente, mientras que el otro tendría que ser almacenado en un *buffer*. Esto se conoce como *contención de salida*.

El esquema empleado para almacenar los paquetes tiene una gran influencia tanto en el coste de implementación del conmutador, como en las prestaciones que proporciona. Con *buffers* a la entrada, los paquetes se almacenan en una cola situada en los puertos de entrada hasta que pueden ser transmitidos hacia el puerto de salida correspondiente. Un serio problema asociado a esta técnica es el *bloqueo de la cabeza de línea* (*head-of-line blocking* o *HOL blocking*). Esto sucede cuando hay paquetes en una cola de entrada que no pueden ser transmitidos, a pesar de que sus puertos de salida estén libres, debido a la contención de salida que sufre la cabeza de la cola. Se ha demostrado que la contención de cabeza de línea reduce la productividad de un *crossbar* al 58 % de su capacidad ante tráfico uniformemente distribuido [25].

Las prestaciones de un *crossbar* con *buffers* a la entrada puede mejorarse si se permite que el conmutador escoja uno de entre varios paquetes de la cola, para ser transmitidos en cada ciclo. Un *buffer* con ventana permite que cualquiera de los W primeros paquetes sea transmitido, siendo W el tamaño de la ventana que se aplica sobre la cola.

El problema de la planificación de paquetes dentro de un conmutador con *buffers* a la entrada puede simplificarse si se divide en dos niveles (ver figura 2.1):

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 2.1, y en general, empleando cualquiera de los existentes en la literatura para

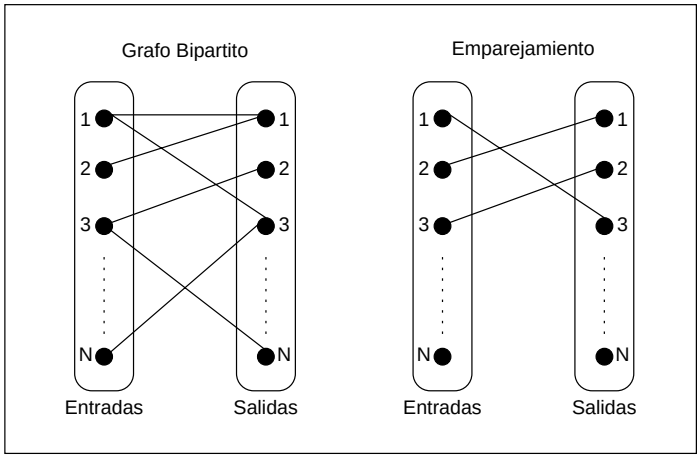


Figura 2.2: Ejemplo de grafo bipartito y de un emparejamiento válido

conmutadores con *buffers* en la salida.

En cuanto al primer problema, conmutar el máximo número de paquetes en un *crossbar* con colas de acceso aleatorio o con ventanas, se puede reducir al *problema del emparejamiento en un grafo bipartito* [1]. 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 2.2). 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 no hayan dos arcos con 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 planificado por el emparejamiento es máximo de entre todos los emparejamientos válidos, entonces tenemos un *emparejamiento máximo*. Es posible asignar pesos a los arcos que componen el grafo, con lo que la maximización puede hacerse también 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, como se verá en la sección 2.1.4.

2.1 Conmutadores con *buffers* a la entrada

2.1.1 Emparejamiento Paralelo Iterativo

La red AN2

El algoritmo de *Emparejamiento Paralelo Iterativo* es el empleado por el planificador diseñado para AN2, una red de área local de altas prestaciones desarrollada por el *Systems Research Center* de Digital, en 1993. El mismo centro desarrolló en 1990 Autonet [42], una red pionera dentro del campo de las redes locales de altas prestaciones. Esta red introdujo en su diseño elementos que hoy día son ampliamente aceptados en las redes de altas prestaciones en general, como por ejemplo, empleo de *wormhole* como técnica de conmutación, red basada en *host*s y conmutadores unidos mediante enlaces punto a punto de elevado ancho de banda, encaminamiento *up/down*...

Ahora bien, Autonet estaba diseñada pensando en un modelo de tráfico *best-effort*, para el cuál se garantizaba el reparto de todos los datos, pero no la ordenación de los mismos, ni una cota para su latencia. Este diseño es inadecuado cuando hay un patrón de tráfico en tiempo real, como transmisiones de vídeo o aplicaciones multimedia en general. Por ello, en el SRC de Digital se empezó a desarrollar un nuevo conmutador de altas prestaciones, que empleaba enlaces de fibra óptica con un ancho de banda de 1 Gb/s, y que es capaz de satisfacer los requerimientos del tráfico de datos en tiempo real, es decir, un ancho de banda garantizado y una latencia acotada. Este conmutador es el que se emplearía para la construcción de la red sucesora de Autonet, AN2 [1]. El conmutador de AN2 pretende ser un estudio de la arquitectura de conmutadores de altas prestaciones para redes locales de topología arbitraria.

En AN2, los datos se transmiten en células de tamaño fijo en lugar de paquetes de tamaño variable, que forman *flujos de datos*. Cada célula lleva un identificador del flujo al que pertenece. En cada conmutador hay una tabla de encaminamiento, construida durante la configuración de la red, que determina el puerto de salida para cada flujo. Se soportan células ATM estándar de 53 bytes, con 5 bytes de cabecera. Aunque el diseño del conmutador es más complejo si las células son pequeñas, pues las decisiones de planificación se deben tomar más frecuentemente, las garantías en cuanto a prestaciones son más fáciles de proporcionar y mantener si todo el *crossbar* se reconfigura después del tiempo que se tarda en transmitir cada célula. Las aplicaciones pueden trabajar con paquetes de longitud variable; son los controladores de red en los extremos de una conexión los encargados de dividir los paquetes en células, cada una con el identificador de flujo adecuado para ser encaminada, y de volver a ensamblarlas en paquetes. El trabajar con células pequeñas de tamaño fijo es necesario

también por el esquema de organización de los *buffers*, como se verá enseguida.

La interconexión interna en el conmutador se realiza mediante un *crossbar*, pues el algoritmo de planificación supone que no hay bloqueos internos en el conmutador. También podría haberse empleado una red *batcher-banyan*, pero es más complicada y tiene mayor latencia.

La organización de los buffers en Autonet en forma de cola FIFO era uno de sus serios inconvenientes para el funcionamiento de la red a cargas elevadas, por culpa del *HOL blocking*. Por ello, el conmutador de AN2 utiliza en su lugar *buffers* de acceso aleatorio, junto con un algoritmo de planificación más sofisticado que el "primero que llega, primero que se considera"(FCFS) de Autonet. Este algoritmo es el *emparejamiento paralelo iterativo*, y su objetivo es encontrar de manera rápida un emparejamiento entre entradas y salidas libre de conflictos, considerando únicamente aquellos pares que tengan encolada una célula a la espera de transmitirse entre ellos.

El algoritmo de planificación

El algoritmo consiste en repetir los siguientes tres pasos un cierto número de veces (inicialmente, todas las entradas y salidas están desemparejadas):

1. **Fase de petición** Cada entrada no emparejada envía una petición a cada una de las salidas para las que tiene una célula almacenada. 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, lo acepta y se lo notifica a esa salida.

Cada uno de estos tres pasos se lleva a cabo en paralelo en cada puerto en entrada/salida; no hay ningún planificador centralizado.

El protocolo se va repitiendo, almacenando los emparejamientos obtenidos en cada iteración, hasta que ya no se alcancen más. Entonces, se emplea el resultado del algoritmo para configurar el *crossbar* para la siguiente rodaja de tiempo. En [1] se demuestra que esto ocurre en una media de $O(logN)$ iteraciones, para un conmutador de tamaño $N \times N$.

Unidad Funcional	Coste del prototipo	Coste de producción (estimado)
Optoelectrónica	48 %	63 %
Crossbar	4 %	5 %
Lógica y RAM de buffers	21 %	19 %
Lógica de planificación	10 %	3 %
CPU de control y encaminamiento	17 %	10 %

Tabla 2.2: Coste de los componentes del conmutador AN2, como proporción del coste total

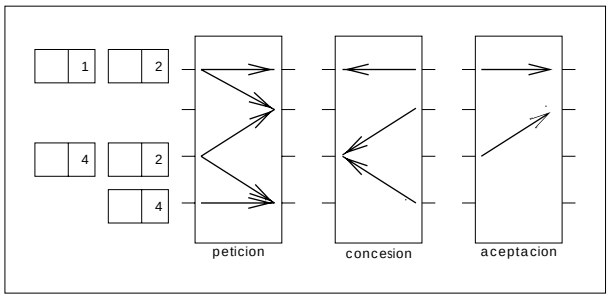


Figura 2.3: Una iteración en el emparejamiento paralelo iterativo

Prestaciones y costes

Las prestaciones que alcanza el algoritmo respecto a las colas FIFO tradicionales se hacen mucho más evidentes a medida que se aumenta la carga de la red. Es en esa situación cuando el bloqueo *HOL* de las colas FIFO empieza a dejarse notar, con lo que la mejora obtenida al emplear el nuevo algoritmo se hace patente. La productividad pico obtenida está muy cercana a la obtenida con colas de salida perfectas, que es el algoritmo más óptimo posible, pero prohibitivo por sus costes de implementación. Esto es similar tanto para una carga uniforme como para cargas del tipo cliente-servidor [1].

El prototipo del conmutador de AN2 (16×16) se llegó a implementar con 4 iteraciones. Tarda en ejecutarse un poco menos que lo que tarda en recibirse una célula (células de 53 bytes, enlaces de 1 Gb/seg, FPGAs). Esto requiere planificar unos 37 millones de células por segundo. La latencia a través del conmutador es de unos 2.2 microsegundos, en ausencia de contención. El conmutador no descarta células y preserva el orden de las mismas.

Las prestaciones obtenidas con este algoritmo de planificación, en cuanto a retraso sufrido por las células en las colas, son aproximadamente un 36 % mayores que con colas FIFO, con unos requerimientos de hardware bastante modestos. En la figura 2.3 se muestra una iteración del algoritmo. El coste de los componentes del conmutador AN2 se reparte según la tabla 2.2.

Se puede observar que en ambos casos, coste del prototipo y coste estimado de producción, el coste del conmutador se halla dominado por el de los componentes optoelectrónicos. Los *buffers* de entrada se organizan en listas, para realizar el acceso aleatorio a las células. Cada flujo tiene su propia cola FIFO de células almacenadas. Un flujo es seleccionable para ser planificado si tiene por lo menos una célula encolada. Se mantiene una lista de flujos seleccionables para cada par de entrada/salida. Si hay por lo menos un flujo seleccionable para un par entrada/salida dado, la entrada requiere la salida durante el emparejamiento paralelo iterativo. Si la petición se concede, uno de los flujos seleccionables se elige para ser planificado de forma *round-robin*.

Aspectos de la implementación

El protocolo de petición, concesión y aceptación se implementa lanzando un hilo entre cada entrada y cada salida. Las señales de petición y concesión pueden codificarse como un solo bit sobre el hilo apropiado. Como una optimización sencilla, no se requiere ninguna comunicación separada en el paso 3 para indicar qué concesiones se aceptan. En su lugar, cuando una entrada acepta la concesión de una salida, simplemente continua pidiendo esa salida en las siguientes iteraciones, pero descarta todas las demás peticiones. Una vez que una salida se concede a una entrada, se sigue concediendo a la misma durante las iteraciones siguientes a menos que la entrada elimine su petición.

Para seleccionar una petición de entre varias de forma aleatoria, se genera un número aleatorio. Para conmutadores de tamaño moderado, esta selección puede realizarse de forma eficiente mediante tablas con valores precalculados. Sin embargo, mediante simulaciones se ha comprobado que el algoritmo de emparejamiento paralelo iterativo es bastante insensible a la técnica utilizada para aproximar la aleatoriedad.

2.1.2 Emparejamiento Paralelo Estadístico

Para garantizar las prestaciones requeridas por el tráfico en tiempo real, no basta con utilizar al algoritmo de emparejamiento paralelo iterativo, sino que debe emplearse una generalización del mismo: el *emparejamiento estadístico* [1]. Este tipo de emparejamiento soporta mejor los cambios frecuentes en la reserva del ancho de banda.

Siguiendo la terminología ATM, se puede distinguir entre dos tipos de tráfico: *CBR* o *constant bit rate*, que requiere la reserva de un ancho de banda determinado, y *VBR*, o *variable bit rate*, también conocido como tráfico de datagramas[2]. Los conmutadores distinguen

[2]En [1], se denomina tráfico VBR a lo que en ATM comprendería las clases VBR, ABR y UBR. Hemos respetado esto al explicar aquí este algoritmo de planificación

las células pertenecientes a un tipo u otro de tráfico mediante el identificador de flujo en la cabecera de la misma. La idea es, puesto que se conoce con antelación el ancho de banda que necesita el tráfico CBR, precalcular una planificación en cada conmutador que lo contemple. Si utilizamos emparejamiento paralelo iterativo, planificará el conmutador rápidamente para cualquier tráfico VBR que llegue al conmutador. Por eso, hay que utilizar una variante suya para soportar el tráfico CBR. Esta variante, el emparejamiento estadístico, intenta compatibilizar ambos tipos de tráfico.

La reserva de ancho de banda se basa en tramas o *frames*, que consisten en un número fijo de ranuras o *slots*. Un *slot* es el tiempo necesario para transmitir una célula. Una petición de ancho de banda se expresa como un cierto número de células por trama. En el conmutador de AN2, el tráfico CBR se maneja haciendo que cada conmutador construya una planificación explícita de pares entrada-salida para cada *slot* de una trama. La asignación de los *slots* puede modificarse de forma dinámica sin interrumpir las prestaciones garantizadas. Así, las células CBR se encaminan a través del conmutador durante los *slots* planificados, mientras que las células VBR se transmiten durante los *slots* que no utilizan las células CBR. Además, las células VBR pueden utilizar un *slot* reservado si no hay ninguna célula del flujo asociado presente en el conmutador. La asignación de los *slots* puede modificarse dinámicamente sin interrumpir las prestaciones garantizadas. Existen conjuntos de *buffers* diferentes para tráfico CBR y VBR. Los *buffers* para tráfico CBR se reservan de manera estática, mientras que los destinados a tráfico VBR están sujetos a control de flujo.

Con emparejamiento paralelo iterativo se tomaban decisiones completamente aleatorias para emparejar las entradas y las salidas, dando la misma probabilidad a todos los flujos que hubiera. Esto, junto con una topología arbitraria, puede llevar a que el ancho de banda no se reparta equitativamente entre los flujos. En el emparejamiento estadístico, sin embargo, se divide el ancho de banda entre los flujos que lo necesitan, de acuerdo a sus reservas. Es decir, 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 células. Con este procedimiento se puede asignar hasta un 72 % del ancho de banda total. El resto se asigna con emparejamiento paralelo iterativo normal. En [1] se demuestra que la latencia extremo a extremo está acotada por $p(2f + l)$, donde p es el número de saltos que debe tomar la célula, f es el tiempo que se tarda en transmitir una trama y l es una cota superior de la latencia del enlace más la sobrecarga por procesar una célula en el conmutador.

El algoritmo de emparejamiento estadístico es similar al de emparejamiento paralelo iterativo, excepto en que no hay fase de petición. Se divide el ancho de banda adjudicable

por enlace en X unidades discretas; $X_{i,j}$ denota el número de unidades reservadas para el tráfico de la entrada i a la salida j. La clave es que disponemos los factores aleatorios de peso en las entradas y salidas de forma que cada entrada recibe hasta X concesiones virtuales, cada una hecha de forma independiente con probabilidad $1/X$. $X_{i,j}$ de las potenciales concesiones virtuales para la entrada i se asocian con la salida j. Si la entrada i escoge de manera aleatoria entre las concesiones virtuales que recibe, se conectará a cada salida con una probabilidad proporcional a su reserva.

Una versión simplificada del algoritmo es la siguiente:

1. Cada salida aleatoriamente escoge una entrada a la cual hacer la concesión; la salida j escoge la entrada i con probabilidad $X_{i,j}/X$, proporcional a la reserva de ancho de banda.

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

   (a) La entrada reinterpreta la 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 la distribución binomial – es decir, la probabilidad de que exactamente k de X eventos independientes ocurran, dado que cada uno ocurre con probabilidad $1/X$.

   (b) Si una entrada recibe concesiones virtuales, escoge una aleatoriamente para aceptarla; la salida correspondiente a la concesión virtual aceptada se empareja con la entrada.

Con una pasada de este algoritmo, se alcanza una probabilidad de conexión entre una entrada i y una salida j de $X_{i,j} \cdot (1 - 1/e)$, o aproximadamente el 63 % de $X_{i,j}/X$. Se puede alcanzar mejor productividad con una segunda iteración del algoritmo, dando la posibilidad de reservar hasta un 72 % del ancho de banda del enlace. Iteraciones adicionales no llevan a mejoras de la productividad significativas.

El emparejamiento estadístico tiene más requerimientos de hardware que el paralelo iterativo para su implementación, aunque el coste no es prohibitivo. Los pasos 1 y 2(a) pueden implementarse como búsquedas en tablas. La tabla se inicializa con el número de entradas para cada resultado proporcional a su probabilidad; un índice aleatorio en la tabla selecciona el resultado. El paso 2(b) es una generalización de la elección aleatoria entre peticiones que necesita el emparejamiento paralelo iterativo; por tanto, se emplean similares técnicas de im-

plementación. Sin embargo, no existe información de ningún prototipo con emparejamiento estadístico.

2.1.3 Emparejamiento Paralelo Iterativo Ponderado

El algoritmo de emparejamiento paralelo estadístico, introducido en la sección anterior, permite una reserva flexible del ancho de banda, distribuyendo éste entre los pares de entrada/salida que compiten entre sí. Sin embargo, tiene algunos inconvenientes. En primer lugar, el proceso de emparejamiento es iniciado por los puertos de salida, que generan una señal de concesión con una probabilidad basada en el ancho de banda reservado por cada puerto de entrada. Puede darse el caso de que se envíe esta concesión a un puerto de entrada cuando no tiene paquetes que transmitir. Esto limita la productividad máxima del emparejamiento estadístico al 72 % de la capacidad del enlace. Además, el algoritmo conlleva el cálculo de varias distribuciones de probabilidad en cada paso, haciendo que sea difícil una implementación hardware.

Una mejora sobre los dos algoritmos de emparejamiento iterativo anteriores es el *Emparejamiento Paralelo Iterativo Ponderado*(*WPIM*) [47], que permite el reparto del ancho de banda de un puerto de salida común entre los puertos de entrada de un conmutador de una manera sencilla. WPIM distribuye el ancho de banda del enlace de salida entre las entradas en base a la reserva efectuada durante la fase de establecimiento de la conexión, y puede garantizar que el tráfico procedente de cada entrada recibe su parte comprometida del ancho de banda de salida. Además, el algoritmo es capaz de proporcionar aislamiento entre flujos que llegan a diferentes puertos de entrada y compiten por el mismo puerto de salida. Si un flujo se comporta de manera anómala (no cumple con lo reservado), esto no afectará a las garantías de ancho de banda o de retardos de las otras entradas. Con WPIM, la cantidad de ancho de banda que puede ser reservada a flujos de tiempo real está limitada únicamente por la eficiencia del algoritmo de emparejamiento en sí, que se halla entre el 85% y el 95%. La sobrecarga hardware es relativamente pequeña, mucho menor que el emparejamiento estadístico.

WPIM resuelve el problema del emparejamiento libre de contención entre entradas y salidas en un conmutador mediante la reserva explícita de una parte del ancho de banda de cada enlace de salida, para cada conjunto de flujos destinados a ellos desde cada enlace de entrada. Si la demanda total de ancho de banda efectuada por los flujos que comparten la misma conexión de entrada/salida está dentro de los límites impuestos por esta reserva, entonces sus garantías de ancho de banda pueden ser satisfechas, independientemente de las demandas de ancho de banda efectuadas por otras conexiones. En otro nivel de planificación

se consideraría el reparto del ancho de banda adjudicado a cada puerto de entrada entre los flujos que lo comparten, como se explica en la página 13.

Para implementar la reserva del ancho de banda, el eje de tiempo se divide en *tramas* (*frames*), con un número fijo de *ranuras*(*slots*) por trama. Cada conexión de entrada/salida reserva su ancho de banda asignándose un cierto número mínimo de ranuras por trama. A esta cantidad se le denomina *crédito* de la conexión. Cada conexión de entrada/salida solicita un crédito correspondiente a la fracción deseada del ancho de banda del enlace de salida. La petición se acepta si los créditos totales solicitados por todas las conexiones que comparten el enlace de salida es menor que la longitud de la trama.

Si $c_{i,j}$ es el crédito reservado para la conexión desde la entrada i a la salida j, y si f es la longitud de la trama, el objetivo del algoritmo de planificación WPIM es garantizar que, en término medio, por lo menos $c_{i,j}$ paquetes pueden ser transmitidos desde la entrada i a la salida j durante cada trama.

Pasamos ahora a ver el funcionamiento básico de WPIM. Asumiremos que cada puerto de entrada del conmutador mantiene una cola distinta para almacenar los paquetes destinados a cada salida del conmutador. Además, cada puerto de salida debe mantener un contador de los paquetes transmitidos desde cada puerto de entrada durante la trama en curso. Cada iteración del algortimo consiste en cuatro pasos:

1. **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. **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. **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. **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.

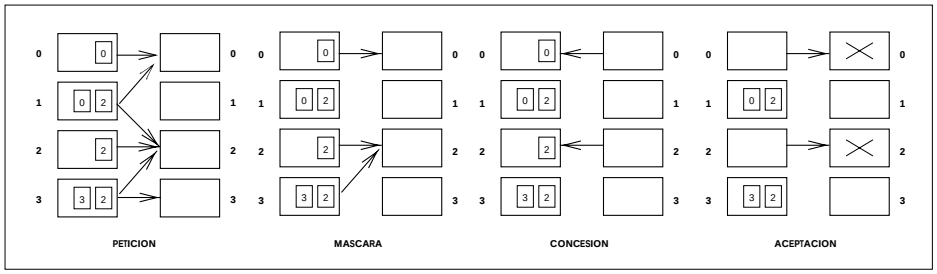


Figura 2.4: Una iteración de la *primera fase* del algoritmo WPIM

Esos puertos de entrada y salida ya están emparejados y pueden eliminarse de sucesivas iteraciones.

Este ciclo se repite un número determinado de veces, o bien hasta que no existen más peticiones no enmascaradas pendientes. La diferencia fundamental entre este algoritmo y el emparejamiento iterativo probabilístico es la adición de la fase de máscara.

En la figura 2.4 se ilustran los cuatro pasos del algoritmo. Para la fase de máscara, se supone que el puerto de entrada 1 ya ha agotado sus créditos para los puertos 0 y 2 para la trama actual. Por tanto, ambas peticiones son enmascaradas. De igual forma, la petición del puerto de entrada 3 al de salida 3 también se enmascara. Durante la fase de concesión, el puerto de salida 2 selecciona aleatoriamente el puerto de entrada 2. Puesto que esta entrada no recibe más concesiones, acepta la del puerto de salida 2.

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, podemos emplear el algoritmo de emparejamiento paralelo normal (página 15 y siguientes) sobre las peticiones no emparejadas para repartir el ancho de banda residual, después de haber satisfecho las reservas de ancho de banda. En el ejemplo anterior, es fácil ver que la petición del puerto de entrada 3 hacia el puerto de salida 3 puede ser atendida también. Las peticiones dirigidas a los puertos de salida 0 y 2 son ignoradas, puesto que ya están emparejados (figura 2.5).

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,

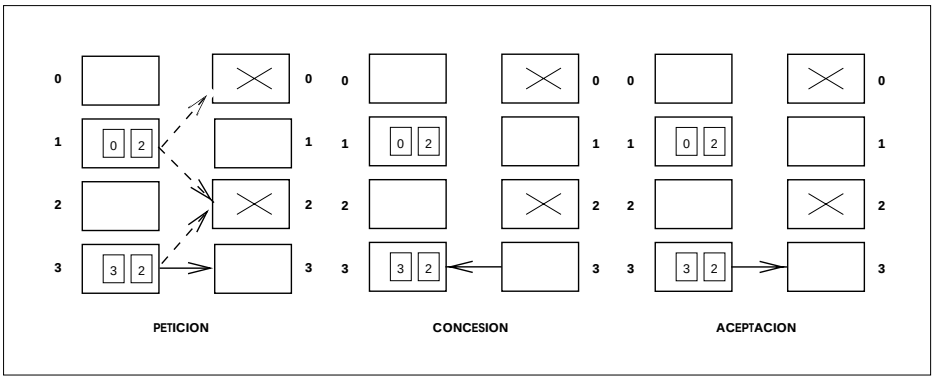


Figura 2.5: Una iteración de la *segunda fase* del algoritmo WPIM

el resto del ancho de banda se reparte por igual entre todas las conexiones que comparten dicho enlace de salida.

Si se asignan $c_{i,j}$ créditos a la conexión entre el puerto de entrada i y el de salida j, y el tamaño de trama es f, la fracción del ancho de banda de salida reservada para dicha conexión viene dada por

$$b_{i,j} \geq \frac{c_{i,j}}{f} \tag{2.1}$$

para cada i, j. Además, si asumimos que se reserva explícitamente una parte del ancho de banda para cada una de las conexiones de entrada/salida, entonces

$$\sum_{i=1}^{N} = \frac{c_{i,j}}{f} \tag{2.2}$$

De las ecuaciones 2.1 y 2.2 se concluye que, bajo condiciones de fuerte carga,

$$b_{i,j} = \frac{c_{i,j}}{f} \tag{2.3}$$

Si parte del ancho de banda de salida no se reserva de forma explícita, cada entrada recibirá la cantidad de ancho de banda que reservó, más una parte igual del ancho de banda que quede. Además, si una conexión entrada/salida incrementa su caudal de tráfico, las prestaciones de todas las demás se verán degradadas por igual, pero todavía recibirán por lo menos el ancho de banda que reservaron.

El funcionamiento de WPIM puede verse como dos aplicaciones consecutivas del algoritmo de emparejamiento iterativo. Para este último, el número máximo de iteraciones necesarias para conseguir un emparejamiento maximal es $\log_2 N + \frac{4}{3}$, partiendo de N^2 peticiones sin resolver. En WPIM, durante su primera fase, se aplica emparejamiento iterativo sobre

las peticiones con créditos pendientes. Durante la segunda fase, se aplica de nuevo emparejamiento iterativo sobre las peticiones que quedan sin resolver de la primera fase, y sobre las que fueron enmascaradas. La peor distribución se da cuando la mitad de las peticiones fueron enmascaradas en la primera fase. En este caso, la primera fase del algoritmo empezará con un máximo de $N^2/2$ peticiones, y convergerá, en media, en menos de $\log_2 N + \frac{7}{6}$. En la segunda fase, las $N^2/2$ peticiones enmascaradas se activarán y el algoritmo convergerá también en menos de $\log_2 N + \frac{7}{6}$ iteraciones, en media. Por tanto, el número medio de iteraciones será menor de $2\log_2 N + \frac{7}{3}$.

Si se combinan las dos fases del algoritmo en una sola, se puede mejorar el número medio de iteraciones. Esto se puede hacer si permitimos que cada puerto de salida borre todos sus bits de máscara cuando todas sus peticiones de entrada están enmascaradas, y el puerto de salida sigue sin emparejar. Esto no afecta a la reserva del ancho de banda. Las máscaras se ignoran únicamente cuando todos los puertos con créditos pendientes no tienen paquetes destinados al puerto de salida, o bien, cuando todas esas entradas con créditos pendientes, han sido emparejadas con otros puertos de salida. De esta forma, la cota superior del número medio de iteraciones que dan lugar a un emparejamiento maximal se reduce a $\log_2 N + \frac{4}{3}$, igual que el emparejamiento iterativo. En [47] se demuestra esta afirmación. También en [47], se ha comprobado mediante simulación que, en la práctica, el número de iteraciones necesario es $\log_2 N$ para un conmutador de N puertos.

En cuanto a la implementación de WPIM en hardware, decir que la arquitectura es básicamente la misma que la empleada para emparejamiento iterativo. El planificador para un conmutador $N \times N$ consiste en N módulos de entrada y N módulos de salida. Las señales de petición y concesión se intercambian entre estos dos módulos. Los paquetes entrantes se almacenan en colas de acceso aleatorio en cada entrada, y se mantiene una cola distinta para los paquetes destinados a cada puerto de salida. El algoritmo WPIM se lleva a cabo entre los módulos de entrada y de salida. El mecanismo de reserva y adjudicación del ancho de banda se implementa mediante registros de máscara, contadores de créditos, y registros de concesiones. Más detalles aparecen en [47].

2.1.4 Otras propuestas

Los algoritmos anteriores buscan un emparejamiento máximo en tamaño, es decir, tienen como objetivo conseguir el máximo número de emparejamientos entre entradas y salidas. En [35] y [36] se discuten una serie de algoritmos para conseguir emparejamientos máximos en peso. La idea general consiste en asignar pesos a los arcos del grafo bipartito que modela el problema, y calcular un emparejamiento que maximice el peso total de los arcos seleccionados.

Dependiendo del criterio seguido para asignar esos pesos, se obtienen distintos algoritmos, con diferentes características. Todos ellos emplean la técnica de *Virtual Output Queuing*, para evitar el bloqueo de la cabeza de línea. 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.

A continuación, comentaremos brevemente estos algoritmos.

Longest Queue First (LQF)

El peso $w_{i,j}$ asignado a cada petición que implica al puerto de entrada i y al de salida j consiste en este caso en la ocupación de su cola correspondiente, 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.

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 2.6.

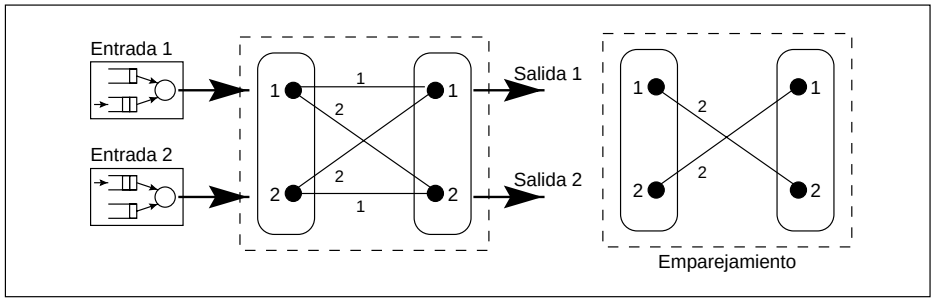


Figura 2.6: Ejemplo de inanición con LQF para un conmutador de 2×2

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 células que hay en cada puerto de entrada destinadas 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.

Oldest Cell First (OCF)

OCF es una alternativa a LQF que soluciona el problema de la inanición. Esto lo consigue asignando como pesos para efectuar el emparejamiento los tiempos de espera de los paquetes almacenados en la cabeza de cada cola. De esta manera, en cada ciclo de transmisión, las células no servidas se hacen más viejas, y así hasta que llegan a ser lo suficientemente viejas como para ser transmitidas.

En [35] se prueba que OCF es capaz de obtener un 100 % de productividad para cualquier patrón de procesos de llegadas independientes.

Longest Port First (LPF)

El principal problema con LQF y OCF es su dificultad de implementación en hardware a alta velocidad. Una primera razón es que el mejor algoritmo conocido para calcular el emparejamiento según LQF o OCF tiene una complejidad temporal de $O(N^3 \log N)$. En segundo lugar, la implementación necesitaría un gran número de comparadores multibit para efectuar muchas comparaciones de peso en paralelo. Los intentos de implementación se han visto limitados por el diseño de un planificador en un sólo chip que tenga comparadores lo suficientemente rápidos, que pueda soportar un número suficiente de los mismos, y que pueda interconectarlos con un patrón lo bastante rico.

Ante esta perspectiva, los mismos autores de LQF y OCF proponen LPF con la intención de combinar los beneficios de un emparejamiento de tamaño máximo con los de un algoritmo de máximo peso, 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:

$$w_{i,j} = \begin{cases} R_i + C_j, & L_{i,j} > 0 \\ 0, & \text{en otro caso} \end{cases} \tag{2.4}$$

donde R_i es la ocupación de entrada (número total de células que esperan en la entrada i), C_j es la ocupación de salida (número total de células, distribuidas entre todas las entradas, esperando a ser transmitidas 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:

$$R_i = \sum_{j}^{N} L_{i,j} \quad ; \quad C_j = \sum_{i}^{N} L_{i,j} \tag{2.5}$$

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

En [36] 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 $O(\log N^2.5)$. Además, se han eliminado del camino crítico todos los comparadores que limitaban las prestaciones de LQF. Existen aproximaciones heurísticas para el cálculo de emparejamientos de tamaño máximo, una de las cuales es PIM (sección 2.1.1).

LPF une la alta productividad instantánea que proporciona aun algoritmo de tamaño máximo, junto con el pequeño múmero de desbordamientos de colas que proporciona un algoritmo de peso máximo, incluso ante tráfico no uniforme.

Los autores proponen una implementación en hardware en dos fases: una primera de ordenación donde se realiza todo el tratamiento de pesos, y una segunda que calcula el emparejamiento de tamaño máximo. Se asegura que es posible implementar LPF en un conmutador de 32×32 puertos, con lógica CMOS de 0.25 micras, obteniendo una latencia de unos 20 ns.

2.2 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 retransmitir, 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. Como aparece en la tabla 2.1, de la página 12, 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. A continuación, veremos ejemplos de ambos tipos.

2.2.1 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, veremos superficialmente algunos algoritmos de planificación basados en prioridades. En primer lugar, repasaremos los orientados al retardo; después, los orientados al ancho de banda. Para cada grupo, veremos en detalle un algoritmo: DEDD y VC, respectivamente.

Esquemas orientados al retardo

Un esquema 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** [49]. 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 [30], 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.

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)** [23]. 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 contínuos 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)**, que veremos con detalle en la siguiente

sección.

Delay Earliest-Due-Date

En [14], Ferrari y Verma proponen un algoritmo de planificación pensado 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 conexi'on 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.

Se 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 tambien 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.

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 comparacion 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)** [50] 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 [37] 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 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)** [11] [3]. 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 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 [39] se introduce un esquema muy similar a FQ, denominado ***Packet-by-Packet Generalized Processor Sharing* (PGPS)** [4]. 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* [7], 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

[3] FQ también se conoce como *Generalized Processor Sharing* (GPS).
[4] PGPS también se conoce como *Weighted Fair Queuing*(WFQ).

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 [51], que veremos en detalle en el siguiente apartado, como ejemplo de algoritmo basado en prioridad y orientado al ancho de banda.

Se han desarrollado otras 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)** [34] 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 [21], 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)** [48, 46], 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; por tanto, también es muy complejo de implementar.

Virtual Clock

En [51] Zhang propone *Virtual Clock* (VC) como algoritmo de planificación apropiado para redes de alta velocidad. 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.

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 variablidad 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 transmitindo 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 completemente 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 2.7 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

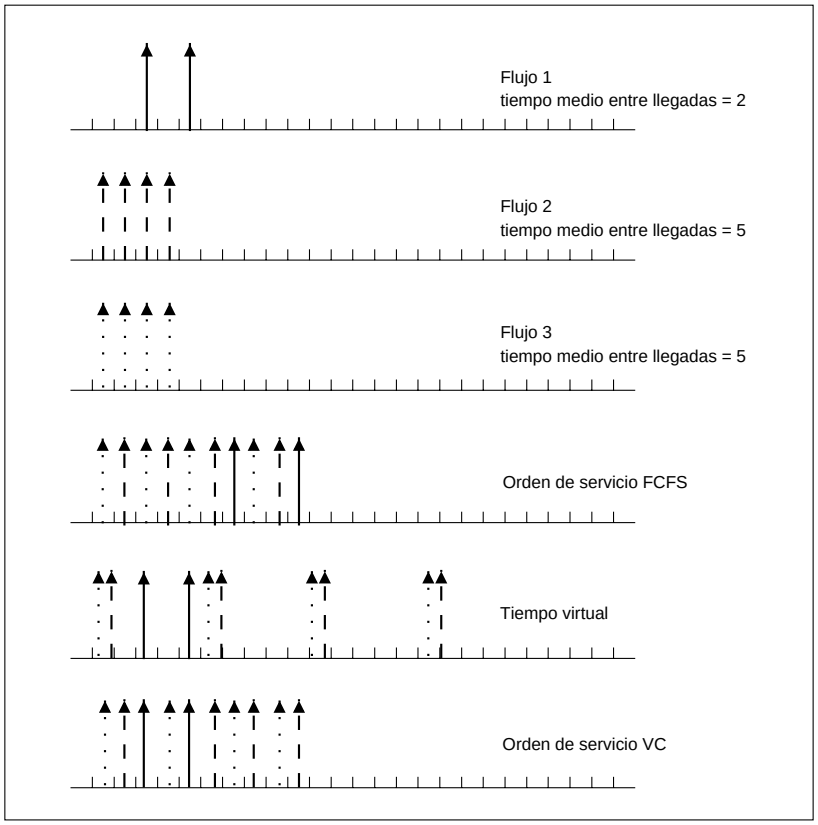


Figura 2.7: Ejemplo de funcionamiento de VC, frente a FCFS

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, 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)** [3] 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 crecerán de forma desmesurada.

2.2.2 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.

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 [26], 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)**, y lo veremos en detalle como ejemplo de este tipo de algoritmos.

Otra disciplina de servicio flexible y simple es ***Deficit Round Robin* (DRR)** [43], 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 un 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 su 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 [5]. DRR puede ser implementado a un bajo coste porque las conexiones activas se atienden en estricto orden *round-robin*.

[5] 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 [31] se ha empleado esta propiedad para ofrecer garantías de ancho de banda a tráfico *best-effort* en redes *wormhole*.

***Round-Robin* Ponderado**

El conmutador propuesto en [26] es un conmutador ATM, es decir, considera células de tamaño fijo (53 bytes). Su diseño se basa en los siguientes mecanismos:

1. Soporte de prioridades : Hay cuatro clases de prioridad para los CV; las células pertenecientes a una clase tienen absoluta prioridad sobre las de clases inferiores.

2. Distribución de ancho de banda dentro de cada clase de prioridades : En cada clase, la contención entre células por un enlace, se resuelve mediante WRR, donde cada VC tiene su propio factor de peso, y por tanto, comparte la productividad del enlace en proporción a este peso.

3. Esquema de control de flujo entre conmutadores, que opera sobre los CV. Un conmutador no envía ninguna célula perteneciente a un determinado CV hasta que recibe un *token* del conmutador destino. Así se asegura que no se transmite ninguna célula a menos que exista un *buffer* para ella.

4. Soporte de diferentes mecanismos de almacenamiento: un subconjunto de CV tienen *buffers* dedicados para cada uno de ellos, mientras que el resto de CVs comparten un conjunto de *buffers* comunes.

El algoritmo de planificación *round-robin ponderado* (WRR) distribuye el ancho de banda disponible en un enlace entre los circuitos virtuales 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 información de cada uno de ellos 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 2.8, donde los cuadrados simbolizan CVs con paquetes destinados a un puerto de salida común.

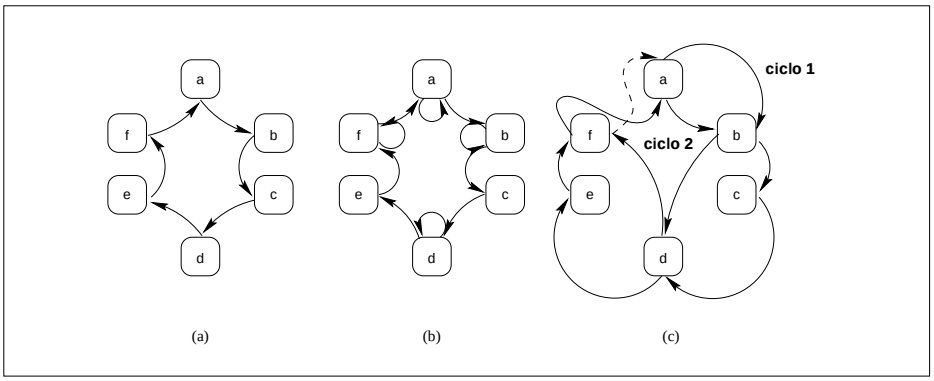


Figura 2.8: (a) Round-robin. (b) Round-robin ponderado (c) Orden de las visitas

En la parte (a) de la figura 2.8 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 [26] 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.

La implementación de este método se lleva a cabo en hardware, guardando toda la información de estado en una *memoria de exploración*. Esta memoria está organizada de manera similar a una memoria asociativa. Para cada CV, se almacena:

Bit de `válido` Indica si este CV está abierto actualmente.

Enlace fuente e identificador del CV Necesarios para saber a quién hay que enviar la información de control de flujo.

Clase de prioridad de este CV Necesaria para escoger el siguiente CV que se va a servir.

Bit de `ready` Representa la información de control de flujo y disponibilidad de células. Es verdad siempre que hay al menos una célula almacenada para el CV correspondiente, y dicho CV no está detenido por causa del control de flujo ejercido por el conmutador destino.

Bit de `no visitado todavia` Se pone a 1 cuando empieza cada ciclo de planificación. Durante el ciclo, después de que el CV ha sido visitado, este bit se pone a cero.

Peso de la frecuencia de servicio Especifica las etiquetas de los subciclos durante los cuales el CV correspondiente debe ser visitado. Si las etiquetas de los subciclos están en el rango 1 a N, un CV visitado durante w de estos N subciclos recibe una productividad de tamaño w/N.

El método para escoger las w etiquetas de subciclo de forma más o menos uniformemente espaciada (y obtener así el efecto ilustrado en la figura 2.8(c)) consiste en escribir las N etiquetas como números binarios, e identificar el dígito de valor 1 situado más a la derecha en esta notación, es decir, el 1 menos significativo (ver la tabla 2.3). Hay $(N+1)/2$ etiquetas con ese 1 en la posición de bit menos significativa (posición 0), $(N+1)/4$ etiquetas donde está en la posición 1, etc. Estos $\log_2(N+1)$ conjuntos de etiquetas contienen cada uno potencias decrecientes de 2, son disjuntos y están distribuidos de forma bastante equilibrada sobre el eje de tiempos (vertical en la tabla 2.3).

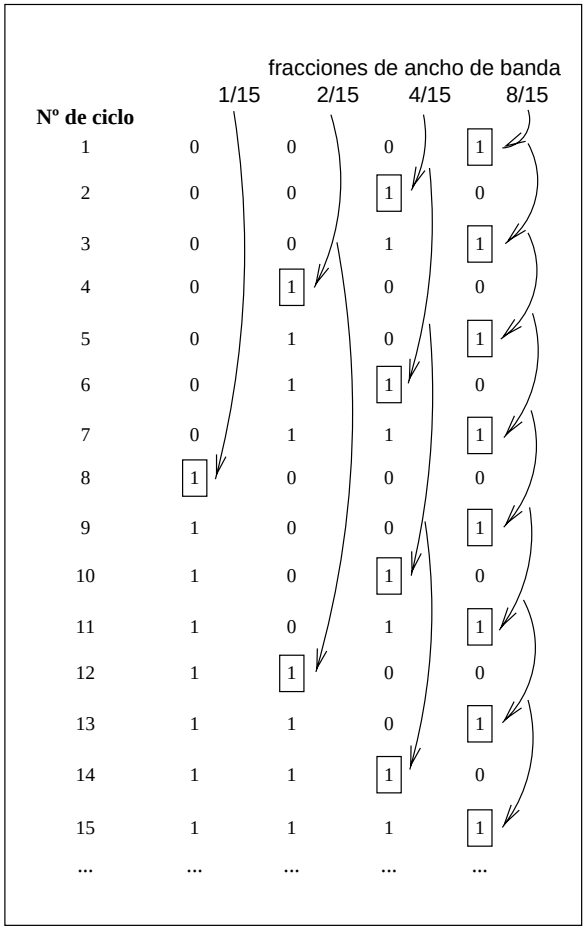
El valor de las etiquetas w se escoge así: w en su representación binaria, es una suma de potencias de dos; se escogen todas las etiquetas del conjunto anterior $(1 \ldots N)$, cuyas cardinalidades sean esas mismas potencias de dos. Por ejemplo, en la tabla 2.3, si w=5, es decir, si queremos servir un CV a una tasa relativa de 5/15=(4/15) + (1/15), entonces visitaremos ese CV durante cada subciclo cuya etiqueta tenga su 1 más a la derecha, en las posiciones de bit 1 o 3 (subciclos 2, 6, 8, 10 y 14).

Este algoritmo se implementa mediante un *contador*, que almacena la etiqueta del subciclo actual, y un mecanismo para hacer cumplir la prioridad, denominado *priority enforcer*. Éste hace circular a través de la parte de pesos de la memoria de exploración, utilizando un bus, un sólo 1 en la posición del 1menos significativo del contador de ciclos. Aquellos CVs cuyos pesos tienen un 1 en la posición donde el mecanismo anterior envía su 1, se convierten en candidatos a ser visitados durante el subciclo actual, es decir, son los que componen el conjunto sobre el que se aplicará *round-robin* en el subciclo actual.

La realización del chip que implementa el conmutador es posible con tecnología CMOS, para tamaños de entre 4 y 10 enlaces, capacidad de enlace entre 0.5 y 1 Gb/s y alrededor de 1000 circuitos virtuales abiertos por cada conmutador.

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



Nº de ciclo	1/15	2/15	4/15	8/15
1	0	0	0	**1**
2	0	0	**1**	0
3	0	0	1	**1**
4	0	**1**	0	0
5	0	1	0	**1**
6	0	1	**1**	0
7	0	1	1	**1**
8	**1**	0	0	0
9	1	0	0	**1**
10	1	0	**1**	0
11	1	0	1	**1**
12	1	**1**	0	0
13	1	1	0	**1**
14	1	1	**1**	0
15	1	1	1	**1**
...	...	...	...	...

Tabla 2.3: Identificación de ciclos y componentes de frecuencia binarios en WRR

justifican:

1. La regulación del tráfico a la entrada de la red, como por ejemplo mediante un algoritmo *leaky bucket* [7], 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)** [20, 19] 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*. Veremos con mayor detalle

este algoritmo en el siguiente apartado.

Stop-and-Go

En [20] 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. 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 *discorcandia 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 2.9 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

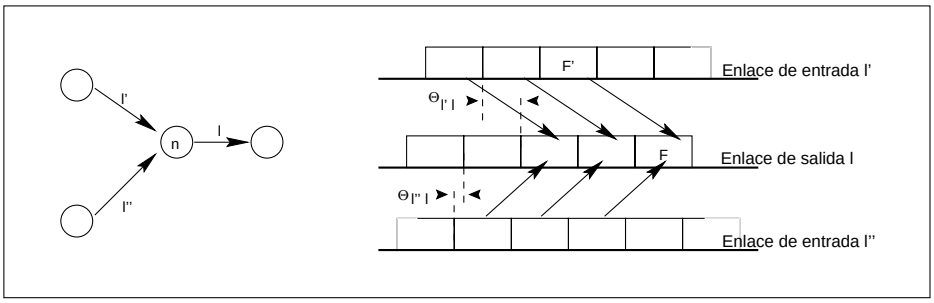


Figura 2.9: Ilustración de la disciplina *Stop-and-Go*

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 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 [19], 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 2.10), aunque también es posible emplear tamaños de trama arbitrarios.

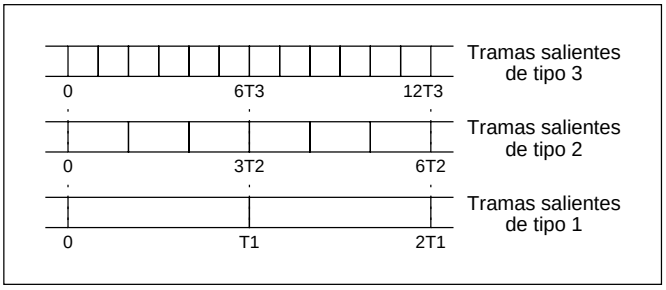


Figura 2.10: Tramas salientes en un nodo para $G = 3$, $T_1 = 3T_2$, y $T_2 = 2T_3$

El algoritmo básico de SG se modifica en lo siguiente:

Política de admisión Cada conexión se establece en asociación con un tamaño de trama T_g, denominándose *conexión de tipo g*. El valor de g se indicará en la cabecera de cada paquete de la conexión. El criterio de admisión de paquetes considera ahora el tiempo de trama T_g: se admiten un máximo de $(1/S) \times r \times T_g$ paquetes de tamaño S en la red, correspondientes a una conexión con tasa de transmisión r, durante cada intervalo de duranción T_g.

Algoritmo de planificación Las dos reglas A y B anteriores pasan a ser tres reglas, para considerar los tipos de tramas:

Regla A Sea un nodo n, un enlace de salida l, uno de entrada l', y un paquete de tipo g 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' (en este caso, las tramas adyacentes serán del mismo tipo g). Entonces, el paquete no llega a ser elegible hasta el inicio de la trama F.

Regla B Cualquier paquete elegible de tipo g, tiene prioridad no-preemptiva sobre los paquetes elegibles de tipo $g' < g$.

Regla C (permanece idéntica) Un enlace no debe permanecer desocupado mientras queden paquetes elegibles en la cola.

Se siguen manteniendo las cotas en los retados y en el espacio de almacenamiento necesario para evitar desbordamientos, aunque en este caso serán diferentes por cada tipo de conexión. Esto es debido a que todas las cotas involucran al tamaño de trama T_g. 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 (por la regla B).

Tanto la estrategia propuesta en [20] como la multitrama de [19], se pueden implementar fácilmente como modificaciones a una estructura de colas FIFO.

Por otro lado, con el algoritmo tal y como se plantea en [20], no existe flexibilidad para soportar eficientemente varias clases de servicio, 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 [18] se propone otra modificación que intenta dar soporte a tráfico VBR, mejorando la utilización de la red, e intentando alcanzar un equilibrio entre ganacia 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)** [24]. 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*).

2.2.3 Redes con 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 [32] se introduce el concepto de *ráfaga*, como la sequencia de paquetes que encapsula una unidad de datos de una aplicación. En este modelo, un flujo de tráfico sigue siendo una sequencia 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*).

Definiciones básicas

Consideramos un flujo que atraviesa un camino de $K+2$ nodos, a través de una red. El nodo 0 es la fuente, el nodo $K+1$ es el destino, y los nodos $\{1 \ldots K\}$ son conmutadores intermedios. Asumimos que los paquetes pertenecientes al mismo flujo se sirven en orden FIFO en cada conmutador.

El *retardo de una ráfaga* en el flujo se define como el tiempo transcurrido desde que el primer paquete de la ráfaga llega a la entrada de la red (nodo 1), hasta que el último paquete de la ráfaga llega a su destino (nodo $K+1$). La *tasa de pérdidas* del flujo se define como la fracción de ráfagas que componen el flujo, que no llegan a su destino.

Se asume que un flujo que requiere una cierta QoS por parte de la red cumplirá una *Especificación de Flujo* a la entrada de la red. Las garantías son condicionales en el sentido de que la red no está obligada a proporcionar garantías a un flujo que no cumple con su Especificación.

La *tasa de una ráfaga* se define como el tamaño de la misma (medido en bits o en paquetes) dividido por la duración de la ráfaga en segundos, que viene especificada por la fuente. La *tasa pico* de un flujo es la mayor tasa de entre las ráfagas que componen el flujo.

En una red de servicios integrados un canal transporta tráfico perteneciente a distintas clases de servicio. Se dice que un canal está *sobrereservado para una clase de servicio* (*overbooked*) si la suma de las tasas pico de todos los flujos de esa clase de servicio que circulan por el canal, excede la parte de ancho de banda del canal reservada para dicha clase de servicio.

Especificación de los Flujos

Para poder proporcionar QoS al nivel de ráfagas, hace falta una Especificación del Flujo basada en ráfagas. De esta forma, se podrá diseñar la red de manera adecuada. Hay dos cuestiones a considerar: (i) los tamaños de las ráfagas varían mucho (por ejemplo, en sequencias de vídeo MPEG) y (ii) para que una red pueda proporcionar una cota superior para el retardo de una ráfaga, los paquetes que la componen deben ser inyectados en la red dentro de un cierto tiempo acotado.

La primera consideración sugiere que las redes deben diseñarse de forma que se utilice la información relativa al tamaño de la ráfaga, o bien, el ancho de banda requerido sobre un cierto intervalo de tiempo, especificados ambos por la propia ráfaga. La segunda consideración sugiere el empleo de una restricción sobre el *jitter* entre los paquetes de una ráfaga, en la especificación del flujo.

Consideramos la siguiente notación para especificar un flujo particular f:

(m, l)	el l-ésimo paquete de la m-ésima ráfaga del flujo f
$A(m, l)$	tiempo de llegada del paquete (m, l)
b_m	tamaño de la ráfaga m, en paquetes
δ_m	duración máxima de la ráfaga m, en segundos
λ_m	$\geq b_m/\delta_m$, tasa reservada para la ráfaga m (paquetes/seg.)

Los valores de b_m y δ_m para la ráfaga m deben ser proporcionados por la fuente del flujo. Un flujo conforme con la Especificación de Flujo, cumple las siguientes tres restricciones:

- El primer y último paquete de una ráfaga están identificados de forma única. El primer paquete de la ráfaga m contiene la información de su tasa λ_m.
- Los paquetes de una ráfaga m satisfacen una *restricción de jitter*:

$$0 \leq A(m, l) - A(m, 1) \leq \frac{l-1}{\lambda_m}, \ para\ l = 1, 2, \ldots, b_m \tag{2.6}$$

- Las ráfagas de un flujo satisfacen una *restricción de separación*:

$$A(m+1, 1) - A(m, 1) \geq \frac{b_m}{\lambda_m}, \ para\ m \geq 1 \tag{2.7}$$

La restricción 2.6 especifica una cota para la separación entre los paquetes de una ráfaga, necesaria para poder garantizar cotas para la latencia de la misma. La restricción 2.7 especifica una separación mínima entre las llegadas de dos ráfagas consecutivas pertenecientes a un

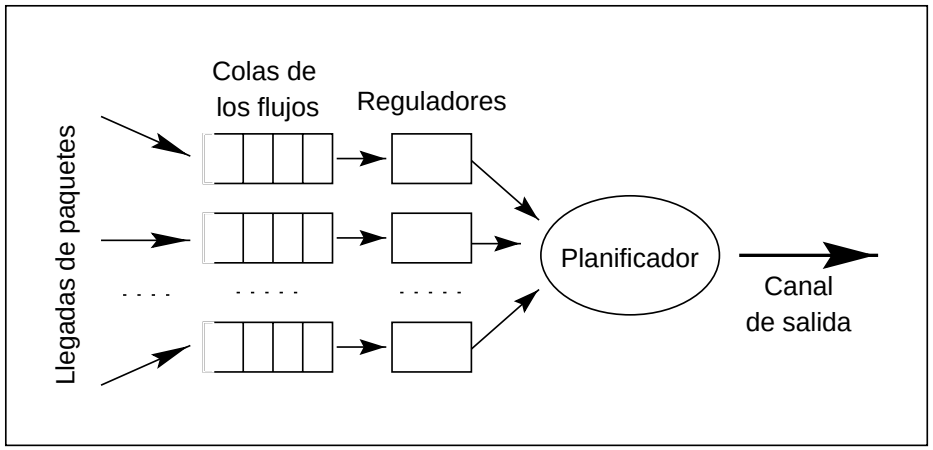


Figura 2.11: Arquitectura de un canal para planificación basada en ráfagas

mismo flujo. Esto es una forma de control sobre las fuentes. Mas aún, esta restricción asegura que, dentro de un conmutador, cada flujo activo tendrá como máximo una sola ráfaga activa en cualquier momento. De esta forma, es sencillo que un conmutador compruebe que la suma de tasas reservadas por los flujos activos no sobrepasa la capacidad de un enlace de salida.

Un flujo que requiere garantías de QoS para sus ráfagas, y cumple la Especificación de Flujo a la entrada de la red, se denomina *flujo garantizado*.

A la entrada de la red, existen *reguladores de la fuente*, que aseguran que los paquetes del flujo cumplen con la especificación. Puesto que la secuencia de paquetes que abandona el nodo 1 puede satisfacer o no los requisitos de la Especificación, y lo mismo sucede a la salida de cualquier nodo intermedio en la red, se necesitan *reguladores de flujo* para asegurar que ambas restricciones se cumplen, en el momento en que los paquetes sean elegibles en el siguiente conmutador. Estos reguladores simplemente retrasan el primer paquete de un ráfaga el tiempo necesario para cumplir con la Especificación. De esta forma, la ráfaga entera se retrasa, debido al orden FIFO de las colas de cada flujo. La arquitectura de un canal que utiliza planificación en ráfagas se muestra en la figura 2.11. Se emplea una organización del conmutador con colas situadas a la salida. Para cada canal, existen colas diferentes para los distintos flujos. Cada flujo garantizado dispone de sus propios *buffers*. Hay un regulador para la cola de cada flujo, y un planificador para cada canal.

Existe otro tipo de tráfico que no requiere garantías de retardo sobre sus ráfagas, como es el tráfico ABR. Un flujo ABR no cumple la Especificación de Flujo. El regulador de flujo para un flujo ABR lo único que ha de hacer es calcular los valores del reloj virtual.

Política de admisión

Para poder ofrecer garantías de QoS a los flujos de datos, es necesario algún mecanismo de control de admisión. Se pueden emplear dos políticas diferentes, que proporcionan dos clases de servicio distintas:

Servicio VBR en tiempo real sin pérdidas Cada flujo de esta clase se admite en la fase de establecimiento de la conexión en base a su tasa pico. Para esta clase no existe *overbooking*. Por tanto, cada ráfaga del flujo será admitida en todos los conmutadores que componen el camino.

Servicio VBR en tiempo real con una tasa de pérdidas especificada Para aplicaciones que pueden tolerar una pequeña tasa de pérdida de ráfagas, sus flujos se admiten en la fase de establecimiento de la conexión en base a sus tasas de pico y sostenida. Se permite el *overbooking* de los canales. La tasa de pérdida de ráfagas de esta clase se calcula y se negocia durante la fase de establecimiento de la conexión.

También se permite el reparto de ancho de banda basado en la tasa de ráfagas, de manera que la tasa reservada para un flujo VBR es variable, esto es, cambia de una ráfaga a la siguiente. De manera más específica, en cada canal, una tasa reservada no se adjudica a un flujo hasta que llega el primer paquete de la ráfaga, y la tasa se libera cuando sale el último paquete de la ráfaga. Este tipo de reparto tiene la ventaja de adjudicar una tasa que es exactamente la que necesita el flujo VBR a lo largo del tiempo. En particular, para cada flujo de tiempo real admitido al establecer la conexión en base a su tasa pico, cualquier diferencia entre esta tasa y la tasa reservada actual no se le adjudica, y permanece disponible para los flujos de otra clase de servicio.

En el caso de flujos admitidos en una clase de servicio sujeta a *overbooking*, cada ráfaga perteneciente a dicho flujo está sujeta a un control de admisión cuando su primer paquete llega a un conmutador. Debido al *overbooking*, el canal de salida del flujo puede que no disponga de suficiente ancho de banda para dar cabida a la ráfaga. La ráfaga solo se admite si la tasa de reserva para la ráfaga no excede la capacidad no asignada del canal.

Por tanto, las decisiones relativas al control de admisión se llevan a cabo por los conmutadores en dos niveles: una vez para el flujo al establecer la conexión, y una vez para cada ráfaga del flujo, cuando llega su primer paquete. Está permitido el *overbooking* al admitir los flujos. Sin embargo, cuando se admite una ráfaga individual, la tasa adjudicada total no debe exceder la capacidad del canal. Además, los paquetes que componen una ráfaga se admiten o se descartan como un todo. En el caso de servicio sin pérdidas, las ráfagas no se

descartan nunca, mientras que en el otro caso, es posible el descarte de ráfagas debido al *overbooking*.

Política de planificación

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

En *Virtual Clock*, el paquete $i+1$ del flujo f se marca con el siguiente valor de reloj virtual:

$$P(i+1) = max\{P(i), A(i+1)\} + \frac{1}{\lambda(f)} \tag{2.8}$$

donde $P(i)$ es el valor de reloj virtual del paquete i, $A(i+1)$ es el tiempo de llegada del paquete $i+1$, y $\lambda(f)$ denota la tasa reservada para el flujo f en paquetes/segundo.

Los paquetes del flujo f transmitidos según la disciplina VC, obtienen la siguiente cota de retardo :

$$L(i+1) \leq P(i+1) + \frac{1}{\gamma} \tag{2.9}$$

donde $L(i+1)$ es el tiempo de salida del paquete $i+1$, y γ es la tasa de transmisión en paquetes/segundo.

Para adaptar el algoritmo VC al esquema basado en ráfagas necesitamos la siguiente notación:

$P(m,l)$	valor del reloj virtual para el paquete (m,l)
$L(m,l)$	tiempo de salida del conmutador del paquete (m,l)

Como consecuencia de la restricción 2.6 de la especificación de flujo, tenemos que el valor del reloj virtual del paquete l-ésimo, de la ráfaga m es:

$$P(m,l) = P(m,1) + \frac{l-1}{\lambda_m} \tag{2.10}$$

y la cota del retardo es, por tanto:

$$L(m,l) \leq A(m,1) + \frac{l}{\lambda_m} + \frac{1}{\gamma} \tag{2.11}$$

La desigualdad 2.11 proporciona una cota del retardo de la ráfaga que depende de su longitud m. Gracias a la expresión 2.10, el valor de reloj virtual puede calcularse de forma más eficiente, simplemente incrementándolo en una cantidad que se calcula una vez por ráfaga. Además, sólo es necesario almacenar un valor de reloj virtual por flujo, en lugar de uno por paquete.

Una optimización propuesta en [33] 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. Esto se debe a que los tamaños de los grupos se escogen de manera que el cociente g_m/λ_m es aproximadamente el mismo para todas las ráfagas pertenecientes al mismo flujo. De esta manera, 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.

2.3 Planificadores para SAN y Redes de Multicomputadores

Una SAN (*System Area Network*) es una forma de combinar elementos hardware estándar para lograr sistemas verdaderamente escalables que proporcionan prestaciones y productividad superiores, de manera fiable. Al igual que los sistemas tradicionales basados en bus, una SAN hace posible que un grupo de recursos computacionales sea compartido para poder ejecutar aplicaciones con altas prestaciones. Por el contrario, una SAN agrupa los componentes más importantes del sistema como elementos independientes. Como resultado, cualquier elemento (sea procesador, disco, o E/S) puede relacionarse con cualquier otro sin la intervención del procesador.

Esta conectividad cualquiera-a-cualquiera es crítica para utilizar aplicaciones intensivas en datos, como multimedia, Internet e intranet. Los recursos agrupados dentro de la SAN no necesitan estar dentro de la caja del computador, sino que pueden hallarse distribuidos por la habitación, el edificio e incluso el campus. Además de esta conectividad, una SAN proporciona el nivel de fiabilidad para datos y aplicaciones esperado en un sistema altamente integrado.

El resultado de la combinación de estas características es que una SAN proporciona la capacidad de soportar tanto el creciente conjunto de aplicaciones intensivas en comunicaciones como las tradicionales aplicaciones de computación intensiva, utilizando para ello productos estándar de la industria.

ServerNet [55] es una SAN creada por Tandem Computers con la idea de interconectar entre sí directamente el(los) procesador(es) y dispositivos de E/S que componen el computador, soportando cualquier topología.

La motivación subyacente es que hoy en día existen muchas aplicaciones que requieren la transferencia de grandes cantidades de datos sin apenas necesitar procesamiento alguno. Servicios como la educación interactiva, prevención en línea del fraude, servicios comerciales en línea y por Internet y telecompra, son ejemplos de aplicaciones que incorporan grandes volúmenes de datos esencialmente sin procesar (por ejemplo, imágenes y voz), además de grandes cantidades de datos procesados (tales como información de marketing y demográfica).

Las estaciones de trabajo están preparadas para manejar este volumen de información; sin embargo, el servidor debe evolucionar para ser capaz de repartir, en lugar de procesar, la información de muchas estaciones de trabajo. Con el sistema tradicional de interconexión de dispositivos, todos los datos deben pasar obligatoriamente por el procesador, independientemente de que tengan que ser objeto o no de algún procesamiento. Esto supone un severo

hándicap para las prestaciones de la máquina, pues penaliza cada transferencia de datos con el paso innecesario por el procesador, a la vez que convierte a éste en cuello de botella del sistema.

ServerNet es una tecnología de interconexión para clusters de altas prestaciones y alta fiabilidad. Al haber sido concebida como una SAN (la primera disponible en el mercado, según sus fabricantes), ServerNet proporciona una arquitectura del sistema que extiende el concepto del mismo más allá de los límites de la caja del computador, pues dotando a cada periférico del interfaz ServerNet apropiado se puede escalar el sistema según se necesite, bien en términos de procesamiento, bien de almacenamiento, etc., de acuerdo a las necesidades de las aplicaciones, tanto tradicionales (de procesamiento intensivo) como nuevas (intensivas en comunicaciones).

ServerNet se compone de encaminadores de 6 puertos, que emplean *wormhole* como técnica de conmutación, y enlaces punto-a-punto de 50 Mbytes/seg *full-duplex*. Además de un *crossbar* como *fabric* de interconexión, los encaminadores tienen *buffers* FIFO en sus entradas, lógica para arbitraje y control de flujo y una tabla de encaminamiento implementada en memoria RAM, todo ello en el mismo ASIC (*Application Specific Integrated Circuit*). Las tablas de encaminamiento pueden ser modificadas mediante software para inicializar o reconfigurar la red.

Los paquetes de ServerNet son de tamaño variable, con un máximo de 70 bytes. Consisten en una cabecera de 8 bytes, una dirección opcional de 4 bytes, un campo de datos de tamaño variable (entre 0 y 64 bytes) y un campo de CRC (4 bytes). Al limitar la longitud del campo de datos, se elimina la posibilidad de altas latencias provocadas por paquetes extremadamente largos. Al restringir la longitud de los paquetes también se atenúa la congestión y se disminuyen los costes, pues se reducen los requerimientos de *buffers*.

Las técnicas de arbitraje utilizadas en los encaminadores de redes de multicomputadores son muy sencillas, como *round-robin* o *first-come-first-served*. Estas técnicas no pueden garantizar en la práctica una calidad de servicio a las aplicaciones. En los multicomputadores existentes la calidad de servicio viene determinada únicamente por las interacciones con otras comunicaciones, generalmente impredecibles y en su mayoría incontrolables. Además, en redes con arbitraje simple, cualquier conexión asimétrica provocará una utilización desequilibrada de la red.

Para dar soporte a las nuevas aplicaciones, los recursos de la red deben ser controlados de manera más flexible en cada encaminador. El encaminador debe ser capaz de reservar ancho de banda y *buffers* para los caminos de comunicaciones según se necesiten, y de

manejar los paquetes en órdenes arbitrarios distintos a sus órdenes de llegada. Un conjunto de mecanismos de control de los recursos de red ya se han implementado en conmutadores ATM comerciales. Sin embargo, la mayoría de los algoritmos adoptados en los conmutadores ATM, que requieren normalmente un procesador de control y algunos megabytes de espacio para buffers por conmutador, son demasiado complicados para poder implementarse en un encaminador barato para multicomputadores, y más aún en un chip ASIC.

Por tanto, los principales objetivos de diseño para el algoritmo de arbitraje son:

- Una cantidad de ancho de banda arbitraria podrá ser reservada para diferentes caminos de comunicaciones.
- El ancho de banda reservado estará garantizado.
- El ancho de banda no asignado se redistribuirá proporcionalmente entre otras conexiones.
- Obtener prestaciones netas del encaminador comparables a las de encaminadores de multicomputadores existentes.
- El algoritmo ha de ser realizable a bajo coste (un encaminador en un chip ASIC).

En sus encaminadores de ServerNet, Tandem implementa una nueva arquitectura para el control de recursos de la red denominada *arbitraje ALU-biasing*.

2.3.1 Planificación *ALU-biasing*

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. En el encaminador, el ancho de banda de salida se distribuye entre los enlaces de entrada según el algoritmo de la figura 2.13.

En *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:

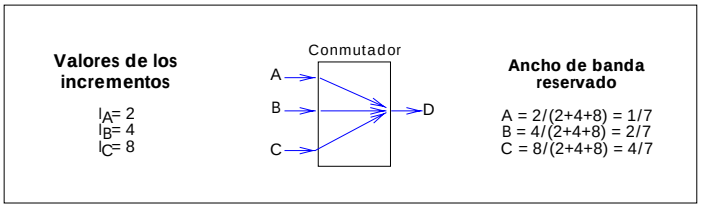


Figura 2.12: Ejemplo de reserva de ancho de banda con *ALU-biasing*

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
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 2.13: Algoritmo *ALU-biasing*

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 2.13. Un ejemplo de asignación del ancho de banda en un conmutador ServerNet se muestra en la figura 2.12. Una traza de la ejecución del algoritmo para esa configuración aparece en la tabla 2.4.

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 2.12.

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 2.4: Ejemplo de ejecución del algoritmo *ALU-biasing*

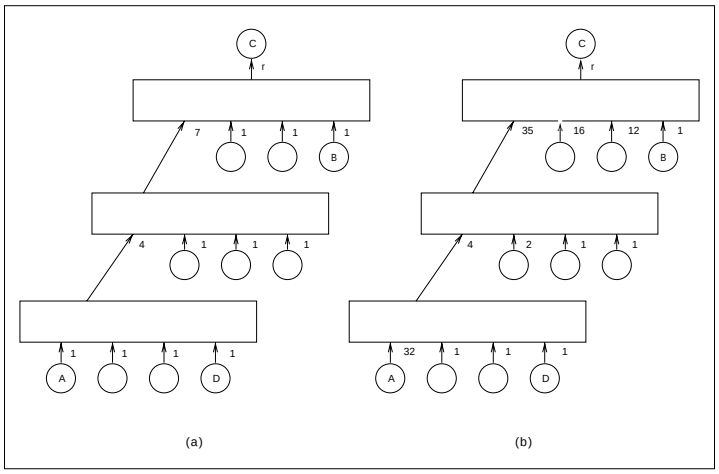


Figura 2.14: Ejemplos de reserva de ancho de banda con el encaminador de ServerNet.

En la figura 2.14 se muestran ejemplos de cómo los encaminadores que implementan el algoritmo de *ALU-biasing* distribuyen distinta cantidad de ancho de banda de la red a diferentes caminos de comunicación. El ancho de banda reservado para un camino desde su origen hasta su destino se determina multiplicando las fracciones de ancho de banda reservadas para los enlaces de entrada a lo largo del camino. En la figura 2.14-a se muestran configuraciones de los encaminadores para una distribución uniforme del ancho de banda de la red. El nodo B en lo alto del árbol recibe 1/10 del ancho de banda total del enlace r, y el nodo A también recibe $(7/10) \cdot (4/7) \cdot (1/4) = 1/10$ del ancho de banda. De hecho, todas las hojas en el árbol reciben la misma cantidad de ancho de banda. Con *ALU-biasing*, en caso de necesidad, incluso se puede reservar más ancho de banda para nodos remotos que para nodos adyacentes. Por ejemplo, en la figura 2.14-b, el nodo A recibe $(35/64) \cdot (4/8) \cdot (32/35) = 16/64$ del ancho de banda r, 16 veces más que la parte del nodo B (1/64).

En [54] se analiza la capacidad de este algoritmo de arbitraje para garantizar ancho de banda a las aplicaciones según lo soliciten. Las simulaciones muestran que el ancho de banda

reservado por los nodos se corresponde con el ancho de banda efectivo, independientemente de su situación en la topología de la red. La mejora más sobresaliente de este algoritmo de arbitraje es una utilización más uniforme de los recursos de la red en topologías asimétricas. Además, reduce la varianza en los retrasos de las comunicaciones, con lo que las prestaciones de muchas aplicaciones pueden ser mejoradas.

Por contra, algunos problemas que deja sin solucionar son la dificultad en definir cotas ajustadas para los retardos, que pueden hacer que las redes que implementan *ALU-biasing* no sean adecuadas para aplicaciones con fuertes exigencias de tiempo real, y la dificultad de repartir el ancho de banda no reservado entre todo el sistema, pues la redistribución queda limitada al encaminador local.

En cualquier caso, 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.

2.3.2 *Rotating Combined Queuing*

El algoritmo de planificación ***Rotating Combined Queuing* (RCQ)** [28] se diseñó para soportar de forma eficiente un amplio rango de garantías de servicio, proporcionando cotas deterministas para los retardos y los anchos de banda en redes de multicomputadores. RCQ permite que el tráfico de ráfagas aproveche los recursos de red no utilizados, de manera que también es capaz de proporcionar prestaciones competitivas a las comunicaciones de datos *best-effort*.

Las características que se buscaban con RCQ son:

- Proporcionar una cota pequeña y determinista para el retardo de cada conexión.
- Garantizar el ancho de banda reservado para cada conexión.
- Proporcionar altas prestaciones al tráfico *best-effort*.
- Poder ser implementado con menor coste que los algoritmos existentes que proporcionan garantías de servicio comparables.

RCQ necesita conmutadores con colas a la salida, a diferencia de la mayoría de conmutadores empleados en redes de multicomputadores. El servicio es orientado a la conexión. La información relativa a cada conexión se almacena en una tabla de búsqueda en el conmutador, que además guarda información relativa al encaminamiento.

El desbordamiento de los *buffers* se evita empleando control de flujo, como en las redes de multicomputadores existentes. Esto hace que RCQ sea adecuado únicamente en redes de corto alcance. En redes de largo alcance con enlaces de alta velocidad, los tiempos de propagación de las señales de control de flujo son elevados, y se necesitarían *buffers* mucho más grandes.

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*).

RCQ es similar a SQ [20] 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.

En la figura 2.15, se muestra la organización de las colas, situadas en los módulos de salida del conmutador. En la tabla 2.5 se muestra la notación empleada. Las colas paralelas son idénticas, y no implican nigún nivel de prioridad. El tamaño de cada cola es lo suficientemente grande como para contener una trama completa.

El algoritmo se muestra en la figura 2.16. Inicialmente, se selecciona la primera cola para transmitir sus paquetes (línea 2) 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 (líneas 4-6).

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

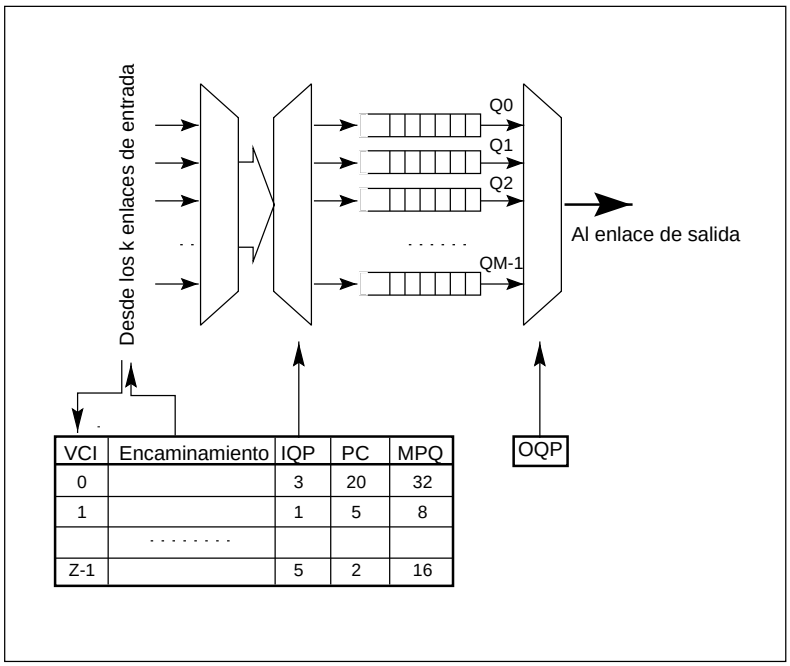


Figura 2.15: Un módulo de salida en un conmutador RCQ

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

Tabla 2.5: Notación para RCQ

Rotating Combined Queuing :

1. Inicialización:
2. $OQP = 0$
3. Para cada nueva conexión $i = 0 \ldots Z-1$
4. $IQP_i = *$
5. $PC_i = 0$
6. $MPQ_i = BW_i$
7. Para cada paquete entrante P_{in}, repetir 8-14:
8. $i = VCI(P_{in})$
9. $iqp = IQP_i$; Si $iqp == 0$, entonces $iqp = OQP$
10. Añadir el paquete a la cola Q_{iqp}
11. $PC_i = PC_i + 1$
12. Si $PC_i == MPQ_i$,
13. entonces $IQP_i = (iqp + 1)\ mod\ M$
14. $PC_i = 0$
15. En cada ciclo vacío del canal de salida, repetir 16-19:
16. Si hay algún paquete pendiente P_{out} en Q_{OQP}
17. entonces transmitirlo
18. Si $IQP_{VCI(P_out)} == OQP$, entonces $IQP_{VCI(P_out)} = *$
19. sino $OQP = (OQP + 1)\ mod\ M$

Figura 2.16: Algoritmo RCQ

ya los paquetes que corresponden dentro de la trama actual, y los siguientes paquetes que lleguen se transmitirán dentro de la trama siguiente (líneas 8-14).

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 (línea 17). Si la cola se queda vacía tras transmitir ese paquete, el puntero OQP se avanza hasta la siguiente cola (línea 19).

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 2.17 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

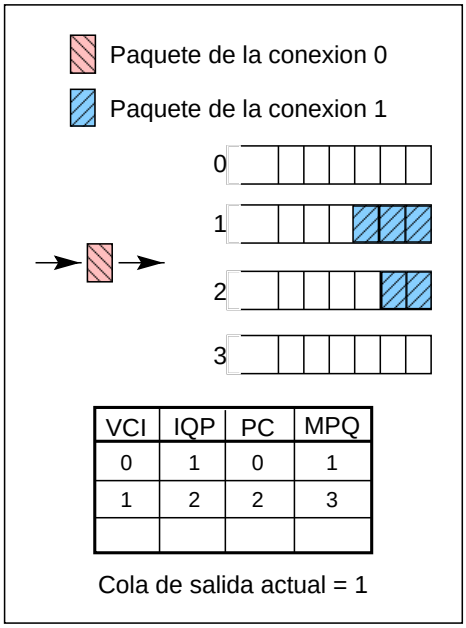


Figura 2.17: Ejemplo de funcionamiento de RCQ

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.

En este esquema del algoritmo falta por considerar el *control de flujo*. La idea básica del mismo consiste en lo siguiente. Si los paquetes de una conexión llegan a una tasa mayor que la reservada, su puntero a la cola de entrada se incrementa más rápidamente que el puntero a la cola de salida. Cuando la diferencia alcanza un determinado valor (*high mark*), calculado de manera que todavía exista suficiente espacio para poder almacenar los paquetes que puedan hallarse en tránsito en el peor de los casos, se envía una señal de control de flujo al nodo anterior que reduce la tasa de transmisión a la tasa reservada. Este modo se denomina "modo de tasa reservada", y una vez en él, el puntero a la cola de entrada nunca va por delante del puntero a la cola de salida, puesto que la tasa de entrada es igual a la de salida. Si la congestión desaparece, y la tasa de salida sobrepasa a la tasa de entrada, entonces

OQP alcanzará a IQP_i. Si la diferencia es menor que otro valor determinado (*low mark*), el conmutador cambia a "modo de tasa completa", enviando la señal de control adecuada al conmutador anterior.

Como se ha podido comprobar, 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[6].

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.

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

Capítulo 3

Conclusiones

En el capítulo previo hemos realizado un repaso del estado del arte en lo que se refiere a algoritmos de planificacin de conmutadores. Por un lado, los conmutadores con *buffers* situados a la entrada necesitan de una planificacin entre flujos por cada enlace físico de entrada, además de una planificación del conmutador que sea capaz de hallar un emparejamiento entre puertos de entrada y de salida libre de conflictos. Los algoritmos que solucionan el primer problema son los mismos que obtienen la planificación en un conmutador con *buffers* a la salida. Estos algoritmos pueden clasificarse en dos grandes grupos: los basados en tramas, y los basados en prioridades.

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 [20, 24]. Esto puede eliminar la limitación en cierta medida, al precio de necesitar un control más complejo.

Otra propuesta interesante son los *algoritmos de planificación por ráfagas*. Estos consisten en una variante sobre los algoritmos basados en prioridades, de forma que tienen en cuenta también los requisitos de QoS que necesitan las aplicaciones. De esta manera, 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 las estructura de los flujos para optimizar la velocidad a la que se realiza la planificación.

Por último, los algoritmos de planificación diseñados para SAN y multicomputadores suponen un compromiso entre las garantías de QoS que pueden llegar a ofrecer, y la necesaria simplicidad de su diseño. Por ese motivo, estos algoritmos no llegan a desacoplar totalmente las garantías ofrecidas al tráfico de otros factores, como la interferencia entre distintos flujos de tráfico.

Bibliografía

[1] T. E. Anderson y otros. *High speed switch scheduling for local area networks.* Digital Systems Research Center Technical Report, núm. 99. También aparece en ACM Transactions on Computer Systems, vol.11, no.4. Noviembre, 1993.

[2] S. Balakrishnan y F. Özgüner. *A priority-based flow control mechanism to support real-time traffic in pipelined direct networks.* Proceedings International Conference on Parallel Processing. Agosto, 1996.

[3] B. Bensaou, K. T. Chan, D. H. K. Tsang, *Credit-Based Fair Queuing(CBFQ): A simple and feasible scheduling algorithm for packet networks.* Proceedings IEEE ATM Wokshop, 1997.

[4] N. Boden y otros, *Myrinet: A gigabit per second LAN.* IEEE Micro. Febrero, 1995.

[5] T. M. Chen, S. S. Liu. *ATM Switching Systems.* Artech House Publishers, 1995.

[6] A. A. Chien, J. H. Kim. *Approaches to quality of service in high-performance networks.* Proceedings Parallel Computer Routing and Communications Workshop. Julio, 1997.

[7] P. Cuenca. *Redes de Interconexión en Sistemas Distribuidos con Tecnología ATM.* Tesina de Licenciatura. Universidad de Valencia, 1996.

[8] 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.* Proceedings IEEE ATM'97 Workshop. Mayo, 1997.

[9] W. J. Dally. *Virtual-channel flow control.* IEEE Transactions on Parallel and Distributed Systems. Marzo, 1992.

[10] B.V. Dao, J. Duato, S. Yalamanchili. *Configurable flow control mechanisms for fault-tolerant routing.* Proceedings 22nd International Symposium on Computer Architecture. Junio, 1995.

[11] A. Demers, S. Keshav, S. Shenker. *Analysis and simulations of a fair queuing algorithm.* Proceedings ACM SIGCOMM, 1989.

[12] J. Duato, P. López, F. Silla, S. Yalamanchili. *A high performance router architecture for interconnection networks.* Proceedings International Conference on Parallel Processing. Agosto, 1996.

[13] J. Duato, S. Yalamanchili, M. B. Caminero, D. Love, F. J. Quiles. *MMR: A high-performance multimedia router. Architecture and design trade-offs.* Enviado al Fifth International Symposium on High Performance Computer Architecture (HPCA-5). Julio, 1998.

[14] D. Ferrari, D. Verma. *A scheme for real-time channel establishment in wide-area networks.* IEEE Journal on Selected Areas in Communications. Abril, 1990.

[15] M. L. Fulgham y L. Snyder. *A comparison of input and output driven routers.* Proceedings Euro-Par'96. Agosto, 1996.

[16] P. T. Gaughan y S. Yalamanchili. *Adaptive routing protocols for hypercube interconnection networks.* IEEE Computer. Mayo, 1993.

[17] 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.

[18] S. J. Golestani. *Duration-limited statistical multiplexing of delay-sensitive traffic in packet networks.* Proceedings IEEE INFOCOM. Abril, 1991.

[19] S. J. Golestani. *A framing strategy for congestion management.* IEEE Journal on Selected Areas in Communications. Septiembre, 1991.

[20] S. J. Golestani. *Congestion-free communication in high-speed packet networks.* IEEE Transactions on Communications. Diciembre, 1991.

[21] S. J. Golestani. *A self-clocked fair queuing scheme for broadband applications.* Proceedings IEEE INFOCOM, 1994.

[22] 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.

[23] J. Jonsson, J. Vasell. *A comparative study of methods for time-deterministic message delivery in a multiprocessor architecture.* Proceedings IEEE International Parallel Processing Symposium, 1996.

[24] C. Kalmanek, H. Kanakia, S. Keshav. *Rate controlled servers for very high-speed networks.* Proceedings IEEE Global Telecommunications Conference, 1990.

[25] M. J. Karol, M. G. Hluchyj, S. P. Morgan. *Input versus output queuing on a space division packet switch.* IEEE Transactions on Communications. Diciembre, 1987.

[26] M. Katevenis, S. Sidiropoulos, C. Corcoubetis. *Weighted round-robin cell multiplexing in a general-purpose ATM switch.* IEEE Journal on Selected Areas in Communications. Octubre, 1991.

[27] M. G. H. Katevenis, y otros. *ATLAS I: A single-chip ATM switch for NOWs.* Proceedings Workshop on Communications and Architectural Support for Network-based Parallel Computing. Febrero, 1997.

[28] J. H. Kim, A. A. Chien, *Rotating Combined Queuing (RCQ): Bandwidth and latency guarantees in low-cost, high-performance networks.* Proceedings Int. Symposium on Computer Architecture (ISCA). Mayo, 1996.

[29] J. H. Kim. *Bandwidth and latency guarantees in low-cost, high-performance networks.* Tesis Doctoral. Universidad de Illinois en Urbana-Champaign. 1997.

[30] B. Kuszmaul. *The RACE network architecture.* Proceedings IEEE International Parallel Processing Symposium, 1995.

[31] B. Kwan y otros. *Best-effort bandwidth reservation in high-speed LAN's using wormhole routing.* Proceedings Int. Conf. on Computer Communications and Networks, 1996.

[32] S. S. Lam, G. G. Xie. *Burst scheduling networks.* Technical Report TR-94-20. Dept. of Computer Science, Univ. de Texas en Austin. Octubre, 1996. Disponible en http://www.cs.utexas.edu/users/lam/NRL/video_services.html

[33] S. S. Lam, G. G. Xie. *Group Priority Scheduling.* Technical Report TR-95-28. Dept. of Computer Science, Univ. de Texas en Austin. Enero, 1997. Disponible en http://www.cs.utexas.edu/users/lam/NRL/video_services.html

[34] P. McKenney. *Stochastic fairness queuing.* Proceedings IEEE INFOCOM, 1990.

[35] A. Mekkittikul, N. McKeown, *A starvation-free algorithm for achieving 100 % throughput in an input-queued switch.* Proceedings Int. Conf. on Computer Communications and Networks, 1996.

[36] A. Mekkittikul, N. McKeown, *A practical scheduling algorithm to achieve 100 % throughput in input-queued switches*. Proceedings INFOCOM, 1998.

[37] J. B. Nagle. *On packet switches with infinite storage*. IEEE Transactions on Communications. Abril, 1987.

[38] S. Pakin, M. Lauria, A. Chien. *High performance messaging on workstationns: Illinois Fast Messages on Myrinet*. Proceedings Supercomputing 95. Noviembre, 1995.

[39] A. Parekh, R. Gallager. *A generalized processor sharing approach to flow control in integrated services networks – the single node case*. Proceedings IEEE INFOCOM. Mayo 1992.

[40] M. Prycker. *Asynchronous transfer mode: solution for broadband ISDN*, Ellis Horwood Limited, 1991.

[41] J. Rexford, J. Hall y K. G. Shin. *A Router Architecture for Real-Time Point-to-Point Networks*. Proceedings International Symposium on Computer Architecture. Mayo 1996.

[42] M. D. Schroeder y otros. *Autonet: A high-speed, self-configuring local area network using point-to-point links*. IEEE Journal on Selected Areas in Communications. Octubre, 1991.

[43] M. Shreedhar, G. Varghese. *Efficient fair queuing using deficit round-robin*. IEEE/ACM Transactions on Networking. Junio, 1996.

[44] F. Silla y otros. *Efficient adaptive routing in networks of workstations with irregular topology*. Proceedings Workshop on Communications and Architectural Support for Network-based Parallel Computing. Febrero, 1997.

[45] F. Silla y J. Duato. *Improving the efficiency of adaptive routing in networks with irregular topology*. Proceedings 1997 Conference on High Performance Computing. Diciembre, 1997.

[46] D. Stiliadis. *Traffic scheduling in packet-switched networks: Analysis, design and implementation*. Tesis Doctoral. Universidad de California en Santa Cruz. Junio, 1996.

[47] D. Stiliadis, A. Varma. *Providing bandwidth guarantees in an input-buffered crossbar switch*. Proceedings IEEE INFOCOM, 1995.

[48] D. Stiliadis, A. Varma. *Design and analysis of frame-based queuing: A new traffic scheduling algorithm for packet-switched networks*. Proceedings SIGMETRICS, 1996.

[49] K. Toda y otros. *Design and implementation of a priority forwarding router chip for real-time interconnection networks*. International Journal of Mini and Microcomputers. Enero, 1995.

[50] D. Verma, H. Zhang, D. Ferrari. *Delay jitter control for real-time communication in a packet switching network*. Proceedings Tricomm, 1991.

[51] L. Zhang. *Virtual Clock: A new traffic control algorithm for packet switching networks*. ACM Transaction on Computer Systems. Mayo 1991.

[52] H. Zhang, D. Ferrari. *Rate-controlled static-priority queuing*. Proceedings IEEE INFOCOM, 1993.

[53] *Generic coding of moving pictures and associated audio*. Recomendación H.262.Draft International Standard ISO/IEC 13818-2. Marzo, 1994.

[54] *Network resource control in the Tandem ServerNet router*. Tandem Computers. Marzo, 1996.

[55] *ServerNet technology: Introducing the world's first System Area Network*. Disponible en http://www.tandem.com/. Tandem Computers. Octubre, 1996.