

CAPÍTULO 4

CONTROL DE CONGESTIÓN Y GESTIÓN JUSTA DE COLAS

4.1. INTRODUCCIÓN

Otro importante aspecto relacionado con el control de congestión al que nos enfrentamos en esta tesis es el que surge de las diferentes características de tráfico que tienen las fuentes. La caracterización del tráfico, como sabemos, es una de las ventajas de ATM, que permite la integración de todo tipo de CoS. Sin embargo, esto puede ser también una importante fuente de problemas por la posibilidad que existan flujos de entrada a los conmutadores que generen situaciones de injusticia con respecto a los recursos de otras fuentes. Esa posibilidad de mal comportamiento de unas fuentes exigentes con respecto a otras que lo son menos puede acabar generando situaciones de inanición y también de congestión que hay que intentar solventar.

En muchas tecnologías de redes el control de congestión se consigue confiando en protocolos extremo-a-extremo como TCP. Esto permite disponer de routers y conmutadores muy sencillos, sin embargo, pueden aparecer situaciones críticas si todos los extremos de la comunicación no cooperan, ya que -de otro modo- los flujos con buen comportamiento no quedarán bien protegidos de los flujos más ambiciosos que, en el límite, pueden acabar quedándose con todos los recursos de la red. Además, los extremos de la comunicación deberán implementar algoritmos de control homogéneos, para evitar que aquellos usuarios que empleen algoritmos de control más estrictos puedan autogenerar situaciones de injusticia respecto a aquellas fuentes de tráfico con control menos riguroso.

En nuestra propuesta de arquitectura TAP se ve clara la necesidad de aportar mecanismos de justicia ya que estamos proponiendo fuentes privilegiadas de tráfico a las que se les ofrece GoS, lo que puede acabar degenerando en situaciones de injusticia para aquellas fuentes que no se definan como privilegiadas. Así, una alternativa al control de congestión en los extremos es diseñar conmutadores que soporten asignaciones de ancho de banda justas para proteger a las fuentes no privilegiadas de las que sí lo son. De este modo, cada fuente tendrá garantizados sus recursos. Pero uno de los principales problemas al que nos enfrentamos es que los algoritmos propuestos para aportar la necesaria justicia son demasiado complejos ya que deben realizar la gestión separada (*per-flow*) de flujos. Siendo más precisos, un router o un conmutador que implemente estos algoritmos deberá realizar: una clasificación separada de paquetes, una gestión separada de buffers para cada flujo y, en la mayoría de los casos, también una planificación separada de flujos. Toda esta complejidad añadida puede hacer peligrar otros parámetros de QoS como el *delay* y *jitter* que son un importante requerimiento para fuentes con necesidades de alta velocidad como las aplicaciones de tiempo real.

Como se ha comentado en capítulos previos, una de las principales características de ATM es su capacidad para garantizar la QoS a diversos tipos de aplicaciones, tanto de tiempo real como no tiempo real. Para las aplicaciones de tiempo real deben garantizarse, entre otros parámetros de tráfico, un *delay* y un *jitter* máximos negociados en el establecimiento de la conexión. Sin embargo, las aplicaciones pensadas para la transmisión de datos no tienen excesivos requerimientos de *delay* pero sí en lo referente a la justicia de unas fuentes de tráfico con respecto a otras que puedan consumir excesivos recursos.

Existen diferentes planteamientos en cuanto a la arquitectura de los conmutadores ATM, pero todos ellos cuentan con la inclusión de colas que, en esencia, se usan para aplicar diferentes esquemas de gestión de

tráfico. Precisamente este planteamiento explica la importancia de las políticas de gestión de colas.

En general, los conmutadores ATM enrutan las células desde un conjunto de puertos de entrada hasta un conjunto de puertos de salida basados en los identificadores de flujo VPI/VCI. En el puerto de salida de un conmutador ATM un controlador de puerto multiplexa células desde diferentes flujos de entrada en la línea de salida. Los puertos de salida de los conmutadores ATM son gobernados por una política de planificación como FIFO, que decide qué células desencolar de la cola de salida. Las políticas de planificación pueden ser clasificadas en *work-conserving* o *non work-conserving*. Una política es *work conserving* si nunca deja el enlace de salida en estado de inactividad mientras haya células dentro de la cola de salida. Al contrario, una política *non work-conserving* puede dejar el enlace de salida en estado de inactividad incluso si la cola de salida no está vacía. La *Tabla 4.1* presenta la clasificación de algunas de las más conocidas políticas de planificación.

TABLA 4.1
MECANISMOS DE CONTROL BASADOS EN VELOCIDAD

Servicios <i>work-conserving</i>	Servicios <i>non-work-conserving</i>
Weighted Fair Queueing	Jitter Earliest-Due-Date
Virtual Clock	Stop-and-Go
Delay Earliest-Due-Date	Hierarchical Round-Robin
Self-Clocked Fair Queueing	

Todas las políticas *work-conserving* tienen, para un conjunto dado de patrón de llegada de células, idéntico retardo medio de células y necesidades máximas de buffer. Una política *non work-conserving* puede necesitar mayor retardo medio de células y necesidades de buffer máximo y medio que una política *work-conserving*. Por otro lado, las políticas *work-conserving* tienden a incrementar las ráfagas de tráfico, mientras las políticas *non work-conserving* pueden ser usadas para limitar el tráfico a ráfagas. En líneas generales las políticas *work-conserving* son más fáciles de implementar y requieren una gestión de buffers más sencilla.

Los sistemas *work-conserving* envían los paquetes una vez que el servidor ha completado el servicio. Por tanto, el servidor nunca queda en estado de inactividad si existen trabajos en una cola del sistema. Los dos ejemplos más claros y clásicos de los sistemas *work-conserving* son FIFO y LIFO. Los esquemas *non work conserving* se caracterizan porque los servidores esperan un espacio de tiempo aleatorio antes de servir el siguiente paquete de una cola, incluso si hay paquetes esperando en las colas.

En ATM las aplicaciones diferentes tienen diferentes requerimientos de tiempo y ancho de banda. Para asegurar que cada aplicación tenga la QoS requerida, la red debe realizar alguna forma de control de admisión a los flujos que intentan acceder a los conmutadores. El control de admisión deberá asegurar que existen suficientes recursos en la red antes de conceder al flujo la QoS requerida, para no comprometer la QoS de las conexiones que ya están dentro de la red. Para garantizar que cada conexión puede tener la QoS solicitada se aplica la función CAC (Call Admission Control). Existen muchas formas de implementar la CAC, sin embargo, muchas de ellas sólo ofrecen garantías estadísticas, es decir, en términos de retardo medio de célula. Para tráfico de tiempo real, donde la QoS debe tener una garantía absoluta, este tipo de CAC no son aplicables. En lugar de ello, una garantía determinista garantiza que nunca se pierde el plazo de entrega.

En general, suele distinguirse entre dos tipos de planteamientos en cuanto a la llegada de los datos: garantía estadística y determinística. La garantía estadística promete que un porcentaje de los datos llegará con un retardo D , o que una media del ancho de banda está disponible para el flujo. Sin embargo, la garantía determinística toma la forma de promesa de que todos los datos llegarán con un retardo D respecto a la transmisión. También garantiza que un flujo accederá, como poco, a X bits por segundo o que las pérdidas no excederán de un cierto valor.

Las colas pueden ser gestionadas de muy diversas formas y el método elegido para hacer esta gestión tiene muy diferentes efectos en el tráfico que fluye a través de las colas. Otro importante aspecto relacionado con las colas es que cada esquema va a suponer en cada conmutador una serie de requerimientos para realizar la implementación del sistema de colas.

Otro aspecto directamente relacionado con este tema es la integración con diferentes redes. Esto causa que, aunque los conmutadores y/o routers gestionen su tráfico perfectamente, el tráfico puede no tener garantizado el buen rendimiento.

Para acabar de situar este importante aspecto de gestión de tráfico destacamos su relación con el control de velocidad. Así, puede realizarse la siguiente división:

- Las redes de datos de conmutación de paquetes suelen usar mecanismos FIFO y control de flujo basado en ventana.
- Por otro lado, las redes de comunicación multimedia suelen usar mecanismos basados en velocidad. Este planteamiento puede clasificarse según lo que puede observarse en la *Tabla 4.2*

Las técnicas de encolamiento *per-VC*¹, que aplican diferentes pesos a las colas, se han demostrado como un mecanismo apropiado para satisfacer los requerimientos de *delay*, *jitter* y justicia comentados al inicio, y se han propuesto múltiples algoritmos de encolamientos *per-VC* basados en velocidades. Algunos de los ejemplos más significativos de estos mecanismos son WFQ (Weighted Fair Queueing) [2], VirtualClock [20], SCFQ (Self-Clocked Fair Queueing) [6], Delay-EDD (Delay Earliest Due Date), Jitter-EDD [21] y HOL-EDD (Head-Of-the-Line EDD) [22]. Todos estos algoritmos, salvo Jitter-EDD, son *work-conserving*, lo que implica que el servidor no descansa mientras existan paquetes en espera de ser procesados.

La *Tabla 4.2* presenta una división de las políticas de planificación *work-conserving* más interesantes, que pueden ser clasificadas en políticas simples en cuanto a su implementación y eficiencia, y en políticas basadas en velocidad, en las cuales a cada flujo se asigna una velocidad de servicio o un peso que determina cuántos recursos se deben asignar al flujo. Como regla general, las políticas basadas en pesos aproximan la disciplina de servicio ideal GPS (Generalized Processor Sharing) y, para todas ellas, es común que el tiempo de respuesta sea directamente dependiente de la velocidad de servicio asignada a cada flujo y la velocidad de servicio se corresponde con el ancho de banda asignado. Debe destacarse que la implementación de las técnicas basadas en velocidad son mucho más complejas de implementar.

TABLA 4.2
CLASIFICACIÓN DE POLÍTICAS DE PLANIFICACIÓN WORK-CONSERVING

Políticas de planificación sencilla	Políticas de planificación basadas en velocidad
FIFO	GPS
Priority Queueing	Virtual Clock (VC)
Earliest Deadline First (EDF)	Weighted Round-Robin (WRR)
Response-Time Analysis (RTA)	WFQ
Calculus for Network Delays (CND)	Multirate PGPS

La *Figura 4.1* muestra los diferentes comportamientos y evolución de los mecanismos *fair queueing* que comentamos brevemente:

- La propuesta (1) es la conocida como solución original de Nagle [1] que se caracteriza por el aislamiento de las fuentes de tráfico que se comportan incorrectamente. Esta solución presenta dos limitaciones destacables: en primer lugar, se ignora la longitud de los paquetes que llegan a las colas que, como podemos ver, pueden ser de diferente tamaño teniendo todos un mismo tiempo de ciclo y, en segundo lugar, este esquema es sensible al patrón de llegadas de paquetes.
- La solución (2) propuesta por Demers, Keshav y Shenker [2] parte de paquetes todos de igual tamaño (1 bit), de forma que se aplica justicia realizando esperas de $n-1$ bits antes de que una fuente vuelva a enviar un nuevo paquete, lo que presenta la limitación de que cada fuente tiene la misma fracción de ancho de banda, sin permitir la caracterización del tráfico.

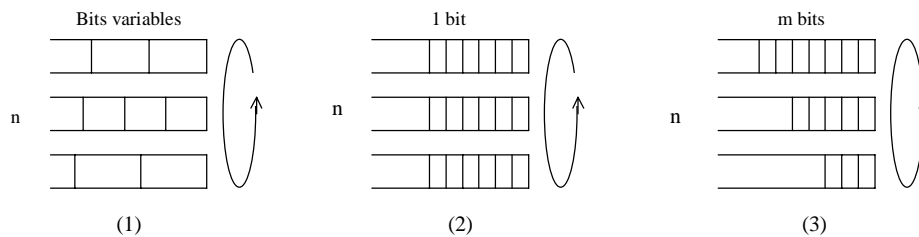


Figura 4.1. Esquemas justos de colas

¹ Las técnicas de colas *per-VC* se caracterizan por emplear colas separadas para cada VC. En nuestro caso tendremos colas separadas, tanto para transmisiones garantizadas, como para las que no lo son.

- El esquema (3) coincide con el conocido WFQ (Weighted Fair Queueing) caracterizado porque las fuentes pueden enviar diferente número de paquetes en cada ciclo. En este caso m es mayor que n y se establecen ciclos de m bits cada uno. Parekh [3,4] en su tesis de 1992 realiza una demostración de las garantías de funcionamiento de esta propuesta.

A continuación vamos a realizar una breve descripción de algunas de las técnicas de control de congestión basadas en la justicia de colas, describiendo sus ventajas e inconvenientes para concluir con la presentación general de nuestra aportación que será explicada en detalle en capítulos siguientes.

4.2. FAIR QUEUEING

Una de las disciplinas de servicio de paquetes a los puntos de encolado de los conmutadores es la PS (Processor Sharing) que emplea colas FIFO separadas para cada sesión que comparte un enlace. En PS todas las sesiones en espera reciben igual tamaño de ancho de banda, independientemente del tamaño de los paquetes que tengan. PS es una disciplina ideal cuando el servidor es capaz de servir N sesiones simultáneas. En realidad, el servidor envía un paquete cada vez. Precisamente, [2] propone un algoritmo de aproximación a paquetes de PS llamado FQ (Fair Queueing) con asignación justa del ancho de banda y protección contra las fuentes malintencionadas. Además, para los servidores que se ajustan o aproximan la disciplina de servicio PS, las fuentes pueden medir más ajustadamente el estado de la red. De este modo queremos destacar que PS es una disciplina de servicio que permite el diseño y la implementación de algoritmos de congestión muy robustos.

Los algoritmos FQ (Fair Queueing) [2,3,4] aportan atractivas ventajas y han sido diseñados para que routers y conmutadores ofrezcan el control de congestión y para la asignación justa de ancho de banda. Sin embargo, su coste de implementación completa en una red puede ser no aceptable para aplicaciones de alta velocidad ya que, aparte de la complejidad de programación, requieren que el mecanismo FQ mantenga información de estado, gestione los buffers de entrada, realice planificación de los paquetes y, todo ello, sobre la base de la separación de los flujos.

No obstante, varias investigaciones han demostrado que las propuestas de control de congestión extremo-a-extremo pueden ser mejoradas de una forma sustancial si en los routers se aplican técnicas justas de asignación del ancho de banda disponible en la red ya que, además de proteger unas fuentes de otras, permiten que en la red puedan coexistir diversas políticas de control de congestión. Del mismo modo, varias de estas investigaciones también reconocen que la asignación justa de ancho de banda juega un papel necesario, aunque a veces no beneficioso, en el control de congestión. Así, la mayor parte de propuestas en este campo parten de dos premisas importantes, por un lado que los mecanismos justos de asignación de ancho de banda son necesarios, y por otro, que la complejidad de estos mecanismos es un importante factor para su adopción en las redes.

Lo que parece claro es que cualquier propuesta que realicemos en este sentido será bastante más compleja de implementar que el clásico encolamiento FIFO que ha sido el más usado tradicionalmente. Como es sabido, en FIFO los paquetes son servidos en el orden de llegada, y los buffers son gestionados siguiendo una sencilla estrategia *drop-tail* según la cual los paquetes entrantes son tirados cuando el buffer está lleno. Pero FIFO carece de la justicia que se buscó con FQ [2,3,4] y sus variantes [5,6,7] y con mecanismos de abandono por flujos como FRED (Flow Random Early Drop) [8]. En cualquiera de estas técnicas se requiere mantener el estado de los diversos flujos por separado, de forma que, para cada paquete que llega, hay que clasificarlo en su correspondiente flujo, actualizar las variables de estado de cada flujo y realizar operaciones (*encolar_paquete*, *tirar_paquete*, *priorizar_flujo*, etc) en función del estado de cada flujo.

RED (Random Early Detection) [9] sirve también los paquetes en el orden de llegada, pero la gestión del buffer es mucho más sofisticada que el *drop-tail* de FIFO. RED tira probabilísticamente paquetes largos antes de que el buffer se llene, ofreciendo indicación de congestión rápida a los flujos que pueden ser afectados antes de que se produzca el rebosamiento del buffer. De este modo RED mantiene dos umbrales en los buffers. Cuando la ocupación media del buffer es menor que el primer umbral, no se tira ningún paquete, pero cuando se supera el segundo umbral, los paquetes comienzan a ser tirados por la red. Cuando la ocupación del buffer se encuentra entre los dos umbrales, es cuando la probabilidad de tirar paquetes incrementa linealmente con la ocupación del buffer.

FRED (Flow Random Early Drop) [8] lo que hace es ampliar RED para ofrecer un cierto grado de asignación justa de ancho de banda. Para conseguir la justicia, FRED mantiene el estado de todos los flujos que tienen, como poco, un paquete en el buffer. Al contrario que en RED, donde la decisión de tirar paquetes se basa sólo en el estado del buffer, en FRED la decisión se basa también en el estado de los flujos. De este modo, FRED tira preferentemente los paquetes de un flujo al cual se le han tirado pocos paquetes

anteriormente, y que poseen una cola mayor que el tamaño medio de las colas. Este algoritmo tiene dos variantes, que se diferencian en que una de las versiones garantiza a cada flujo un número mínimo de buffers.

Todos los algoritmos comentados presentan diferentes niveles de complejidad. FRED se encarga de clasificar los flujos entrantes, mientras FIFO y RED no lo hacen. En suma, ninguno de ellos debe implementar ningún algoritmo de planificación de paquetes, sin embargo, todos aplican una sencilla planificación FIFO.

4.3. GPS, PGPS (WFQ) Y PFQ

Este apartado comenta brevemente el modelo de tráfico continuo ampliamente aceptado, para pasar luego a las propuestas de aproximación a este modelo para las redes de paquetes. De este modo tratamos de aclarar algunas confusiones en torno a estas disciplinas e identificamos algunos aspectos claves que luego empleamos en nuestra propia propuesta.

La disciplina GPS (Generalized Processor Sharing) tiene dos propiedades importantes: 1) puede garantizar un servicio de retardo limitado extremo-a-extremo a una sesión cuyo tráfico está controlado por leaky bucket; y 2) puede asegurar asignación justa de ancho de banda a todas las sesiones que tienen trabajos en espera, estén o no sometidas al control de tráfico. Mientras la primera propiedad es importante para soportar tráfico best-effort y servicios jerárquicos de enlace compartido, la segunda propiedad es la base para aportar GoS al tráfico.

Debido a que GPS usa un modelo fluido ideal que no puede ser realizado en el mundo real, se han propuesto varios algoritmos de paquetes como aproximación a GPS. En la literatura se denominan PGPS (Packet Generalized Processor Sharing) [3,4] a estas aproximaciones, aunque en otros casos [5,10] también se denominan como PFQ (Packet Fair Queueing). Entre los algoritmos que implementan GPS en redes de paquetes destaca WFQ (Weighted Fair Queueing) [2] que es actualmente el más popular método para ofrecer garantía de funcionamiento y es también conocido como PGPS (Packet Generalized Processor Sharing) [3,4].

Contrariamente a la idea que se ha extendido de forma generalizada, existen importantes diferencias entre los servicios ofrecidos por el sistema de paquetes WFQ y el sistema fluido GPS y estas diferencias son estudiadas en detalle en [5].

Precisamente, uno de los aspectos más importantes en el diseño de redes de servicios integrados es la elección de la disciplina de servicio de los paquetes en los puntos de encolado de la red. GPS es una de esas disciplinas, a la que se ha prestado una especial atención. GPS es una forma general de procesador *head-of-line* compartiendo disciplinas de servicio PS. Con PS tenemos una cola FIFO separada para cada sesión que comparte el mismo enlace. Durante un intervalo de tiempo, cuando hay exactamente N colas no vacías cada una de ellas asociada a un servicio compartido, el servidor GPS sirve los N paquetes de la cabecera de las colas simultáneamente, cada uno a una velocidad de N veces la velocidad del enlace. Mientras un servidor PS sirve todas las colas no vacías a la misma velocidad, GPS permite a diferentes sesiones tener diferentes servicios compartidos y sirve las colas no vacías en proporción al servicio compartido de las correspondientes sesiones.

Un servidor GPS, sirviendo N sesiones, se caracteriza por N números reales positivos $\phi_1, \phi_2, \dots, \phi_N$. Estos valores indican el tamaño relativo de servicio para cada sesión. El servidor trabaja a una velocidad fija r y es *work-conserving*. Sea $W_i(t_1, t_2)$ el tamaño de sesión i servido en el intervalo $[t_1, t_2]$, así, un servidor GPS se define como aquel para el cual

$$\frac{W_i(t_1, t_2)}{W_j(t_1, t_2)} \geq \frac{\phi_i}{\phi_j} \quad j = 1, 2, \dots, N \quad (1)$$

se mantiene para una sesión i que está pendiente durante el intervalo $[t_1, t_2]$. Si $B_{GPS}(\tau)$, el conjunto de sesiones pendientes en el tiempo τ , permanece inalterable durante el intervalo de tiempo $[t_1, t_2]$, la velocidad de servicio de la sesión i durante el intervalo será exactamente

$$r_i^*(t_1, t_2) = \frac{\phi_i}{\sum_{j \in B_{GPS}(t_1)} \phi_j} r \quad (2)$$

donde r es la velocidad del enlace. Como $B_{GPS}(t_1)$ es un subconjunto de todas las sesiones del servidor, puede verse como

$$r_i^*(t_1, t_2) \geq r_i \quad (3)$$

se mantiene, donde

$$r_i = \frac{\phi_i}{\sum_{j=1}^N \phi_j} r \quad (4)$$

Así, la sesión i tiene garantizada una mínima velocidad de servicio r_i durante un intervalo en el que está pendiente. Dada la longitud del intervalo de tiempo hasta llegar a cero, obtenemos la velocidad de servicio instantánea de la sesión $r_i^*(\tau)$.

Hay que destacar que GPS es un servidor ideal que no transmite paquetes como entidades. Esto asume que el servidor puede servir todas las sesiones pendientes simultáneamente y que el tráfico es infinitamente divisible. En un sistema de paquetes más realista sólo una sesión puede recibir servicio en un instante, y un paquete entero debe ser servido antes de servir otro. La *Figura 4.2* presenta un diagrama de tiempos de la llegada y transferencia de paquetes pertenecientes a la sesión i en un sistema GPS.

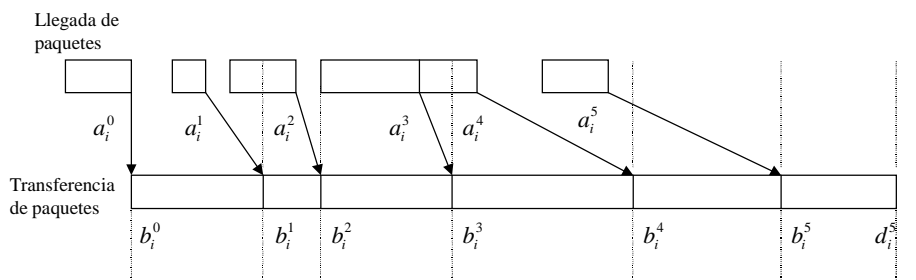


Figura 4.2. Diagrama de tiempos de llegada y salida de paquetes en la disciplina de servicio GPS

Una de las líneas de trabajo de GPS está centrada en el diseño de algoritmos de control de congestión basadas en realimentación para ser usados en redes de datagramas. Así, en este caso las fuentes muestréan constantemente el estado de la red para detectar síntomas de congestión y, cuando los detectan, las fuentes reducen la velocidad de envío para aliviar el efecto de la congestión. En el caso de que todas las fuentes empleen la misma cola FIFO para el envío de su tráfico, si una de las fuentes no reacciona al control de congestión, nos encontraremos con el hecho de que todas las sesiones acabarán sufriendo el problema de congestión. Para evitar este problema se propuso emplear una cola FIFO separada para cada sesión y atender las colas existentes en una política Round-Robin, como la propuesta (1) de Nagle [1] en la *Figura 4.1*. Este esquema se comporta mejor que un FIFO puro, pero en el caso que existan fuentes con diferentes tamaños de paquetes, puede ocurrir que las que generen paquetes más grandes acaben consiguiendo más ancho de banda que las que los generan más pequeños. Destacamos que en la disciplina de servicio PS, descrita antes, este problema no aparece porque las sesiones en espera reciben el mismo ancho de banda independientemente del tamaño de los paquetes.

El algoritmo GPS tiene por tanto una serie de interesantes propiedades para las redes de servicios integrados como ATM y, de hecho, se han propuesto muy diversos algoritmos PFQ (Packet Fair Queueing) [10] para aproximar GPS a las redes de conmutación de paquetes, aunque no demasiadas para las de tecnología ATM. El coste de implementación de los algoritmos PFQ está influido por dos importantes aspectos:

- 1) Cálculo de la función de tiempo virtual del sistema que es una medida dinámica del tamaño de servicio justo normalizado que debe recibir cada sesión. La mayor parte de algoritmos PFQ propuestos se mueven entre costes $O(1)$ y $O(\log N)$ en cuanto a la complejidad de la función de tiempo virtual.

- 2) El coste del mantenimiento del orden relativo de los paquetes a través de su marca de tiempos en un mecanismo basado en colas con prioridad desde donde se transfieren los paquetes. La complejidad algorítmica de implementación de una cola con prioridad para N números arbitrarios es $O(\log N)$. Pueden conseguirse menores costes, pero son de difícil implementación en redes de alta velocidad. Las colas con prioridad en PFQ pueden basarse en el tiempo virtual de inicio, en el de finalización, o en ambos. Como cualquiera de estos dos tiempos crece monótonicamente con cada sesión, solo necesitamos considerar las cabeceras de cada sesión cuando el servidor está tomando el siguiente paquete a ser transferido. Así, el número de entidades en la cola con prioridad coincide con el número de sesiones activas.

Existen bastantes aproximaciones de algoritmos PFQ que procesan paquetes basados en GPS, pero todos usan un mecanismo similar de colas con prioridad, basado en la noción de una función de tiempo virtual. Las propuestas se diferencian en la elección de la función de tiempo virtual y en la selección de la política de paquetes.

Para poder aproximarse a GPS, los algoritmos PFQ mantienen una función de tiempo virtual $V(\cdot)$, un tiempo de inicio virtual $I_i(\cdot)$ y un tiempo de finalización virtual $F_i(\cdot)$ para cada una de las i sesiones. En el caso de las redes ATM es lógico representar el tiempo virtual en términos de la cantidad de células servidas.

$I_i(\cdot)$ y $F_i(\cdot)$ son actualizados cada vez que a una sesión i llega un paquete o está activa, o cada vez que un paquete de esa sesión i acaba su servicio. Podemos representar este funcionamiento con la siguientes expresiones,

$$I_i(t) \begin{cases} \max(V(t), F_i(t-)) \\ F_i(t-) \end{cases} \quad (5)$$

$$F_i(t) = I_i(t) + \frac{L_i^k}{r_i} \quad (6)$$

donde $F_i(t-)$ es el tiempo de finalización virtual de la sesión i antes de la actualización, y L_i^k es la longitud del paquete de la cabecera de la cola de la sesión i .

Podemos intuir que $V(t)$ es el tamaño de *tiempo de servicio normalizado justamente* que cada sesión debería recibir en el tiempo t . $I_i(t)$ representa el tamaño de *tiempo de servicio normalizado* que la sesión i ha recibido en el tiempo t . Como norma general, el objetivo de todos los algoritmos PFQ es el de minimizar las diferencias entre los tiempos virtuales $V(t)$ e $I_i(t)$.

El papel que juega la función de tiempo virtual es el de reiniciar el valor del tiempo inicial virtual de la sesión cuando una sesión, que no estaba pendiente de servirse, pasa a estar pendiente de nuevo. Las funciones de tiempo virtual usadas en los diversos algoritmos PFQ existentes aportan diferencias entre su complejidad y su precisión².

En la mayor parte de los casos de PFQ, cuando el servidor está eligiendo el siguiente paquete para transmitir, los algoritmos optan, de entre todos los paquetes del sistema, por aquel que tenga el tiempo final virtual más pequeño, es decir, se aplica una política de selección de paquetes “primero los tiempos virtuales finales más pequeños” (SFF). Esta política de elección de paquetes suele dar lugar a límites de retardo casi idénticos a los del sistema fluido GPS, sin embargo, acaba dando lugar a mayores tiempos de servicio que GPS [10]. Desde luego, puede optarse por otras políticas de elección de paquetes como las descritas en [5,10,11], donde los algoritmos aplican la política SEFF (Smallest Eligible virtual Finish time First) optando, de entre todos los elegibles³, por aquel con el menor tiempo de finalización.

Otra de las líneas importantes de GPS está en la forma de ofrecer servicios con garantía de retardo limitado en redes de conmutación de paquetes. El esquema PGPS (Packet Generalized Processor Sharing [2] responde al planteamiento (2) ilustrado en la *Figura 4.1*, con evolución hacia el (3), bajo el nombre de WFQ (Weighted Fair Queueing) que es el algoritmo PFQ más conocido. [2] demuestra que, empleando servidores GPS en los conmutadores, puede garantizarse un límite de retardo extremo-a-extremo a sesiones cuyo tráfico

² Una función de tiempo virtual se considera precisa si el algoritmo PFQ que la usa ofrece casi idéntico servicio que GPS.

³ Un paquetes es elegible si su tiempo virtual de inicio es menor que el tiempo virtual actual.

es leaky bucket. La disciplina de servicio de PGPS se centra en el tratamiento igualitario de todos los usuarios y, a partir de estos planteamientos, han surgido posteriormente nuevas investigaciones que han dado lugar a variantes extendidas de PGPS.

Una de las variantes más interesantes de PGPS es descrita en [3,4] donde se emplea una disciplina de servicio también basada en topologías de redes arbitrarias de servidores GPS. Se destaca que para soportar servicios de tiempo real es básico resolver el problema de ofrecer garantía de funcionamiento a los diversos usuarios de una red de servicios integrados. Y este problema es especialmente difícil de resolver cuando aparecen las congestiones, que es cuando es importante el uso eficiente del ancho de banda de los enlaces de la red. En [3] se propone la combinación de control de admisión (CAC) leaky bucket y una disciplina de servicio de paquetes *work-conserving* en los nodos de la red para acomodar el retardo y el rendimiento de un amplio número de sesiones simultáneas. Mientras [3] analiza un sistema con un solo nodo, en [4] los mismos autores extienden el análisis a una red de topología arbitraria de servidores GPS, y relacionan los resultados GPS a redes en las cuales los nodos siguen la disciplina PGPS. En este segundo caso se parte de una red con una serie de valores fijados para los servidores, y un conjunto de sesiones sometidas a leaky bucket, y se intenta calcular cuál es el valor de retardo y de trabajo pendiente en el peor de los casos para cada una de las sesiones del conjunto. Para paquetes pequeños, como es el caso de ATM, el comportamiento de los sistemas PGPS y GPS es prácticamente idéntico.

Parekh en [2] estableció varias relaciones entre un sistema fluido GPS y su correspondiente sistema de paquetes WFQ que se pueden resumir en lo siguiente:

- En términos de retardo, un paquete acabará servicio en un sistema WFQ después que en el correspondiente sistema GPS, pero en no más tiempo de transmisión que el de un paquete del tamaño máximo.
- En términos del número total de bits servidos por cada sesión, un sistema WFQ no es peor que su correspondiente sistema GPS en más de un paquete del tamaño máximo.

Según lo anterior parece entenderse, y así se ha extendido la creencia, que la disciplina de paquetes WFQ y la disciplina fluida GPS ofrecen casi idéntico servicio diferenciándose únicamente en un paquete. Pero [5] se encarga de aclarar esta confusión ya que, aunque se puede probar que WFQ no cae en más de un paquete por debajo de GPS, sin embargo, WFQ sí que puede tener un comportamiento muy alejado del de GPS si lo expresamos en término de número de bits servidos por una sesión. Estas diferencias demuestran que WFQ puede acabar teniendo un comportamiento poco eficiente y [5] propone un nuevo y mejorado algoritmo de aproximación a GPS para la gestión de paquetes que llama Worst-case Fair Weighted Fair Queueing (WF^2Q y WF^2Q+) que ofrece casi idéntico servicio que GPS con una diferencia máxima de un paquete, y comparte el límite de retardo y las propiedades de justicia de GPS.

WF^2Q y WF^2Q+ son descritos en las referencias [10,11] y son presentados por sus autores como los algoritmos PFQ más precisos y, para ambos, se propone una mejora en [5], particularizada en ATM, en la que se consigue una mejora de la complejidad total, reduciendo la complejidad de las operaciones básicas de las colas además de los costes relacionados con las funciones de tiempo virtual. La idea de esta propuesta es la de alcanzar los siguientes objetivos:

- Soportar un gran número de VCIs con diferentes requerimientos de ancho de banda.
- Trabajar a altas velocidades del estilo de OC-3 o superiores a 155,52 Mbps.
- Mantener las propiedades de GPS en cuanto a límites de retardo, justicia y justicia en el peor de los casos.

Aunque WFQ es el algoritmo más conocido y reconocido como mecanismo de planificación ideal en términos de sus propiedades combinadas de límite de retardo y proporcionalidad de justicia, varios trabajos demuestran que la complejidad asintótica del algoritmo crece linealmente con el número de sesiones atendidas por el planificador, lo que limita su uso en aplicaciones de elevada velocidad. De este modo en [12] se proponen dos algoritmos de complejidad constante $O(1)$ en gestión de las marcas de tiempos (pesos) y además, conservan los mismos valores de retardo extremo-a-extremo y tamaños de buffers de WFQ. El primer algoritmo FFQ (Frame-based Fair Queueing) emplea un mecanismo de tramas para recalibrar periódicamente una variable global rastreando la evolución de los trabajos en el sistema y limitando la injusticia en periodos cortos de tiempo determinados por el tiempo de las tramas. El segundo algoritmo SPFQ (Starting Potential-based Fair Queueing) realiza la recalibración en el límite de los paquetes.

Otra interesante alternativa es DRR (Deficit Round Robin) [7], que es un algoritmo que presenta una mejora de la implementación de WFQ. El esquema de gestión del buffer asume que cuando éste se llena, el

paquete de la cola más larga será tirado. DRR usa un algoritmo de encolado sofisticado y acaba consiguiendo un elevado grado de justicia.

RTA (Response-Time Analysis) [13] se propone como control de admisión de tráfico en tiempo real en redes ATM. RTA usa un sencillo mecanismo de prioridad de encolado que ofrece una clara separación entre ancho de banda y requerimientos de entrega. Esto da a RTA la posibilidad de mejorar la utilización del ancho de banda respecto a WFQ. Además, se compara RTA con CND (Calculus of Network Delays) que también emplea priorización en las colas.

Investigaciones posteriores se centran en la propuesta de entornos donde la computación de los límites de retardo y ancho de banda estén más controlados, al contrario de las propuestas GPS que se basan en garantías de QoS determinísticas que pueden ser consideradas como más conservadoras debido a los relajados límites que se aplican y que conducen a limitaciones de capacidad. De este modo se proponen en [14] varios algoritmos de CAC basados en sistemas con retardo y ancho de banda desacoplados.

Por último, RFQ (Rainbow Fair Queueing) [15] parte de los esquemas previos [16] como CSFQ que no emplean *per-flow* en el mecanismo justo para el reparto del ancho de banda entre los diferentes flujos. RFQ divide cada flujo en un conjunto de capas basadas en velocidades. Los paquetes de cada flujo son “marcados con un color”, de forma que los routers mantienen un umbral de color y se encargan de tirar todas las capas cuyo color excede del umbral.

4.4. PROPUESTA QPWFQ PARA TAP

A la vista del estudio de las propuestas ya comentadas podemos decir que WFQ, también denominado PGPS (Packet-by-packet Generalized Processor Sharing), aporta interesantes prestaciones en cuanto a retardo y justicia [3,4], sin embargo, su coste computacional y complejidad de implementación han impedido su implantación.

VirtualClock y SCFQ proponen un mecanismo de planificación de paquetes más simple que WFQ, aunque las tres propuestas usan la longitud de paquetes, el peso y el tiempo virtual como parámetros para la planificación de los paquetes que llegan a las colas de entrada de los conmutadores ATM.

Delay-EDD, Jitter-EDD y HOL-EDD se basan en la asignación de marcas de tiempos de servicio a las colas o a los paquetes que son enviados en función de esos tiempos de servicio.

Todos estos algoritmos de planificación de colas se basan en marcas de tiempos que se asignan a las colas o a los propios paquetes garantizando límites en los retardos mediante el tratamiento aislado del tráfico. Sin embargo, la introducción de las marcas de tiempos provoca los excesivos costes computacionales ya comentados que puede determinarse en $O(\log_2 N)$ donde N es el número de VPIs/VCIa a la espera en las colas. El coste logarítmico es aceptable cuando el número de conexiones no es elevado, pero cuando se multiplexan varios miles de conexiones en un único enlace ATM este coste se considera excesivo. Han aparecido propuestas con costes $O(1)$ pero, por una causa u otra, se ajustan mal a las redes de tecnología ATM. Otro inconveniente de los algoritmos basados en marcas de tiempos está en uno de los más importantes parámetros de QoS en las redes de tecnología ATM, que es la variabilidad de los retardos (*jitter*). Esto ha llevado a varios investigadores a indicar lo inadecuado de estos algoritmos desde el punto de vista de la compartición de recursos.

No obstante, la literatura presenta diversas variantes mejoradas sobre las propuestas anteriores. Así, por ejemplo, en [17] se presenta una variación de SCFQ con coste constante $O(1)$. En nuestro caso queremos centrarnos en una de estas propuestas, QLWFQ (Queue Length based WFQ) descrita por Ohba en [18] y donde se presenta un algoritmo *work-conserving* de coste constante $O(1)$ para la planificación de colas *per-VC* en redes de conmutación de paquetes de longitud fija como ATM. QLWFQ es compatible con FQ [1] y FIFO y se basa en la asignación de pesos a las colas y en el establecimiento de créditos para determinar el orden de atención de las células ATM en función de la comparación de la longitud de cada cola con su peso particular.

El algoritmo compara -a la llegada y partida de las células- la longitud de las colas con los pesos que cada una tiene asignados. QLWFQ es en realidad una simplificación de WRR (Weighted Round Robin) [19] consiguiendo un mejor equilibrio entre aislamiento de tráfico y compartición de recursos que los algoritmos basados en marcas de tiempos. Ohba demuestra unos aceptables límites de retardo e interesantes índices de justicia a través de simulaciones de escenarios de elevada y de baja carga de tráfico.

En nuestro caso proponemos el algoritmo QPWFQ (Queue PDU Weighted Fair Queueing), inspirado en QLWFQ, para conseguir aportar un tratamiento justo a las PDU que llegan a los conmutadores AcTMs. Dado

que debemos dar un tratamiento privilegiado a las PDU pertenecientes a las conexiones con GoS, el mecanismo de pesos de colas es de suma utilidad para nosotros. Además, nos encontramos con otra particularidad no tratada por ninguno de los algoritmos estudiados que es el hecho de tener que dar respuesta a las retransmisiones de PDUs congestionadas. Esto nos va a determinar el tener que incluir en el algoritmo QPWFQ un tratamiento prioritario para las retransmisiones.

A continuación vamos a describir el funcionamiento general del algoritmo QPWFQ y, a un tiempo, comentaremos el mecanismo aplicado a las colas de entrada de los conmutadores AcTMs para demostrar la justicia del mismo, así como su sencillez de implementación que acaba resultando en un coste constante $O(1)$. Igualmente, veremos que QPWFQ procesa, tanto células como PDU, y describiremos la sencilla técnica usada para mantener las características *work conserving* de WFQ, así como la posibilidad de tratamiento prioritario de las solicitudes de retransmisiones de PDUs pertenecientes a las conexiones garantizadas.

La *Figura 4.3* ilustra el comportamiento de QPWFQ que es gestionado por el agente programable WFQA⁴. En esta figura podemos observar cómo las fuentes de tráfico llegan al agente WFQA a través del agente CoSA que se encarga de aplicar las características de tráfico negociadas para las conexiones. Supondremos que el tráfico de llegada es de células nativas ATM (aunque igualmente podrían ser PDU si suponemos estar en el conmutador de inicio de la conexión). Todo el tráfico es controlado por CoSA que se encarga de detectar las conexiones privilegiadas y comenzar el reensamblado de las células pertenecientes a cada una de ellas. CoSA sirve las PDU o las células a su correspondiente cola de llegada que es donde después serán tratadas por el algoritmo QPWFQ.

Como puede observarse en la *Figura 4.3*, tenemos tres colas de espera, cada una de ellas con su correspondiente peso p . En nuestro caso la Cola 1 tiene peso 3, la cola 2 peso 2 y la cola 3 peso 1. Las posiciones de cada cola pueden almacenar, tanto PDU como células para ser procesadas hasta el buffer del conmutador. Cuando una PDU, o una célula, llega a una cola, la longitud l de esta cola es incrementada en una unidad y es introducido el número de cola en la *cola de turnos* siempre que $l \leq p$.

Cuando la cabecera de una cola de espera es servida, la longitud de esta cola es decrementada en una unidad y se comprueba si $l \geq p$ para seguir atendiéndola, siempre y cuando en las restantes colas no se cumpla $l \leq p$. De este modo se garantiza la característica *work-conserving* (el servicio no se detiene mientras exista tráfico que procesar), que permite atender colas con tráfico continuado siempre y cuando en las restantes colas no exista tráfico.

En el escenario de la *Figura 4.3* hemos supuesto que en el instante 0 en la cola 1 había una PDU, la cola 2 estaba vacía y en la cola 3 existía otra PDU. Así, la longitud de la cola 1 se incrementa en 1, y en la *cola de turnos* se ha introducido el crédito 1 (suponiendo que atendemos las colas circularmente y de forma ascendente) ya que su peso es 3 y la longitud de la cola en ese instante es 1 ($l \leq p$). Posteriormente es tratada la cola 3, su longitud es incrementada en uno, y es anotado su turno en la *cola de turnos* ya que su peso es 1, lo mismo que su longitud ($l \leq p$). Si suponemos mantenernos en esta situación sin servir las cabeceras de las colas 1 y 3 a la salida, y en ese momento llegan tres nuevas unidades a la cola 1, éstas serán atendidas de forma continuada por la característica *work-conserving*, ya que a las colas 2 y 3 no llega tráfico. De este modo se atienden dos nuevas entradas de la cola 1 pero al ser atendida la tercera llega una célula a la cola 2 que pone su longitud a 1 y es pasado su turno a la *cola de turnos* ya que $l=1 \leq p=2$.

El procedimiento de salida de las células o PDU de las colas de espera funciona de la siguiente forma. La cabecera de la cola de turnos indica la cola de espera que va a ser atendida en cada momento. Volviendo a la *Figura 4.3*, se observa cómo es el turno de atender la cola de espera 1, entonces lo que se hace es enviar la cabecera de la cola 1 hasta el buffer, a continuación se decrementa en 1 la longitud de la cola 1, se elimina la cabecera de la *cola de turnos* y de la cola 1, para pasar a comprobar primero si $l \leq p$ y después si $l \geq p$.

En el escenario descrito no hemos considerado aún la posibilidad de llegada a las colas de espera de PDU pertenecientes a solicitudes de retransmisión de conexiones privilegiadas. Cuando esto ocurre, el agente WFQA realiza un tratamiento privilegiado de estas retransmisiones para darles prioridad en el servicio, tanto con respecto al resto de PDU de esa misma conexión como con respecto al resto de conexiones. Es decir, cuando a una cola de espera llega una PDU retransmitida se prioriza con respecto a las PDU que ya estuvieran en la cola de espera de su propia conexión y, además, también se priorizará su cola de espera (en la *cola de turnos*) con respecto al resto de colas de espera. Para implementar esta posibilidad recurrimos también a un sencillo mecanismo que mantenga el coste constante ya comentado y sin necesidad de usar excesivas variables para su implementación.

⁴ El sistema multiagente es descrito en el Capítulo 6.

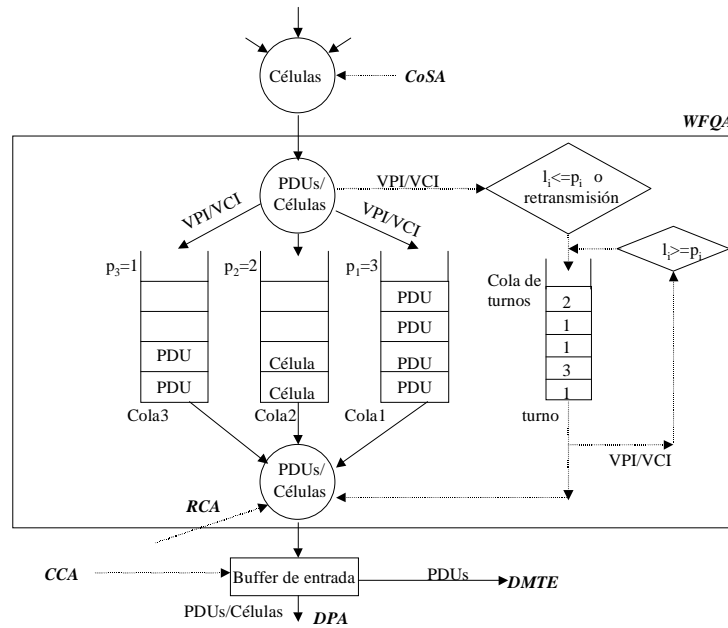


Figura 4.3. Esquema de planificación del algoritmo QPWFQ

El WFQA tendrá conocimiento de la inminente llegada de una PDU retransmitida a través de la comunicación del agente RCA que le indica el índice VPI/VCI/PDUid/Port. Esta comunicación activa una nueva variable asociada a cada cola para saber que va a llegar una retransmisión. Cuando llega la PDU es introducida en su correspondiente cola y es apuntada para garantizar el acceso directo. De este modo, las colas de espera pueden servir células y PDU, no sólo desde las cabeceras, sino también desde los punteros que identifican las PDU retransmitidas. Así, las colas FIFO, implementadas con punteros, permitirán priorizar las PDU retransmitidas que, a medida que van llegando, son insertadas, no al final de las colas, sino al principio de las mismas con la intención de garantizar el coste constante.

Pero para poder realizar la priorización también será necesario actuar en la *cola de turnos* para priorizar la cola a la que acaba de llegar la PDU retransmitida. Para poder realizar esta última operación es necesario comprobar si se trata de una retransmisión para poder acceder a la cola de turnos. Cuando se cumple esta condición el número de cola es priorizado en la *cola de turnos* siendo colocada en la cabeza de la misma.

Aunque las colas pueden contener células nativas ATM y PDU privilegiadas y, aparentemente esto puede conducir a situaciones injustas en las células respecto a las PDU, la estrategia aplicada, basada en los pesos y las longitudes de las colas, permite que no se produzcan situaciones de inanición de paquetes pequeños respecto a los de mayor tamaño.

En capítulos posteriores es descrito detalladamente el algoritmo QPWFQ así como su funcionamiento sobre varios escenarios de simulación.

4.5. CONCLUSIONES

En la actualidad, el control de congestión se delega en protocolos que las resuelven mediante retransmisiones extremo-extremo. Esta es una técnica sencilla de implementar a altas velocidades que también simplifica los conmutadores, pero toda la red se ve sobrecargada con las retransmisiones y tampoco aporta protección contra fuentes de tráfico egoístas.

Hemos revisado diferentes esquemas de planificación de paquetes inspirados en el método de tráfico continuo GPS, y hemos podido identificar sus ventajas e inconvenientes. Los esquemas de ancho de banda justo protegen a las fuentes bienintencionadas de las que no se comportan adecuadamente, y permiten un amplio y variado conjunto de mecanismos de control de congestión extremo-extremo. La clave de los algoritmos descritos en la literatura es su propia dificultad de implementación debido a la complejidad de mantener las colas de prioridad y la computación de la función de tiempo virtual. Esta complejidad crece con el incremento del número de sesiones que son procesadas pero, en nuestro caso, esta cuestión se ve amortiguada por el hecho de que las conexiones privilegiadas que van a disponer de GoS constituyen un número reducido y controlado.

Nuestra propuesta está basada en QLWFQ por las atractivas ventajas que aporta este algoritmo. No obstante, las características propias de nuestra arquitectura nos llevan a proponer el algoritmo QPWFQ para soportar los siguientes aspectos básicos inherentes a TAP:

- En nuestro caso, la unidad de procesamiento en las colas de entrada a los conmutadores ActMs son, tanto PDUs, como células nativas ATM. Este aspecto en realidad no afecta a las características de planificación, ya que aprovechamos los tiempos de espera en las colas para realizar el reensamblado de las células ATM y obtener las PDU que son la base de funcionamiento del protocolo TAP.
- En segundo lugar, las colas de espera van a requerir mayores tamaño de memoria por la significativa diferencia entre el tamaño de las PDU y las células ATM nativas.
- En tercer lugar, una de las prestaciones más importantes de TAP es el mecanismo de retransmisiones de PDU que no es soportado en QLWFQ. Para nuestra propuesta es vital la búsqueda del equilibrio entre la justicia en el procesamiento del tráfico, y la atención privilegiada de las solicitudes de retransmisión de PDU. Por tanto, QPWFQ mantiene los índices de justicia y el coste computacional constante soportando la priorización de las retransmisiones.
- Por último, y como aspecto más destacable, ante la posibilidad de soportar la solicitud por parte de los usuarios, o de protocolos de control de flujo de cambios dinámicos en los pesos de las colas, QPWFQ es parte de un agente programable de nuestra arquitectura. Este planteamiento implica el establecimiento y control de las comunicaciones del agente WFQA con el resto de agentes del sistema multiagente propuesto para el soporte de TAP.

REFERENCIAS

- [1] J. Nagle, "On packet switches with infinite storage," *IEEE Transactions on Communications*, pp. 435-438, (1987).
- [2] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *Proceedings of ACM SIGCOMM'89*, pp. 3-12, (1989).
- [3] Abhay Parekh and Robert G. Gallager, "A generalized processor sharing approach to flow control - the single node case," *ACM/IEEE Transactions Network*, (1993).
- [4] Abhay Parekh and Robert G. Gallager, "A generalized processor sharing approach to flow control in integrated service networks: the multiple node case," *IEEE/ACM Transactions on Networking*, pp. 137-150, (1994).
- [5] J. C. R. Bennet and H. Zhang WF²Q: Worst-case Fair Weighted Fair Queueing," *Proceedings of IEEE INFOCOM'96*, pp. 120-128, (1996).
- [6] S. J. Golestani, "A Self-Clocked Fair Queueing scheme for broadband applications," *Procs. IEEE 13th INFOCOM*, pp. 636-646, (June 1994).
- [7] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," *Proceedings SIGCOMM'95*, pp. 231-243, (1995).
- [8] D. Lin and R. Morris, "Dynamics of random early detection," *Proceedings of ACM SIGCOMM'97*, pp. 127-137, (1997).
- [9] S. Floyd and V. Jacobson, "Random early detection for congestion avoidance," *IEEE/ACM Transactions on Networking*, pp. 397-413, (1993).
- [10] Jon C. R. Bennett, D. C. Stephens and Hui Zhang, "High speed, scalable, and accurate implementation of packet fair queueing algorithms in ATM networks," *Procs. International Conference on Network Protocols*, pp. 7-14, (Oct. 1997).
- [11] Jon C. R. Bennet and Hui Zhang, "Hierarchical packet fair queueing algorithms," *IEEE/ACM Transactions on Networking*, pp. 675-689, (Oct. 1997).
- [12] D. Stiliadis and A. Varma, "Efficient fair queueing algorithms for packet-switched networks," *IEEE/ACM Transactions on Networking*, pp. 175-185, (April 1998).
- [13] H. Hansson and M. Sjodin, "Response time guarantees for ATM-networked control systems," *Procs. IEEE International Workshop on Factory Communication Systems*, pp. 213-222, (Oct. 1997).
- [14] R. Szabo, P. Barta, F. Nemeth, J. Biro and C.-G. Perntz, "Call admission control in generalized processor sharing (GPS) schedulers using non-rate proportional weighting of sessions," *Procs. IEEE 19th INFOCOM*, pp. 1243-1252, (March 2000).
- [15] Zhiruo Cao, Zheng Wang and E. Zegura, "Rainbow fair queueing: fair bandwidth sharing without per-flow state," *Procs. 19th INFOCOM*, pp. 922-931, (March 2000).

- [16] Ion Stoica, Scott Shenker and Hui Zhang, "Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocation in High Speed Networks," *Proceedings of SIGCOMM'98*, (1.998).
- [17] J. L. Lexford, A. G. Greenberg, and F. G. Bonomi, "Hardware-Efficient Fair Queueing Architectures for High-Speed Networks," *Proceedings IEEE INFOCOM*, pp.638-646, (1996).
- [18] Yoshihiro Ohba, "QLWFQ: A Queue Length Based Weighted Fair Queueing Algorithm in ATM Networks", *INFOCOM'97, Proceedings IEEE*, pp. 566-575 vol.2 (1997).
- [19] M. Katavenis, S. Sidiropoulos, and C. Courcoubetis, "Weighted Round-Robin Cell Multiplexing in a general-purpose ATM switch chip," *IEEE JSAC*, pp. 1265-1279, (1991).
- [20] N. R. Figueira and J. Pasquale, "An Upper Bound on Delay for the VirtualClock Service discipline," *IEEE/ACM Transactions Networking*, pp. 399-408, (1995).
- [21] H. Zhang and S. Keshav, "Comparison of Rate-Based Service Disciplines," *Proceedings ACM SIGCOMM*, pp. 113-121, (1991)
- [22] M. Vishnu and J. W. Mark, "HOL-EDD: A flexible Service Scheduling Scheme for ATM Networks," *Proceedings IEEE INFOCOM*, pp. 647-654, (1.996).