CAPÍTULO 10

DESCRIPCIÓN DETALLADA DE LA ARQUITECTURA DISTRIBUIDA Y MULTIAGENTE

10.1. VISIÓN GENERAL

En la *Parte I* de esta tesis hemos tenido la ocasión de revisar los trabajos relacionados con nuestras investigaciones donde, además, hemos adelantado los aspectos más importantes de nuestras aportaciones en cada uno de los capítulos correspondientes. En la *Parte II* hemos presentado nuestras motivaciones generales y también hemos identificado las limitaciones de la tecnología ATM con las que nos enfrentamos para conseguir la GoS donde, también, hemos destacado las soluciones que proponemos para conseguir nuestros objetivos. En este capítulo realizaremos la descripción detallada de la arquitectura TAP para explicar sus componentes principales así como su funcionalidad y también el funcionamiento de los conmutadores activos AcTMs que la soportan.

Así, en este primer apartado del *Capítulo 10* retomamos la introducción realizada en el *Capítulo 6* para obtener una visión general sobre la arquitectura de los AcTMs que será sucesivamente ampliada en los apartados siguientes del capítulo, donde serán justificados cada uno de los componentes, desde el punto de vista de las ventajas que aportan, así como desde las implicaciones que supone su inclusión en los conmutadores. Recordamos que las dos tareas principales de un conmutador ATM, o de un nodo *cross-connect*, son la traslación de los valores VPI/VCI de entrada a los correspondientes valores VPI/VCI de salida y el transporte de las células desde sus puertos de entrada hasta sus respectivos puertos de salida.

Nuestra propuesta de arquitectura se basa plenamente en el modelo arquitectónico propuesto [1] para la tecnología ATM, de forma que nuestro principal interés es soportar el tráfico ATM nativo y también el procedente de otros protocolos tan extendidos como TCP. Por esto, la arquitectura TAP aprovecha las posibilidades que le aportan los propios estándares para aportar soluciones que esos estándares no ofrecen a la propia tecnología. La *Figura 10.1* intenta aportar la visión más general de TAP aprovechando el propio modelo ATM, de forma que se vea de forma clara la situación de cada uno de los bloques que componen la arquitectura. Así, podemos comprobar cómo hemos sustituido la capa AAL-5 por nuestra EAAL-5 para aprovechar las ventajas que ésta nos aporta como ya hemos discutido. Las capas ATM y Física permanecen intactas; sin embargo, sobre la capa EAAL-5 se sitúa el protocolo TAP que está formado por un conjunto de algoritmos que permiten obtener nuestros objetivos. Por encima de TAP podemos encontrar alguno de los protocolos clásicos de capas superiores, aunque éstos no serían necesarios en el caso de emplear tráfico nativo ATM. La *Figura 10.1* nos permite observar también la posición que ocupa el subsistema SMA-TAP que abarca desde el Plano de Gestión hasta el Plano de Control del modelo tridimensional de ATM.

La arquitectura está formada por los aspectos que destaca la *Figura 10.1* y, además, debe estar soportada sobre los conmutadores activos AcTMs que deben estar específicamente equipados con los bloques hardware y código software que vamos a describir en los siguientes apartados. En líneas generales el software constituye el protocolo TAP en su conjunto, que es la unión del código que implementa cada uno de los agentes software y que controlan cada uno de los bloques hardware de los conmutadores que soportan el protocolo y, por tanto, la arquitectura completa. Sabemos también ya que los nodos extremos de la comunicación deben soportar completamente la arquitectura, mientras los conmutadores activos únicamente necesitan disponer de capa Física, ATM y parte de EAAL-5 para soportar la identificación de las PDU.

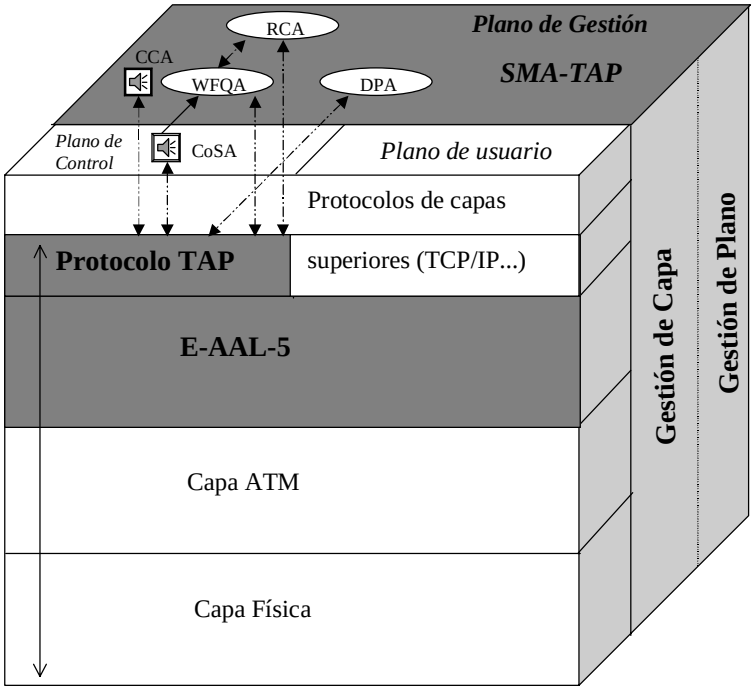

Figura 10.1. *Arquitectura TAP sobre el modelo arquitectónico estándar ATM*

Por otro lado, sabemos también que proponemos una VPN, por lo que no es obligatorio que todos los nodos que componen la red privada soporten la arquitectura TAP, para que se pueda disponer de GoS en las conexiones. Por tanto, los nodos que no implementen TAP podrán tratar el tráfico de la forma tradicional, sin la posibilidad de recuperar congestiones de forma local, pero sí de transferir las células RM estándares entre nodos AcTMs, para lo cual sólo necesitaremos adaptar levemente las labores de la señalización del estándar.

Tenemos ya una noción de los problemas que deseamos resolver con nuestra propuesta, por lo que debemos destacar que hemos incluido componentes específicos en la arquitectura para enfrentarnos a esos problemas. En líneas generales, podemos identificar (en la *Figura 10.2*) los siguientes bloques en la arquitectura de los nodos activos de la red, cada uno de los cuales intenta solventar problemas concretos de los identificados en el *Capítulo 9*:

- Colas de entrada, equipadas con un mecanismo justo de control de congestión que permita caracterizar el tráfico de entrada a los conmutadores mediante asignación de pesos.
- Buffer del conmutador, sobre el que se multiplexa el tráfico de las conexiones provenientes de las colas de entrada. Este buffer está gestionado con mecanismos de control de congestión que evitan la fragmentación de las PDU. El adecuado tratamiento del buffer nos permite también evitar el *interleaving*.
- Memoria dinámica DMTE, que realiza las veces de buffer temporal en el que cada AcTMs almacena el tráfico con requerimientos de GoS y desde donde se realizan las retransmisiones en el caso de aparecer congestiones.
- Tablas de Entrada/Salida, que están asociadas a cada uno de los puertos de los conmutadores AcTMs con dos funciones concretas. Por un lado las Tablas de E/S se encargan de mantener la relación entre los datos de entrada y de salida de cada una de las conexiones para evitar la posibilidad de valores repetidos en los VPI/VCI en fuentes diferentes. Por otro lado, estas tablas nos sirven como índice de acceso a la memoria DMTE en las operaciones de retransmisión locales.
- Los cuatro componentes anteriores son controlados por el subsistema SMA-TAP que ya hemos comentado que actúa como un sistema multiagente en el que, mediante un grupo de cinco agentes software, se realiza la supervisión y control de toda la actividad del conmutador. En realidad, el SMA-TAP representa el componente software que hemos desarrollado para el control de la equipación hardware que se ha indicado en los cuatro puntos anteriores. La mayor parte de las ventajas de este subsistema ya han sido explicadas en el *Capítulo 6*, aunque tendremos ocasión en este capítulo de explicar la función de cada uno de los agentes y las ventajas de su inclusión para disponer de conmutadores programables con funciones activas en la VPN que proponemos.

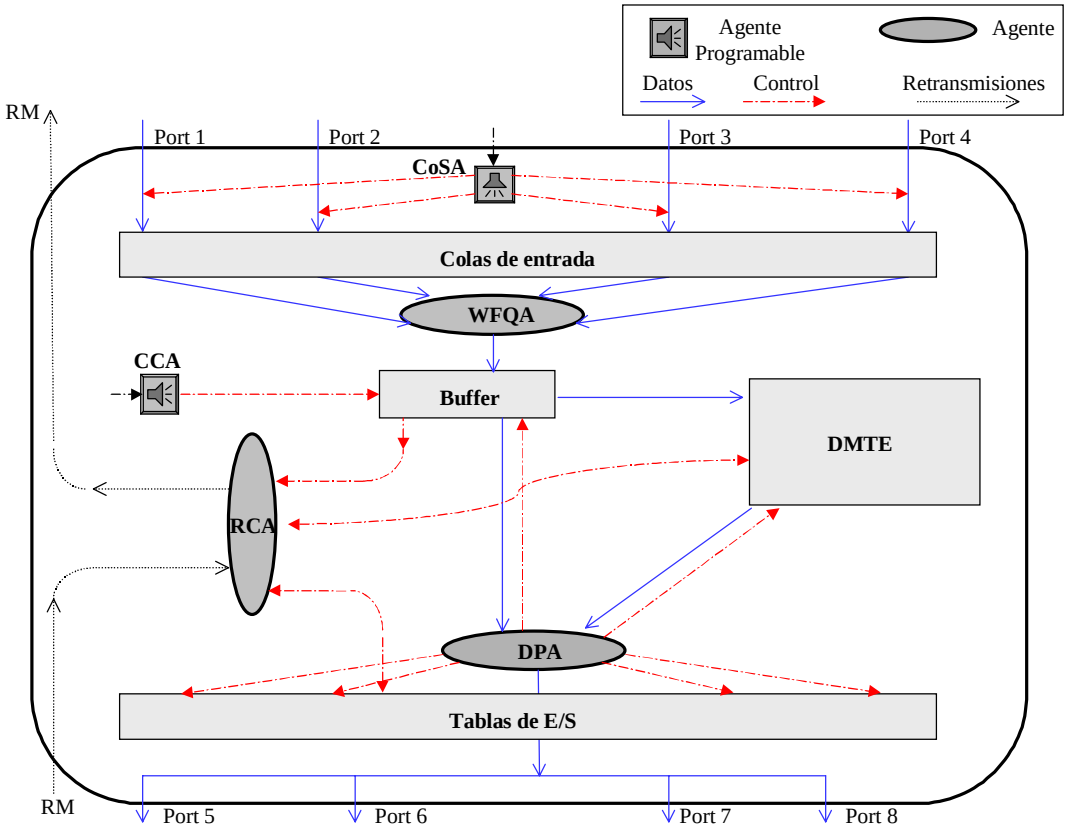

Figura 10.2. *Esquema de bloques de la arquitectura de los conmutadores AcTMs*

La *Figura 10.2* presenta una visión general de los conmutadores AcTMs con todos los componentes hardware anteriores así como cada uno de los agentes software que intervienen en la gestión del tráfico y control sobre cada uno de los componentes hardware. Podemos observar que el agente programable CoSA actúa sobre el tráfico de entrada que llega a las colas de entrada. Del mismo modo, el agente WFQA decide la política de atención de las colas de entrada en función de los pesos del tráfico para llegar al buffer del conmutador. El tráfico que llega al buffer es controlado según las características de los agentes CCA y DPA. Así el tráfico va dirigido a la memoria DMTE y a los puertos de salida. El agente DPA también controla los índices de las Tablas de E/S a medida que las PDU son enviadas a los puertos de salida de los conmutadores activos. Por último, el agente RCA genera y atiende las peticiones de retransmisión en caso de congestiones, teniendo relación también con las Tablas de E/S.

En las secciones que siguen describiremos cada uno de los módulos de la arquitectura del conmutador activo, destacando la relación entre cada uno de ellos, para pasar después a estudiar los flujos de datos y control dentro y fuera del conmutador.

10.2. CONMUTADORES ACTMS

Existen en la literatura múltiples arquitecturas de conmutadores ATM, aunque la mayor parte de ellas se basan en el uso de una matriz de conmutación donde confluyen las células de las conexiones de entrada que son conmutadas (tras asignarles sus correspondientes valores de VPI/VCI) a los puertos de salida. Sobre esta propuesta base de arquitectura se han realizado múltiples variaciones entre las que podemos situar TAP que, con la intención de solventar las problemáticas que ya conocemos, ha sido equipada con los módulos citados en el apartado anterior, los cuales van a ser seguidamente comentados en detalle.

10.2.1. COLAS DE ENTRADA

Los conmutadores que constituyen las redes ATM deben encargarse de mezclar los flujos de tráfico que provienen de fuentes diferentes y enrutarlos después hacia destinos distintos a través de *paths* de conmutación y enlaces de transmisión que las células de las diversas fuentes pueden compartir durante gran parte de su tránsito. Para que todo este proceso sea posible, los nodos de la red deben disponer de un almacenamiento temporal de las células en buffers de tamaño finito, de forma que el flujo de los datos en

cada momento pueda hacer que el tamaño de estas colas de almacén temporal puedan crecer y disminuir en su tamaño de forma dinámica. En realidad, este no es ningún mecanismo nuevo si lo comparamos con las redes de conmutación de paquetes clásicas, aunque lo realmente novedoso en el caso de ATM es la muy superior velocidad de conmutación (superiores a 622 Mbps en ATM frente a poco más de 256 Kbps en X.25). Estos requerimientos de elevada potencia de conmutación obliga a que cualquier propuesta que pueda hacerse para la labor de encolado del tráfico de entrada a los conmutadores deba ser optimizada en cuanto a los tiempos de acceso y de procesamiento.

Por tanto, las labores principales de los conmutadores de la red son las de ofrecer un almacén temporal de las células en tránsito hasta su destino (o al menos hasta el siguiente conmutador), reenrutar estas células desde este almacén al correspondiente puerto de salida del conmutador y, sobre todo, conseguirlo de la forma más eficiente y rápida posible, aunque en nuestro caso deseamos añadir una nueva prestación como la GoS para aportar esa fiabilidad que la red no aporta, pagando por ello un precio importante en prestaciones cuando el almacén temporal de las células experimenta las impredecibles congestiones. Veremos después que las técnicas de encolado en los nodos de la red juegan un papel fundamental en el diseño, operación y rendimiento de las redes ATM, por lo que existen una gran variedad de formas de solventarlo en los conmutadores, aunque los esquemas principales se dividen en las tres siguientes categorías (representadas en la *Figura 10.3*), que colocan las colas en los siguientes puntos de los conmutadores:

- En la entrada del conmutador, que se corresponde con la opción a) de la *Figura 10.3*, donde los buffers de células se sitúan a la entrada del conmutador. Si se emplean buffers FIFO, se producen colisiones cuando dos o más cabeceras de colas compiten simultáneamente por la misma salida, lo que provoca que una de las células pueda quedar bloqueada. Pero también quedan bloqueadas las células que siguen a la célula bloqueada, aunque no tengan la misma salida. Este tipo de desventaja puede solventarse con memorias RAM, aunque se requiere para ello un control de las colas más complejo. Sucesivamente se han ido proponiendo mejoras para este tipo de colas de entrada, y un ejemplo de ello es la propuesta que comentaremos en TAP.

- En la salida del conmutador, que se corresponde con la opción b) de la *Figura 10.3*, donde podemos observar que, en este caso, las colas de buffers se sitúan en los puertos de salida del conmutador. En este tipo, el elemento de conmutación consta de una matriz con buffers de salida y, sólo cuando la matriz opera a la misma velocidad que las líneas entrantes, se provocarán las colisiones; es decir, varias células compiten simultáneamente por el mismo puerto de salida. Este problema puede atacarse reduciendo el tiempo de acceso al buffer, o bajando la velocidad de la matriz de conmutación. En muchos casos el bloqueo interno se debe resolver con buffers adicionales.

- En los puntos de cruce (*crosspoints*) de la matriz de conmutación de los conmutadores, como puede observarse en el caso c) de la *Figura 10.3*. A este tipo de elementos de conmutación suele denominarse *butterfly* o de encolado central [2], y se caracterizan porque los buffers sólo se colocan en los puntos de cruce de la matriz, con lo que se previene que las células que compiten por puertos diferentes no se afecten mutuamente. Además, mediante un control lógico, puede elegirse qué buffer es el primero en el caso de que varias células o paquetes compitan por el mismo puerto de salida.

La situación c) de los buffer en los *crosspoint* supone disponer de una cola en cada uno de los puntos de cruce de la matriz de conmutación, lo que implica que hay que dividir el espacio de buffer total en N^2 colas diferentes, cuando podrían ser N como ocurre en las otras dos opciones citadas. Esta división de recursos de almacenamiento puede acabar provocando el incremento de la probabilidad de pérdida de células al incrementar el número de colas potencialmente congestionables, además de aumentar considerablemente la labor de gestión de las colas.

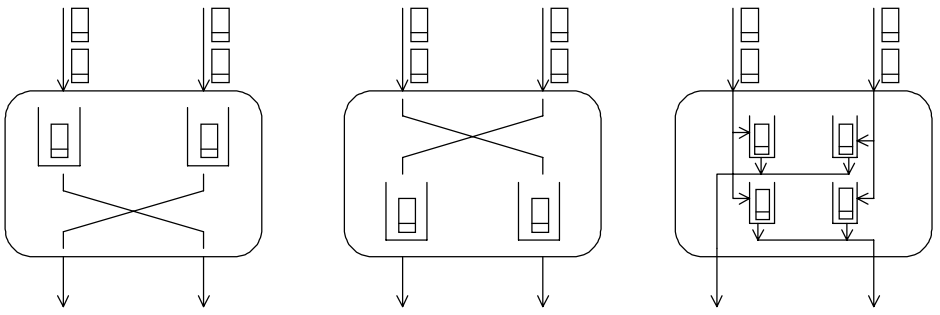

Figura 10.3. *Alternativas para situar las colas en los conmutadores*

La mayor parte de los estudios de estos tres diseños principales se inclinan por el uso de colas de entrada, aunque otros lo hacen por los de colas de salida si se combinan con la posibilidad de compartir el espacio de almacenamiento disponible entre todas las salidas [3]. Como sabemos ya, en nuestro caso hemos optado por el uso de colas de entrada para poder realizar un óptimo tratamiento de los flujos a la entrada de los conmutadores. Para nuestros intereses la gestión del tráfico a la entrada nos aporta importantes ventajas que han sido descritas a lo largo de toda esta tesis y, sobre todo, en el *Capítulo 4*. Si deseamos poder caracterizar el tráfico con pesos que además nos aporten la posibilidad de aplicar mecanismos justos de servicio de las colas, la mejor posibilidad es la de aplicar el buffering del tráfico en los propios puertos de entrada y antes de ser pasados por la matriz de conmutación. Además, la mayor parte de limitaciones de la tecnología y de las soluciones propuestas identificadas en el *Capítulo 9* apuntan a la posibilidad de poder tratar el tráfico en los puntos de llegada a los conmutadores. Debemos destacar que TAP aporta una novedad con respecto a las tres opciones representadas en la *Figura 10.3*, de forma que introducimos en la arquitectura de nuestros conmutadores un buffer (que puede ser identificado en la *Figura 10.2*) que es en realidad el punto de multiplexación del tráfico que llega desde las colas de entrada y que es donde solventamos las congestiones.

Una aspiración clave de TAP es la de aportar soluciones óptimas cuando aparecen congestiones en los conmutadores. Pues bien, con la intención de evitar la aparición de esas congestiones se propuso la utilización de colas de entrada que realizan las labores de buffer de entrada para el almacenamiento temporal de las células de entrada mientras son procesadas por la matriz de conmutación. La mayor parte de propuestas basan su trabajo en mecanismos de separación de flujos en las colas de entrada. Es decir, cada una de las conexiones dispone de una cola de entrada en la que se procesan separadamente células con valor de VPI/VCI diferentes, en un intento por realizar un tratamiento individualizado de cada una de las conexiones.

Como también hemos visto en capítulos anteriores resulta muy interesante la aplicación de técnicas que apliquen justicia en el procesamiento de las colas. El objetivo de estas técnicas es conseguir evitar las situaciones de inanición de las fuentes con menos requerimientos con respecto a aquellas que sean más ambiciosas en cuanto a los recursos de red que necesitan. Las técnicas de justicia son muy apropiadas en cualquiera de las situaciones, por lo que hemos decidido soportar esta posibilidad sobre TAP; sin embargo, hemos de destacar que nuestra principal aspiración es la de aportar GoS a una serie de conexiones privilegiadas, por lo que, en cierto modo, nos alejamos del tratamiento igualitario de las fuentes. No obstante, para soportar esta posibilidad y evitar el comportamiento inequitativo de las conexiones privilegiadas con respecto a las que no lo son, hemos decidido aplicar un mecanismo de atención de las colas basado en pesos. Los pesos de las conexiones están basados en dos parámetros importantes como son el PCR de cada fuente y también en la longitud de cada una de las colas de entrada. Por tanto, atenderemos las colas de entrada, no sólo en función de la velocidad pico de las fuentes, sino también en función del tamaño de todas las colas de espera del resto de fuentes. De este modo, conseguimos atender en primer lugar las fuentes con mayores requerimientos, pero a la vez conseguimos evitar la inanición de las fuentes más lentas. En nuestro caso debemos añadir a los mecanismos propuestos en la literatura dos nuevas características como son la atención prioritaria a las retransmisiones provocadas como consecuencia de las congestiones y, además, el soporte de unidades de procesamiento en forma de células o de PDU que se va a disponer en las colas. Esto quiere decir que al basar el mecanismo de justicia en el tamaño de las colas, el tener unas colas con PDU y otras con células individuales, tenemos que aplicar una proporcionalidad entre el parámetro de tamaño de las colas. En nuestro caso particular consideramos que el tamaño de las PDU es de 1.500 octetos, lo que da lugar a 32 células. Por esto, el tamaño de las colas está calculado en relación al valor proporcional entre PDU y células. La adaptación a PDU de más de 1.500 octetos es inmediata y únicamente requiere de la asignación de una cantidad mayor de memoria.

Cada esquema de gestión de tráfico requiere gestión de colas, y los diversos métodos de gestión de colas tienen diferentes efectos en el tráfico que fluye a través de las colas. Sabemos que los sistemas *Work conserving* (FIFO) envían las PDU cuando el conmutador completa el tiempo de servicio. Por tanto, el conmutador no permanecerá en estado de inactividad (*idle*) si hay PDU en cualquiera de las colas. Por otro lado, los esquemas *Non-work conserving* esperan un espacio de tiempo aleatorio antes de servir la siguiente PDU de las colas, incluso si hay PDU esperando en las colas. Mientras las redes de conmutación de paquetes usan control de flujo basado en ventana (FIFO), las redes de alta velocidad necesitan mecanismos basados en tasas de transmisión y usan servicios *work-conserving* y mecanismos como Fair Queueing (FQ). FQ espera *n-1* bits veces antes de enviar y tiene el problema de que cada fuente dispone de la misma fracción de ancho de banda. Sin embargo sabemos que Weighted Fair Queuing (WFQ) aporta una elevada garantía de rendimiento. Los algoritmos diseñados para conseguir asignaciones de ancho de banda justas ofrecen importantes ventajas para el control de congestión, pero su complejidad de implementación (planificación *per-flow*, gestión de buffer *per-flow* y clasificación *per-PDU*) es un obstáculo importante para su aplicación en redes de alta velocidad. En nuestro caso proponemos una variante de WFQ, con coste constante que actúa por delegación sobre el agente WFQA y cuyo funcionamiento describiremos siguiendo la *Figura 10.4*.

Para comprender el funcionamiento del algoritmo QPWFQ que hemos propuesto en el *Capítulo 4*, retomamos la *Figura 4.3* que es reinterpretada en la *Figura 10.4* en la que podemos observar 4 colas de entrada de nuestro modelo de AcTMs, en la cual podemos observar dos conexiones privilegiadas que generan PDU, y otras dos que generan células independientes y que usamos como tráfico de *background*[1] para estudiar el comportamiento del algoritmo QPWFQ que será retomado en el siguiente capítulo. Para comprender mejor el funcionamiento hemos etiquetado con números cada uno de los pasos que se dan en el procesamiento del tráfico.

Seguidamente describimos los apartados más importantes del flujo de datos y control dentro de las colas de entrada que, como observamos en la figura, está controlado por el agente WFQA. En la Figura hemos distinguido con trazos diferentes los datos, del control y de la petición de retransmisión, y hemos etiquetado cada una de las etapas para comentarlas seguidamente (el algoritmo será descrito en el *Capítulo 11*).

① Esta etiqueta intenta destacar el papel jugado por el agente programable CoSA en el tráfico de datos, para que podamos observar que al conmutador llegan los datos en forma de células que el propio agente de CoS se encarga de ensamblar en unidades de PDU antes que pasen a las colas de entrada para ser planificadas hasta el buffer, según lo que describimos a continuación.

② En este punto intentamos representar el punto de comunicación entre los agentes CoSA y WFQA, de forma que el flujo de datos acaba llegando a WFQA a través del propio agente de CoS. Como sabemos ya, cada una de las colas soporta el tráfico de VPI/VCI distintos, por lo que en este apartado se realiza la elección de la cola a la que va a parar cada una de las células en función de los valores de VPI/VCI que éstas traen en sus cabeceras y de los pesos asignados a las fuentes que se asocian también a la correspondiente cola de entrada que es la que soportará todo el tráfico de esa conexión.

③ Cada una de las colas tendrá un punto de acceso, de forma que las células o PDU se incorporan a cada una de sus correspondientes colas de entrada, tal que, una vez introducidas, debe incrementarse el tamaño actual de la cola, ya que el algoritmo que empleamos para la gestión de las colas se basa, no sólo en los pesos de éstas, sino también en su longitud que permite conseguir la justicia de forma que las colas de menor peso sean atendidas a medida que va creciendo su longitud.

④ En este punto se establecen las condiciones de atención de las cabeceras de las colas, para lo cual mantenemos una cola de turnos en la que se almacena el número de la cola que debe ser transferida hasta el buffer. Como podemos comprobar en la *Figura 10.4*, la condición principal de entrada en la cola de turnos es que la longitud de cada cola sea menor o igual que el peso de esa cola, o bien que se trate de una retransmisión que decidimos priorizar sobre el resto de transferencias. Al introducir cada PDU o célula en su correspondiente cola de entrada se comprueban estas dos condiciones, y si se cumplen, es introducido el número de la cola en la cola de turnos[2]. Así se consigue que las conexiones con un mayor PCR (peso) tengan un límite de entrada en la cola de turnos lo que evita que los VPI/VCI con menor velocidad sean relegados a favor de las fuentes más rápidas.

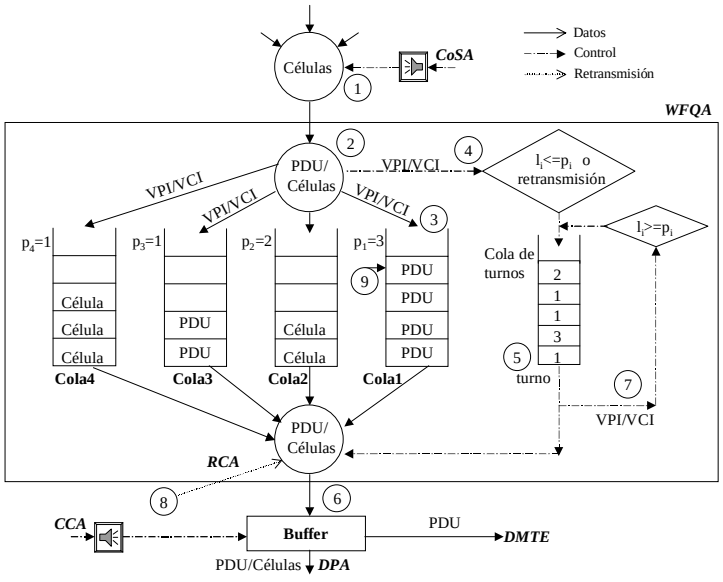

Figura 10.4. *Ejemplo de funcionamiento de las colas de entrada mediante el algoritmo QPWFQ*

[1] Empleamos fuentes de tráfico no privilegiadas adicionales para estudiar el comportamiento de sobrecarga en los conmutadores.

[2] Cuando $l_i > p_i$ se introduce la célula o PDU en su cola, pero no se pasa el número de cola a la cola de turnos.

⑤ En esta etapa se decide el número de la cola cuya cabecera va a ser procesada de modo que será transferida hasta el buffer del conmutador. En el caso de la Figura se indica que la cabecera de la cola número 1 será sacada de la misma, enviada al buffer y su espacio es liberado de la cola 1 y también de la cola de turnos, en la cual la siguiente cola en atenderse será la número 3.

⑥ Este punto indica la operación de transferencia desde las colas hasta el buffer, donde el algoritmo QPWFQ deja el control de la transferencia y pasa al mecanismo que gestiona el propio buffer que se haya indicado desde el agente CCA y que es descrito en apartados siguientes.

⑦ De este modo se garantiza la característica *work conserving* del mecanismo de control de las colas, de forma que, aunque el acceso principal a la cola de turnos especifica que la longitud de cada cola debe ser menor que el peso de esa cola para garantizar justicia, por este punto se permite el acceso a la cola de turnos, aunque la longitud de la cola sea mayor que el peso. Esto permite que, mientras existan paquetes en una cola pendientes de ser procesados, éstos puedan ser procesados sin tener que esperar el turno de otra cola en la que no existen paquetes que tratar.

⑧ Con esta etiqueta identificamos el punto por el cual el agente RCA notifica al WFQA que va a realizar una petición de retransmisión, de forma que, cuando ésta llegue, debe ser priorizada sobre el resto de transferencias pendientes de ser servidas. Este mecanismo de sincronización entre agentes facilita y optimiza la labor de retransmisión. El agente RCA transfiere el índice de la PDU que se va a retransmitir.

⑨ Una vez que el agente WFQA conoce por el agente RCA de la posible llegada de una PDU retransmitida, mantendrá el índice para identificarla rápidamente, y cuando llega la PDU esperada ésta es apuntada de forma específica para pasar a ser la cabecera de su cola correspondiente y ser priorizada con respecto a su cola y con respecto al resto de las colas.

Podemos intuir que el mecanismo que acabamos de describir consigue que los flujos de datos sean etiquetados en función de sus pesos (PCR para nosotros), evita que las fuentes rápidas releguen a las más lentas, garantiza que no se pierda tiempo de procesamiento mientras exista actividad en las colas y, sobre todo, garantiza un acceso constante en la atención de las colas de forma que podemos implementar eficientemente una variante del algoritmo WFQ. Esta sección ha mostrado sólo la forma de atender el tráfico, mientras en el siguiente capítulo tendremos la oportunidad de analizar el algoritmo que implementa esta importante sección de la gestión de las colas de entrada.

10.2.2. BUFFER

En nuestro prototipo de conmutador, el buffer es el punto más susceptible de congestiones, ya que la capacidad de almacenamiento de las colas de entrada es superior a la de éste, y el tráfico de las colas acaba multiplexándose en el buffer de entrada donde pueden acabar produciéndose congestiones en función de la velocidad de transmisión de las fuentes. Por esta causa, hemos decidido aplicar un mecanismo de actuación para detectar las congestiones del buffer, así como otras técnicas para evitar la fragmentación de las PDU y también el *interleaving* de los distintos flujos que acaban confluyendo en el buffer y que tienen el mismo puerto de salida. Como ya sabemos, la política de gestión del buffer es gestionada por el agente CCA que intenta controlar las congestiones aplicando el algoritmo EPDR (o cualquier otro que se desee ya que éste es un agente programable) y también el valor umbral del buffer que es de vital importancia para controlar las congestiones y para evitar la fragmentación de las PDU.

El dimensionado del buffer es una tarea que presenta una gran variedad de facetas que convierten a esta labor en realmente compleja. Así, la elección del tamaño del buffer depende de aspectos como: el número de fuentes simultáneas que vayan a soportarse; la velocidad máxima de cada una de las fuentes en particular y de las características diferentes de cada una de las fuentes de tráfico, de forma que las fuentes a ráfagas hacen bastante inviable la toma de decisión de la dimensión del buffer. Hay que destacar que, cuanto más grande elijamos el buffer, más se tarda en el acceso del tráfico a la red y viceversa. Por esto la elección de la dimensión tiene un relativo impacto sobre la función CAC de control de admisión. En general se asume que las aplicaciones sensibles al retardo requieren buffers de pequeño tamaño, mientras las aplicaciones sensibles a las pérdidas (para las que es adecuada TAP) tendrán mejor comportamiento con buffers de gran tamaño.

La CoS UBR, para la que principalmente se propone TAP, podría ser considerada dentro del tráfico a ráfagas, por lo que podemos enfrentarnos al problema de la elección del tamaño del buffer según los dos planteamientos más tradicionales [3]:

- Multiplexación sobre la velocidad disponible que consiste en restringir el número de fuentes con tráfico a ráfagas para que la velocidad total de entrada no exceda el CSR (Cell Slot Rate) total del enlace, y si lo exceden se supone que todo el exceso de tráfico se perderá.
- Multiplexación estadística compartiendo la velocidad, que asume que se dispone de un buffer de gran tamaño para soportar las células debidas al exceso de velocidad, de forma que sólo se perderá una porción de ellas mientras el resto de las que no se pierden experimentan retardos en el buffer.

En el caso de emplear un buffer pequeño, y apoyándonos en un modelo finito de cola M/D/1, puede calcularse el retardo máximo en la cola multiplicando la capacidad del buffer por el tiempo por slot de células, *s* en la velocidad apropiada del enlace. El retardo medio *d* en la cola *q* depende de la carga, ρ y es calculada usando la siguiente fórmula para un sistema infinito M/D/1:

$$d_q = s + \frac{\rho s}{2(1-\rho)} \tag{1}$$

De aquí puede deducirse que, para un sistema M/D/1 finito, el retardo medio puede quedar bastante ajustado ya que las pérdidas son muy bajas; es decir, los retardos sólo tendrán una diferencia apreciable en el caso que las pérdidas en el sistema finito sean muy elevadas, lo cual es bastante improbable. La teoría de colas demuestra que el retardo sólo incrementa a medida que nos acercamos al 80% de carga. Esto nos permite determinar que los buffers de pequeño tamaño son aplicables a redes que ofrecen la capacidad de transferencia de DBR (Deterministic Bit-Rate) y la capacidad de transferencia SBR (Statistical Bit-Rate) basadas en lo que antes hemos denominado como multiplexación sobre velocidad disponible.

En el caso de emplear buffers de gran tamaño para el tráfico a ráfagas, el dimensionado es mucho más complicado que en el caso anterior porque intervienen muchos más parámetros de caracterización de tráfico. En este caso suele asumirse un tráfico medio de muchas fuentes ON/OFF, cada una de ellas con las mismas características de tráfico (PCR, MCR y MBL en el estado activo). Los parámetros clave suelen ser el mínimo número de velocidades pico requeridas por ráfaga de encolado, N_0, el ratio de la capacidad del buffer para la longitud media de ráfaga, *X/b*, la carga media ρ, y la probabilidad de pérdida de células *CLP*. Por tanto, es necesario calcular el valor de *X/b* que es encontrado resolviendo la ecuación:

$$\frac{CLP_{fuente}}{CLP_{bsl}} = CLP_{bsd} = \exp\left[-N_0 \frac{X}{b} \frac{(1-\rho)^3}{4\rho+1}\right] \tag{2}$$

donde *bsl* es el factor *burst scale loss* y *bsd* el factor *burst scale delay*. Adaptando debidamente la fórmula anterior obtenemos la siguiente expresión para poder calcular la dimensión del buffer que necesitamos,

$$\frac{X}{b} = -\frac{4\rho+1}{(1-\rho)^3} \frac{\ln(CLP_{fuente} / CLP_{bsl})}{N_0} \tag{3}$$

En el caso de TAP vamos a requerir un buffer de gran tamaño, por las propias características del tráfico. Como acabamos de comprobar, es realmente complejo ajustar un tamaño acertado por la dificultad de caracterizar todos los parámetros del tráfico. Por esto, en nuestro caso particular, y apoyándonos en las consideraciones expuestas, hemos fijado el tamaño del buffer en 2.000 células. Este es un valor ampliamente usado en la literatura que permite equilibrar el efecto entre retardo en los accesos y la probabilidad de pérdida provocada por la velocidad de transferencia de las fuentes y por el número de fuentes que pueden acabar congestionándolo. En nuestro caso, la elección de 2.000 células como tamaño de buffer supone una zona de memoria de 106.000 octetos, lo que, usando PDU de 1.500 octetos, acaba dando que en el buffer pueden tener cabida un total de 70 PDU almacenadas en espera de ser servidas a sus correspondientes puertos de salida. Consideramos este valor como aceptable para poder obtener del algoritmo EPDR un mejor rendimiento, con una reserva de memoria razonable y asumible por cualquier conmutador en la actualidad.

10.2.3. MEMORIA DMTE

La memoria DMTE (Dynamic Memory Trusted EAAL-5 PDU) puede ser considerada como el módulo clave de la arquitectura TAP. Se comporta como una zona de memoria compartida en la que se copian las PDU en los conmutadores activos. Cuando una PDU de una conexión privilegiada llega al buffer, antes de ser aplicada la técnica del VC Merge es realizada una copia en la memoria DMTE. Por tanto, DMTE lo que hace es almacenar temporalmente las PDU generadas por la extensión de AAL-5 que hemos propuesto (EAAL-5) para la arquitectura. Una vez realizada la copia de cada PDU, éstas son enviadas a sus correspondientes puertos de salida. De este modo, estas copias de PDU contenidas en la DMTE pueden ser usadas para atender las peticiones de retransmisión en el caso que se produzcan congestiones en los conmutadores siguientes al AcTMs que ha realizado la copia.

Como podemos ver, por tanto, la DMTE es la que aporta al protocolo TAP la característica *trusted* que hemos justificado en el *Capítulo 3* y que acaba dando como resultado la GoS. Por la forma en que hemos diseñado la memoria, ésta permite almacenar varias PDU de cada conexión privilegiada, con la intención de aportar una mayor garantía de servicio en función de la cantidad de PDU que se almacenan de cada conexión. De todos modos, la propia característica de las PDU no nos permite disponer de un gran número de unidades de PDU en la memoria, porque, a medida que crece el número de conexiones fiables, el tamaño de la DMTE crece también considerablemente. Para evitar el crecimiento incontrolado del índice de ocupación de la memoria aplicamos un control sobre la misma de forma que las PDU de cada conexión permanecen en la memoria, sólo si se dispone de suficiente espacio de almacenamiento en la misma. Es decir, cuando se requiere almacenar una nueva PDU, se aplica un mecanismo de *aging* que permite sacar de la DMTE aquellas PDU de una conexión que llevan más tiempo en la memoria. Esta posibilidad es la que aporta la característica Dinámica a la memoria DMTE.

Debemos destacar que las PDU de AAL-5 pueden tener un elevado tamaña (hasta 65.535 bytes) y, dado el potencialmente alto número de conexiones (VPI/VCI) con requerimientos de GoS, el tamaño de la DMTE puede ser excesivo. Esta es la razón por la que hemos limitado el número de PDU que se pueden almacenar por cada conexión fiable como hemos comentado en el párrafo anterior. De este modo, sólo soportamos un número reducido de VCI privilegiados con transferencias garantizadas. Es evidente que el tráfico de una conexión es mucho más seguro cuando la DMTE almacena más PDU de esta conexión. Pero el tamaño de la DMTE también depende del tamaño de las PDU que, como sabemos, es variable y de considerable tamaño en su valor máximo como hemos visto antes. Por esta causa hemos calculado el tamaño que requerimos de memoria para poder garantizar un número concreto de conexiones con mayor o menor grado de confianza en cuanto a la GoS.

Podemos entender que el tamaño del la memoria DMTE es un importante factor en la arquitectura. Así, trabajamos con dos parámetros para dimensionar esta memoria que son el número de PDU almacenadas de cada conexión y, por otro lado, el tamaño de cada una de esas PDU. Se estudian a continuación las combinaciones de estos dos parámetros.

En el caso de usar las PDU de EAAL-5 con tamaño máximo requeriremos los siguientes tamaños de memoria si decidamos mantener en la DMTE 3 PDU almacenadas por cada conexión privilegiada:

3 PDU * 64 Kbytes cada PDU = 192 Kbytes * 10 conexiones con GoS = 1,9 Mbytes de DMTE.

3 PDU * 64 Kbytes cada PDU = 192 Kbytes * 20 conexiones con GoS = 3,8 Mbytes de DMTE.

Si consideramos un tamaño de PDU más razonable con 1.500 bytes, como es lo más lógico para el tráfico TCP que es una de nuestras motivaciones generales, tendremos los siguientes requerimientos de tamaño en DMTE:

3 PDU * 1,5 Kbytes cada PDU = 4,5 Kbytes * 10 conexiones con GoS = 45 Kbytes.

3 PDU * 1,5 Kbytes cada PDU = 4,5 Kbytes * 100 conexiones con GoS = 450 Kbytes.

3 PDU * 1,5 Kbytes cada PDU = 4,5 Kbytes * 1.000 conexiones con GoS = 4,5 Mbytes.

La *Tabla 10.1* muestra esquemáticamente algunos de los resultados obtenidos, donde podemos observar los resultados requeridos de memoria en función del número de conexiones que se desea privilegiar y el número de PDU de cada conexión a fiabilizar. Podemos ver así cómo aportar GoS a 1.000 fuentes de tráfico TCP sólo requiere de una memoria de 3 Mbytes que podemos considerar como un tamaño más que razononable y asumible en los conmutadores actuales.

TABLA 10.1
TAMAÑO REQUERIDO DE DMTE PARA OFRECER GOS A X CONEXIONES

Nº de PDU almacenadas por cada conexión	Tamaño de cada PDU	Nº de Conexiones con GoS	Tamaño de DMTE
3	64 Kbytes	10	1.9 Mbytes
3	64 Kbytes	20	3.8 Mbytes
3	1,5 Kbytes	10	45 Kbytes
3	1,5 Kbytes	100	450 Kbytes
3	1,5 Kbytes	1.000	4,5 Mbytes
4	1,5 Kbytes	1.000	6 Mbytes
2	1,5 Kbytes	1.000	3 Mbytes

Podemos comprobar por tanto cómo el grado de GoS que van a tener las fuentes privilegiadas depende del número de PDU que se almacenan en la DMTE. Vemos también cómo el grado de confianza deseado afecta también al tamaño de memoria consumida. Para nuestros planteamientos partimos de un tamaño de DMTE fijo, con el que se equipa a cada conmutador AcTMs. Así, con un tamaño fijo podemos determinar el número total de fuentes privilegiadas que podemos garantizar, o bien el grado de fiabilidad que desea aportarse. Debemos, por tanto, buscar un punto de equilibrio entre los tres factores: Tamaño de PDU, Número de fuentes con GoS y grado de fiabilidad deseado para cada fuente que depende del número de PDU de cada conexión que se almacena en la DMTE. Si acordamos que el grado de GoS (G) depende del tamaño de memoria DMTE con que equipemos a un conmutador podremos expresar este grado de la siguiente forma,

$$G = \frac{T}{\sum_{1}^{n} N * t} \tag{4}$$

donde T es el tamaño total de la memoria DMTE, n es el número de fuentes, N es el número de PDU que se almacena de cada fuente o VCI y t es el tamaño de cada una de esas PDU. Así, en el último ejemplo de la *Tabla 10.1* el grado de GoS G es 1, ya que T=3 Mbytes; tenemos 1.000 fuentes de las cuales se almacenan 2 PDU, de 1.500 octetos cada una. En cierto modo, y obviando otras características de las fuentes y del conmutador, el valor de G debería tener un valor mayor o igual a 1 para poder ofrecer a un número n de VCI su GoS. O dicho de otro modo, y si expresamos la fórmula (4) de otra forma, podemos obtener el grado de garantía de servicio g de la fuente i o VCI i en función de el número de PDU que deseamos almacenar y del tamaño de cada una de esas PDU,

$$g_i = \frac{T}{N * t} = \frac{3Mbytes}{2 * 1.500bytes} = 1.000 \tag{5}$$

La expresión (5) está calculada nuevamente sobre el último ejemplo de la *Tabla 10.1*, y podemos ver cómo el valor obtenido para la g coincide con el número de fuentes de la característica de la fuente i que podrían soportarse con una memoria DMTE de 3 Mbytes. Llamamos la atención sobre el hecho de que el parámetro g para las fuentes individuales ha de ser normalizado o entendido como que su valor ideal sería 1, ya que a mayores valores de N (y también de t) dispondremos de mayor GoS aunque el valor de g_i obtenga valores menores. Por tanto, mientras en el caso general de G lo deseable es obtener valores positivos cercanos a 1, en el caso de g_i lo razonable es obtener también valores positivos pero bastante alejados de la unidad, ya que en realidad lo que expresa es el número de VCI de iguales características que podrían soportarse con ese valor de g_i.

Los accesos a la DMTE son otro aspecto de importancia ya que debemos disponer de un mecanismo óptimo para poder localizar de forma eficiente las PDU en las solicitudes de retransmisión. Como podemos observar en la *Figura 10.2*, la copia de cada PDU es realizada en la memoria DMTE desde el buffer, justo antes de ser enviada a su correspondiente puerto de salida. Para poder localizar y acceder con rapidez a las PDU de la DMTE hemos decidido utilizar un índice que está formado por los valores *VPI/VCI/PDUid/Port_Entrada*. Con estos valores mantenemos la DMTE como una tabla de *hash* cuya función de *hashing* elimina las posibles colisiones en los accesos de escritura y lectura.

Por tanto, cada PDU dentro de la memoria es identificada por el VPI/VCI de la conexión a la que pertenece, su propio identificador de PDUid (que se corresponde con los campos UU y CPI de AAL-5 como ya sabemos) que las distingue con respecto al resto de PDU de su misma conexión y, además, hemos introducido el valor del puerto de entrada para distinguir la posibilidad de que a un mismo conmutador

accedan conexiones diferentes en las que se hayan elegido valores idénticos de VPI/VCI por provenir de usuarios diferentes que, en principio, no tienen porqué conocer los valores de identificación de las conexiones de otros usuarios. Esta probabilidad de coincidencia es relativamente alta, ya que existe la tendencia a elegir valores de VPI/VCI similares en el establecimiento de la conexión, por lo que hemos decidido pagar este precio para evitar la posibilidad de claves repetidas en la función de *hash* de la DMTE. Además de las colisiones en la función de *hash* también esto daría lugar a retransmisiones erróneas, de forma que se podrían retransmitir PDU de una conexión a otra por esa posibilidad de repetición en el índice.

La operación de escritura en la memoria es la que se encarga también de controlar el número de PDU que se ha decidido almacenar por cada una de las conexiones, por lo que aplica la citada técnica de *aging*, mediante la cual se evita (cuando no existe suficiente espacio de memoria) que una PDU entrante no tenga cabida. Es decir, mientras existe espacio libre de memoria se va a permitir la entrada de PDU, de forma que cuando el espacio comienza a ser insuficiente es cuando se eliminan las PDU cuyo PDUid es menor (bajo la suposición que son las que mayor tiempo llevan en la memoria) y por lo tanto son las menos susceptibles de requerir una retransmisión. Una situación extrema que puede describirse es la que se produce cuando al retransmitir una PDU desde la DMTE llega una nueva PDU desde el buffer con el mismo VPI/VCI y no existe espacio libre en la memoria, en este caso lo que se hace es descartar la PDU entrante de forma similar a la pérdida que se puede producir cuando se carece de buffers en *VC Merge*.

Lo razonable en los conmutadores activos es disponer de un espacio fijo y concreto de memoria que será repartido entre todos los VCI que se deseen fiables. Mediante un parámetro de tráfico puede expresarse el número de PDU que se desea almacenar de cada VCI, de forma que mediante el sistema multiagente que explicaremos en secciones siguientes, se pueda controlar este valor en los accesos a la memoria DMTE de cada uno de los conmutadores AcTMs. Es decir, el agente CoSA soporta un parámetro de tráfico que es el del grado de GoS deseado por la fuente, tal que el valor normalizado de la g_i expresa en cada conmutador la reserva máxima de DMTE que debe hacerse para cada VCI. Destacamos también que en nuestras simulaciones podemos determinar el tamaño deseado de DMTE, aunque este valor es el mismo en todos los conmutadores de la red. En el *Capítulo 12* tendremos ocasión de explicar las características del simulador de TAP, donde veremos cómo éste nos permite elegir, tanto los tamaños de DMTE, como el número de PDU y el tamaño de cada una de ellas. El objetivo final es el de conseguir el mayor número posible de conexiones garantizadas, o bien el de lograr el mayor grado posible de GoS para un número más reducido de fuentes. El simulador nos permitirá demostrar que es aceptable el tamaño de memoria que hay que aportar a los conmutadores para conseguir un aspecto tan importante como es el de las transferencias garantizadas que la propia tecnología ATM no es capaz de soportar con los actuales estándares.

10.2.4. TABLAS DE E/S

Como sabemos, a cada puerto de un conmutador AcTMs le corresponde una Tabla de Entrada/Salida que almacena los índices de acceso a la memoria DMTE. Estos índices nos permiten un acceso eficiente a través de la función de *hash* de la memoria DMTE en el caso de solicitudes de retransmisión. Pero, además, estas tablas desempeñan otra importante función que es la de evitar la repetición de valores VPI/VCI en conexiones diferentes. Como es conocido, en el establecimiento de la conexión los usuarios de ATM pueden especificar el VPI/VCI que desean usar en sus conexiones, por esto es posible que en un conmutador confluyan dos conexiones que provienen de usuarios diferentes que han decidido usar el mismo valor de VPI/VCI. Debemos buscar entonces un sistema que nos permita identificar las PDU de cada emisor que se han almacenado en la DMTE, ya que las dos conexiones pueden coincidir, no solo en VPI/VCI, sino también en el identificador de PDU, PDUid, y también en el puerto de entrada que son los valores que configuran el índice de acceso a la DMTE. Como los VPI/VCI también pueden cambiar de la entrada a la salida el sistema adoptado debe permitir identificar la secuencia para poder localizar las PDU en las retransmisiones.

El sistema de identificación de las PDU implementado se basa en el uso de las Tablas de E/S asociadas a cada uno de los puertos de salida de los conmutadores que permitan relacionar los valores de entrada de cada PDU con los valores de salida una vez que han sido conmutadas a la salida. Es decir, cuando una PDU es segmentada y enviada a su puerto de salida, en la Tabla de E/S de este puerto se almacena el siguiente índice,

InPort/VPIIn/VCIIn/PDUid/OutPort/VPIOut/VCIOut/OutPortPrev/InPortNext

El significado de cada uno de los campos que componen el índice de las tablas se explica a continuación:

- *InPort*: Representa el valor numérico del puerto de entrada de las PDU de cada una de las conexiones privilegiadas. Normalmente será un valor comprendido entre 1 y 16 (en el caso de conmutadores con 16 puertos) de forma que identifica la conexión física de cada conmutador por la que se va a realizar el transporte de los datos entre conmutadores.

- *VPIIn*: Este valor representa el identificador de camino virtual (VPI) con el que cada una de las conexiones privilegiadas llega a un conmutador. Este valor estará comprendido entre 0 y 255 en el caso de los nodos UNI, y entre 0 y 4.095 en el caso NNI como puede deducirse de la *Figura 1.1*. Este valor está íntimamente ligado al valor del puerto de entrada *InPort* anterior, ya que ésta es la vía física por la que el camino virtual se va a establecer en cada conmutador.

- *VCIIn*: Este campo identifica el circuito virtual (VCI) por el que se realiza el envío de las células entre conmutadores. Este valor está comprendido entre 0 y 65.535, tanto en los nodos UNI como NNI, de forma que cada VPI puede tener asociados, como máximo, hasta los 65.535 VCI por los que se pueden realizar otras tantas conexiones. Como es sabido, la conmutación de un VPI concreto permite la conmutación automática de todos los VCI que lleva asociados. El *VCIIn* indica entonces el valor con el que las células acceden a cada conmutador, dentro de un VPI y a través de un *InPort* concreto.

- *PDUid*: Este es el campo que identifica cada una de las PDU de una determinada conexión con GoS. Como ya sabemos, este valor toma valores entre 0 y 65.535 según se explica en la *Figura 9.1* y se corresponde con los campos UU y CPI de un octeto cada uno. Este campo permanecerá invariable e-e, por lo que no cambiará de valor al atravesar los conmutadores de la red. En realidad este es el campo más importante para poder identificar las PDU a retransmitir, pero sin los tres campos anteriores no será posible identificar PDU con mismo valor de PDUid pero pertenecientes a diferentes conexiones *Port/VPI/VCI*.

- *OutPort*: Con este valor se identifica el puerto de salida por el que las células de entrada de una conexión concreta con valores *InPort/VPI/VCI* acabarán llegando al siguiente conmutador en sentido hacia el destino. Puede tomar el mismo rango de valores que el *InPort* ya comentado.

- *VPIOut*: Identifica el valor del VPI de salida que la matriz de conmutación asignará a las PDU de una determinada conexión. Este valor, como el *VPIIn*, se asignará a cada una de las cabeceras de las células en las que se segmente cada PDU, con la característica que ese será el valor con el que llegarán las células al siguiente conmutador, de forma que el valor de *VPIOut* en el conmutador *n* será el mismo que el *VPIIn* en el conmutador *n+1*. El rango de valores que puede tomar es el mismo que en el caso de las entradas.

- *VCIOut*: Su función es la misma que la del campo anterior, sólo que en este caso se trata del valor del VCI de salida. Sus valores potenciales se corresponden con los del *VCIIn*, con la característica que ahora el valor del *VCIOut* del conmutador *n* coincidirá con el valor *VCIIn* del conmutador *n+1*. Tenemos por tanto ya la relación de los valores de entrada con los de salida; es decir, cada identificador de PDU fiable PDUid, llegará a un conmutador activo a través del camino indicado por la terna */InPort/VPIIn/VCIIn*[3] y será servido al siguiente conmutador a través del camino representado por */OutPort/VPIOut/VCIOut*.

- *OutPortPrev*: Este campo del índice de acceso de las Tablas de E/S identifica el número de puerto de salida del conmutador previo al actual por el cual salieron las células de una PDU para llegar hasta el conmutador actual. Es decir, si estamos en el conmutador *n*, el valor de *OutPortPrev* es el valor del puerto de salida del conmutador *n-1* (previo al *n*) desde el que se han enviado las células al *InPort* o puerto de entrada del conmutador *n*.

- *InPortNext*: Es similar al campo anterior, sólo que en este caso este valor indica el número de puerto de entrada por el que se va a acceder al siguiente conmutador. Es decir, si estamos en el conmutador *n*, el *InPortNext* indica el puerto de entrada por el que el conmutador *n+1* (siguiente a *n*) va a recibir las células de una PDU fiable y que el conmutador *n+1* identificará como el *InPort*. Estos dos últimos campos del índice son necesarios para que cada conmutador conozca la relación entre los números de los puertos de entrada y salida con sus conmutadores vecinos. Es decir, para nuestros objetivos necesitamos saber la correspondencia de cada uno de los puertos de salida del conmutador *n* con cada uno de los puertos de entrada del conmutador *n+1*. Del mismo modo, es necesario conocer la relación de cada uno de los puertos de entrada del conmutador *n* con los puertos de salida del conmutador *n-1*. Para ello requeriremos actuar sobre alguno de los aspectos de la señalización en la red, ya que es vital disponer de estas relaciones para poder resolver las solicitudes de retransmisión.

Podemos observar en la *Figura 10.5* la situación de las Tablas de E/S del puerto 6 con cada uno de los campos explicados. Observamos también la relación entre el índice de la Tabla con el índice de la DMTE, de forma que sólo coinciden en los valores de entrada de la PDU, dejando los valores de salida para relacionarse con el conmutador siguiente.

[3] Destacamos que el valor formado por /InPort/VPIIn/VCIIn/PDUid constituye el índice de acceso a la DMTE.

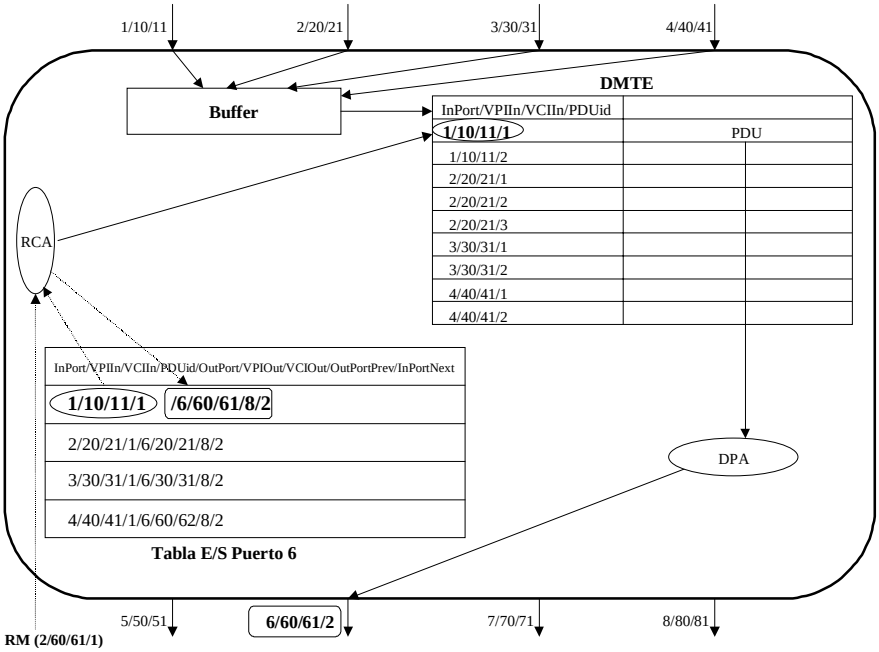

Figura 10.5. *Contenidos de las Tablas de Entrada/Salida de los AcTMs*

Como puede observarse en la *Figura 10.5*, a cada uno de los puertos de entrada del conmutador se ha asignado un VPI/VCI, de forma que, por ejemplo, por el puerto 1 accede la conexión cuyo VPI/CVI es el 10/11. Las PDU van llegando a la DMTE a través del buffer de modo que cada PDU del mismo VPI/VCI va aumentando en su número de secuencia. En la Tabla E/S del puerto 6 puede verse cómo la PDU 1 que proviene de la conexión VPI/VCI=10/11 a través del puerto 1, cuyo índice en la DMTE es 1/10/11/1, tiene en la Tabla el mismo comienzo de índice. Pero esta PDU va a salir por el puerto 6 asignando como VPI/VCI de salida el 60/61. El puerto por el que va a entrar en el siguiente conmutador es el 2 y el puerto de salida del conmutador anterior por el que llega al de la figura es el 8.

La *Figura 10.5* muestra también cómo desde el conmutador siguiente al representado se está pidiendo retransmitir, mediante una célula BRM, la PDU cuya identificación en ese conmutador es 2/60/61/1, es decir, ha entrado por el puerto 2, su VPI/VCI es el 60/61 y el PDUid es el número 1. Este índice parcial es recibido por el agente RCA que se encarga de usarlo para acceder a la Tabla en la que se localiza la PDU 1 cuyos datos de entrada son 1/10/11/1, que es el índice devuelto al agente RCA y que éste empleará para acceder a la DMTE y localizar los datos de la PDU que será retransmitida nuevamente por el agente DPA.

El mecanismo ideado es mantenido por el agente DPA que, antes de despachar las células de una PDU, se encarga de añadir al índice que se dispone de la entrada los nuevos valores de la salida, dando lugar a un nuevo índice que será empleado cuando a este conmutador llegue una célula BRM con el índice de retransmisión. Éste es el que se empleará como acceso a la Tabla de E/S para encontrar el valor del índice que nos sirve para poder luego acceder a la DMTE a localizar los datos de la PDU que se debe retransmitir.

Podemos observar que en realidad estas tablas de E/S se comportan a modo de tablas de *routing* ya que se encargan de mantener la relación de los datos con que entra cada PDU en los conmutadores con los datos de salida del mismo conmutador. Mediante esta relación podrá posteriormente encontrarse el índice de acceso a la memoria de retransmisiones y evitarse un problema más grave que la propia pérdida de una PDU que es la retransmisión de una PDU equivocada o perteneciente a una conexión diferente. Para analizar el comportamiento y explicar su funcionamiento disponemos de la *Figura 10.6* en la que presentamos una traza de los valores que toman las tablas de E/S en varias conexiones, empleando en este caso tres conmutadores AcTMs consecutivos que hemos etiquetado para explicar a continuación su funcionamiento.

① Podemos observar en esta etiqueta cómo en la Tabla E/S del puerto 6 se mantiene la relación de la información de las PDU a la entrada de cada conmutador en la DMTE con lo que se tiene a la salida del conmutador. Por ejemplo, por el puerto 1 VPI/VCI=10/11 recibimos la PDU 1 de esta conexión que después saldrá del conmutador por el puerto 6 con el VPI/VCI=60/61, y acabará llegando al siguiente conmutador por su puerto 2. Suponemos que esta PDU proviene del puerto 8 de salida del conmutador anterior. En la tabla lateral de la derecha anotamos estas relaciones, donde la PDU destacada en negrita ha experimentado una congestión en el buffer del segundo conmutador.

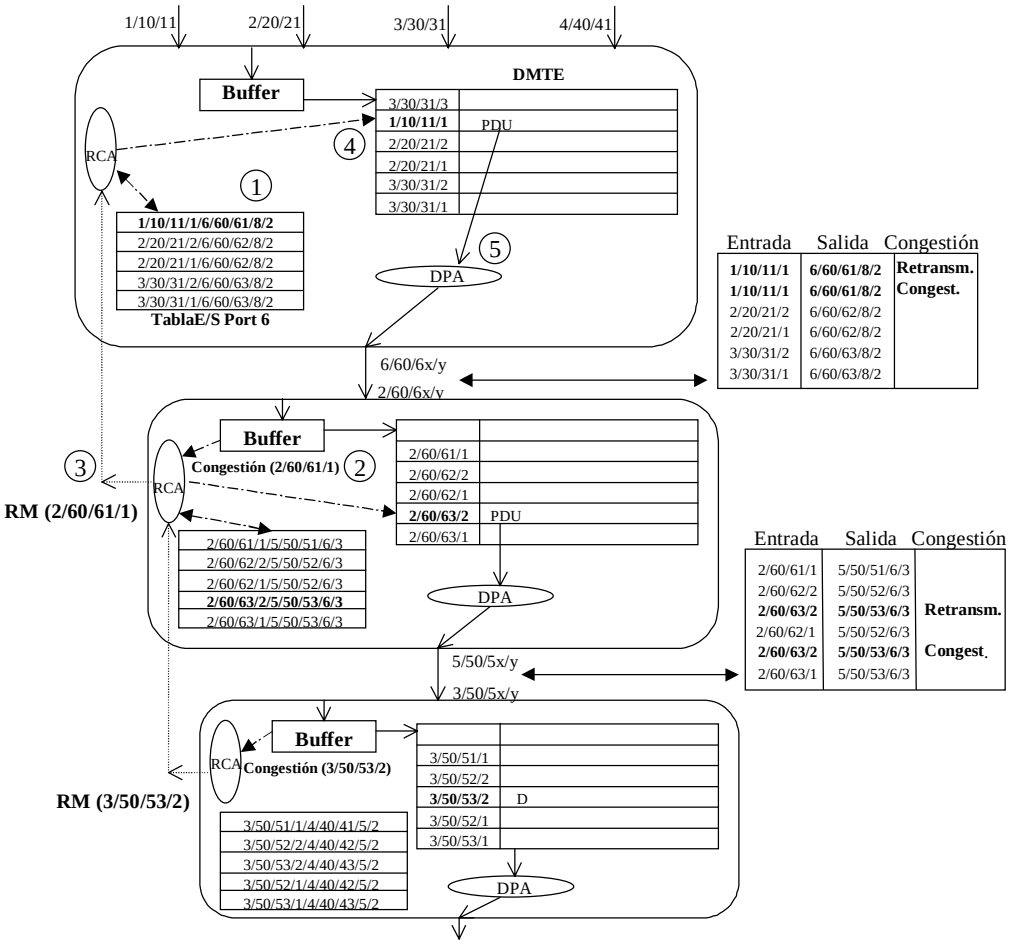

Figura 10.6. *Escenario para analizar las Tablas de E/S en tres AcTMs*

② En este punto es donde suponemos la congestión del buffer del segundo conmutador con la PDU de índice de entrada 2/60/61/1. Podemos ver cómo en el paso del primer conmutador 1 al 2 se pasa desde el puerto 6 en el primero al puerto 2 del segundo, manteniéndose los valores VPI/VCI asignados en el primer conmutador.

③ Al detectarse la congestión del buffer el agente RCA genera la célula BRM con los datos de la PDU, que acaban llegando al primer conmutador a través de su RCA local que localiza la relación entre los datos de entrada y de salida de la PDU según hemos explicado anteriormente.

④ Con el índice de acceso a la DMTE se toma la PDU para retransmitirla. Destacamos que el mecanismo de retransmisión lleva de forma inherente el desorden de las PDU, por lo que al retransmitir una PDU no se puede garantizar el orden en la secuencia. Esta situación se produce en el tercer conmutador que solicitada una PDU al segundo conmutador y que es localizada en la DMTE local de este conmutador. La PDU retransmitida acabará provocando que la PDU retransmitida llegue después que otra PDU que en el origen estaba delante de ella. Hemos marcado esta posibilidad de desorden con una D en la DMTE del tercer conmutador.

⑤ El agente DPA se encarga de la retransmisión de la PDU solicitada hasta su correspondiente puerto de salida y VPI/VCI, según los datos anteriormente empleados en la primera transmisión.

El escenario anterior lo podemos ampliar con la inclusión en la red de conmutadores no activos, que no implementan el protocolo TAP y que, por tanto, no disponen de ninguna de las características hardware y software de los conmutadores AcTMs. En este caso los conmutadores no activos actúan de la forma tradicional, conmutando las células y PDU en dirección al destino, y las BRM en sentido contrario al flujo de datos hasta encontrar un conmutador activo. Este comportamiento provocaría que se perdiese la relación de los puertos de E/S y VPI/VCI en el caso de las solicitudes de retransmisión. Para controlar esta situación lo que se hace en el proceso de establecimiento de la conexión es registrar en todas las tablas de E/S de los

conmutadores activos los VPI/VCI junto a los puertos de E/S de cada uno de los conmutadores no activos que forman parte de la ruta seguida por cada fuente con requerimientos de GoS. Esta labor de registro de la ruta es realizada por el agente CoSA en el establecimiento de la conexión y, como hemos comentado anteriormente, supone actuar sobre los mecanismos de señalización de la red. Todo esto nos da una idea más definida de que las tablas de E/S de los AcTMs actúan a modo de tablas de routing.

En lo relativo a los costes de acceso a las tablas hemos de destacar su acceso constante tanto para lectura como escritura, y su dimensión la realizamos en función del tamaño de la DMTE. Es decir, cada una de las tablas mantiene tantas posiciones como números de entrada tiene la memoria dinámica con la diferencia con respecto a ésta en que las Tablas de E/S sólo almacenan los índices y no los datos reales que son registrados únicamente en la DMTE.

10.2.5. SISTEMA MULTIAGENTE

Como sabemos, la arquitectura TAP dispone de un subsistema constituido por cinco agentes software que hemos dado en denominar SMA-TAP y que es el que aporta la vertiente software a los conmutadores AcTMs y que confiere a estos su característica activa. Como hemos podido comprobar en apartados anteriores de la arquitectura, la mayor parte de bloques hardware son gestionados por alguno de los agentes software que forman el sistema multiagente SMA-TAP. En esta línea vamos a describir la funcionalidad de cada uno de los agentes de la arquitectura en relación a los aspectos de la arquitectura que controlan y de las interacciones entre cada uno de ellos.

La *Figura 10.7* representa la relación entre la arquitectura del SMA-TAP y el modelo arquitectónico de ATM presentado en la *Figura 10.1*. Como podemos observar en la *Figura 10.7* los agentes que intervienen en la arquitectura dependen directamente del protocolo TAP que, en realidad, es el conjunto de algoritmos que se ejecutan sobre la arquitectura TAP. Podemos observar los cinco agentes software, donde dos de ellos son programables y algunos de ellos están coordinados entre sí para poder realizar las labores que tienen encomendadas. Seguidamente vamos a comentar algunos de los aspectos de interés del proceso de comunicación entre los agentes, así como los aspectos de mayor interés de SMA-TAP y, para ello, vamos a emplear las etiquetas colocadas en la *Figura 10.7*.

① El protocolo TAP es el que concentra toda la labor del conjunto de algoritmos que ejecutan cada uno de los agentes software del SMA-TAP. Estos algoritmos que constituyen el protocolo serán descritos en el *Capítulo 11*, pero debemos de destacar que el protocolo TAP debe ser soportado por los nodos extremos de la comunicación de la VPN. Gracias a la característica distribuida de la arquitectura se permitirá el soporte de las conexiones privilegiadas desde el nodo emisor de la conexión y, a través de los conmutadores activos que existen en la VPN, se establecerá la comunicación con el nodo extremo de la comunicación que debe implementar también TAP. Podemos observar cómo el protocolo TAP está situado por encima de la capa EAAL-5 y, a su vez, puede tener por encima los protocolos de capas superiores como TCP, Frame Relay, etc.

② El agente CoSA (Class of Service Agent), situado en la Capa de Plano de Control del modelo arquitectónico ATM, se encarga de identificar las características del tráfico de entrada en la red (PCR, Ton, Toff, etc.), de forma que se responsabiliza del establecimiento de la conexión creando el VPI/VCI para cada una de las fuentes de datos. Este agente programable también se comunica con el resto de agentes CoSA de los conmutadores activos con el fin de mantener las relaciones de caracterización del tráfico en todos los AcTMs que intervienen en la red extremo-extremo. Podemos observar en la *Figura 10.7* que este agente también se comunica con el agente WFQA local a cada uno de los conmutadores, con la función ya explicada en secciones precedentes. CoSA se encarga de reensamblar cada una de las células de las conexiones fiables para constituir las PDU que serán pasadas a las colas de entrada y servidas a las colas ya en unidades de PDU con el objetivo de poder identificar cada una de las PDU de forma individualizada. Para que esto sea posible, los conmutadores deben soportar también la extensión EAAL-5 que hemos explicado en puntos anteriores.

③ El agente programable CCA (Control de Congestion Agent) tiene como función principal la de definir el algoritmo de control de congestión que se establece en el buffer que, en nuestro caso, se trata de EPDR. De este modo, se encarga también de aplicar el valor del umbral necesario para aplicar EPDR sobre el buffer y evitar las fragmentaciones de las PDU. Este agente mantiene también coordinación con el resto de agentes CCA de los conmutadores activos de la red, ya que la labor de programación establecida debe ser común en todos los conmutadores que intervienen en las conexiones garantizadas. Queda clara por tanto la relación del agente software CCA con el buffer como bloque

hardware de los conmutadores. Podemos observar cómo este agente está situado en la Capa de Plano de Gestión de la arquitectura ATM.

④ El agente WFQA, situado también sobre la Capa de Plano de Gestión, se responsabiliza de la gestión de las colas de entrada y, para ello, implementa el algoritmo QPWFQ. Hemos descrito ya las labores que realiza el agente sobre las colas en la sección dedicada a este componente hardware de la arquitectura, y el algoritmo será también comentado en detalle en el *Capítulo 11*, pero destacamos en este punto que este agente juega un papel primordial en la optimización del procesamiento del tráfico de entrada, ya que de él depende el garantizar que se cumplen los pesos aplicados a las fuentes, a la vez que se garantiza la justicia en todas ellas. Destaca también su función en la atención de las retransmisiones de las PDU congestionadas en el mismo o en conmutadores que están por debajo y en dirección al destino. Su coordinación es también importante para conseguir servir las PDU y células al buffer con eficiencia.

⑤ DPA es un agente que actúa desde la Capa de Plano de Gestión, de forma autónoma con respecto al resto de agentes del SMA-TAP. Dispone de funciones que le permiten solicitar al buffer la siguiente unidad de transferencia, tanto desde el buffer en el caso de las transmisiones, como desde la DMTE en el caso de las retransmisiones que hayan tenido éxito en el conmutador local. Para ello debe sincronizarse con estos dos bloques hardware para aprovechar las ventajas de una operación atómica que permite que, en el caso de las PDU, éstas sean transferidas de forma íntegra a su correspondiente puerto de salida para evitar el conocido problema del *interleaving*. Es también función del agente DPA la asignar los nuevos valores de VPI/VCI antes de ser enviadas a la salida las células que la red es capaz de transportar. Antes de comenzar a enviar las células de una PDU, DPA se encarga también de actualizar la correspondiente tabla de E/S según lo que hemos indicado en la sección anterior, para poder tener en el índice el valor del puerto de salida de cada una de las células de una PDU perteneciente a una conexión garantizada.

⑥ Por último, el agente RCA (Retransmission Control Agent) actúa desde la Capa de Plano de Gestión de la arquitectura, realizando una de las labores más importantes en el mecanismo de recuperación de las PDU congestionadas. Este agente recibe notificaciones de congestión desde el buffer local, que se encargar de proporcionarle el índice de la PDU que ha sido descartada del buffer por no tener cabida. Con este índice RCA se encarga de coordinarse con WFQA para notificarle a éste que va a generar una célula RM solicitando la retransmisión de esa PDU al conmutador anterior. El agente puede recibir también notificación de retransmisión desde el conmutador que le sucede, de forma que mediante el índice recibido en una célula RM se encargará de localizarla en la DMTE según detallaremos más adelante. RCA, por tanto, tiene acceso a los bloques hardware DMTE, Tablas de E/S y al buffer. Para realizar su trabajo necesita coordinarse con su agente local WFQA y con los agentes RCA de otros conmutadores a través del VPI/VCI dedicado a las células RM.

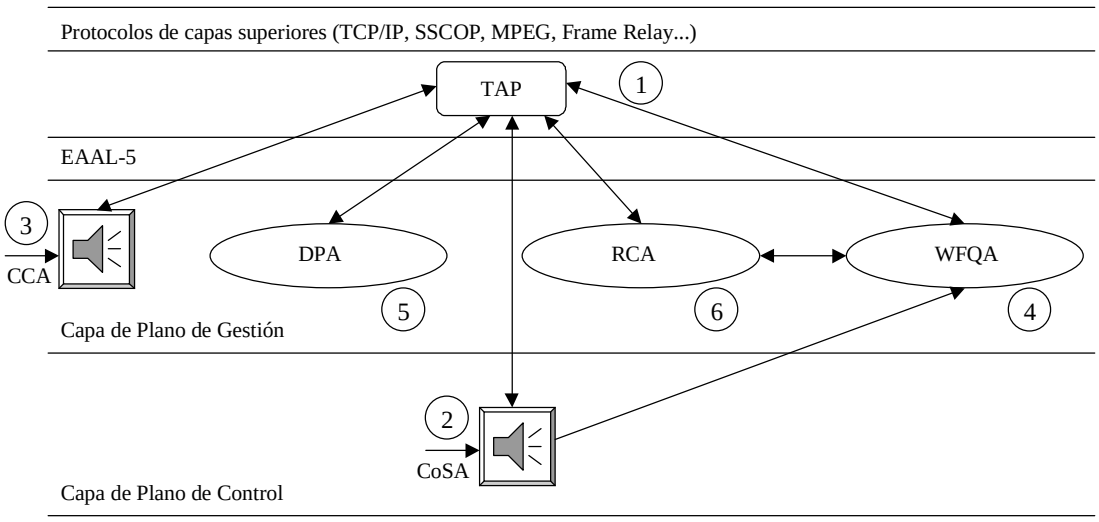

Figura 10.7. *Arquitectura del SMA-TAP en capas*

10.3. DESCRIPCIÓN DEL FLUJO EN CONEXIONES PUNTO-PUNTO

En esta sección vamos a presentar los flujos de datos y de control entre los diferentes elementos que constituyen la arquitectura TAP, en un intento por aclarar el funcionamiento de los conmutadores AcTMs que constituyen la VPN formada por nodos activos y multiagente. Para ello vamos a usar la *Figura 10.8* donde hemos etiquetado con un número cada una de las etapas que consideramos de mayor interés.

Antes de comenzar el análisis pormenorizado de cada uno de los apartados destacamos que en la *Figura 10.8* se han representado también detalladamente todos los bloques que componen la arquitectura de los nodos activos, así como cada uno de los agentes software que controlan, tanto el flujo de datos, como el control. Presentamos por esto los agentes diferenciados entre los que son programables y los que no lo son. Del mismo modo, se representa con líneas de trazo diferente el tráfico de datos, el de control y el provocado por las solicitudes de retransmisión entre conmutadores adyacentes. Aparecen también representadas las interacciones entre los agentes CoSA-WFQA y entre WFQA-RCA. Queremos destacar también la posibilidad de comunicación entre agentes RCA de conmutadores adyacentes que se realiza a través de las células BRM indicadas para las retransmisiones. La *Figura 10.8* no representa la comunicación entre agentes CoSA y CCA entre conmutadores AcTMs consecutivos. Seguidamente iremos describiendo cada una de las etapas de forma secuencial, tal como sería el flujo normal de datos dentro de los conmutadores activos.

① El agente programable CoSA es el encargado de la función de establecimiento de conexión en la red. Por esto permite identificar cada una de las fuentes de tráfico que se van a poner en marcha, y por tanto, otra de sus funciones es la de caracterizar los parámetros de tráfico de cada una de las fuentes. Una vez establecida la conexión, asignadas las tablas de routing e identificados los parámetros de tráfico, mantiene la comunicación con cada uno de los agentes CoSA del resto de los conmutadores activos presentes en la red y que están implicados en las conexiones negociadas.

② Antes de comenzar la comunicación es necesario actuar sobre el agente programable CCA para indicar los mecanismos de control de congestión que desean aplicarse a los flujos de datos. Es decir, elegir el umbral de EPDR, decidir si se desea usar PPD u otro algoritmo de control de congestión sobre el buffer etc. Las características definidas por CCA sobre el buffer afectarán a todo el tráfico de datos; sin embargo, al tratarse de un agente programable permitirá cambiar en tiempo de ejecución el valor del umbral cuando se están detectando excesivas congestiones.

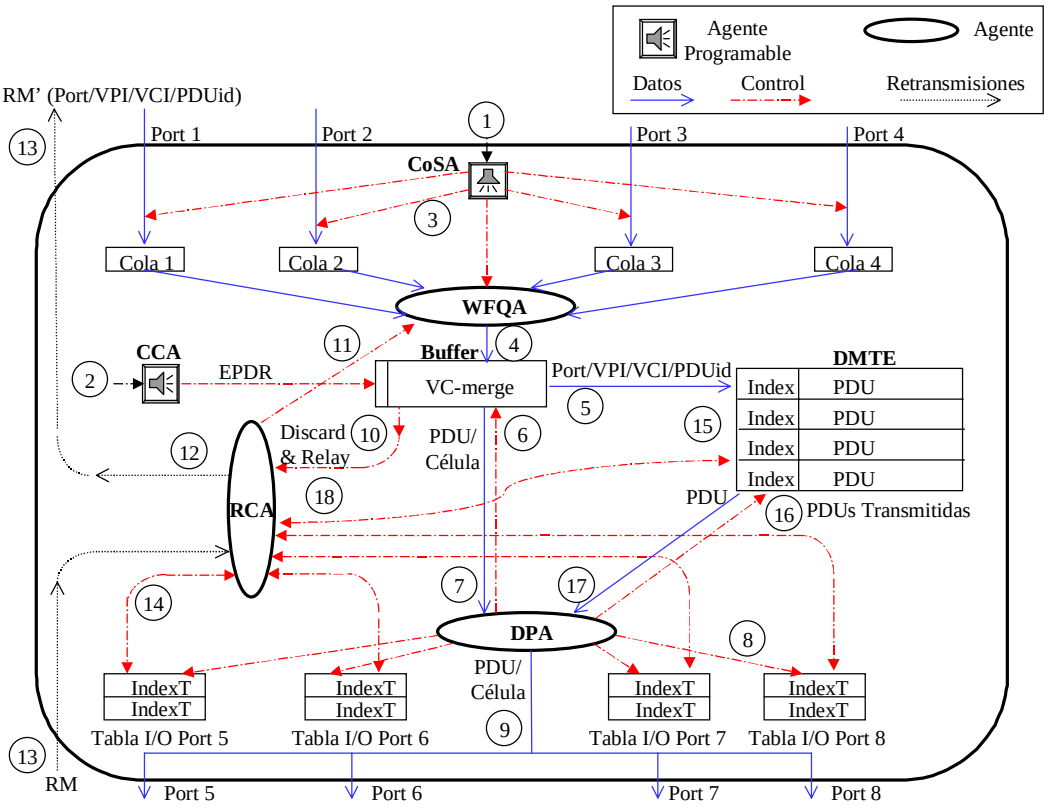

Figura 10.8. *Flujo de datos y control de un conmutador AcTMs*

③ Una vez elegidas las características del tráfico y del buffer comienza la transferencia desde las fuentes que generan PDU en el caso de las fuentes privilegiadas y células ATM nativas si se trata de fuentes de background. El tráfico llega a cada una de las colas correspondientes de los puertos de entrada de forma que, como sabemos, cada VPI/VCI tendrá su propia cola de entrada en el conmutador. Tal como hemos descrito anteriormente tendremos separado el flujo en colas diferentes para cada una de las fuentes. En este punto es donde se establece la comunicación del agente CoSA con el agente WFQA que entra en acción en el siguiente punto.

④ El agente WFQA se encarga de servir cada una de las cabeceras de las colas según lo que hemos comentado en secciones anteriores. De este modo el flujo de datos irá llegando al buffer del conmutador en forma de PDU y/o de células, donde van a ser atendidas según las características especificadas en el punto ② por el agente programable CCA.

⑤ Una vez que las datos están en el buffer, se realiza la función de copia de las PDU en la memoria DMTE, que es indexada con los datos de la PDU (*VPI/VCI/PDUid/Port*). Para ello se usa la función *copy(index,PDU)*, pero antes de depositar la PDU en la memoria son comprobadas las PDU ya existentes de cada conexión para garantizar el grado de GoS ya comentado. De esta forma, si ya se tienen en memoria el número de PDU máximo, la que acaba de llegar elimina la de PDUid más pequeño. El índice empleado en la entrada de la PDU será posteriormente empleada en los accesos para lectura en el caso de las solicitudes de retransmisión.

⑥ Una vez realizada la copia de la PDU entrante en la DMTE el buffer espera desde el agente DPA una petición de envío. Para esto el agente DPA dispone de la función *get (siguiente)*. Así, el *dispacher* se encarga de comprobar si existen PDU o células pendientes de procesar en el buffer, de modo que cuando su nivel de actividad lo permite, solicita que le sea enviada la siguiente unidad de transferencia a enviar a los puertos de salida.

⑦ Cuando el buffer recibe una orden de *get(siguiente)* desde el agente DPA, se activa la función *send(siguiente)* que es la responsable de enviar al *dispacher* cada una de las PDU que han sido previamente copiadas en la memoria DMTE o de las células independientes que no se copian en memoria. La función *get* debe sincronizarse adecuadamente con las funciones *copy* y *send* asociadas al buffer. Cuando *get* obtiene respuesta positiva, *copy(index,PDU)* debe haber terminado de realizar la copia de la PDU solicitada en la DMTE antes de enviarla al agente DPA. Una vez enviada cada PDU o célula, éstas son eliminadas del buffer liberando espacio para el tráfico entrante que pueda estar a la espera en las colas de entrada. Tenemos que destacar que las funciones *get* y *send* son las que se encargan de garantizar la operación atómica de envío desde el buffer que permite que puedan procesarse todas las células de una PDU de forma completa evitándose así el *interleaving*.

⑧ En este punto el agente habrá recibido la siguiente PDU/célula desde el buffer, de forma que deberá encargarse de enviar cada unidad de transferencia a su correspondiente puerto de salida. Pero antes de realizar el envío deberá ocuparse de la operación de asignación de VPI/VCI a las células que le hayan llegado. Cada vez que el *dispacher* recibe una PDU destinada a un *port* de salida, se encarga de actualizar su correspondiente tabla de E/S para relacionar los puertos de E/S y tener el índice de acceso a la DMTE en las retransmisiones. Por tanto, el agente DPA debe acceder a la correspondiente tabla de E/S del puerto de salida para registrar el índice de la PDU que se va a enviar a través de ese puerto. Este índice es el mismo que se ha utilizado en la DMTE, sólo que se le añade también el puerto de salida de cada PDU garantizada para evitar la posible repetición entre los VPI/VCI de conexiones diferentes.

⑨ Actualizadas todas las estructuras, el *dispacher* envía cada PDU a su correspondiente *port* de salida hasta el siguiente conmutador en dirección al destino que puede ser un nodo activo o no. Los datos enviados entre conmutadores son células ATM nativas como si de una troncal ATM se tratase.

⑩ En este punto volvemos de nuevo al buffer del conmutador para explicar la forma de proceder en el caso que se produzca una congestión del mismo. Hasta ahora hemos visto el flujo de datos en la situación en que las PDU tienen cabida en el buffer, pero cuando no es así entra en acción el algoritmo EPDR que se encarga de descartar las PDU que no tienen cabida completa en el buffer al superar el valor del umbral. Cuando esto se produce actúa el agente RCA que se encarga de evitar la fragmentación de las PDU que superan el valor del umbral del buffer. Al RCA se pasan los datos de la PDU mediante el valor del índice de la misma.

⑪ La primera función que debe realizar el agente RCA al recibir una notificación de congestión desde el buffer es la de comunicar al agente WFQA la solicitud que va a realizar de retransmisión al conmutador anterior. De este modo el agente WFQA tendrá conocimiento de la llegada inminente de una PDU retransmitida que deberá priorizar sobre el resto de PDU que estén en las colas.

⑫ El siguiente paso realizado por el agente RCA es el de preparar la célula BRM que debe enviar al conmutador anterior, en la que introduce el índice de la PDU que ha sido descartada y que hay que retransmitir desde conmutadores anteriores. Antes de preparar la PDU RCA consulta si el enlace afectado dispone de suficiente tiempo de OFF para poder atender la petición de retransmisión. Realizados estos pasos la BRM es enviada por el VPI/VCI reservado al efecto.

⑬ Como puede comprobarse en la *Figura 10.8*, este paso está etiquetado en dos puntos diferentes, aunque en realidad hace referencia al mismo proceso. Con esto queremos indicar que la solicitud de retransmisión puede realizarse desde dos puntos diferentes en cada conmutador activo. Es decir, en cada AcTMs puede generarse una BRM en el agente RCA, pero también puede llegarle una BRM desde el conmutador que le sigue que habrá, por tanto, experimentado una congestión. Esta etiqueta identifica entonces el VPI/VCI reservado para las células RM, en las que se indica el índice de la PDU según se ha explicado ya en capítulos anteriores. Destacamos que la etiqueta de la parte superior de la Figura acabará llegando al conmutador anterior que atenderá la RM desde el agente RCA, tal como lo hace el que está representado en la Figura. En el caso que el conmutador anterior no sea un AcTMs, y no soporte TAP, la célula RM será reenviada hasta el siguiente conmutador en dirección al emisor, y así sucesivamente hasta encontrar un conmutador activo que sea capaz de atender la solicitud de retransmisión, o hasta llegar a la fuente de tráfico.

⑭ Cuando al RCA llega la célula RM de retransmisión, el agente accede al campo que contiene el índice de la misma. Mediante este índice se accede a la correspondiente tabla de E/S del conmutador para poder obtener la relación de puertos de E/S y así tener el índice real de acceso a la memoria DMTE. Una vez conseguido el índice de la tabla de E/S es pasado al agente RCA.

⑮ Cuando conocemos el índice de acceso a la memoria DMTE se intenta localizar en ésta la PDU que el conmutador siguiente ha perdido por congestión. Una vez que se ha accedido a la memoria mediante la función de *hash*, pueden ocurrir dos cosas: que la PDU esté en memoria aún, o que ya no se encuentre en la misma, porque hayan llegado otras PDU con posterioridad que hayan ocupado el espacio de memoria de las PDU más antiguas de esa conexión concreta, dando lugar a lo que llamamos fallo de memoria DMTE.

⑯ Si la PDU solicitada está aún en la memoria, el protocolo TAP que se ejecuta dentro del conmutador y controla el buffer y la DMTE, lo notifica al agente DPA que solicita la PDU a la DMTE para retransmitirla. En realidad, este paso es parecido al ya descrito en la etiqueta ⑥, porque se dispone de una operación *get* similar para solicitar el envío de una PDU en este caso desde la DMTE en lugar de hacerlo desde el buffer.

⑰ Una vez solicitada la PDU desde el DPA, es la memoria DMTE la que se encarga de servirla al *dispacher* del mismo modo que se hacía en la etiqueta ⑦ desde el buffer. Por tanto, en este caso también se recurre a la atomicidad para evitar el *interleaving*, tal como ocurría en el paso ya descrito. Una vez que la PDU retransmitida llega al DPA se repetirán los pasos ⑧ y ⑨ descritos anteriormente, para reenviar la PDU hasta el conmutador siguiente que se congestionó en la fase anterior.

⑱ Este paso de la secuencia se da cuando en la DMTE no existe la PDU solicitada. RCA de este modo conoce que la PDU no ha sido localizada por lo que debe solicitar la retransmisión hacia el conmutador previo. Esta etapa coincide realmente con la indicada en la etiqueta ⑩ y, por tanto, se llega al agente RCA con una PDU no localizada (en lugar de congestionada en el conmutador actual), pero que RCA debe resolver de la misma forma que si la petición viniese del buffer. El mecanismo de solicitud de retransmisiones puede continuar en sentido al emisor hasta localizar la PDU en un AcTMs, teniendo en cuenta que los conmutadores no activos sólo se encargan de reenviar la BRM en sentido contrario al flujo de datos hacia el conmutador previo.

Antes de concluir hemos de destacar que acabamos de describir un escenario de comunicación punto-punto, y que las extensiones necesarias para las transferencias punto-multipunto, multipunto-punto y multipunto-multipunto sólo requieren de la incorporación en los nodos emisores y en el resto de los nodos

activos de la red, de las rutas o secuencias de VPI/VCI de las múltiples fuentes de tráfico y receptores de las conexiones multipunto. En general, en ATM las investigaciones relacionadas con las comunicaciones multipunto apuntan en la dirección de la unión de *n* comunicaciones punto-punto.

10.5. CONCLUSIONES

En este capítulo se ha presentado el detalle de cada uno de los bloques que componen la arquitectura de los conmutadores activos AcTMs. La memoria DMTE desempeña un papel fundamental para conseguir aportar la GoS a las fuentes privilegiadas, pero otros componentes como las colas de entrada permiten la separación de los flujos en colas separadas a la vez que permiten establecer pesos a cada uno de los VCI con la intención de poder aportar un tratamiento específico, pero a la vez justo a cada una de las fuentes. El buffer del conmutador también desempeña una función básica en la arquitectura ya que con una adecuada gestión del mismo se consiguen amortiguar muchos de los efectos negativos descritos en el *Capítulo 9*. Las Tablas de E/S aportan los índices de acceso a la memoria DMTE en las retransmisiones y nos dan la relación entre los puertos de entrada y de salida en cada una de las conexiones privilegiadas. Toda esta equipación hardware es gestionada por los mecanismos software que aportan los cinco agentes que conforman el SMA-TAP cuyo funcionamiento también ha sido descrito en este capítulo. Se han justificado determinadas decisiones de diseño como el tamaño de la DMTE, el de las colas de entrada y el del buffer, con la intención de demostrar que su coste está justificado por la GoS aportada y por la optimización del goodput que aportan a la red. Una vez analizados los bloques separadamente se han estudiados los flujos de datos y del control en conexiones punto-punto para comprender su funcionamiento.

REFERENCIAS

[1] J. M. Pitss, “Introduction to ATM. Design and Performance”, *Ed. Wiley*, (1997).

[2] M. de Prycker, “Asynchronous Transfer Mode. Solution for Broadband ISDN (3ª Ed.),” *Ed. Prentice Hall*, (1995).

[3] Handël, R., Huber, M. N., Schoröder, S., “ATM Networks Concepts, Protocols. Applications (2ª Ed.),” *Addison-Wesley*, (1995).