

CAPÍTULO 5

MECANISMOS DE CONTROL DE CONGESTIÓN

5.1. INTRODUCCIÓN

Sabemos que la CoS UBR fue diseñada y propuesta para las aplicaciones de datos sin especiales requerimientos de retardo, que no son sensibles a las pérdidas de células y que aprovechan la capacidad disponible en la red generando tráfico a ráfagas. De este modo, no están controladas por la función CAC y tampoco se ven sometidas a políticas de rendimiento. Si aparecen congestiones, las células pueden perderse y las fuentes no se ven obligadas a reducir su velocidad de transmisión. Por esto, son las aplicaciones las que se encargan de aplicar sus propios mecanismos de retransmisión y de recuperación de las pérdidas, generalmente a través de la ventana de control de flujo de TCP.

Una CoS alternativa a UBR es, como también sabemos, ABR que aporta su propio mecanismo de control de congestión basado en velocidades en un bucle cerrado, en el que el uso de las células RM juega un importante papel. Este mecanismo consiste en que las fuentes de entrada se encargan de ajustar sus velocidades a los niveles de congestión de las conexiones en cada momento. En esta línea, se han propuesto múltiples esquemas de control de congestión basados en las velocidades de las fuentes, que se diferencian básicamente en la forma en que los conmutadores determinan la congestión y en la forma en que las fuentes condicionan sus velocidades de transmisión a las situaciones de congestión. Los propuestas iniciales para aplicar esos mecanismos de control de congestión en ABR aportadas por el ATM Forum [1,2,3,24] buscan la justicia de las fuentes sin aplicar ninguna técnica de encolado *per-VC* que, posteriormente, han acabado popularizándose una vez que se ha demostrado la ventaja del tratamiento diferenciado de los flujos en colas separadas.

En suma, ABR y UBR son las dos clases de servicio diseñadas por el ATM Forum [4] para soportar las aplicaciones que generan datos sin requerimientos de tiempo real. Aunque ABR aporta mejores características de QoS, UBR es más simple en su implementación con una complejidad y coste menores que ABR. Por esto UBR es una interesante alternativa a ABR y, sea una u otra la CoS usada, hemos de buscar soluciones para controlar las situaciones de congestión y para eliminar los problemas de injusticia que puedan producirse en las fuentes. La existencia de tráfico a ráfagas provoca que las redes ATM acaben alcanzando situaciones de sobrecarga y colapso más a menudo de lo esperado.

Recordamos que UBR y ABR son las CoS destinatarias de la arquitectura que hemos diseñado y destacamos que, tanto ABR como UBR, han sido las propuestas para soportar las aplicaciones de datos (principal objetivo de la arquitectura TAP) en las redes ATM. Precisamente, TCP es de los protocolos de la capa de transporte más usados en redes de datos y se han realizado numerosos esfuerzos para su soporte sobre las redes ATM. Es de suma importancia, por tanto, dar respuesta a múltiples requerimientos de este tipo de tráfico best-effort, ya que en la actualidad es el más abundante en las redes.

En el Capítulo 3 se comenta que las principales investigaciones para conseguir fiabilidad están inspiradas en mecanismos de generación de código redundante (como CRC o FEC), los cuales introducen importante overhead generando sobrecargas en la red y afectando al throughput negativamente. Sabemos también que estos métodos redundantes solventan el problema de la recuperación de células corruptas y perdidas, pero no pueden solventar otros indeseables, y no menos frecuentes, problemas como son la congestión y la

fragmentación¹ de paquetes en los conmutadores, debidas al exceso de tráfico, a problemas en los buffers, etc. Todos estos problemas acaban afectando al rendimiento de la red y degradando el goodput en la misma.

Los esquemas de control de congestión más conocidos son RCD (Random Cell Discard), PPD (Partial Packet Discard) [5] y EPD (Early Packet Discard) [1,6]. Avanzamos ya que el esquema EPD se ha demostrado como una solución interesante para conseguir óptimo funcionamiento desde el punto de vista de la justicia y utilización de los enlaces. Sin embargo, existen otros esquemas con idénticos objetivos que son maximizar el throughput y la justicia, mientras se minimiza el retardo en las redes ATM. Algunos de estos mecanismos son: ESPD (Early Selective Packet Discard) [3], FBA (Fair Buffer Allocation) y RED (Random Early Detection) [8,9].

En los trabajos previos de nuestras investigaciones [10] implementábamos PPD para aliviar el efecto de las congestiones y de la fragmentación de paquetes. Posteriormente hemos soportado y modificado una variación de EPD en TAP de forma que, cuando una de las PDU llega al buffer, esperamos a la célula final de cada PDU para aplicar EPD en los conmutadores ActMs, desechando aquellas PDU que provoquen congestión en el buffer y solicitando su retransmisión mediante NACKs al conmutador previo. Las PDU que no provocan congestión en el buffer son enviadas a los puertos de salida de forma completa aplicando VC merge para aliviar también los problemas de interleaving².

Cuando se llena el buffer de un conmutador congestionado, las células que llegan serán descartadas y, con una elevada probabilidad, muchos paquetes perderán células y mientras otras células seguirán su transmisión a través de la red, provocando el indeseable fenómeno que [11] de la fragmentación³ de paquetes. Al evitar la fragmentación estamos optimizando el ancho de banda de los enlaces de la red que no se ven sobrecargados con las retransmisiones extremo-extremo, ni con la transmisión de PDU completas que pueden estar corruptas por la pérdida de una sola de sus células.

En este capítulo vamos a revisar diferentes técnicas para eliminar o aliviar los problemas de congestión de buffers y fragmentación de paquetes que provocan la degradación progresiva del throughput de la red. Varios de estos mecanismos buscan también un cierto punto de equilibrio entre el índice de justicia que son capaces de aportar y el goodput que se consigue en la red. Es decir, se busca no sacrificar la justicia de las fuentes por incrementar el throughput en la red, teniendo en cuenta que el índice de justicia puede ser calculado con la siguiente expresión [12],

$$I_j = \frac{\left(\sum_{i=1}^n x_i\right)^2}{n \sum_{i=1}^n x_i^2}$$

donde x_i es el throughput efectivo de la i -ésima fuente, y n es el número de fuentes.

En el caso de la arquitectura TAP la justicia es garantizada por la estrategia planteada en el capítulo anterior donde se presenta el algoritmo QPWFQ. Por esto, el buffer que usamos, y donde aplicaremos una variante del protocolo EPD, se propone como un mecanismo para evitar la fragmentación de las PDU y también como el punto en que realizamos la detección de congestiones y solicitud de retransmisiones. Para evitar el problema de la fragmentación proponemos la aplicación de técnicas VC-merging⁴ que también van a ser descritas en el presente capítulo. El buffer es gobernado por un agente programable que permite al operador de la red elegir el algoritmo a aplicar en la gestión del buffer.

5.2. PPD (PARTIAL PACKET DISCARD)

PPD fue el esquema propuesto por G. Armitage y K. Adams [5] para amortiguar el efecto negativo de la fragmentación de paquetes, lo que supone mejorar sustancialmente el throughput conseguido por el esquema original RCD (Random Cell Discard) basado en desechar las células aleatoriamente. PPD tira las células cuando, al llegar un paquete, se llena el buffer, y continua tirando las siguientes células del mismo paquete.

¹ La fragmentación de los paquetes se produce al perderse células de forma incontrolada provocando la inconsistencia completa de los paquetes (PDUs) a los que pertenecen las células perdidas.

² El interleaving se produce al intercalarse en un mismo enlace las células pertenecientes a conexiones diferentes.

³ Cuando se tira alguna célula de un paquete, éste no podrá ser reconstruido por el destinatario, lo que requerirá la retransmisión completa del paquete extremo-extremo, afectando negativamente al goodput de la red.

⁴ VC merge es una de las técnicas propuestas para evitar el problema del interleaving.

Este comportamiento de PPD conduce a tirar las colas de cada paquete. No obstante, PPD aporta un mejor comportamiento a las conexiones TCP aplicando un mecanismo selectivo (en lugar de aleatorio como RCD) en las células que son tiradas por la red.

En el caso del envío de paquetes (PDU) a través de las redes ATM supone que éstos sean segmentados en células de 53 octetos que son las que realmente viajan por la red. Por tanto, todas las células de un paquete deben llegar íntegras al destinatario para que el paquete llegue sin problemas. Cuando una sola célula se pierde, el paquete completo acabará también perdiendo su integridad por lo que será descartado. Es por tanto evidente el riesgo de afectar a muchos paquetes aunque sólo se pierdan unas cuantas células si se desechan las células de forma aleatoria. Pero además, ocurre que el resto de las células de un paquete que acaba de perder una célula continúan adelante en dirección al destino, pudiendo volver a provocar nuevas congestiones que acaben afectando a otras conexiones que pueden también experimentar pérdidas. Con lo cual, células que pertenecen a paquetes que ya se sabe que son corruptos pueden colaborar a la congestión de nuevos conmutadores. Este es el citado fenómeno que se definió en [11] como fragmentación, para el que en [5] se propuso el algoritmo PPD presentado en la *Figura 5.1*.

En este algoritmo la *lista-descartes* registra una relación de VPI/VCI que quedan marcados cuando ya se les ha tirado alguna célula en el paquete que se está procesando para cada VPI/VCI. Con EOM se determina cuándo se llega a la célula final de un paquete. Cada vez que se acaba de procesar un paquete que ha perdido alguna célula se actualiza la *lista-tirados* eliminando el VPI/VCI de esta lista para comenzar el siguiente paquete de esa conexión sin tenerlo marcado con células descartadas.

```

Mientras una célula está llegando a un conmutador;
  Si el VPI/VCI de la célula pertenece a lista-descartes
    tirar la célula
    Si la célula es una célula EOM
      borra el VPI/VCI de la célula de la lista-descartes
    Fsi
  en otro caso
    Si el buffer está lleno
      tirar la célula
      añadir el VPI/VCI de la célula a la lista-descartes
    en otro caso
      aceptar la célula en el buffer
    Fsi
  Fsi
FMientras

```

Figura 5.1. Algoritmo PPD

Como puede observarse en el algoritmo, PPD tirará una célula cuando se llena el buffer pero, además, como se registra el VPI/VCI de todas las células tiradas en la *lista-descartes*, también se tirarán todas las células que pertenezcan al mismo paquete y que siguen a la que ya ha sido descartada. De este modo, no sólo se tiran las partes finales de los paquetes, sino que se pueden tirar paquetes completos. Aunque no se elimina completamente el problema de la fragmentación, sí se reduce de una forma importante el número de las células inservibles.

5.3. EPD (EARLY PACKET DISCARD)

EPD es una técnica de gestión de buffers implementada para asegurar elevado throughput extremo-extremo a las aplicaciones de datos a ráfagas durante los periodos de sobrecarga. EPD aporta mejoras a PPD, con la intención de garantizar que se tiran paquetes enteros en lugar de partes de paquetes como hace PPD. EPD fue introducido en [6] para tirar las células BOM (Begin Of Message) y las siguientes células del paquete a que pertenecen las cabeceras, una vez que se ha superado un determinado umbral de llenado del buffer. EPD tirará los paquetes completos antes de que el buffer acabe llenándose, con la intención de que los paquetes que puedan ser susceptibles de perder alguna célula, y por tanto su integridad, no sean transmitidos por la red.

En resumen, cuando un buffer se llena, en lugar de tirar las células pertenecientes a diferentes conexiones que generan paquetes, o PDU en nuestro caso, se propuso en [11] EPD para tirar los paquetes completos antes de que el buffer se llene. De este modo se evita el problema de la fragmentación y se consigue que no sigan adelante las transmisiones de los paquetes corrompidos por la pérdida de células aleatorias. En realidad,

podemos entender [13,14] que EPD es un algoritmo para el descarte de paquetes, que puede ser aplicado a protocolos basados en paquetes soportados sobre ATM como TCP, UDP o IPX. En nuestro caso, lo emplearemos para soportar la posibilidad de tirar PDUs generadas por nuestro protocolo TAP ATM nativo. Varias investigaciones [7,14,15,16] han demostrado el rendimiento de TCP sobre ABR y UBR en términos de throughput en la red y justicia en las fuentes de tráfico. Se ha demostrado también que EPD, aplicado a tráfico TCP sobre UBR, no consigue buenos índices de justicia y, sobre todo, en redes muy congestionadas. Sin embargo, mejorando EPD con mecanismos *per-VC accounting* y *per-VC queueing* se consigue un buen comportamiento en cuanto a justicia en el tratamiento de los VC.

A continuación vamos a comprobar la simplicidad del algoritmo para comprender que su coste algorítmico es asumible desde el punto de vista de la justicia y goodput conseguidos. Para soportar EPD se establece en cada conmutador un valor umbral que marca el punto de llenado del buffer en que, al ser alcanzado, se supone que no se aceptará la entrada de ningún otro paquete. Es decir, cuando llega la primera célula de un paquete a un buffer que está lleno hasta, o por encima de, su valor umbral, esa célula es desechada, y también todas las demás células pertenecientes a ese paquete, aunque la cola descienda inmediatamente su nivel de llenado por debajo del umbral. Por otro lado, las células de un paquete cuya primera célula llegó antes de que el buffer alcanzase el umbral no serán tiradas a no ser que el buffer acabe llenándose. En suma, cuando la primera célula de un paquete llega en un margen de llenado elevado del buffer, es descartada junto a todas las demás que pertenecen a su mismo paquete, ante el riesgo de provocarse la fragmentación del paquete.

Lo importante está en elegir el valor del umbral para que sea capaz de evitar el rebosamiento del buffer que conduce a la fragmentación. Pero además, es necesario buscar el punto de equilibrio al establecerlo para que al intentar evitar la fragmentación no acabemos provocando demasiadas retransmisiones de paquetes desechados completamente. Es decir, si se establece el umbral demasiado alto, el mecanismo no surtirá ningún efecto y, si el umbral se fija muy bajo, el porcentaje de paquetes tirados será demasiado elevado degradando también el goodput.

La *Figura 5.2* ilustra el comportamiento del algoritmo EPD cuyos detalles de funcionamiento pueden ser ampliados en la referencia [11]. Además, la *Figura 5.3*, presenta el algoritmo EPD implementado en [13,14] donde se demuestra que el ancho de banda es mejor aprovechado con EPD aplicado tanto a ABR como a UBR. Hemos adaptado la *Figura 5.3* a nuestro escenario concreto y podemos observar la llegada de las PDU al buffer del conmutador, en el que hemos definido el *tamaño del umbral*, el *tamaño actual de la cola (TAC)* y la *capacidad máxima del buffer (CMB)*. En la parte superior de la *Figura 5.3* (caso a)), puede observarse cómo al buffer ha llegado la PDU 7, que no cabe completa en el buffer. Sin embargo, como a la llegada de la cabecera (primera célula) de esta PDU el valor actual de la cola está por debajo del umbral ($\delta = \text{Umbral} - \text{TAC} > 0$), la PDU es aceptada en el buffer, y el tamaño de la cola se ve incrementado en el volumen de la PDU. En el escenario b), al crecer el tamaño de *TAC*, éste se acaba igualando con el valor del umbral, lo que provoca que, a la llegada de la primera célula de la PDU 8 ($\delta = \text{Umbral} - \text{TAC} = 0$), ésta no pueda ser admitida aunque aún queda espacio en el buffer del conmutador. De este modo la PDU 8 es tirada por la red ante el riesgo de experimentar la fragmentación debida al probable rebosamiento del buffer. Si la PDU 8 no fuese descartada, sus células finales no cabrían en el buffer, mientras las primeras seguirán adelante provocando la fragmentación de la PDU.

```

TAC=Tamaño_actual_de_la_cola
CMB=Capacidad_máxima_del_buffer_del_conmutador
Mientras una célula está llegando a un conmutador;
  Si la célula es la primera célula de un paquete
    Si TAC >= Umbral
      tirar la célula
    En otro caso
      aceptar la célula en la cola FIFO
    Fsi
  En otro caso
    Si ya se ha tirado alguna célula de este paquete
      tirar la célula
    En otro caso
      Si TAC >= CMB
        tirar la célula
      En otro caso
        aceptar la célula en cola FIFO
      Fsi
    Fsi
  Fsi
Fsi
Fsi
Fsi

```

Figura 5.2. Algoritmo EPD

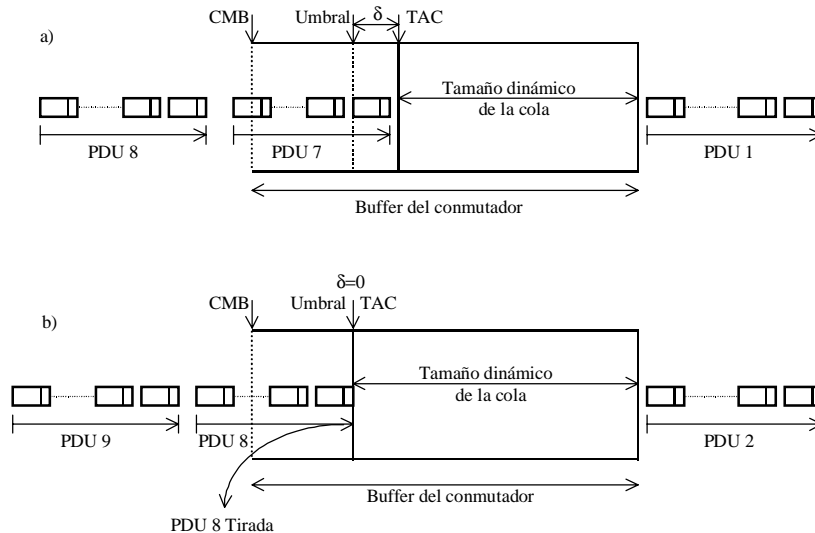


Figura 5.3. Representación gráfica de EPD procesando PDUs AAL-5

Investigaciones como [13,14] demuestran que el índice de justicia en ABR y UBR con control de congestión basado en velocidades, puede ser mejorado aplicando esquemas EPD en colas *per-VC*. Los autores de estas investigaciones han demostrado también en trabajos previos que TCP sobre ABR tiene mejor comportamiento que sobre UBR, necesitando incluso menos requerimientos en cuanto a hardware.

El algoritmo de EPD puede tener otros planteamientos o versiones, aunque el problema general es el mismo. De hecho, en la sección 5.7 explicaremos nuestra propia variante adaptada a ATM, que es la que implementamos en nuestra arquitectura. Usamos la condición de EOM (End Of Message) que es la delimitación de PDU que soportamos en nuestra propuesta nativa de EAAL-5.

No obstante, la aplicación de EPD al tráfico UBR no permite garantizar el tratamiento justo de las fuentes de tráfico y se ha comprobado [13,14] que el tráfico de varias fuentes acaba teniendo un comportamiento bastante injusto, lo que justifica la utilización de técnicas *per-VC* para intentar evitar el efecto de unas fuentes sobre otras. Para ello se han propuesto dos variantes del algoritmo EPD original [17] para incorporar asignación justa de buffer, *per-VC queueing* y *per-VC accounting*. Estos dos mecanismos intentan mejorar la técnica aleatoria de tirado de paquetes que provoca la asignación de ancho de banda injusta entre fuentes que se disputan el buffer.

El mecanismo *per-VC accounting* se basa en el uso de una nueva condición para tirar las PDU, de forma que, además de cumplir que $TAC \geq umbral$, se debe cumplir que $TAC_i \geq UM$ para cada fuente F_i . En este caso, TAC_i es el número de células pertenecientes a la fuente F_i . UM representa el porcentaje del *umbral total* que corresponde a cada fuente calculado dividiendo TAC entre el número de N fuentes que tienen células en el buffer, y normalizado con un parámetro de control C que está comprendido entre 1 y 2. Es decir, $UM = C * TAC/N$ y representa la ocupación media del buffer para cada fuente cuando $C=1$.

Aunque *per-VC accounting* aporta un mejor índice de justicia a las conexiones, sólo se mantiene la asignación justa de buffer en el momento de tirar las células. Por esto es necesario conseguir la justicia en el reparto del ancho de banda entre todos los VC mediante una asignación justa del buffer en el tiempo de la transmisión. Surge así la posibilidad de distribuir el buffer mediante técnicas *per-VC queueing* cuyo comportamiento podemos ver en la Figura 5.4. En este caso, cada VC dispone de su propia cola en el buffer del conmutador, así como una indicación del tamaño medio de buffer ocupado por cada uno de los VC. Como puede observarse en la Figura 5.4 se han considerado tres conexiones VC, cada una con su propia situación de ocupación de colas. Contamos también, como en el caso general de EPD, con una *CMB general* del buffer, y con un valor de UM calculado como en el caso de *per-VC accounting* $UM = C * TAC/N$. La diferencia la tenemos en que las células de los VC son servidas a la salida del buffer siguiendo un planificador Round Robin, como puede observarse en la Figura 5.4, que garantiza que en cada ciclo se sirve una célula de cada uno de los VC. Esta es la técnica que va a aportar justicia a la forma de atender las fuentes. En [13,14] se realizan diversas simulaciones con el parámetro de control C demostrándose que a medida que se incrementa su valor, crece también el índice de justicia. Sin embargo, a medida que crece C también se degrada el throughput por elevarse las probabilidades de rebosamiento en el buffer del conmutador. No obstante, la investigación no demuestra la justificación ni relación directa de este parámetro con los efectos que produce.

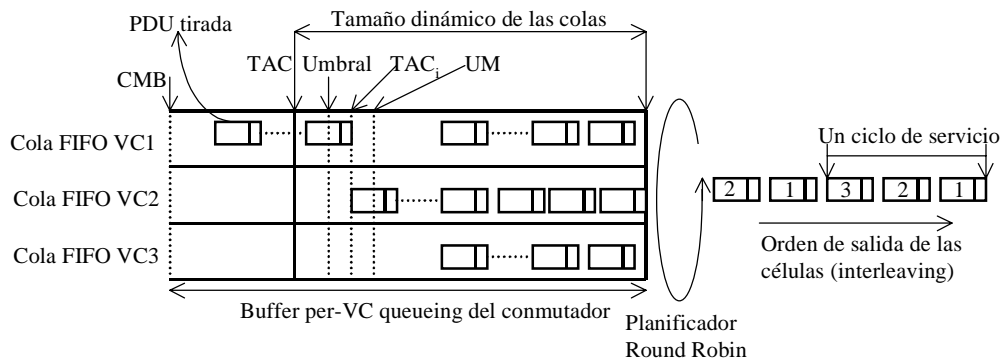


Figura 5.4. Esquema EPD con per-VC queueing y Round-Robin

El algoritmo que implementa el comportamiento descrito por la Figura 5.4 es presentado en la Figura 5.5 y, como puede analizarse, cuando se alcanza el tamaño del umbral de EPD, ningún VC deberá exceder su porción de buffer correspondiente, garantizando el grado de justicia que se busca. Además, en cada ciclo de emisión de células a la salida se va a servir una sola célula de cada uno de los VC. Así, se garantiza la justicia final de cada VC sin afectar el tamaño de cada una de las colas de los VC. Como veremos más adelante, este mecanismo aporta justicia, sin embargo, produce otro fenómeno indeseable que es el del *interleaving* ya que a la salida se mezclan los flujos pertenecientes a todos los VC. Nuestra propuesta de EPD elimina este problema además de aportar un mecanismo de retransmisión que no está soportado en ninguna de las propuestas de la literatura.

```

N=Número_de_VCs_en_el_buffer
C=Parámetro_de_Control_(1<= C <=2)
TACi=Tamaño_actual_de_la_cola_de_cada_VCi
TAC=Suma_de_todos_los_tamaños_TACi
UM=Umbral_medio_de_cada_VC
CMB=Capacidad_máxima_del_buffer_del_conmutador
Mientras una célula de un VCi está llegando al conmutador;
  Si la célula es la primera célula de una PDU
    Calcular UM=C * TAC/N
    Si TAC >= Umbral y TACi >= UM
      tirar la célula
    En otro caso
      aceptar la célula en la cola del VCi
      TACi = TACi + 1
    Fsi
  En otro caso
    Si ya se ha tirado alguna célula de este paquete
      tirar la célula
    En otro caso
      Si TAC >= CMB
        tirar la célula
      En otro caso
        aceptar la célula en cola del VCi
        TACi = TACi + 1
      Fsi
    Fsi
  Fsi
Fsi
FMientras

```

Figura 5.5. Algoritmo EPD per-VC queueing

Hemos de destacar que también han sido propuestas en [23] variaciones de EPD en las que se establecen varios umbrales en el buffer con la intención de mejorar aún más el rendimiento. Una de estas variantes del algoritmo PPD es ESPD que va a ser descrito a continuación.

5.4. ESPD (EARLY SELECTIVE PACKET DISCARD)

Cuando se produce congestión, EPD comienza a descartar los paquetes entrantes independientemente de la sesión a la que pertenecen. EPD no se fija por tanto en la sesión que tiene mayor actividad o que consume más recursos. Sin embargo, ESPD intenta tirar sólo los paquetes de las fuentes con mayor actividad.

La referencia [7] presenta ESPD (Early Selective Packet Discard) demostrando que consigue mejorar los índices de justicia y throughput aportados por EPD, todo ello a costa de un mínimo incremento de la complejidad de implementación. La congestión continuada es el resultado de la sincronización en la expansión y reducción de la ventana de TCP. Para evitar este problema ESPD propone una estrategia para que las sesiones tengan turnos en el acceso a la capacidad de la red, tirando las células de forma selectiva en lugar de hacerlo aleatoriamente. Aunque EPD tira las células selectivamente, cuando se trata de paquetes el mecanismo para desecharlos es aleatorio, por lo que esto puede acabar afectando a pérdidas en múltiples conexiones. Estas pérdidas activan los timeout en esas sesiones TCP que deben resincronizarse, ajustar sus ventanas y acabar sobrecargando la red. Para evitar los problemas que EPD no resuelve se propone ESPD como esquema para desechar paquetes.

ESPD tiene un *reloj* de control de paquetes descartados y además incluye tres umbrales: *umbral_1_de_buffer*, *umbral_2_de_buffer* y *umbral_de_lista-descartes*.

El primer umbral del buffer se usa para capturar los VPI/VCI en la *lista-descartes* que tiene la misma función del PPD. El segundo umbral del buffer se usa para liberar los VPI/VCI de la *lista-descartes*. La función del umbral de la *lista-descartes* se usa para poner un límite al número de VPI/VCI que se almacenan en la *lista-descartes*. Por último, el *reloj* evita las situaciones de injusticia entre las fuentes de tráfico.

La *Figura 5.6* muestra el algoritmo propuesto en [7] para implementar ESPD que, como podemos observar, incrementa sustancialmente el grado de complejidad del EPD original, aunque podemos considerar que el coste de implementación es relativamente factible para las funcionalidades que aporta en la garantía de justicia y mantenimiento del buen comportamiento de la red. Los simulaciones de ESPD demuestran un mejor comportamiento que RCD, PPD y EPD. Además es capaz de encontrar un adecuado punto de equilibrio entre el índice de justicia y throughput conseguidos, de modo que no se sacrifica la justicia por incrementar el throughput.

Debemos destacar también la existencia de otras investigaciones en el contexto de EPD, como [15], donde se estudia la peor situación en que se puede encontrar EPD para conseguir mantener la integridad de los paquetes mientras se producen elevadas sobrecargas en la red. Estos trabajos demuestran que, para mantener el 100% del goodput en la red durante las sobrecargas bajo las peores condiciones posibles, se requerirá del uso de un buffer con suficiente capacidad de almacenamiento para un paquete de longitud máxima provenientes desde cada uno de los VC. No obstante, se comprueba también que puede alcanzarse el 100% del goodput con buffers sustancialmente más pequeños que los que se predicen para el peor caso. Se observa también que puede conseguirse un adecuado goodput con buffers más pequeños, pero EPD experimenta malos comportamientos a medida que se reduce el tamaño del buffer e, incluso, el incremento del buffer también acaba provocando situaciones de degradación del goodput. Queda por tanto un amplio campo de investigación para determinar el punto de equilibrio en que conseguir, con el mínimo tamaño de buffer, el máximo goodput. La experiencia demuestra que las redes ATM experimentan sobrecargas por lo que es importante estudiar el funcionamiento de la red en estos periodos.

Si se observa el número de células en una cola gestionada por EPD como una función del tiempo, podrá observarse un comportamiento cíclico en el cual el número de células en la cola se incrementa por encima del umbral, entonces, como varios VC permanecen inactivos (porque han acabado sus paquetes o porque se les han tirado paquetes), el número de células detenidas incrementan y descienden dinámicamente. Cuando caen por debajo del umbral y llegan nuevos paquetes a la cola, el nivel del buffer se detiene cayendo y comenzando para elevarse de nuevo. Así, [15] presenta la siguiente expresión que permite calcular el goodput, donde p_{ok} denota el número de paquetes que pueden completarse durante un ciclo; T es la longitud del periodo de tiempo y l es el número de células en cada paquete,

$$goodput = \frac{p_{ok}}{T/l}$$

En este caso, el goodput es el ratio del número de paquetes que se completan en un ciclo para el máximo número de paquetes que podrían completarse en ese ciclo, durante el cual el enlace es usado por paquetes que son transmitidos completos. Así, en un periodo de tiempo de longitud T (donde la unidad de tiempo es el tiempo requerido para enviar una única célula), podemos enviar T/l paquetes por el enlace.

```

Mientras una célula está llegando a un conmutador;
Si el reloj_de_control ha expirado
    Borrar todas las entradas de la lista-descartes
FSi
Si el VPI/VCI de la célula está en la lista- descartes
    Si la célula es una célula EOM
        Si longitud_cola < tamaño_buffer
            insertar la célula en el buffer
        En otro caso
            tirar la célula
        FSi
        Si longitud_cola < umbral2_buffer
            borrar el VPI/VCI de la lista- descartes
            desactivar el reloj_de_control si está activo
        FSi
        En otro caso
            tirar la célula
    FSi
En otro caso
    Si longitud_cola <= umbral1_buffer
        insertar la célula en el buffer
    En otro caso Si ((# de entradas en la lista- descartes < umbral_lista- descartes
        y célula BOM) o el buffer está lleno))
        tirar la célula
        capturar el VPI/VCI en la lista- descartes
        Si el primer VPI/VCI está en la lista- descartes
            activar el reloj_de_control
        FSi
        FSi
    En otro caso
        insertar la célula en el buffer
    Fsi
Fsi
FMientras

```

Figura 5.6. Algoritmo de esquema ESPD

5.5. RED (RANDOM EARLY DETECTION)

Para evitar los problemas de sincronización del protocolo TCP se propuso RED [8] (Random Early Detection gateways) que se encarga de tirar o marcar cada paquete que llega cuando se tiene una cierta probabilidad de ser descartado, condición que es detectada cuando la longitud media de la cola excede un umbral preestablecido. RED intenta mantener el tamaño de las colas tan bajo como sea posible, mientras se permiten ráfagas ocasionales. Se ha demostrado que RED mantiene un elevado grado de throughput mientras se consigue minimizar el retardo. La idea es monitorizar la longitud de las colas y evitar que los paquetes sean tirados cuando en la red se producen cambios. Además, garantiza que las conexiones pueden tener un alto grado de compartición del ancho de banda como se comentó en el Capítulo 4. Partiendo del trabajo original [8], en el artículo [9] se presenta una adaptación de RED a la CoS UBR, adaptando el algoritmo RED a las redes ATM.

Es conocido que las redes ofrecen mecanismos de realimentación que aportan a las aplicaciones diversas técnicas para poder conocer el estado de la red y poder monitorizar los envíos de sus datos. En las redes de conmutación de paquetes como ATM puede hablarse de dos tipos de realimentación. La explícita permite el uso de campos especiales en las células como es el caso de ABR con sus células RM para indicar congestión. Por otro lado, la realimentación implícita se basa en respuestas de la red a las variaciones en el comportamiento de las fuentes. Esta última puede ser deducida gracias a las variaciones en los retardos o por las pérdidas de los paquetes. Pues bien, RED emplea notificación de congestión implícita a través de los paquetes tirados. En lugar de esperar a que las colas se llenen y comenzar a desechar los paquetes que lleguen a partir de entonces, RED decide descartar los paquetes que llegan con una probabilidad elevada cada vez que la longitud media de las colas excede de un determinado umbral. Para cada paquete que llega al conmutador se estima el tamaño medio de la cola mediante un filtro paso bajo:

$$Long_Media = (1 - w_q) * Long_Media + w_q * Tama\~{n}o_cola;$$

donde w_q es una constante que satisface $0 \leq w_q \leq 1$.

El algoritmo de RED es resumido en la *Figura 5.7*, donde *contador* es el número de paquetes encolados tan largos como *Long_Media* quedando entre los dos *umbrales*. El *contador* es puesto a cero en cada pérdida. El valor Max_p es la máxima probabilidad de desecho de paquetes, probabilidad que es también función del tamaño de los paquetes.

```

 $P_a = P_b / (1 - contador * P_b)$ 
 $P_a = Max_p * (Long\_Media - Umbral\_Mínimo) * (Umbral\_Máximo - Umbral\_Mínimo)$ 
 $P_b = P_b * Tama\~{n}o\_Paquete / Tama\~{n}o\_Máximo\_Paquete$ 
Si  $Long\_Media \leq Umbral\_Mínimo$ 
    Aceptar el paquete
FSi
Si  $Umbral\_Mínimo < Long\_Media < Umbral\_Máximo$ 
    Calcular probabilidad  $P_a$ 
    Tirar los paquetes que llegan con probabilidad  $P_a$ 
FSi
Si  $Umbral\_Máximo \leq Long\_Media$ 
    Tirar los paquetes que llegan
FSi
    
```

Figura 5.7. Algoritmo RED

Tomando como base el algoritmo original RED, que se ha demostrado que no escala del todo bien en redes de alta velocidad, en [9] se han propuesto dos variantes C-RED y P-RED para conseguir esta escalabilidad en redes ATM, consiguiéndose reducir la complejidad de implementación del algoritmo original y obteniendo un mayor grado de justicia.

5.6. VC MERGE

Con la intención de reducir los cuellos de botella provocados en los routers, (inundados por el creciente tráfico IP), que constituyen las redes de alta velocidad, se han propuesto múltiples variantes de integración de la capa 3 de routing con la capa 2 de conmutación. La mayoría de ellas se apoyan en el establecimiento de etiquetas en la capa 2 que permitan mapearse en las tablas de routing de capa 3, dando lugar así a la evolución del clásico mecanismo de routing de capa 3 hacia la conmutación de capa 2. Con este planteamiento parece lógico que ATM sea la tecnología de capa 2 de conmutación y, de este modo, se han propuesto varias posibilidades para realizar el mapeo de la información de las rutas IP a las etiquetas ATM.

La técnica de mapeo más simple es que cada pareja inicio-destino se mapeen en un único valor de VC en el conmutador ATM. Este método es conocido como *non-VC merging* [17,18] y permite a los receptores reensamblar células de una forma muy sencilla en los respectivos paquetes, ya que el propio valor de VC puede ser usado para diferenciar los emisores. El problema aparece cuando se necesita que esta técnica escale adecuadamente con el crecimiento de los emisores y receptores. Es decir, si tenemos n emisores y receptores, cada conmutador deberá soportar $O(n^2)$ etiquetas de VC para poder soportar una conectividad completamente mallada entre todos los emisores y receptores. Esto hace que las tablas de rutas VC crezcan sustancialmente a medida que crecen los nodos que se incorporan a la red (en el caso de 1.000 nodos será necesario que las tablas soporten 1 millón de entradas).

El segundo método propuesto es conocido como *VP merging* y consiste en etiquetar los VP de forma que las células etiquetadas con VP con el mismo destino se mapean al mismo valor de VP saliente en los conmutadores. Esto permite una sustancial reducción de VP ya que, para cada VP, se emplea el valor de VC como identificador del emisor con la intención de que el receptor sea capaz de reconstruir los paquetes, incluso, aunque las células de diferentes paquetes acaben experimentando el problema de interleaving. En este caso, para cada destino, un conmutador debe localizar $O(p)$ etiquetas de VP, donde p representa el número de puertos⁵ de cada conmutador. En cualquier caso, también se depende del tamaño de la red, ya que si tenemos n destinos, cada conmutador deberá gestionar $O(np)$ etiquetas de VP que, desde luego, es una considerable mejora respecto *non-VC merging*. Sin embargo, aunque el espacio de etiquetas en las tablas es

⁵ Un número de puertos habitual en entornos locales es 16.

razonable, el mayor problema de *VP merging* es que el espacio de VP utilizables es muy pequeño 2^{12} (4.096) en los interfaces NNI⁶.

El tercer método se encarga de mapear las etiquetas de VC entrantes en los conmutadores, dirigidas a un mismo destino por la misma etiqueta de VC saliente. Esta solución es tan escalable como *VP merging O(np)* y además, no se encuentra con el problema del espacio de valores que pueden tomar los VC en la red, ya que, tanto en UNI como NNI el espacio de valores 2^{16} (65.535) es más que suficiente. Con *VC merging* las células que tienen el mismo destino no son distinguibles ni diferenciables en los puertos de salida de los conmutadores. Así, las células que pertenecen a paquetes distintos, pero que van dirigidos al mismo destino no podrán entremezclarse con otras y además, el receptor no tendrá la necesidad de realizar labores de reensamblado. Precisamente, esta característica es la que queremos aprovechar en nuestro protocolo para garantizar que los paquetes que van a salir por el mismo puerto no puedan entremezclarse con otros, por compartir el mismo valor de VC. En realidad, en nuestro caso realizamos una reinterpretación de la técnica *VC merge* aplicada al buffer, ya que nuestro objetivo es evitar que a la salida del buffer se produzcan mezclas de diferentes conexiones. Es decir, cuando un paquete sale del buffer garantizamos que va a ser tratado por completo antes de atender ningún otro. De este modo, las PDU son transferidas a su correspondiente puerto de salida sin ninguna interferencia de entrelazado con células de otras conexiones.

Como podemos comprobar, la técnica *VC merge* parece la más interesante de las tres, aunque es necesario partir de la base que requiere más espacio de buffer que *non-VC merge*, además los conmutadores tienen requerimientos hardware para soportar el reensamblado y evitar el *interleaving*. Así en [12,13] se proponen OM (Output Modules) encargados de realizar la traslación de VCI a las salidas de los conmutadores. A cada célula ATM que llega se le añaden dos campos que contienen un número de puerto de entrada y un número de puerto de salida. Partiendo del número de puerto de salida, los conmutadores reenvían cada célula al correspondiente puerto de salida de los OM. Cuando no se aplica *VC merging*, los OM se comportan como simples buffers de salida. Para desempeñar esta labor los OM disponen de RB (Reassembly Buffers) que se encargan de relacionar cada VC de entrada con un puerto de entrada, garantizando que las células de un paquete no se mezclan con las células de otros paquetes que comparten el mismo VC⁷. Durante la transferencia de un paquete hasta el buffer de salida, el VCI entrante es convertido al VCI saliente y, para ahorrar traslaciones de VCI, lo que se hace es que VCI entrantes diferentes se mezclan (*merge*) asignándose el mismo VCI saliente durante la traslación y siempre que las células vayan dirigidas al mismo destino. Puede optarse por una mezcla de VCI total o parcial y elegir una u otra depende de las características de las fuentes de tráfico. Por ejemplo, en el caso de tener que atender conexiones con diferentes necesidades de CoS se puede implementar *merging* parcial de forma que los VCI entrantes con mismo destino y misma necesidad de QoS se mapeen por el mismo VCI de salida, pero se separen de éste aquellos flujos que no requieren esa QoS como podría ser el tráfico best-effort.

La técnica *VC merge* permite que varias conexiones sean mapeadas con la misma etiqueta de VC, aportando un mecanismo escalable para soportar mucho miles de conexiones. Para escalar adecuadamente, la técnica *VC merge* requiere la presencia de buffers de reensamblado [17,18] para que las células pertenecientes a los diversos paquetes, y preparadas para el mismo destino⁸, no se mezclen con otras. Así, es importante eliminar el problema de *interleaving*, aunque para ello sea necesaria la presencia de buffers adicionales que soporten la característica *VC merge*.

Las investigaciones realizadas [17] demuestran que *VC merge* genera un overhead mínimo si lo comparamos con *non-VC merging* en términos de buffers adicionales. En realidad, el overhead descende a medida que el tráfico se incrementa o a medida que se generan más ráfagas. Pero también se ha demostrado que el retardo adicional en que se incurre por aplicar este mecanismo es mínimo para la mayoría de aplicaciones.

Otras interesantes investigaciones [19-22] han realizado propuestas en la línea de la técnica de *merging* con variantes hacia el problema del multicasting como el *compound VC* introducido en [19,22] para integrar el IP multicast sobre ATM multicast nativo. En este caso los grupos multicast de IP son asociados a grupos de VC (*compound VC*) de forma que, con una sola entrada en la tabla de conmutación, se puede conmutar el tráfico de un grupo multicast completo gracias al uso de máscaras. Se emplea multiplexación ID per PDU como en [21], pero aportando mayor escalabilidad en cuanto al número de grupos multicast que se pueden soportar. Se evita el *interleaving* de las células mediante la asignación dinámica de identificadores de cada PDU.

⁶ La longitud del campo VPI de 12 bits en las células ATM limita sustancialmente la capacidad de direccionamiento de *VP merging*.

⁷ Este mecanismo se conoce con el nombre de store-and-forward de paquetes.

⁸ En este contexto la palabra destino se refiere a la red destino (prefijo CIDR), pero puede también entenderse como mismo nodo destino, mismo puerto de destino, mismo usuario destino, misma CoS destino, misma QoS destino, etc.

5.7. PROPUESTA EPDR (EARLY PDU DISCARD AND RELAY) PARA TAP

Como ya hemos comentado, la arquitectura TAP incorpora un esquema de control del buffer basado en EPD. Nuestro algoritmo es novedoso respecto al esquema EPD original y sus variantes existentes porque:

- Incluye un mecanismo de retransmisión cuando se detectan PDU que van a provocar el rebosamiento del buffer. Por tanto, cuando se produce la situación de buffer lleno, la PDU que es tirada será automáticamente solicitada al conmutador previo para ser retransmitida. De este modo, se optimiza el goodput de una forma muy superior a cualquier otro esquema de la literatura que suponen que la retransmisión será realizada extremo-extremo. Las retransmisiones sólo se solicitan en el caso que la fuente tenga suficiente tiempo de inactividad para atenderlas sin afectar a la transmisión en curso. Así, nuestra propuesta permite ahorrar, en el peor de los casos, el tiempo *RTT extremo-extremo* y además, el proceso de solicitud de retransmisión de los protocolos de las capas superiores como TCP. Y lo que es más importante, la implosión en los emisores de tráfico es solventada por la delegación realizada en los conmutadores AcTMs que soportan la arquitectura TAP.
- El problema de la fragmentación de las PDU es solventado empleando el campo EOM de las EAAL-5 que es otra de las propuestas del protocolo que soporta TAP. De este modo se reduce al máximo la posibilidad de desechar PDU, aunque la fragmentación no es completamente eliminada como en el resto de variantes de EPD.
- El problema de la justicia no es un objetivo del esquema de control del buffer ya que esta labor es desempeñada por el algoritmo QPWFQ comentado en el capítulo anterior. Por tanto, la justicia está garantizada, pero también la caracterización del tráfico que requiere que ciertas fuentes de tráfico tengan prioridad sobre otras en función de los pesos asignados a cada una de las colas de entrada de la arquitectura.
- El buffer es controlado por un agente programable que permite la coordinación con otros agentes del sistema para ajustar los umbrales más convenientemente en función de las situaciones del estado de cada conmutador. Este agente permite al operador de la red dimensionar el buffer y elegir el esquema a aplicar. En estos momentos puede elegirse entre PPD y EPDR.
- El problema del interleaving es evitado por una variante de *per-VC queueing* que nos permite procesar las PDU existentes en el buffer de forma separada sin provocar la mezcla de células a la salida. De todos modos, este problema también es amortiguado por el esquema QPWFQ que se encarga de enviar al buffer de forma continua todas las células de la misma PDU. Además, QPWFQ ya atiende el tráfico aplicando técnicas *per-VPI/VCI*.
- El coste algorítmico es similar a los esquemas que hemos tenido la ocasión de revisar en apartados previos de este capítulo. En cuanto a requerimientos hardware tampoco necesitamos recursos extraordinarios.

Todos estos aspectos serán discutidos detalladamente en capítulos siguientes, pero queremos avanzar tanto el algoritmo, como una representación gráfica de la sección de la arquitectura que se encarga de esta labor en los conmutadores AcTMs.

La *Figura 5.8* ilustra el algoritmo que implementa el esquema EPDR de control del buffer basado en [7] en el que hemos introducido la función *Retransmitir(indexPDU)* que describiremos también detalladamente en el *Capítulo 10*. El valor del umbral es determinado por el agente programable CCA en función de la determinación del operador de la red y del estado de la propia red. Como podemos comprobar en el algoritmo, en cuanto la longitud actual de la cola iguala el valor del umbral, el conmutador se dispondrá a tirar todas las células de la PDU que ha provocado la situación. Pero en lugar de empezar a descartar células en cuanto se iguala el umbral, se espera a que llegue la primera célula de una PDU (BOM de AAL-3). Es decir, cuando llega la primera célula de una PDU y ya se ha igualado el umbral, se tira esa célula y se marca el VPI/VCI de la célula en la *lista-descartes* para, a continuación, tirar todas las células de esa misma PDU hasta llegar a su última célula (EOM). Este es el modo en que se garantiza que no se produzca la fragmentación de la PDU, impidiendo que no se envíe ninguna célula de una PDU que ha provocado una situación de congestión. Una vez ha llegado la última célula de la PDU congestionada, y puede accederse al campo PDUid, se solicita la retransmisión de la PDU a través del agente RCA. Una vez solicitada la retransmisión de la PDU se procede a eliminar el valor VPI/VCI de la *lista-descartes* para que la siguiente PDU de esa misma conexión pueda ser aceptada en el buffer si ya ha pasado la situación de congestión.

Como puede comprobarse también, la célula EOM de la PDU congestionada se introduce en el buffer si tiene espacio para ello, ya que esta célula es la que usa EAAL-5 (como el estándar AAL-5) para delimitar

unas PDU de otras. De este modo se sabe cuándo acaba una PDU congestionada. Destacamos también que, al igual que en las propuestas originales de EPD, el algoritmo EPDR no solventa la fragmentación cuando una vez insertadas las células iniciales de una célula se llena el buffer. La mejor forma de evitar que esto se produzca es la elección adecuada del valor de umbral, que debería ser menor en unas tres o cuatro PDU que el límite del buffer. No obstante, aunque puedan seguir adelante las células iniciales de la PDU, las restantes células hasta llegar a la célula EOM no serán transmitidas para no seguir enviando células de una PDU que se sabe ya corrompida.

```

Mientras una célula llega al buffer
Si el VPI/VCI de la célula pertenece a la lista-descartes
  Si la célula es una célula EOM
    Si Longitud_de_Cola < Tamaño_Buffer
      insertar la célula en el buffer
    En otro caso
      tirar la célula
    FSi
      Retransmitir(indexPDU)
      borrar el VPI/VCI de la lista-descartes
    FSi
  En otro caso
    tirar la célula
  FSi
En otro caso
  Si Longitud_de_Cola < umbral
    insertar la célula en el buffer
  En otro caso Si (primera célula de PDU o (el buffer está lleno))
    tirar la célula
    capturar el VPI/VCI en la lista-descartes
  FSi
En otro caso
    insertar la célula en el buffer
  FSi
FSi
FMientras
  
```

Figura 5.8. Esquema de planificación del algoritmo EPDR

La *Figura 5.9* muestra una representación gráfica del buffer en el que podemos observar todos los elementos que intervienen. Podemos observar el punto de llegada del tráfico de entrada, así como los límites de los valores de la *Longitud Actual de Cola (LAC)*, el valor *Umbral (U)* y el valor de *Tamaño Máximo del Buffer (TMB)*. También puede comprobarse la comunicación existente entre el buffer y los agentes WFQA, CCA, RCA y DPA, así como la conexión del buffer con la memoria DMTE en la que se realizan las copias de las PDU que han llegado completas al buffer.

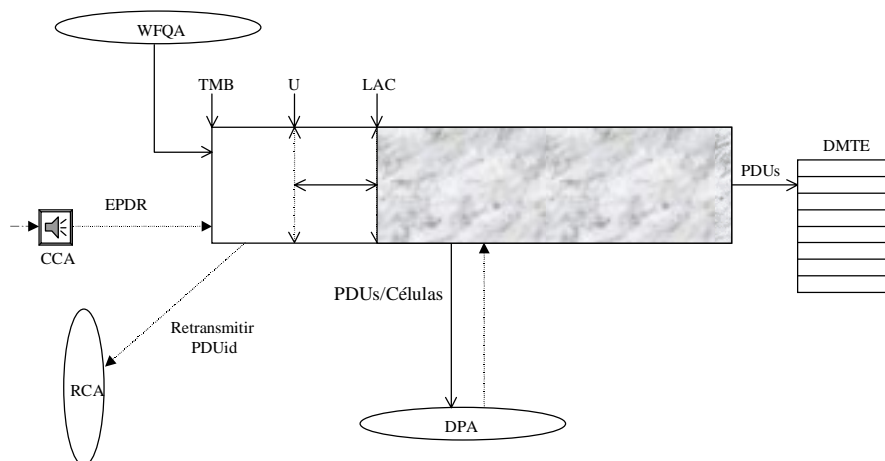


Figura 5.9. Esquema gráfico del buffer

Por último destacamos que en nuestro caso el VC-merging se basa en *store-and-forward* de paquetes en el buffer, de forma que se almacenan las células de cada paquete que llegan al buffer hasta que llega la última célula de cada PDU. Cuando llega la última célula de una PDU se transfieren de una forma atómica⁹ al correspondiente puerto de salida del conmutador. Desde el punto de vista de la implementación, las operaciones atómicas nos suponen no mucho más que la sincronización de dos agentes software y el movimiento de un puntero en el buffer.

5.8. CONCLUSIONES

Resumiendo, el hecho de que los protocolos extremo-extremo envíen información en paquetes conteniendo múltiples células ATM provoca que el impacto de periodos de sobrecarga empeoren aún más el comportamiento de la red porque la pérdida de una sola célula puede conducir a la pérdida y retransmisión entre los extremos de la PDU completa de la capa de transporte. Esto hace que durante los periodos de sobrecarga, las redes ATM puedan experimentar congestión y colapso. Mediante los esquemas de control del buffer pueden garantizarse el goodput y la justicia entre las diversas fuentes de tráfico. En nuestro caso incorporamos un algoritmo que solventa los problemas de fragmentación, implosion e interleaving.

REFERENCIAS

- [1] M., Hluchyj, "Closed-Loop Rate-based Traffic Management," *ATM Forum Contribution 94-0438R2*, (1993).
- [2] R. Jain, S. Kalyanaraman, and Viswanathan, "Simulation Results: The EPRCA+ scheme," *ATM Forum Contribution 94-0988*, (1994).
- [3] H. Ohsaki, M. Murata, H. Suzuki, C. Ikeda, and H. Miyahara, "Rate-based Congestion Control for ATM Networks," *Computer Communications Review*, pp. 60-72, (1995).
- [4] _____ ATM Forum, "Traffic Management Specification Version 4.0," *ftp://ftp.atmforum.com/pub/approved-specs/afm-0056.000.ps*, (1996).
- [5] G. Armitage and K. Adams, "Packet Reassembly during Cell Loss," *IEEE Networks*, vol. 7, no 5, pp. 26-34, Sept. (1993).
- [6] Allyn Romanov and R. Oskouy, "A performance enhancement for packetized ABR and VBR+data," *AF-TM 940295*, (1994).
- [7] Kangsik Cheon and Shivendra S. Panwar, "Early Selective Packet Discard for Alternating Resource Access of TCP over ATM-UBR," *IEEE LCN'97*, pp. 306-316, (1997).
- [8] Sally Floyd and Van Jacobson, "Random Early Detection Gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, pp. 397-413, (1993).
- [9] Omar Elluomi and Hossam Afifi, "RED Algorithm in ATM Networks," *IEEE SIGCOMM'2000*, pp. 312-319, (2000).
- [10] José Luis González-Sánchez and Jordi Domingo-Pascual "RAP: Protocol for Reliable Transfers in ATM Networks with Active Switches," *International Conference on Communications in Computing (CIC'2000)*, pp. 141-148, (2000).
- [11] Allyn Romanov and Sally Floyd, "Dynamics of TCP Traffic over ATM Networks," *IEEE Journal on Selected Areas in Communications*, pp. 633-641, (1995).
- [12] R. Goyal, G. Jain, S. Fahmy, and S. Kim, "Performance of TCP over UBR+," *AF-TM 96-1269*, (1996).
- [13] Hongqing Li, Kai-Yeung Siu, Hong-Yi Tzeng, C. Ikeda, and H. Suzuki, "Performance of TCP over UBR service in ATM networks with Per-VC Early Packet Discard schemes," *Proceedings IEEE IC3N*, pp. 350-357, (1996).
- [14] Yuang Wu, Kai-Yeung Siu and Wenge Ren, "Improved Virtual Queueing and Dynamic EPD Techniques for TCP over ATM," *IEEE International Conference on Network Protocols*, pp. 212-219, (1997).
- [15] Maurizio Casoni and Jonathan S. Turner, "On the Performance of Early Packet Discard," *IEEE Journal on Selected Areas in Communications*, Vol. 15, no 5, pp. 892-902, (Jun. 1997).
- [16] Maurizio Casoni "Early Packet Discard with diverse management policies for EOM cells," *IEEE Proceedings International Workshop on Broadband Switching Systems*, pp. 33-37, (1997).
- [17] I. Widjaja, and I. I. Elwalid, "Performance issues in VC.merge capable switches for IP over ATM networks," *IEEE Proceedings INFOCOM'98*, pp. 372-380, (1998).
- [18] I. Widjaja, and A. I. Elwalid, "Performance issues in VC.merge capable switches for multiprotocol label switching," *IEEE Journal on Selected Areas in Communications*, pp.1178-1189, (1999).

⁹ La atomicidad la implementamos como el envío independiente de cada paquete a la salida en una sola operación.

- [19] Josep Manges-Bafalluy and Jordi Domingo-Pascual, "Performance Issues of ATM Multicasting Based on Per-PDU ID assignment," *Proceedings IEEE Int'l Conference Communications (ICC'00)*, (2000).
- [20] M. Baldi, D. Bergamasco, S. Gai, and D. Malagrino, "A Comparison of ATM Stream Merging Techniques," *Proceedings of IFIP High Performance Networking*, pp. 212-227, (1998)
- [21] J. Calvignac, P. Droz, C. Baso, and D. Dykeman, "Dynamic Identifier Assignment (DIDA) for merged ATM connections," *ATM Forum 97-0316* (1997).
- [22] Josep Manges-Bafalluy and Jordi Domingo-Pascual, "Compound VC Mechanism for Native Multicast in ATM Networks," *Proceedings of the ICATM'99*, pp.115-124, (1999).
- [23] Jonathan S. Turner, "Maintaining High throughput during overload in ATM switched," *Proceedings IEEE INFOCOM'96*, pp. 287-295, (1996).
- [24] RR. Jain, "Congestion control and traffic management in ATN networks:Recent advances and a survey," *Comput. Networks ISDN Syst.*, (1996).