

CAPÍTULO 11

PROTOCOLO TAP (TRUSTED AND ACTIVE PROTOCOL)

11.1. INTRODUCCIÓN

En el capítulo anterior hemos descrito cada uno de los componentes de la arquitectura TAP, poniendo especial atención en los bloques hardware de los conmutadores activos AcTMs. Hemos comentado ya que cada uno de esos bloques hardware son controlados por diversos algoritmos que, en la mayor parte de los casos, se corresponden con la implementación de los agentes que componen la arquitectura. Por tanto, la arquitectura hardware propuesta está equipada con una importante faceta software que constituye el protocolo TAP (*Trusted and Active Protocol*) es implementado sobre la propia arquitectura hardware.

En el punto 3.2 del *Capítulo 3* hemos justificado el calificativo *trusted* de nuestro protocolo, en comparación con el concepto *reliable*. Como sabemos, TAP busca la aportación de la garantía de servicio a las conexiones que lo requieran y que puedan verse afectadas por las impredecibles congestiones de los conmutadores tradicionales. Nuestra aportación *trusted* es, por tanto, un complemento para los *reliable protocols* que aportan la fiabilidad a las fuentes que pueden experimentar errores en sus transferencias.

Por otro lado, la característica *active* de TAP ha sido argumentada en el apartado 6.11 del *Capítulo 6*, donde hemos introducido las ventajas de aportar a las redes de comunicaciones componentes software en forma de agentes que las conviertan en programables. Los agentes software que hemos incorporado a la arquitectura TAP convierten a los conmutadores que los soportan en elementos activos en cuanto al procesamiento de los flujos de datos y a las propias labores de control de la red. La incorporación de los agentes software en nuestra propuesta nos permite, por un lado identificar a los conmutadores que los incorporan como conmutadores ATM activos (que denominamos AcTMs) y, por otro, definir al protocolo distribuido sobre la VPN constituida por los AcTMs también como activo.

Además, el *Capítulo 7* nos ha permitido extraer las ventajas de los sistemas y las arquitecturas distribuidas y que hemos incorporado a la visión que pretendemos dar de la red privada virtual sobre la que se ejecuta TAP, que se convierte así en un protocolo distribuido a lo largo de toda la red. Esta visión distribuida permite evitar las dependencias de nodos centralizadores, de forma que TAP funciona independientemente de las características de los nodos que componen la VPN. Es decir, pueden existir nodos en la red que no soporten TAP. Además, no existe ningún nodo que gestione la información completa sobre el estado de la red, de modo que cada conmutador puede tomar decisiones basándose sólo en su información local. Todo esto acaba beneficiando a la integridad, eficacia y robustez del protocolo.

A la vista de todo lo anterior puede entenderse el protocolo TAP como un conjunto de algoritmos distribuidos a lo largo de toda la red, algunos de los cuales aportan la característica activa a los conmutadores, mientras otros se responsabilizan de ofrecer la garantía de servicio a aquellas conexiones que deciden usarlos. Según esto, dispondremos de un algoritmo principal que es ejecutado extremo-extremo entre los terminales de la comunicación, y del que dependen el resto de algoritmos que son implementados en los conmutadores AcTMs de la red. A todo el conjunto de algoritmos lo denominamos TAP y hemos de destacar que existe una relación muy estrecha entre ellos y el subsistema que constituye el SMA-TAP.

En este capítulo, por tanto, se van a describir las consideraciones más importantes acerca de cada uno de los algoritmos que componen TAP, con la intención de conocer sus detalles de implementación y su funcionamiento. Destacamos que los algoritmos presentados son sólo una descripción de las funciones a realizar en el simulador de TAP, más que de la propia implementación hardware de los AcTMs que no es objetivo de esta tesis. Los algoritmos intentan dar una idea de la forma en que se realizarían por software las funciones más importantes; sin embargo, la arquitectura hardware final permitirá realizar las funciones de transferencia de forma más eficiente. En este capítulo demostraremos además la base de funcionamiento de nuestra idea de aprovechar los estados de inactividad de las fuentes de tráfico para atender las solicitudes de retransmisión, de forma que no se afecte al rendimiento de la red. Este es uno de los fundamentos de la filosofía de funcionamiento de TAP y para estudiarlo nos apoyaremos en la teoría de fuentes ON/OFF, que serán también empleadas en el prototipo de TAP que hemos desarrollado y que será explicado en el *Capítulo 12*. Estudiaremos también en este capítulo las implicaciones sobre la señalización de la red.

11.2. PROTOCOLO TAP EN LOS NODOS TERMINALES Y AcTMs

En primer lugar vamos a describir el algoritmo TAP desde el punto de vista de los extremos terminales de la comunicación. Para ello nos vamos a basar en un escenario con un canal unidireccional y punto-a-punto. Destacamos que los algoritmos presentados se centran principalmente en los aspectos de mayor interés para entender el comportamiento de TAP. Por esto hemos obviado diversos aspectos estándares de la propia tecnología ATM (control de errores, segmentación y reensamblado, primitivas, etc.) para no distraer la atención del objetivo principal de este capítulo. Con la intención de no hacer esta descripción demasiado extensa presentamos también un diseño estructurado y en sus primeros niveles de abstracción, por lo que no entramos a describir los detalles de implementación de las funciones empleadas.

El primer aspecto que debemos comentar es la implementación del protocolo sobre el terminal emisor, en el que se soportan todas las capas que fueron presentadas en la *Figura 10.1*, donde puede observarse que TAP está situado en el modelo ATM por debajo de protocolos de capas superiores, y por encima de nuestra propuesta EAAL-5. De este modo, el terminal emisor debe encargarse de los primeros aspectos de la comunicación y, tal como muestra la *Figura 11.1*, podemos comprobar cómo el algoritmo TAP negocia los parámetros de la comunicación y, además, se encarga de iniciar todos los demás algoritmos del conjunto que constituyen el protocolo completo. Por esto inicia y pasa los parámetros a cada uno de los agentes que están soportados en los nodos AcTMs de la VPN.

Una vez iniciada la conexión extremo-extremo entre los dos nodos terminales y la secuencia de PDU garantizadas, podemos observar en la *Figura 11.1* cómo el nodo emisor recibe los paquetes del tráfico desde capas superiores que serán después debidamente procesados por EAAL-5. Cuando concluye el tiempo de la transferencia, o se decide liberar la conexión por alguno de los extremos, termina también el algoritmo TAP en el nodo emisor.

```

Negociacion_Conexion(VPI,VCI,Ton,Toff,PCR,tiempoSimul,gradoGoS,ToS,algoritmo,umbral)
Inic_CoSA(VPI,VCI); /* Inicia el agente de Clase de Servicio */
Inic_CCA(algoritmo,umbral); /* Establece el algoritmo y umbral a aplicar sobre el buffer */
Inic_DMTE(VPI,VCI,gradoGoS); /* Determina N° PDU almacenadas en DMTE por conexión */
Inic_RCA(evento); /* Inicia el agente RCA */
Inic_WFQA(VPI,VCI,PCR); /* Inicia el agente WFQA y establece peso de cola para VCI*/
Inic_DPA(evento); /* Inicia el agente DPA */
Inic(tiempoSimul); /* Activa el reloj de duración de la simulación */
PDUid:=0;

Mientras tiempoSimul > 0 o Liberacion_Conexion(VPI,VCI) = "N"
    paq_TAP:=Desde_Protocolo_Superior(paquete); /* Toma paquete de capa superior */
    Hasta_EAAL-5(VPI,VCI,paq_TAP); /* Pasa el paquete a la capa EAAL-5 */
FMientras;
Liberacion_Conexion(VPI,VCI); /* Concluida la transferencia se libera la conexión */

```

Figura 11.1. Algoritmo TAP en el terminal emisor

La *Figura 11.2* presenta el algoritmo TAP que se ejecuta sobre el terminal receptor por lo que, en este caso, el flujo de datos sigue el sentido contrario al del nodo emisor. Es decir, en el receptor se reciben las PDU desde la capa EAAL-5 para ser transferidas hasta los protocolos de capas superiores. La comunicación concluye de forma similar a lo visto en el caso del nodo emisor.

```

Mientras tiempoSimul > 0 o Liberacion_Conexion(VPI,VCI) = "N"
    paq_TAP:=Desde_EAAL-5(VPI,VCI,PDU); /* Recibe PDU desde capa EAAL-5 */
    Hasta_Protocolo_Superior(paq_TAP); /* Envía la PDU al protocolo superior */
FMientras
    Liberacion_Conexion(VPI/VCI); /* Concluida la transferencia se libera la conexión */
    
```

Figura 11.2. Algoritmo TAP en el terminal receptor

11.2.1. EAAL-5 (EXTENDED ATM ADAPTATION LAYER 5)

El siguiente algoritmo que vamos a comentar es precisamente el de la capa EAAL-5 del nodo emisor que está debajo del algoritmo TAP que se comentó en la *Figura 11.1*. Este algoritmo presentado en la *Figura 11.3* se encarga, principalmente, de generar en el terminal emisor las unidades de PDU de las conexiones privilegiadas. De este modo, se asigna a cada PDU su identificador correspondiente que, como ya sabemos, es la base del mecanismo de recuperación de congestiones. Una vez generadas las PDU, éstas son segmentadas en células independientes y transferidas a la capa ATM estándar que se encargará de ponerlas en la red a través de la capa Física del emisor. Puede observarse cómo cuando se agota la secuencia de 65.536 PDU generadas, ésta es reiniciada nuevamente.

```

Mientras No_Liberacion_Conexion(VPI,VCI);
    paq:=Desde_TAP(VPI,VCI,paq_TAP); /* Recibe un paquete desde la capa TAP */
Mientras NoFin(paq)
    PDU:=Generar_PDU(VPI,VCI,paq,PDUid); /* Genera PDU y le asigna PDUid */
    PDUid:=PDUid+1;
Mientras NoFin(PDU)
    celula:=SAR(VPI,VCI,PDU); /* Segmentación de la PDU en células */
    Hasta_ATM(celula); /* Pasa cada célula a la capa ATM del emisor */
FMientras;
Si PDUid=65.536 entonces PDUid:=0 /* Inicia la secuencia de numeración de PDU */
FMientras;
FMientras;
    
```

Figura 11.3. EAAL-5 en el terminal emisor

A continuación nos encontramos con el algoritmo EAAL-5 ejecutado en el nodo receptor de la conexión. Como puede comprobarse en la *Figura 11.4*, en este caso el sentido del flujo es el contrario al de la *Figura 11.3*, ya que ahora se reciben las células desde la capa ATM, por lo que EAAL-5 se encarga de reensamblarlas en unidades de PDU que, una vez localizada su célula final (EOM), son transferidas hasta la capa en que se encuentra el algoritmo TAP. Como vimos antes, TAP se responsabilizará de hacerlas llegar a los protocolos superiores.

```

Mientras No_Liberacion_Conexion(VPI,VCI)
Hasta EOM(VPI,VCI,cel)
    cel:=Desde_ATM(celula); /* Recibe una célula desde la capa ATM */
    PDU:=SAR(VPI,VCI,cel); /* Reensamblado de las PDU */
FHasta;
Si ToS="D" Hasta_TAP(VPI,VCI,PDU) /* Pasa PDU al protocolo TAP */
En otro Caso entonces
Si Longitud <= ventana_ordenacion entonces Comenzar
    Ordenar(VPI,VCI,PDUid,ListaPDUs); /* Servicio ordenado */
    Longitud:=Longitud(ListaPDUs)+1;
FSi;
FSi;
Si Longitud = ventana_ordenacion entonces
Mientras Longitud > 0
    PDU:=Cabecera(ListaPDUs);
    Hasta_TAP(VPI,VCI,PDU); /* Pasa PDUs ordenadas a TAP */
    Longitud:=Longitud-1;
FMientras;
FSi;
FMientras;
    
```

Figura 11.4. EAAL-5 en el terminal receptor

En este caso EAAL-5 se debe encargar también de, en función del tipo de servicio (ToS) elegido en el establecimiento de la conexión por el terminal emisor, hacer la ordenación de las PDU que se reciben. Como ya sabemos por capítulos anteriores, el mecanismo de retransmisiones puede acabar generando desorden en las PDU, por lo que el protocolo dispone de dos tipos de servicio diferentes: uno ordenado y otro secuencial. Mientras el primero se encarga de pasar a la capa TAP las PDU ordenadas y detectando fallos de secuencia o pérdidas; el segundo transfiere a TAP las PDU nada más ensamblarlas, delegando la ordenación y detección de pérdidas a los protocolos superiores. Mientras el primer servicio es más elaborado y costoso en cómputo, el segundo es menos cuidadoso pero más rápido. Recordamos antes de concluir que el algoritmo EAAL-5 es sólo implementado en los equipos terminales de la comunicación, y no en los nodos AcTMs que sólo consultan los identificadores de PDU que procesan. Esto es así para evitar introducir retardos en la red, y también para mantener las características estándares de ATM, donde los conmutadores sólo disponen de las capas Física y ATM.

11.2.2. ALGORITMO TAP SOBRE LOS AcTMs

Los nodos terminales de la comunicación soportan el protocolo TAP descrito en las figuras anteriores e, igualmente, éstos incluyen la extensión EAAL-5 que acabamos de comentar. Sin embargo, los nodos activos de la red no se equipan con EAAL-5 para no afectar al rendimiento de la red y, del mismo modo, tampoco soportan las características del protocolo TAP que hemos descrito en el caso de los nodos terminales emisor y receptor. No obstante, y dado que ya sabemos que TAP es un protocolo distribuido en toda la VPN constituida por los nodos AcTMs que la forman, queremos comentar algunos de los aspectos de los que se encarga esta variante de TAP que se ejecuta sobre los conmutadores activos.

En cierta forma, este algoritmo que vamos a comentar actúa a modo de columna vertebral sobre la que se articulan el resto de algoritmos que funcionan en los AcTMs. Esto es así porque el algoritmo de la *Figura 11.5* se encarga de copiar las PDU fiables desde el buffer a la DMTE, y también de atender las peticiones del agente DPA, ya sean desde el buffer en las transmisiones, o desde la DMTE en el caso de retransmisiones. Para ello, este algoritmo necesitará cooperar con los agentes CCA y DPA.

```

Sincronizacion_CCA(VPI,VCI,PDUid);          /* Sincronización con el agente WFQA local */
Sincronizacion_DPA(evento);                 /* Sincronización con otros agentes RCA de la VPN*/
Inic_DMTE(VPI,VCI,gradoGoS);               /* Determina N° PDU almacenadas en DMTE por conexión */
Mientras cierto
  Si Fincelula() entonces put(buffer,célula); /* Notificación desde CCA de fin de célula en buffer */
  Si FinPDU(VPI,VCI,PDUid) entonces Comenzar /* Notificación desde CCA de fin de PDU en buffer */
    h:=hash(VPI,VCI,PDUid);                 /* Función de hash de acceso a la DMTE */
    g:= Inic_DMTE(VPI,VCI,gradoGoS);        /* Obtenemos el grado de GoS de la conexión */
    Si get(buffer,PDU) entonces Comenzar /* Si solicitud de envío desde el agente DPA */
      Si NumPDUVCI(DMTE,h) >= g entonces
        borrar(DMTE,h); /* Elimina de la DMTE la PDU de id más pequeño */
        copiar(buffer,VPI,VCI,PDUid,DMTE,h); /* Copia la PDU del buffer a la DMTE */
        put(buffer,PDU); /* Envía la PDU al DPA */
        borrar(buffer,PDU); /* Se elimina del buffer la PDU que se acaba de transferir */
      FSi;
      Si get(DMTE,PDU) entonces put(buffer,PDU); /* Solicitud retransmisión de PDU desde la DMTE al DPA */
    FSi;
FMientras;

```

Figura 11.5. Algoritmo TAP en los nodos AcTMs

El bucle del algoritmo recibe desde CCA la notificación de fin de PDU cuando acaba de procesar una célula EOM en el buffer. Cuando esto ocurre, se calcula la función de *hash* para los accesos a la DMTE para cada PDU. Antes de enviar una PDU a la DMTE es necesario conocer el grado de GoS que se especificó para la conexión, ya que hay que controlar cuántas PDU de cada conexión se tienen ya almacenadas en la memoria. Para entrar una nueva PDU de una conexión que ya tiene el máximo número de PDU permitido (expresado por GoS) es necesario borrar previamente la PDU más antigua (*aging* sobre el PDUid) para liberar espacio de memoria. Una vez que se dispone de espacio, el algoritmo se encarga de realizar la copia de la PDU desde el buffer a la DMTE, para después enviar esa misma PDU hasta el agente DPA (que la habrá solicitado previamente con la función *get*). Después de enviar la PDU ésta es borrada para liberar el espacio ocupado en el buffer. Destacamos los accesos constantes $O(1)$ a la tabla DMTE, gracias a la función de *hash*, que garantiza este coste mientras no aparezcan colisiones en los accesos a la tabla.

En el caso que el algoritmo reciba una petición de retransmisión desde DPA (mediante la operación *get*), entonces la PDU solicitada será enviada a este agente, tal como muestra la *Figura 11.5*. Este es el escenario en que una solicitud de retransmisión desde el conmutador siguiente al que está ejecutando el algoritmo, ha tenido éxito en la DMTE local, y donde ha sido localizada la PDU solicitada para su retransmisión.

11.3. AGENTE CoSA (CLASS OF SERVICE AGENT)

Como ya se ha explicado, este agente se encarga de atender diversos aspectos relativos a la clase de servicio que requieren de la red las fuentes de tráfico. La *Figura 11.6* formaliza las labores más destacables realizadas por el agente, así como las relaciones con el resto de agentes del SMA. El agente es inicializado por el terminal emisor de la comunicación desde donde se definen los parámetros de tráfico de cada una de las fuentes. Estos parámetros van a determinar, en cierto modo, la clase de servicio que desea cada fuente, entre los que se encuentra, por ejemplo, el grado de garantía de servicio que se requiere. De este modo, podemos decir que CoSA actúa como agente programable ya que, en función de sus características y acciones, se podrá lograr un mayor o menor grado de GoS, que es uno de los objetivos primordiales de TAP.

Además, tal como podemos comprobar en el algoritmo, este agente desempeña importantes funciones de comunicación y sincronización con el resto de agentes CoSA de la VPN en las labores de señalización y para el establecimiento de la conexión. Como ya hemos descrito en el *Capítulo 10*, el trabajo cooperativo de este agente con el agente WFQA es también imprescindible para lograr nuestros objetivos, por lo que en ambos se establecen las operaciones de sincronización, tanto para conseguir la GoS de las fuentes privilegiadas, como la justicia de todas las fuentes de tráfico en general, sean o no privilegiadas.

Otro de los aspectos destacables del algoritmo es el que este agente se encarga también de inicializar dos de las estructuras de datos más importantes de la arquitectura TAP, como son la memoria DMTE y las Tablas de E/S. Así, mediante la función *Inic_Grado_GoS_DMTE*, se realiza la reserva de memoria oportuna para soportar el número de PDU que se indica en el parámetro GoS para la conexión VPI/VCI. Por otro lado, las labores de establecimiento de la conexión, y sus implicaciones con la señalización, nos permiten determinar las relaciones de las conexiones con sus respectivos puertos de E/S en los AcTMs. Relaciones que deben ser registradas en la Tablas de E/S de cada puerto para poder obtener los índices de acceso a las PDU de la DMTE en el caso de retransmisiones. Todos estos aspectos han sido también aclarados en el *Capítulo 10*, y la función *Inic_TablaE/S* es usada en este agente para inicializar esta estructura al inicio de cada conexión.

El cuerpo del bucle del algoritmo de la *Figura 11.6* se encarga de procesar todas las células que le llegan desde la capa ATM de los AcTMs. En el caso que las células que llegan pertenezcan a conexiones privilegiadas éstas son ensambladas en PDU y transferidas al agente WFQA para llegar a sus correspondientes colas de entrada. En el caso que la conexión no sea privilegiada se transfieren las células independientes, también a su correspondiente cola de entrada al conmutador.

```

Inic_CoSA(VPI,VCI);                /* Sincronización para el establecimiento de la conexión */
Sincronizacion_Agentes_CoSA(VPI,VCI); /* Sincronización entre todos los agentes CoSA de la VPN*/
Establecimiento_Conexion (VPI,VCI); /* Asignación de los VPI/VCI extremo-extremo */
Sincronizacion_WFQA(VPI,VCI);      /* Sincronización con el agente WFQA local */
Inic_Grado_GoS_DMTE(VPI,VCI,GoS); /* Determinación de la GoS en la DMTE local */
Inic_TablaE/S(VPI,VCI,InPort,OutPortPrev); /* Iniciación de la TablaE/S local */
Mientras Desde_ATM(celula)        /* Se procesan todas las células que lleguen desde la capa ATM */
    Si Privilegiado(VPI,VCI) = "N" entonces Hasta_WFQA(VPI,VCI,celula,p);
    En otro caso Comenzar          /* Cuando se trata de conexión privilegiada se ensambla la PDU */
        PDU:=Ensamblar_PDU(VPI,VCI,celula);
        Si EOM(VPI,VCI,celula) entonces Hasta_WFQA(VPI,VCI,c,PDU); /* Pasar la PDU al agente WFQA */
    FSi;
FMientras;
    
```

Figura 11.6. Algoritmo del Agente de CoS

11.4. AGENTE WFQA (WEIGHTED FAIR QUEUEING AGENT)

Aunque el objetivo principal de TAP es el de ofrecer GoS a las fuentes privilegiadas, también ofrece la ventaja añadida de la atención justa del tráfico generado por todas las fuentes, sean privilegiadas o no. Nos encontramos con la posibilidad que al intentar garantizar las transferencias con GoS, podría darse la posibilidad que las fuentes no privilegiadas acabasen siendo relegadas. Para controlar esta situación

proponemos el algoritmo QPWFQ que permite asignar pesos a las colas de entrada de los ActMs en función de los valores de PCR de las fuentes. Para evitar la inanición de fuentes con PCR reducido, con respecto a las que lo tengan más elevado, incluimos el control sobre la longitud de las colas de entrada antes de procesar el tráfico hasta el buffer del conmutador. El funcionamiento del algoritmo QPWFQ ha sido ya explicado en el apartado 4.4 del *Capítulo 4*, y los flujos en las colas de entrada que son gestionadas por QPWFQ pueden ser consultados en el punto 10.2.1 del *Capítulo 10*. Ahora vamos a complementar todos estos aspectos explicando el funcionamiento del agente WFQA (mostrado en la *Figura 11.7*) que, entre otras funciones importantes, implementa el algoritmo QPWFQ.

Como podemos comprobar en la *Figura 11.7*, el agente WFQA es también inicializado desde el terminal emisor de la comunicación, donde ya se indica el PCR de la fuente de tráfico, lo que acabará reflejándose como el peso que cada fuente tendrá asignado por el agente en su correspondiente *per-VC* cola de entrada.

```

Inic_WFQA(VPI,VCI,PCR); /* Sincronización para iniciar el agente de gestión justa de colas de entrada*/
Sincronizacion_CoSA(VPI,VCI); /* Sincronización con el agente CoSA local */
Sincronizacion_RCA(VPI,VCI,PDUid); /* Sincronización con el agente RCA local para atender retransmisiones*/
pi:=Inic_WFQA(VPI,VCI,PCR); /* Se asigna a la cola PerVC como peso el valor del PCR de la fuente i */
li:=0; /* Longitudes de las colas de espera en cada momento */
Retransmission_RCA(VPI,VCI,PDUid):=falso; /* No hay retransmisiones al inicio */

Mientras Desde_CoSA(VPI,VCI,celula,PDU) /* Se procesan las PDU o células llegadas desde CoSA */
i:=VCI; /* Asignación de la PDU o de la célula del VCI i a la cola i */
Si Retransmission_RCA(VPI,VCI,PDUid) entonces comenzar
    encolar(ColaVPI/VCIi,c,PDUid); /* Encolar PDU en cola VPI/VCIi */
    apuntar(ColaVPI/VCIi,PDUid); /* Apunta PDU retransmitida como más prioritaria a la cabecera de la cola i */
Sino encolar(ColaVPI/VCIi,celula,PDU); /* Introduce en cola de datos i la PDU o célula del VCI i */
FSi;
li:=li+1; /* Cuando llega una PDU o célula incrementa en 1 la longitud de su cola */
Si (li<=pi o Retransmission_RCA(VPI,VCI,PDUid)) entonces comenzar
    Caso
        Retransmission_RCA(VPI,VCI,PDUid): encolar(turnos,VCIi); /* Encola en turnos VCI=i como más prioritario que cabecera */
        li<=pi : encolar(turnos,VCIi); encolar(turnos, VCIi); /* No se trata de una retransmisión */
    FCaso;
FSi;
FMientras;

Mientras Novacias(colas_datos) /* Si hay PDUs o células pendientes de transmitir o retransmitir en las colas de entrada */
desencolar(turnos,i); /* Obtener el turno i de la cabecera de cola de turnos o del puntero a PDU retransmitida */
Hasta_Buffer(ColaVPI/VCIi,PDU,celula); /* Envío de la PDU o célula de la cola i hasta el buffer */
desencolar(ColaVPI/VCIi,PDU,celula); /* Eliminar PDU o célula de la cola i enviada al buffer */
li:=li-1; /* Decrementa en uno la longitud de la cola de espera procesada */
Si (li>=pi) y (No_Retrasmission_RCA(VPI,VCI,PDUid) y resto_de_colas_idle) entonces
    encolar(turnos,VCIi); /* Reencola el turno VCI=i en la cola de turnos garantizando política work-conserving */
FSi;
FMientras;

```

Figura 11.7. Algoritmo del agente WFQA

El algoritmo muestra también las funciones de sincronización del agente WFQA con el agente CoSA con los objetivos que ya hemos comentado en el apartado anterior. Además se establece también la necesaria sincronización con el agente RCA que es el encargado, como hemos explicado en el *Capítulo 10*, de notificar a WFQA su solicitud de retransmisión de una PDU al conmutador anterior.

Las líneas 4 y 5 del algoritmo tienen como misión la de inicializar los pesos y las longitudes de las colas del conmutador, ya que este es el mecanismo ideado para aportar a la vez la justicia y el tratamiento privilegiado de las células de las fuentes de tráfico.

En el primer bucle de la *Figura 11.7* se reciben las células o PDU procesadas previamente por el agente CoSA. En primer lugar se realiza la labor de asignación *per-VC* para que cada conexión VPI/VCI tenga su propia cola, para pasar después a analizar la posibilidad de que haya llegado una solicitud de retransmisión (que deberá haber sido antes notificada por el agente RCA). Si es así, se encola la PDU en su correspondiente cola de entrada y se actualiza el puntero de prioridades para darle a esta PDU un tratamiento especial con respecto al resto que puedan existir ya en esa cola. En el caso de no ser una retransmisión, se encola la célula o PDU en su correspondiente *per-VC* para, posteriormente, incrementar en una unidad la longitud de ésta. Seguidamente se realiza la inclusión del número de la cola de datos en la cola de turnos que es la que lleva el orden de atención de las cabeceras de las colas de datos. Podemos observar cómo la condición de encolado en la cola de turnos es que la longitud de cada cola sea menor que el peso (PCR) asignado a la fuente (lo que evita los comportamientos injustos), o bien que se trate de una retransmisión.

El segundo bucle del algoritmo se encarga de servir los datos al buffer del conmutador en función del orden establecido por la cola de turnos, en cuya cabecera está el número de cola cuya célula o PDU será la siguiente en ser enviada *Hasta_Buffer*. Una vez procesadas las cabeceras de las colas, éstas son descoladas y se decrementa en una unidad la longitud de la cola de datos procesada. Llamamos la atención sobre el hecho que las dos últimas líneas de este segundo bucle aportan la característica *work-conserving* al tratamiento de las colas que, como puede comprobarse, permite atender un VPI/VCI concreto sin atender al resto de colas, y siempre que éstas estén en estado de inactividad y/o no llegue una retransmisión.

La característica *work-conserving* permite optimizar el rendimiento del sistema por no requerir de tiempos de espera fijos para atender secuencialmente las colas. Además, es importante recordar, como explicamos en el *Capítulo 4*, el coste constante $O(1)$ de mantenimiento y de acceso del sistema de colas de entrada y también el mínimo nivel de complejidad (juego reducido de variables y simplicidad en las sentencias de flujo del algoritmo) que acaba redundando en la optimización del rendimiento de la red.

11.5. AGENTE CCA (CONTROL CONGESTION AGENT)

Hemos identificado ya (*Capítulos 5,7,9 y 10*) el buffer como uno de los puntos clave de los AcTMs donde localizar los problemas relativos a las gestiones que deseamos evitar. El buffer es, por tanto, la principal fuente de problemas en cuanto a la aparición de congestiones por ser el punto de competencia entre las conexiones entrantes a los conmutadores. En nuestro caso disponemos de un agente que ofrece la posibilidad de elegir la política de evitación de pérdidas y respuesta ante las congestiones.

Este agente tiene también múltiples funciones encomendadas, aunque la más importante de ellas es la implementación del algoritmo EPDR ya descrito en el apartado 5.7 del *Capítulo 5*, donde la *Figura 5.8* presenta nuestro algoritmo EPDR, propuesto como variante del conocido EPD. En este caso, la *Figura 11.8* muestra el algoritmo del agente CCA que, como puede observarse, incluye EPDR. Seguidamente comentamos los aspectos más destacables de este algoritmo.

```

Inic_CCA(algoritmo,umbral); /* Inicializa Agente CCA con Algoritmo y umbral del buffer */
Sincronización_Agentes_CCA(algoritmo,umbral); /* Sincronización con resto agentes CCA de la VPN */

Si Algoritmo="EPDR"
    Mientras Hasta_Buffer(VPI,VCI,celula) /* Mientras llegan células al buffer */
        Si ConexionNo_Privilegiada(VPI,VCI) y buffer(lleno)="N" entonces insertar(buffer,celula);
        Si no Comenzar
            Si celula(VPI,VCI) en Lista-descartes entonces Comenzar /* Una célula con este VPI/VCI ya ha sido descartada */
                Si EOM(celula) entonces Comenzar /* Es la última célula de una PDU */
                    Si Longitud_Actual_Cola < Tamaño_Maximo_Buffer entonces
                        insertar(buffer,celula); /* La célula EOM cabe en el buffer y se inserta */
                    En otro caso
                        tirar(celula); /* Célula EOM no cabe en el buffer y es tirada */
                    FSi;
                    RCA_Retransmitir(VPI,VCI,PDUid); /* Solicitud de retransmisión al agente RCA */
                    eliminar (VPI,VCI,Lista-descartes); /* Se borra el VPI/VCI de la lista-descartes */
                En otro caso
                    tirar(celula); /* Célula no EOM que no cabe en el buffer y es tirada */
                FSi;
            En otro caso
                Si Longitud_Actual_Cola < umbral entonces Comenzar
                    insertar(buffer,celula); /* La célula cabe en el buffer y se inserta */
                    Si EOM(celula) entonces FinPDU(VPI,VCI,PDUid); /* Notifica a TAP del AcTMs el fin de una PDU en buffer */
                En otro caso Si (primera_célula(PDU) o (buffer(lleno)="S"))
                    tirar(celula); /* Evitar fragmentación de la PDU */
                    insertar (VPI,VCI,Lista-descarte); /* Marcar VPI/VCI con una célula ya descartada */
                FSi;
            En otro caso comenzar
                insertar(buffer,celula); /* Tamaño de buffer sobrepasa el umbral pero se acepta */
                Si EOM(celula) entonces FinPDU(VPI,VCI,PDUid); /* Notifica a TAP del AcTMs el fin de una PDU en buffer */
            FSi;
        FSi;
    FSi;
    FMientras;
FSi;

```

Figura 11.8. Algoritmo del agente CCA y EPDR

CCA es también inicializado por el emisor de las conexiones, donde el usuario o administrador de la red, en el proceso de establecimiento de la conexión, especificará el algoritmo que desea emplear para controlar las congestiones del buffer. Este aspecto es el que aporta la característica programable a CCA, ya que está diseñado para que se pueda optar por diversos mecanismos de control de congestión (*Capítulo 5*) y por algunos de los parámetros de funcionamiento de estos mecanismos. En nuestro caso empleamos EPDR, por lo que permitimos elegir el umbral, pero podría optarse por EPD, PPD, o cualquier otro que desee implementarse como soporte de TAP que, al tener un diseño modular, permitirá optar por uno u otro en función de las necesidades concretas.

Puede observarse en la *Figura 11.8* también, como CCA establece mecanismos de comunicación con otros agentes de la VPN que soporten TAP, de forma que se mantenga una misma política de gestión del buffer en todos los conmutadores.

En el caso que al buffer lleguen células provenientes de conexiones no privilegiadas éstas no reciben ningún tratamiento especial, siendo insertadas en el buffer si éste no está lleno. En cambio, si se reciben desde las colas de entrada varias PDU, comienza a aplicarse la política de evitación de pérdidas y/o fragmentación de PDU en el caso de congestión que, como sabemos, depende del tamaño del umbral que se haya definido en el algoritmo EPDR. Podemos comprobar en la *Figura 11.8* cómo, en el caso de aparecer la congestión, se invoca al agente RCA que se encargará de iniciar el mecanismo de recuperación, actualizándose después la lista de VPI/VCI que ya ha experimentado descartes de algunas de sus células.

11.6. AGENTE DPA (DISPACHER PDU AGENT)

El algoritmo presentado en la *Figura 11.9* aclara las explicaciones dadas en el *Capítulo 10* sobre el funcionamiento del agente encargado de despachar las PDU desde el buffer hasta las correspondientes salidas de los conmutadores. El agente despacha, tanto PDU con GoS, como células pertenecientes a conexiones no fiables. Para ello es iniciado mediante un evento en el proceso de establecimiento de comunicación. Podemos ver también la relación de DPA, tanto con las Tablas de E/S, como con el buffer y con la DMTE.

```

Inic_DPA(evento);                               /* Inicializa Agente DPA para recibir células y PDU */
Buffer, DMTE, TablasE/S                         /* Acceso a estructuras de datos */

Mientras cierto
  Si retransmision(DMTE,PDU) entonces Comenzar /* Sincronización con TAP para retransmisión desde DMTE */
    PDUR:=get(DMTE,PDU);                        /* Solicita a la DMTE la PDU a retransmitir */
    retransmitir(PDUR);                         /* Retransmisión de PDU desde la DMTE local */
  Si no Comenzar
    paq:=get(buffer,siguiente)                  /* Solicita al buffer la siguiente unidad de transferencia */
    Si celula(paq)="S" entonces Comenzar        /* Cuando se trata de una conexión no privilegiada */
      paq.VPI:=VPIOut;
      paq.VCI:=VCIOut;
      Hasta_ATM(paq);
    Si no Comenzar
      celulaEOM:=EOM(VPI,VCI,paq);             /* Conmutación al nuevo VPI/VCI y envío directo a capa ATM */
      Generar(IndexT,celulaEOM);               /* Estamos ante una PDU transferida completa desde el buffer */
      inserta(TablaE/S,IndexT);                /* Obtenemos primero la célula EOM de la PDU */
      celulaEOM.VPI:=VPIOut;                   /* Se genera el índice de acceso a la TablaE/S */
      celulaEOM.VCI:=VCIOut;                   /* Inserción del nuevo índice en la TablaE/S */
      Hasta_ATM(celulaEOM);                    /* Transferencia de la célula EOM hasta siguiente conmutador */
    Mientras No_Fin(paq)
      celula:=cell(VPI,VCI,paq);                /* Obtención del resto de las células de la PDU */
      celula.VPI:=VPIOut;
      celula.VCI:=VCIOut;
      Hasta_ATM(celula);                        /* Se envía cada célula a la capa ATM del ActMs */
    FMientras;
  FSi;
FMientras;

```

Figura 11.9. Algoritmo del agente DPA

Dentro del bucle principal del algoritmo se atienden en primer lugar las situaciones de retransmisión que pueden provenir desde la DMTE y notificadas desde el algoritmo TAP soportado en los AcTMs. Cuando DPA recibe esta notificación solicita con *get* el envío de la PDU desde la DMTE y, una vez que la tiene, se encarga de transmitirla al correspondiente puerto de salida del conmutador. La función *get* (junto con *put* desde el buffer o la DMTE) son las que aportan la atomicidad en el tratamiento de las PDU fiables que son tratadas con el mecanismo *VC-Merge*.

Cuando no existen retransmisiones se atiende el flujo normal de datos, siempre previa solicitud al buffer de la siguiente unidad disponible que puede ser una PDU o una célula, y que el buffer se encarga de servir atómicamente con su operación *put*. Si se procesa una célula independiente se le asigna su correspondiente valor de VPI/VCI de salida y se envía hasta la capa ATM del conmutador que las procesa a la salida. Si la siguiente unidad a transferir desde el buffer es una PDU con GoS, se comienza procesando el final de ésta para disponer del VPI/VCI/PDUid y se aplica la posible asignación de nuevos VPI/VCI de salida para poder enviar los datos en forma de células hasta la capa ATM. Antes de pasar las células a la capa ATM se genera el índice y se inserta en la correspondiente Tabla de E/S según lo explicado en el *Capítulo 10*.

11.7. AGENTE RCA (RETRANSMISSION CONTROL AGENT)

Este agente juega un importante papel en el mecanismo de recuperación de PDU perdidas por congestión, ya que es el responsable de recibir los eventos cuando aparecen las congestiones. Estos eventos pueden provenir, o bien desde el buffer local del conmutador en que está ejecutándose RCA, o bien a través de las células BRM que provienen del conmutador siguiente a donde se encuentra el agente que recibe el evento.

Podemos observar en la *Figura 11.10* el algoritmo de RCA, que es iniciado desde el emisor. Este agente colabora, como hemos visto ya, con WFQA y con el resto de agentes RCA de otros conmutadores desde donde, o hasta donde envía las células BRM.

```

Inic_RCA(evento); /* Inicializa Agente RCA para recibir células y PDU */
Sincronizacion_WFQA(VPI,VCI,PDUid); /* Sincronización con el agente WFQA local */
Sincronizacion_RCA(VPI,VCI,PDUid); /* Sincronización con otros agentes RCA de la VPN */

Mientras cierto
  Si RCA_Retransmitir(VPI,VCI,PDUid) entonces comenzar /* Cuando llega una solicitud de retransmisión desde EPDR */
    Si Toff_agregado(enlace) entonces Comenzar /* Si suficiente Toff en el enlace */
      Retransmision_RCA(VPI,VCI,PDUid); /* Notificación de la solicitud de retransmisión al WFQA */
      GenerarBRM(celdaRM,VPI,VCI,PDUid); /* Se genera la celda BRM para la solicitud de retransmisión */
      TransferirBRM(celdaRM); /* Transferencia de la BRM al conmutador previo */
    FSi;
  FSi;
  En otro caso
    Si LlegaBRM(celdaBRM) entonces comenzar /* RCA recibe una BRM del conmutador siguiente a él */
      Index:=Obtener_Index(celdaBRM); /* Se obtiene el índice de la BRM */
      IndexDMTE:=Acceso(TablaE/S,Index); /* Acceso a la TablaE/S para obtener el índice de acceso a DMTE */
      Si Esta_PDU(DMTE,Index)="S" entonces retransmision(DMTE,PDU); /*Notifica retransmisión exitosa al DPA */
      Si no Si Toff_agregado(enlace) entonces Comenzar /* Fallo de DMTE pues ya no contiene la PDU solicitada */
        Retransmision_RCA(VPI,VCI,PDUid); /* Solicitud de retransmisión al conmutador previo */
        GenerarBRM(celdaRM,VPI,VCI,PDUid); /* Se genera la celda BRM para solicitud retransmisión */
        TransferirBRM(celdaRM); /* Transferencia de la BRM al conmutador previo */
      FSi;
    FSi;
  FSi;
  FMientras;

```

Figura 11.10. Algoritmo del agente RCA

Cuando RCA recibe una solicitud de retransmisión desde el buffer local al conmutador, la primera labor que realiza es evaluar el tiempo de inactividad de que se dispone en el enlace y, si es suficiente, comienza el proceso de recuperación, notificando en primer lugar la situación a WFQA, para pasar después a generar la BRM con los datos de la PDU a retransmitir. Una vez generada la célula RM, ésta es enviada al conmutador previo a través del VPI/VCI estándar reservado para las células RM.

En cambio, como puede observarse, cuando la solicitud de retransmisión no proviene del buffer local, sino que llega en una RM desde el conmutador siguiente que puede haber sufrido una congestión, lo primero que se hace es obtener el índice de acceso a la DMTE local para localizar la PDU cuyo PDUid viene indicado en la RM. Obtenido el índice de la célula RM, se accede a la correspondiente Tabla de E/S en la que se

encuentra el índice real que se emplea en la función de *hash* para acceder a la DMTE desde donde podrá hacerse la retransmisión en el caso que la PDU esté allí localizada. En el caso que la PDU no esté ya en la DMTE, se enviará la petición de retransmisión al conmutador previo, siguiendo un procedimiento idéntico al explicado en el párrafo anterior en que la pérdida era local.

11.8. RETRANSMISIONES DURANTE LOS TIEMPOS DE INACTIVIDAD

Uno de los aspectos básicos del rendimiento del protocolo TAP está en la idea intuitiva de aprovechar los estados de inactividad de los enlaces y de las fuentes con GoS para recuperar las pérdidas debidas a las congestiones. Ya adelantamos en el *Capítulo 8* los inconvenientes de las retransmisiones extremo-extremo en las que se basan protocolos tan exitosos como TCP que acaban pagando un precio demasiado elevado en cuanto a su rendimiento se refiere. Vimos entonces que TAP puede aportar importantes mejoras en goodput, porque las recuperaciones se realizan de forma local a los conmutadores que experimentan las pérdidas.

Además, otra ventaja del protocolo está en su posibilidad de no afectar a las transmisiones para atender las retransmisiones que, en muchas aplicaciones como sabemos, pueden ser perniciosas. Por tanto, en esta sección queremos demostrar que se satisface nuestra idea de partida, según la cual las retransmisiones pueden ser atendidas en los estadios en que las fuentes no generan tráfico o que haciéndolo, el enlace que emplean tiene disponibilidad de ancho de banda para soportarlas junto al tráfico normal. Vamos a comprobar, mediante fuentes ON/OFF, que las conexiones ATM pueden aprovechar muchos de esos estados de inactividad. Para ello debemos analizar y evaluar los tiempos de OFF agregado que nos quedan en la red después de multiplexar en un solo enlace N fuentes de tráfico. Realizaremos esa evaluación en primer lugar de forma conceptual para pasar luego a su estudio formulado, donde podremos comprobar la viabilidad para aplicar esta idea en TAP.

Por tanto, el protocolo analiza previamente la disponibilidad del tiempo de OFF agregado que queda en un enlace antes de atender las retransmisiones, ya que no tiene ningún sentido realizar retransmisiones cuando se sabe de antemano que éstas no tendrán éxito por el propio estado de sobrecarga de los enlaces. Cuando se detecta esta posibilidad TAP no pone en marcha el mecanismo de recuperación para no desaprovechar innecesariamente el rendimiento de la red.

11.8.1. MODELO DE UNA FUENTE ON/OFF Y POSIBILIDADES DE RECUPERACIÓN

Según la teoría de colas, el análisis del proceso de encolado es una parte fundamental para la evaluación del comportamiento de las redes. En el caso de ATM puede considerarse que una cola es una expresión matemática de la idea de contención de recursos. A las colas ATM llegan células, ráfagas o conexiones que necesitan un cierto servicio y, mientras son atendidos, esperan una determinada unidad de tiempo en una zona de almacenamiento (buffer, cola o línea de espera). El sistema de colas puede ser descrito según los siguientes aspectos: el patrón de llegadas de las conexiones, el patrón de servicio de las conexiones, el número de canales de servicio y la capacidad del sistema. Todos estos patrones han sido estudiados amplia y metódicamente en la literatura [1,2,3], aunque en el caso de ATM, suele descuidarse el parámetro de la sincronización en el análisis matemático de los buffers. Así, se asume que una célula es servida inmediatamente después de entrar en un buffer vacío, en lugar de esperar hasta el comienzo del siguiente slot libre. No obstante, nosotros tampoco deseamos entrar en estos niveles de detalle, ya que nuestro objetivo es mucho más generalista, en un intento por demostrar la viabilidad de TAP. No hemos de perder de vista que la mayor parte de propuestas de la arquitectura TAP están pensadas para optimizar el throughput de la red en general y, además, de forma particular, el mecanismo de retransmisiones mejora también el goodput, por lo que nos vamos a centrar en esta visión más amplia y para ello vamos a basarnos, como hemos dicho, en el modelo de fuentes ON/OFF.

Las fuentes ON/OFF pueden considerarse como un modelo basado en dos estados en los cuales la velocidad de llegada de células en cada estado es fija y el periodo de permanencia en cada estado puede ser exponencialmente, geoméricamente o arbitrariamente distribuido. El modelo ON/OFF [1,2] lo usamos para caracterizar el tráfico ATM por conexiones unidireccionales y las fuentes ON/OFF a ráfagas nos van a permitir analizar el comportamiento de la pérdida de células. La *Figura 11.11* muestra este modelo como una fuente que: o bien envía datos (estado ON) durante un tiempo T_{on} a una velocidad CAR (*Cell Arrival Rate*); o bien permanece inactiva (estado OFF) sin producir células durante un tiempo T_{off} .

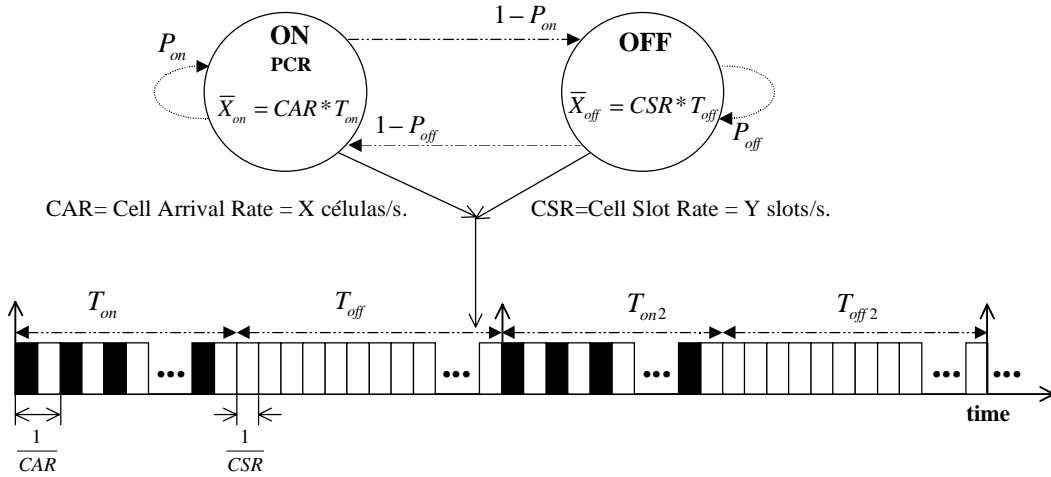


Figura 11.11. Patrón de generación de células para una fuente ON/OFF

Por tanto, en la fuente ON/OFF los procesos conmutan entre un estado de silencio (en que no se producen células) y un estado que produce una velocidad fija y determinada de células. Las fuentes ON/OFF con duraciones distribuidas como exponenciales negativas han sido frecuentemente estudiadas y aplicadas a tráfico de datos y como modelo general para tráfico a ráfagas en multiplexores ATM. Después de la llegada de cada célula se genera otra célula con probabilidad P_{on} , o bien la fuente cambia al estado de silencio OFF (*idle*) con una probabilidad $1-P_{on}$. De la misma forma, en el estado de silencio OFF la fuente genera otro slot de tiempo vacío con probabilidad P_{off} , o bien cambia al estado activo ON con una probabilidad $1-P_{off}$.

En lugar de ver el proceso de generación de células y el de slots de tiempo como procesos Bernoulli, simplemente los consideramos como procesos distribuidos geoméricamente. Al final de un periodo de estancia en un estado concreto, los procesos conmutan al otro estado con probabilidad 1. Destacamos que las distribuciones geométricas de los estados de silencio y activos tienen diferentes bases de tiempos. Además, podemos comprobar en la Figura 11.11 que en los estados de actividad pueden existir también slots vacíos, esto ocurre cuando el CAR es menor que el CSR (*Cell Slot Rate*) que es lo habitual.

Podemos calcular el valor de la unidad de tiempo de cada estado activo, o *CIT* (*Cell Inter-arrival Time*), como el inverso del CAR ,

$$CIT = \frac{1}{CAR} \tag{1}$$

Así, la duración media de cada estado activo es,

$$T_{on} = \frac{1}{CAR} \bar{X}_{on} \tag{2}$$

donde \bar{X}_{on} es el número medio de células que se produce en cada estado activo.

Por analogía con la expresión (2), en el estado de silencio, la unidad de tiempo es el inverso de CSR , por lo que la duración media del tiempo que la fuente permanece en el estado de silencio es,

$$T_{off} = \frac{1}{CSR} \bar{X}_{off} \tag{3}$$

donde en este caso \bar{X}_{off} es el número medio de slots de células vacías que se producen en cada estado de silencio.

La expresión (2) nos permite calcular la media de células en un estado ON como el inverso de la probabilidad de salir del estado activo, aunque también lo podemos expresar como la velocidad de llegada de células CAR , multiplicada por el periodo de tiempo en que la fuente está activa. Así, podemos representar

este valor medio con la siguiente expresión, que calcula el número de células que se producen en cada estado activo de una fuente:

$$\bar{X}_{on} = \frac{1}{(1 - P_{on})} = CAR * T_{on} \quad (4)$$

A través de la fórmula (3) podemos calcular también el número medio de slots de células que se producen en cada estado de silencio OFF como el inverso de la probabilidad de salir del estado *idle*. Como antes, podemos calcular este valor medio multiplicando el valor de *CSR* por el tiempo en que la fuente permanece en inactividad. De este modo, podemos expresar este valor medio con la siguiente expresión que indica el número medio de slots que se producen en cada estado de silencio de una fuente:

$$\bar{X}_{off} = \frac{1}{(1 - P_{off})} = CSR * T_{off} \quad (5)$$

De esta forma, podemos generalizar el modelo de las fuentes ON/OFF para distribuciones arbitrarias, tanto del número de células generadas en el estado activo, como del número de slots vacíos generados en los estados de silencio.

Podemos aplicar todo esto a un ejemplo concreto como es la supresión de los silencios telefónicos (en las cuales no se transmitirían células durante los periodos en los que los interlocutores están en silencio). Estudios empíricos [1] demuestran que se pueden considerar los siguientes tiempos medios en los estados de ON y OFF

$$\begin{aligned} T_{on} &= 0,96s. \\ T_{off} &= 1,69s. \end{aligned} \quad (6)$$

Seguidamente vamos a aplicar toda la formulación anterior a diversos casos que nos permitirán argumentar nuestros objetivos. Supongamos el caso de una fuente con un flujo de llegadas periódicas a una velocidad de 64 Kbps sobre un enlace de 155,52 Mbps. Si expresamos este flujo en forma de células por segundo tendremos el valor de *CAR* según la siguiente expresión,

$$CAR = 64 Kbps = \frac{\left(\frac{64.000}{8}\right)}{48} = 167 \quad (7)$$

Tenemos, por tanto, que la velocidad de llegada de células para esta fuente es de 167 células por segundo. De forma parecida podemos calcular el valor del *CSR* si suponemos usar un enlace de 155,52 Mbps, con la siguiente fórmula que expresa el número de slots por segundo que se producen en cada estado de OFF,

$$CSR = \frac{\left(\frac{155.520.000}{8}\right)}{53} = 366.792 \quad (8)$$

Partiendo del valor obtenido en (8) podemos calcular el *Tiempo de Servicio por Célula (TSC)* lo que arroja un resultado de 2,726 μ seg. como tiempo de proceso de cada célula.

Podemos afinar el resultado de (8) si tenemos en cuenta que en ATM se genera una células *OAM* (*Operation And Management*) cada 27 células, y obtendremos el número total de slots de células válidos que se generan en cada estado OFF,

$$\frac{26}{27} 366.792 = 353.208 \quad (9)$$

En el caso de (9) podemos calcular el nuevo TSC que es de 2,831 μ seg para cada uno de los slots de células generadas.

Con los valores de los tiempos de ON y OFF fijados en (6) podemos calcular el número medio de células producidas en un estado activo:

$$\bar{X}_{on} = CAR * T_{on} = 167 * 0,96 = 160 \quad (10)$$

De forma similar, el número medio de slots vacíos generados en un estado de silencio es,

$$\bar{X}_{off} = CSR * T_{off} = 353.208 * 1,69 = 596.921 \quad (11)$$

Podemos calcular también las probabilidades de paso P_{on} y P_{off} entre estados ON y OFF respectivamente. Sabemos que,

$$\bar{X}_{on} = \frac{1}{(1 - P_{on})} = 160 \quad (12)$$

por tanto,

$$P_{on} = 1 - \left(\frac{1}{160} \right) = 0,99375 \quad (13)$$

Del mismo modo,

$$\bar{X}_{off} = \frac{1}{(1 - P_{off})} = 596.921 \quad (14)$$

por tanto,

$$P_{off} = 1 - \left(\frac{1}{596.921} \right) = 0,99999832 \quad (15)$$

Según todo esto, llegamos a las siguientes conclusiones:

- En el estado de silencio se generan 353.208 slots por segundo, aunque la media de slots generados es de 596.921. Además, se permanece en el estado de silencio durante 1,69 seg., y la probabilidad de continuar en él es de 0,99999832, por lo que se cambia del estado de silencio al de actividad con una probabilidad de 0,00000168.
- En el estado de actividad se generan células a una velocidad de 167 por segundo. Como se permanece en el estado de actividad durante 0,96 seg., el número medio de células en cada estado activo es de 160 células y la probabilidad de permanecer en este estado es de 0,99375, mientras la probabilidad de cambiar al estado de silencio es de 0,00625.
- Sin considerar que en los estados activos puede haber slots vacíos, las 160 células de cada estado ON representan el 0,000268 % de los slots que se producen en el T_{off} . Para realizar los cálculos se han usado valores fijos y concretos de T_{on} y T_{off} que pueden ser diferentes en otras clases de servicio, pero se puede comprobar que, con otros valores de T_{on} y T_{off} más próximos, los resultados finales son muy parecidos. En realidad, lo que nos sirve para apoyar nuestra idea es la gran diferencia entre el valor de CSR del enlace con respecto al CAR de la fuente ON. Cada vez que se entra en un estado OFF se generan muchos más slots de células por segundo que las células realmente generadas en cada estado ON. Esto es lo habitual en las redes ATM donde los conmutadores se caracterizan por su gran potencia de conmutación para poder soportar las necesidades de cada enlace que, generalmente, no son completamente aprovechadas por las fuentes.

Podemos realizar planteamientos similares para otro ejemplo de fuente cuatro veces más rápida que la anterior (512 Kbps) sobre un enlace también de 155,52 Mbps. En este escenario los resultados obtenidos son los siguientes:

$$CAR = ((512.000 / 8) / 48) = 1.334cps$$

$$CSR = 353.208sps$$

$$\bar{X}_{on} = CAR * T_{on} = 1.334 * 0,96 = 1.280c.$$

$$\bar{X}_{off} = CSR * T_{off} = 353.208 * 1,69 = 596.921slots$$

Supongamos ahora un tercer escenario, donde la fuente es de 1,5 Mbps (que podría ser adecuado para tráfico multimedia) sobre el mismo enlace de 155,52 Mbps. Este escenario se ha representado en la *Figura 11.12* donde podemos observar la confluencia de la fuente ON/OFF en un conmutador ActMs con puertos de ancho de banda de 155,52 Mbps. La *Figura 11.12* representa también los resultados que demuestran que el CSR, para una sola fuente, se mantiene en un valor muy superior al de CAR lo que permite disponer de tiempos de inactividad suficientes como para atender las solicitudes de retransmisión.

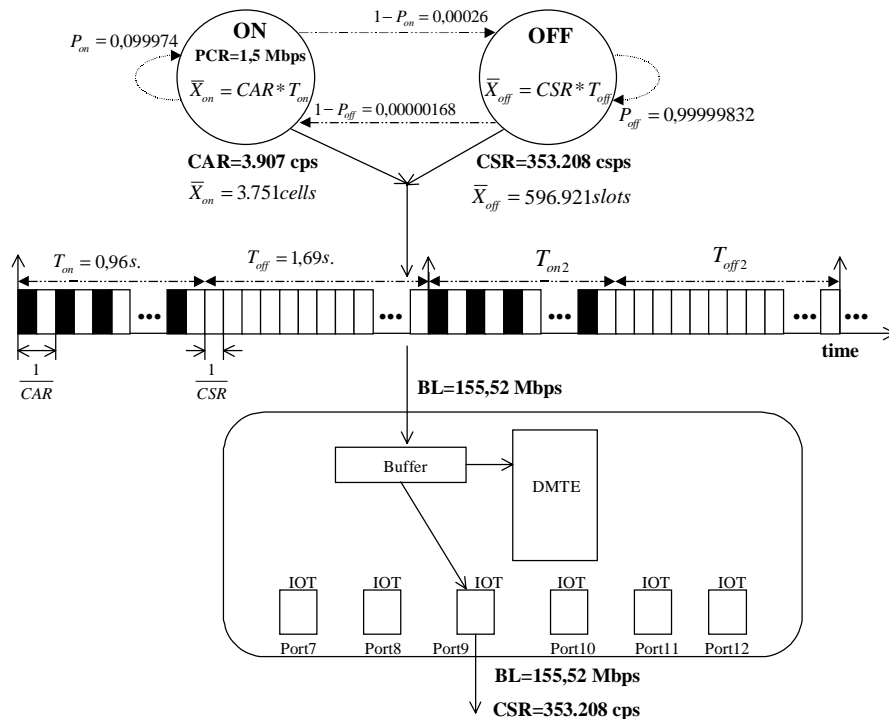


Figura 11.12. Patrón de llegada de células para una fuente ON/OFF concreta en un ActMs

En el siguiente punto estudiamos el efecto del T_{off} agregado con n fuentes; no obstante, supongamos ahora el caso de un conmutador ATM real tipo LE-155 con 16 ports de 155,52 Mbps cada uno y una potencia de conmutación de 2,488 Gbps a la salida del multiplexor interno del conmutador:

$$CAR = ((155.520.000 / 8) / 48) = 405.000cps$$

$$CSR = ((2.488.320.000 / 8) / 53) * (26 / 27) = 5.651.322sps$$

$$\bar{X}_{on} = CAR * T_{on} = 405.000 * 0,96 = 388.800c.$$

$$\bar{X}_{off} = CSR * T_{off} = 5.651.322 * 1,69 = 9.550.735slots$$

En este caso, si suponemos tener las 16 fuentes simultáneamente a 155,52 Mbps se generarían $16 * 405.000 = 6.480.000$ células que son también menores que los slots que pueden producirse en cada estado de OFF, lo que justifica la posibilidad de realizar retransmisiones durante las diferencias entre los tiempos de actividad y de silencio como veremos en la siguiente sección.

Los ejemplos anteriores demuestran conceptualmente la idea intuitiva de que es factible la realización de retransmisiones en los estados de silencio en el caso de emplear una sola fuente. Hemos partido de valores empíricos y concretos para T_{on} y T_{off} , pero podemos ver cómo realizando leves cambios en estos valores sigue existiendo un amplio margen para poder realizar las retransmisiones en los estados de inactividad.

11.8.2. DISPONIBILIDAD DE T_{OFF} AGREGADO CON N FUENTES ON/OFF

En los escenarios anteriores hemos podido comprobar las posibilidades de retransmisión aprovechando los estados de inactividad de una única fuente. Parece evidente que las posibilidades de retransmisión empiezan a disminuir a medida que introducimos más fuentes en los escenarios, porque a un conmutador llegan más conexiones que van a terminar siendo multiplexadas sobre un mismo puerto de salida. En este caso parece que el tiempo T_{off} total de un enlace será compartido por varias fuentes, por lo que es importante analizar qué ocurrirá en esta situación. Pues bien, podemos generalizar el modelo ON/OFF de dos estados a N estados con velocidades fijas en cada uno de los estados. Estos escenarios multi-estado (llamados procesos determinísticos modulados) pueden usarse para modelar un número concreto de N fuentes ON/OFF multiplexadas sobre un mismo enlace:

- Si los tiempos de permanencia siguen una distribución arbitraria, el proceso resultante es llamado *GMDP* (*Generally Modulated Deterministic Process*).
- Si las duraciones de los estados son exponencialmente distribuidas, el proceso es llamado *MMDP* (*Markov Modulated Deterministic Process*). En este caso, cada estado produce un número de células distribuidas geoméricamente durante cada periodo de permanencia. Esto es así porque, habiendo generado la llegada i , se genera la llegada $i+1$ con una probabilidad dada por la posibilidad de que el tiempo de permanencia no acabe antes del tiempo de la siguiente llegada. Esta probabilidad es una constante si los periodos de permanencia son distribuidos exponencialmente debido a la propiedad de “no-memorización” de las distribuciones exponenciales negativas.
- Puede evitarse el restringir el modelo para tener una velocidad constante de llegadas en cada estado. Si el proceso de llegada por estado sigue una Poisson, tendremos una *MMPP* (*Markov Modulated Poisson Process*) que puede ser muy útil para representar un proceso de fuentes agregadas.

Pero en nuestro caso, puede tener mayor interés otro punto de vista más práctico para el estudio de múltiples fuentes ON/OFF que confluyen en un solo buffer o puerto de salida. Supongamos tener N fuentes ON/OFF idénticas que envían células a un buffer de capacidad de servicio de C células por segundo y de un tamaño fijo y finito de T células. T_{on} y T_{off} son las duraciones medias en los estados ON y OFF respectivamente como ya vimos antes. *CAR* es de nuevo la velocidad de llegada de células por segundo en el estado activo pero, en este caso, introducimos el parámetro *MCR* (*Mean Cell Rate*) que expresa la velocidad media de llegadas de células en cada fuente:

$$MCR = CAR \left(\frac{T_{on}}{T_{on} + T_{off}} \right) \quad (16)$$

En este caso, calculamos la probabilidad de que la fuente esté en el estado activo, o lo que podemos denominar como el factor de actividad (*FA*) de cada fuente como,

$$FA = \frac{MCR}{CAR} = \frac{T_{on}}{T_{on} + T_{off}} \quad (17)$$

Podemos ver aquí un aspecto interesante para demostrar nuestra propuesta en el caso de N fuentes ON/OFF, que es el estudio de un parámetro que nos permita saber cuántas veces cabe (o encaja) la velocidad *CAR* de las N fuentes de tráfico en la capacidad de servicio C del buffer y que podemos denominar como número máximo de fuentes privilegiadas (*FP*), a las que podemos garantizar que dispondrán de suficiente tiempo de OFF agregado en el enlace para poder atender las retransmisiones en el caso de congestiones. Denotamos este parámetro con la expresión,

$$FP = \frac{C}{CAR} \quad (18)$$

La *Figura 11.13*, con N fuentes ON/OFF idénticas operando independientemente, representa los conceptos que acabamos de comentar y aplicados al tercer escenario que hemos representado en la *Figura 11.12*. Podemos comprobar cómo en este caso $FP=90,4$ fuentes, cada una de ellas de 1,5 Mbps sobre un enlace de 155,52 Mbps y con un factor de actividad FA del 36%.

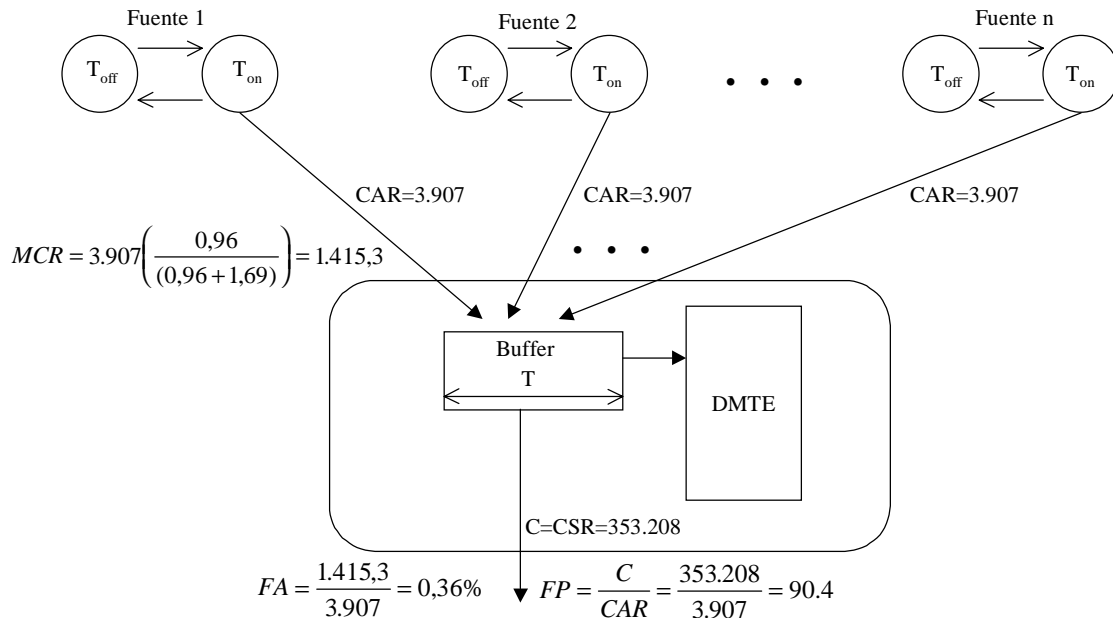


Figura 11.13. Modelo de múltiples fuentes ON/OFF

El valor de FP puede ser un valor no entero. Si lo redondeamos por encima tendremos el número mínimo de fuentes a partir del cual se puede producir la congestión del buffer. Si redondeamos el valor de FP por debajo tendremos el número máximo de fuentes del sistema para no producir congestiones del buffer, o también aquel número máximo de fuentes a las que podremos garantizar las retransmisiones por disponer de suficiente tiempo de inactividad agregado.

En la arquitectura que proponemos tenemos un solo buffer de entrada para las fuentes de tráfico y podemos considerar que la salida del buffer es el punto de multiplexación de todas las fuentes. La multiplexación de las fuentes de entrada debe ser, por tanto, menor o igual que la capacidad de servicio del buffer para que no se produzca la congestión a la salida de éste que, en nuestro caso, consideramos como el puerto de salida del conmutador en el que confluyen las fuentes de entrada para ser multiplexadas.

Según los apartados anteriores, para nosotros es importante aprovechar los tiempos de inactividad de los enlaces de salida en los conmutadores sobre los que se produce la multiplexación de diversas fuentes de entrada. La *Figura 11.14* aporta esta visión intuitiva aprovechando las fuentes ON/OFF que venimos usando. En esta figura podemos observar los modelos de llegada de tres fuentes con sus correspondientes tiempos de actividad y de silencio. Se presenta también el modelo que obtendríamos a la salida de un enlace en el que se multiplexan las tres fuentes, de forma que podemos observar cómo los tiempos T_{on} de cada una de las fuentes se reflejan a la salida. En la salida el tiempo de inactividad es menor, ya que la multiplexación de las tres fuentes se hace a costa de reducir el tiempo de inactividad del enlace. Precisamente, nuestro objetivo es el de atender las retransmisiones en el caso que el T_{offAg} agregado que queda en los enlaces así lo permita. La *Figura 11.14* nos muestra, por tanto, que el T_{on} a la salida es la suma de los T_{on} de las entradas, por lo que el T_{offAg} agregado que queda disponible a la salida para las retransmisiones es el T_{off} original al que hay que restar la suma de los T_{on} de las entradas. De este modo, antes de realizar la solicitud de una retransmisión es necesario evaluar el tiempo de inactividad que tenemos disponible en un enlace, ya que puede ocurrir que el T_{offAg} agregado sea menor que el T_{on} de la fuente cuya retransmisión desea realizarse. En esta situación la retransmisión no será solicitada, ya que no haríamos más que sobrecargar la red con una solicitud que no tendrá cabida en el enlace y que, con seguridad, acabará provocando lo que en las simulaciones hemos denominado como fallos de memoria DMTE.

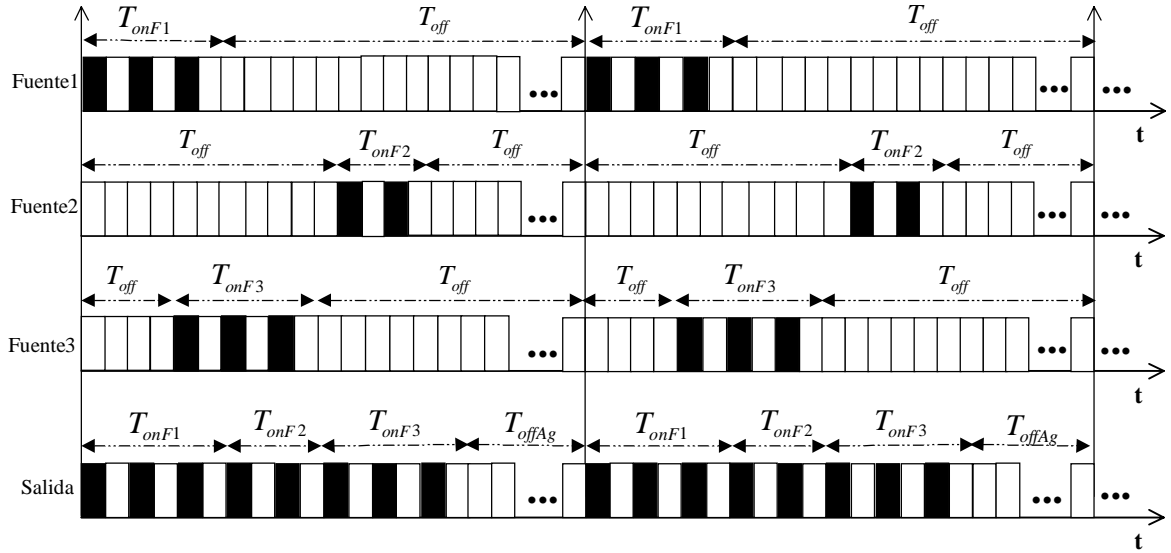


Figura 11.14. Multiplexación de tres fuentes de entrada en un puerto de salida con T_{offAg} agregado

A continuación vamos a formalizar estas reflexiones para justificar nuestros planteamientos. Para ello fijamos la siguiente notación:

T_{onE} ; Es el tiempo de actividad a la entrada

\bar{T}_{onE} ; Es el tiempo de actividad medio a la entrada

T_{offS} ; Es el tiempo de silencio a la salida

\bar{T}_{offS} ; Tiempo de silencio medio a la salida.

PCR_E ; PCR a la entrada

PCR_S ; PCR a la salida

N ; Representa el número de fuentes

\bar{R}_E ; Velocidad media de llegada de células a la entrada (CAR medio a la entrada).

\bar{R}_S ; Velocidad media de llegada de células a la salida (CAR medio a la salida).

En el caso de una sola fuente, el tiempo de actividad medio se representa en general según la siguiente expresión,

$$\bar{T}_{on} = \frac{\bar{R}}{PCR} \bar{T}_{off} \quad (19)$$

Si suponemos disponer de N fuentes de entrada, podemos calcular el tiempo de actividad a la salida de un conmutador sobre el que se multiplexan las N fuentes,

$$\bar{T}_{onS} = \frac{N\bar{R}_S}{PCR_S} \bar{T}_{offS} \quad (20)$$

De (20) podemos obtener entonces el tiempo de inactividad agregado disponible según,

$$\bar{T}_{offS} = \frac{PCR_S}{NR_S} \bar{T}_{onS} \quad (21)$$

Considerando la expresión como valor a la entrada de un conmutador, y despejando la velocidad media R_E de llegada de células a la entrada (variante de entrada en la ecuación (20)), y sustituyéndola en (21) obtendremos,

$$\bar{T}_{offS} = \frac{1}{N} \frac{PCR_S}{PCR_E} \frac{\bar{T}_{onS}}{\bar{T}_{offE}} \bar{T}_{onS} = \frac{1}{N} \frac{PCR_S}{PCR_E} \frac{\bar{T}_{onS}}{\bar{T}_{onE} / \bar{T}_{offE}} \quad (22)$$

Si ajustamos (22) podemos llegar a la siguiente expresión

$$\frac{\bar{T}_{offS}}{\bar{T}_{onS}} = \frac{1}{N} \frac{PCR_S}{PCR_E} \frac{\bar{T}_{offE}}{\bar{T}_{onE}} \quad (23)$$

Agrupando los tiempos en (23) obtenemos,

$$\frac{\bar{T}_{onS} + \bar{T}_{offS}}{\bar{T}_{onE} + \bar{T}_{offE}} < \frac{1}{N} \frac{PCR_S}{PCR_E} \quad (24)$$

Si consideramos que $PCR_S = N * PCR_E$, podemos comprobar cómo las medias de tiempos a la entrada coinciden con la suma de los tiempos medios de entrada,

$$\bar{T}_{onS} + \bar{T}_{offS} = \bar{T}_{onE} + \bar{T}_{offE} \quad (25)$$

No obstante, nuestro objetivo principal es el de poder expresar el valor del tiempo de OFF a la salida y en función de las entradas, lo que nos servirá para estimar el valor agregado de que disponemos para la retransmisión. Así, podemos expresar el valor medio de llegada de células de una nueva forma,

$$\bar{R} = \frac{PCR_E}{T_{onE} + T_{offE}} \Rightarrow T_{offE} = \frac{PCR_E}{\bar{R}} - T_{onE} \quad (26)$$

Usando (26) podemos expresar (24) despejando el valor que nos interesa, y sustituyendo $T_{onS} = NT_{onE}$,

$$\bar{T}_{offS} = \frac{1}{N} \frac{PCR_S}{PCR_E} (\bar{T}_{onE} + T_{offE}) - T_{onS} = \frac{1}{N} \frac{PCR_S}{PCR_E} (\bar{T}_{onE} + \frac{PCR_E}{\bar{R}} - T_{onE}) - NT_{onE} \quad (27)$$

El reajuste de (27) nos lleva a

$$\bar{T}_{offS} = \frac{PCR_S}{NR} - NT_{onE} \quad (28)$$

Para poder atender las retransmisiones, el tiempo de OFF debe ser superior a 0, y para que esto sea así, de (28) deberemos tener,

$$\frac{PCR_S}{NR} > NT_{onE} \quad (29)$$

De este modo podemos llegar a la expresión,

$$PCR_S > N^2 T_{onE} \bar{R} \quad (30)$$

Que, debidamente ajustada, acaba dando lugar a,

$$PCR_S > N^2 PCR_E \frac{T_{onE}}{T_{onE} + T_{offE}} \quad (31)$$

La fórmula (31) expresa, por tanto, una referencia válida del límite que debe tener el PCR en la salida, expresado en los valores conocidos de la entrada, para poder disponer de un T_{offAg} agregado suficiente para garantizar que la retransmisión es factible en esa situación.

11.9. IMPLICACIONES SOBRE LA SEÑALIZACIÓN

Como es sabido, ATM es una tecnología orientada a la conexión, por lo que es necesario disponer de un protocolo de señalización encargado de la configuración, modificación, control y realización de cada una de las conexiones. Los parámetros que identifican cada conexión (VPI, VCI, ports, PCR, GoS, QoS, etc.) son acordados entre los dos extremos de la comunicación y con la red que debe comprobar si dispone de suficientes recursos y si no existe incompatibilidad entre los extremos que impida la comunicación. El protocolo de señalización es el encargado de realizar estas funciones [6] y, para ello, debe contar con un proceso en el que estén implicados los tres elementos que deben acordar los citados parámetros. Cuando la red está compuesta de múltiples nodos, éstos deberán acordar también los parámetros entre ellos [7].

Aunque los protocolos UNI y NNI son muy parecidos, existen diferencias apreciables como, por ejemplo, que en el caso de NNI debe controlarse la multiplexación en la situación habitual en que se soporten múltiples conexiones en un enlace. Así, el protocolo de señalización ha sido estandarizado por ITU-T en su recomendación Q.2931[8], y por ATM Forum en UNI 4.x [6] que no especifica ningún estándar para NNI. Las normas ITU-T y ATM Forum son muy parecidas y, aunque tienen ligeras diferencias, ambas requieren de un canal virtual de señalización propio y separado de los canales de datos (*out band*) para la negociación de los parámetros entre los usuarios y la red. Las dos recomendaciones establecen los mensajes definidos para realizar todas las negociaciones iniciales y el control durante todo el tiempo que dure la comunicación.

Es interesante destacar también las propuestas de señalización pensadas para servicios específicos, como SSCOP (Service-Specific Connection-Oriented Protocol) [9], que se ha propuesto situado en la pila de protocolos ATM sobre la capa AAL-5 y se encarga de aspectos como el control de flujo, la corrección de errores basándose en un mecanismo de retransmisiones, etc..

No es nuestro objetivo desarrollar los aspectos relativos a la señalización en la red; sin embargo, debemos destacar que la aplicación del protocolo TAP implicaría emplear la señalización estándar por un lado, y por otro habría que adaptar ligeramente algunos de los aspectos de la misma para disponer de información relativa a la red que es controlada por estos protocolos específicos. Así, y aunque no se ha especificado expresamente en los algoritmos estudiados en este capítulo, en las funciones y procedimientos donde se realizan las labores de negociación y liberación de la conexión, en las transferencias entre capas, en la actualización de las Tablas de E/S, etc., intervienen también los protocolos de señalización.

Algunos de los aspectos de la señalización de mayor interés para alcanzar nuestros objetivos están relacionados con la identificación de los canales de comunicación, tanto en UNI, como en NNI, ya que de esta identificación (VPI, VCI, ports) depende el mecanismo de recuperación propuesto en nuestras investigaciones. Según esto, las funciones de negociación de la conexión, de establecimiento y liberación de la misma, los procedimientos de creación de índices -tanto en las Tablas de E/S como en la memoria DMTE- y la propia transferencia de células BRM como resultado de una solicitud de retransmisión nos exigen acceder al canal de señalización para identificar cada una de las conexiones implicadas en las transferencias que han negociado la GoS con la red. Estos accesos nos permitirán conocer los VPI, los VCI y los puertos de entrada y de salida de cada una de las conexiones que deseamos controlar para poder garantizarles un servicio privilegiado con respecto al resto de conexiones que no lo demanden.

11.10. CONCLUSIONES

En este capítulo hemos detallado los aspectos de mayor interés del conjunto de algoritmos que constituyen el protocolo TAP. Hemos centrado nuestra atención en destacar la visión que tienen de TAP, tanto los nodos terminales de la comunicación, como los conmutadores activos que soportan el SMA-TAP. Por esto hemos presentado también los algoritmos que implementan cada uno de los agentes software del SMA, describiendo sus puntos de comunicación, así como la relación con los elementos hardware de la arquitectura que fueron comentados en el capítulo anterior.

Hemos modelado mediante fuentes ON/OFF los patrones de llegada de células a los conmutadores para formalizar la idea intuitiva de aprovechar los estados de inactividad de las fuentes y de los enlaces para atender las solicitudes de retransmisión cuando uno de los conmutadores esté experimentando congestiones. Esta formalización nos permite comprobar cómo es factible aprovechar los periodos de silencio para recuperar las pérdidas, sin afectar por ello ni al goodput de la red, ni tampoco al resto de las fuentes que puedan estar compartiendo un mismo conmutador o enlace.

Por último, hemos comentado brevemente algunos de los aspectos relativos a la señalización y control que están relacionados con TAP, de forma que vamos a depender de los protocolos estándares de la propia tecnología ATM para alcanzar nuestros objetivos.

REFERENCIAS

- [1] J. M. Pitss, "Introduction to ATM. Design and Performance", *Ed. Wiley*, (1997).
- [2] A.L. Roginsky, L.A. Tomek and K.J. Christensen, "Analysis of ATM Cell Loss for Systems with on/off Traffic Sources," *IEE Proc. Commun.* Vol. 144. No. 3, pp. 129-134, (1997).
- [3] Sebastián Galmés Obrador, "Algoritmos de captura a gran escala de fuentes de tráfico ATM," *Memoria de tesis doctoral*, (1998).
- [4] K. Kvols and S. Blaabjerg, "Bounds and approximations for the periodic on/off queue with applications to ATM traffic control," *INFOCOM'92*, pp. 487-494, (1992).
- [5] K. Sohraby, "On the theory of general on-off sources with applications in high-speed networks," *INFOCOM'93*, pp. 401-410, (1993).
- [6] ATM Forum, "Traffic Management Specification Version 4.0," *ATM Forum Technical Committee, ATM Forum Document af-tm-0056.000*, (April 1996).
- [7] M. de Prycker, "Asynchronous Transfer Mode. Solution for Broadband ISDN (3^a Ed.)," *Ed. Prentice Hall*, (1995).
- [8] ITU-T, Q.2931, "BISDN User Network Interface Layer 3 Protocol," (1994).
- [9] ITU-T, Q.2110, "BISDN Signalling ATM Adaptation Layer (SAAL). Service Specific Connection Oriented Protocol SSCOP," (1994).