Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Contribución al Estudio y Diseño de Mecanismos Avanzados de Servicio de Flujos Semi-Elásticos en Internet con Garantías de Calidad de Servicio Extremo a Extremo

TESIS DOCTORAL

Tesis Doctoral presentada en la Universidad Politécnica de Cataluña para la obtención del título de Doctor Ingeniero de Telecomunicación

Autor: **Marcos Postigo Boix**

Director: **Dr. Joan García Haro**

Año: **2003**

Tribunal nomenat per l'Il.lm. Senyor Rector de la Universitat Politècnica de Catalunya, el dia de de 20

President Dr.

Vocal Dr.

Vocal Dr.

Vocal Dr.

Secretari Dr.

Realitzat l'acte de defensa i lectura de la Tesi Doctoral

El dia de de 20 a

Qualificació:

EL PRESIDENT ELS VOCALS

EL SECRETARI

A ti, que lees.

"Saber que se sabe lo que se sabe y que no se sabe lo que no se sabe;
he aquí el verdadero saber."

Confucio. 551 AC-478 AC. Filósofo chino.

Agradecimientos

Con estas líneas pretendo agradecer a todas aquellas personas que durante el transcurso de este trabajo de Tesis Doctoral me han ayudado, me han animado y han hecho más agradables los momentos difíciles.

En primer lugar quiero agradecer el apoyo absoluto de mi director de Tesis, Joan García Haro, quien a pesar de la distancia ha estado siempre totalmente disponible para realizar el seguimiento de este trabajo, realizando en cada momento las indicaciones oportunas, así como animándome a no desfallecer en aquellos momentos más difíciles. También quiero agradecerle la confianza que siempre ha depositado en mi carrera profesional universitaria, sin la cual este trabajo nunca se habría realizado.

Aprovecho también para agradecer a Mónica Aguilar sus constantes dosis de ánimo y soporte, y muy en especial, cuando en algún momento necesité que me ayudara a reducir la carga de trabajo que podía alejarme de la investigación de la Tesis.

También a José Luis Melús, quien gustosamente ha realizado las labores de Tutor de esta Tesis, y con quien durante muchos momentos he analizado muy constructivamente el trabajo que iba realizando.

No puedo olvidarme tampoco de Francisco José García y Eduardo Gascón quienes colaboraron en la implementación del sistema real y la documentación del sistema simulado.

Asimismo, agradecer al resto de mis compañeros de línea de investigación el respaldo recibido en todo momento, así como sus consejos siempre provechosos. Muy especialmente a Josep Pegueroles (mi compañero de despacho) quien ha debido soportarme durante prácticamente todos los días que he dedicado a este trabajo, y de quien siempre he recibido buenos consejos, además de su total apoyo y su amistad incondicional. A José Luis Muñoz y a Óscar Esparza, por esos momentos de descanso junto al calor del café o el frío de un refresco de cola de cuyo nombre no quiero acordarme, además de su siempre acertada colaboración en el ámbito docente. A Esteve Pallarés, quien me acompañó en algún viaje y que siempre se ha ofrecido a ayudarme en lo que pudiera. A Juanjo Alins por su ayuda en la instalación de la red con calidad de servicio, que permitió la implementación real del sistema. A Jorge Mata, Miquel Soriano, Luis Javier de la Cruz, Jordi Forga, Francisco Rico, Jordi Forné, Isabel Martín, por todo lo que he podido aprender escuchándolos en tantos momentos. Y como no, a Marcel Fernández quien también en este momento, está pasando por la situación de finalizar su Tesis, por lo que compartimos juntos estos momentos.

También agradecer su apoyo a todo el departamento de Ingeniería Telemática por haberme permitido realizar este trabajo poniendo a mi disposición todos los recursos que fueron posibles.

No quiero olvidarme de agradecer también a todos los miembros del tribunal por su desinteresado trabajo de evaluación de esta Tesis.

Finalmente, agradecer a toda mi familia toda su ayuda y apoyo, muy especialmente a mis padres y hermano, y como no a la persona que me ha soportado durante todo el tiempo que no he dedicado a este trabajo: a Mireia.

Resumen

Los servicios y aplicaciones ofrecidos en Internet demandan, con mayor frecuencia, un determinado nivel de servicio para su correcto funcionamiento. Esto supone el desarrollo de mecanismos que permitan ofrecer calidad de servicio diferenciada entre extremos distantes de la red. En una red con estas características el coste debido al uso de este servicio diferenciado es mayor que el de utilizar el modo tradicional de transferencia *best-effort*. Ello obliga a los usuarios a realizar un uso responsable de los recursos reservados. Por tanto, será crucial reservar sólo aquellos recursos estrictamente necesarios para lograr un mínimo coste.

En este trabajo de Tesis Doctoral, se aborda el problema de minimizar los recursos reservados para una transmisión de un flujo semi-elástico. Este tipo de flujo se caracteriza por necesitar la reserva de más o menos recursos en función del estado de la red, por lo que se plantea el diseño de un sistema cliente-servidor capaz de realizar de forma automática las reservas estrictamente necesarias. Para ello, se propone un mecanismo de control de la ocupación de la memoria del cliente que permite determinar los periodos en que es necesario utilizar un modo de transferencia *best-effort* (aquellos en los que la ocupación garantiza la disponibilidad de datos en recepción), y los periodos en que se necesita un modo de transferencia con reserva de recursos para garantizar el llenado de dicha memoria. Del análisis de este mecanismo, se deducen expresiones para el coste de transmisión en función de los diversos parámetros que afectan a la ocupación de la memoria del cliente. De estos parámetros destaca especialmente la tasa de llegada de datos, que depende del estado de la red, y el umbral máximo de ocupación que indica cuándo se puede transmitir en modo *best-effort*. Gracias al correcto dimensionado de este umbral máximo, el cliente es capaz de minimizar los recursos reservados de la red. Para calcular este umbral, uno de los problemas que se plantea es la necesidad de una estimación inicial de la tasa de llegada de datos en modo *best-effort*. Para ello, se propone el uso de un umbral inicial que permita pasar a modo *best-effort* y realizar así, la estimación de esta tasa. Por otra parte, se propone también el uso de un mecanismo que mejora la minimización del coste de transmisión cuando la variabilidad de la tasa de llegada de paquetes en modo *best-effort* es grande, sin incrementar desmesuradamente la cantidad de señalización asociada necesaria.

Finalmente, se propone un mecanismo para dar servicio de forma simultánea a varios clientes. Concretamente, se estudia mediante simulación un caso homogéneo, donde todos los clientes presentan los mismos requisitos. El esquema propuesto permite dar servicio simultáneo a un número determinado de clientes reservando recursos, mientras que el resto se sirve en modo *best-effort*. Del estudio se determina la existencia de un valor óptimo para el número de clientes a los que se da servicio en modo de reserva de recursos, que minimiza la señalización extra necesaria para gestionar el servicio.

Asimismo, como complemento adicional para realizar este trabajo, se ha desarrollado una implementación del sistema cliente servidor de flujos semi-elásticos en el simulador ns-2, y además otra implementación real en lenguaje C sobre sistema Linux.

Índice

Capítulo 1

Capítulo 2

Capítulo 3

Capítulo 5

Capítulo 6

Anexo A

Índice de Figuras

Índice de Tablas

Capítulo 1
Introducción

1.1 Motivación

Actualmente, los usuarios de Internet utilizan el servicio clásico de *best-effort*, en el que no se ofrece ningún tipo de garantía de calidad de servicio (*Quality of Service*, QoS). Sin embargo, en los últimos años, se han desarrollado varios protocolos y mecanismos que permiten proveer de la QoS necesaria a aplicaciones con requerimientos temporales y semánticos[i] muy estrictos.

En este escenario, se plantea el problema del coste por uso de la QoS diferenciada, inexistente en las redes *best-effort*. Como parece lógico pensar, la utilización de este nuevo servicio diferenciado (que permite satisfacer las exigencias, cada vez más estrictas, de QoS de los usuarios) será más caro que la transmisión clásica en modo *best-effort*, ya que en caso contrario, el uso de este modo de transferencia no tendría sentido, desde el punto de vista del usuario final. Este nuevo coste, afecta al precio de los nuevos servicios que se puedan ofrecer en esta Internet. Así, el coste aumentará a medida que se reserven y requieran más recursos de la red.

[i] En cuanto a pérdida de datos.

Por otro lado, el coste tiene un factor correctivo en el comportamiento de los usuarios de la red. Asimismo, un proveedor de servicio que oferte (venda) un determinado producto en Internet (requiriendo un cierto nivel de QoS), y que realice un uso excesivo o inadecuado de la reserva de recursos de red, deberá pagar más al proveedor (operador) de red, y en consecuencia verá reducidos sus beneficios.

Estos requerimientos de QoS, y por tanto de reserva de recursos, dependen del tipo de flujo de datos que necesite enviar una determinada aplicación para proporcionar el servicio que un determinado usuario solicite. De esta manera, habrá aplicaciones que necesiten enviar toda la información garantizando una determinada QoS, por lo que deberán reservar recursos siempre. Otras no necesitarán garantías de servicio, y el servicio *best-effort*, será suficiente. Finalmente, las habrá que puedan necesitar el reservar recursos durante un determinado tiempo, mientras que el resto de la transmisión se podrá utilizar el servicio *best-effort*.

Para estas últimas (denominadas semi-elásticas), será posible minimizar la cantidad de información enviada mediante el modo de transmisión con reserva de recursos, y de esta forma optimizar el coste asociado a las mismas. Este tipo de aplicaciones, se caracterizan por requerir que la información llegue al extremo cliente en un determinado instante, de forma que sea necesaria la reserva de recursos para garantizar que los datos lleguen antes y puedan ser posteriormente consumidos sin problemas a la tasa de lectura de la aplicación. Para que los datos puedan llegar a tiempo, deben estar previamente disponibles en el servidor, lo que permite avanzar el envío antes de su uso, implicando su almacenamiento en la memoria del cliente. Así será necesario reservar recursos para garantizar la disponibilidad de datos en dicha memoria. De forma que si se detecta una cantidad suficiente de datos almacenados, se podrá transmitir en modo *best-effort*, reduciendo el coste de la transmisión.

1.2 Objetivos

En este trabajo, se pretende diseñar un sistema cliente-servidor de flujos semi-elásticos, que optimice de forma automática el coste de la transmisión en función del estado de la red en modo *best-effort*. El diseño se centra en la parte de control de optimización del coste, es decir, la que se encarga de minimizar la cantidad de información que envía la aplicación servidor mediante un modo de transmisión basado en reserva de recursos, o lo que es semejante, el tiempo que emplea para

transmitir los datos. En la Figura 1.1, se muestra como fluye la información desde la aplicación servidor hacía la aplicación cliente. Como se puede observar, tanto el servidor como el cliente son aplicaciones genéricas, que tienen como condición el generar un flujo de datos semi-elástico. Este flujo de datos, está controlado por un proceso de optimización del coste de la transmisión, objeto de estudio de este trabajo. Es, por tanto, este proceso el que se encarga de recoger, de forma transparente, los datos que debe enviar la aplicación servidor hacia el cliente, y de enviarlos hacía las capas de protocolos de transporte (p.e., TCP/UDP) en forma de paquetes, controlando el modo de transmisión adecuado (reserva de recursos o *best-effort*). Por su parte, el cliente (mediante el proceso de optimización del coste) recibe los datos de la capa TCP/UDP, para posteriormente, pasarlos a la aplicación que los consumirá.

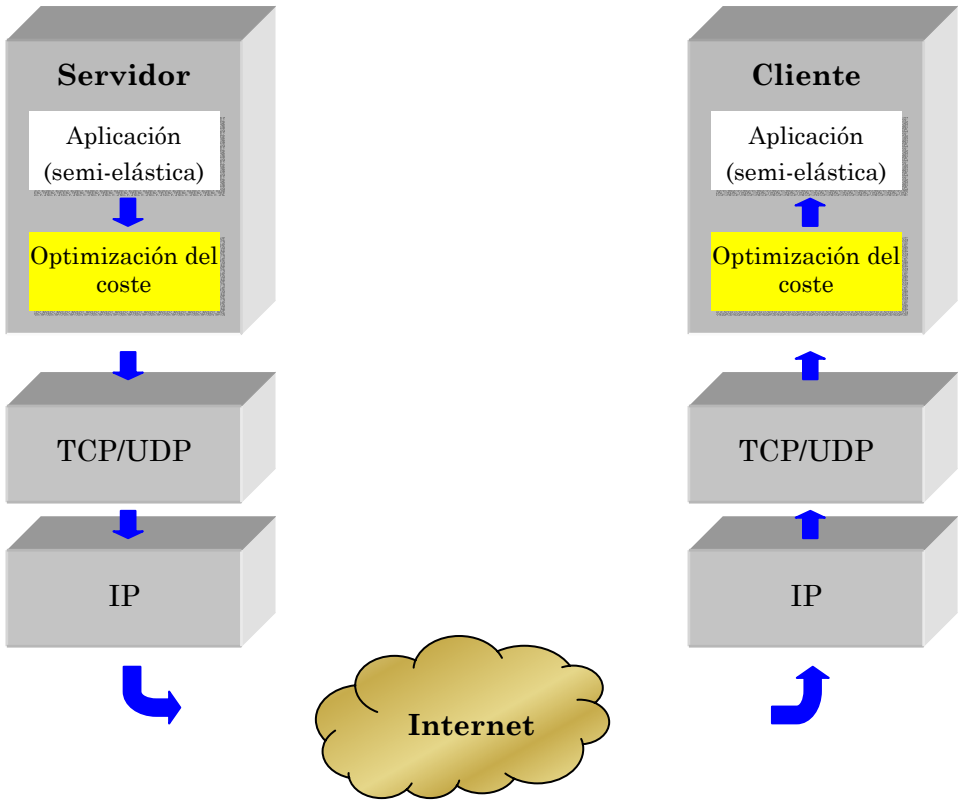

Figura 1.1: Flujo de información y capas de protocolos.

Se pretende caracterizar el proceso de optimización del coste de forma totalmente independiente tanto de la aplicación, como de los protocolos inferiores y de los mecanismos utilizados para proporcionar QoS extremo a extremo. Debido a ello, el proceso optimizador del cliente sólo tendrá noción de la tasa de llegada de datos provenientes de la capa TCP/UDP, que a su vez dependerá del modo de transmisión utilizado, que se verá influido por los protocolos inferiores y por los

mecanismos de provisión de QoS en uso. Por otro lado, se supone que la tasa de traspaso de datos a la aplicación es constante, para que la variabilidad de datos almacenados en la capa de optimización sólo dependa del estado de la red. Asimismo, este proceso se encarga de almacenar los datos en una memoria, manteniendo controlada dicha ocupación entre dos umbrales que garanticen que la aplicación cliente disponga a tiempo de los datos que necesita. En este trabajo, se realizará el análisis del comportamiento del coste en función de los diversos parámetros del sistema optimizador del cliente. Para realizar dicho análisis, primero se determina el comportamiento genérico del sistema, fundamentado en la utilización del modo de reserva de recursos cuando es necesario llenar la memoria del cliente, mientras que si se alcanza un determinado nivel de ocupación suficiente, será posible utilizar el modo *best-effort*. Una vez determinado este comportamiento, se diseñará el control de la ocupación de la memoria del cliente, analizando mediante un modelo matemático cómo evoluciona el coste en función de los diversos parámetros que afectan a dicha ocupación. Además, se implementa y desarrolla la simulación del sistema para comprobar el funcionamiento del proceso optimizador del coste. También se analizarán diversos factores que afectan al comportamiento óptimo del sistema, tales como la variabilidad de la tasa de llegada de datos debida al tráfico existente o presente en la red (de fondo), el tamaño de la memoria del cliente, el tamaño del paquete utilizado para enviar la información, etc. Paralelamente a este estudio, se determinará el comportamiento del proceso optimizador, situado en el servidor, que simplemente se encargará de enviar la información de la forma en que se lo indique el cliente.

Finalmente, se analizará el comportamiento del sistema servidor, cuando se realiza el servicio simultáneo de varios clientes. En este caso, el problema principal es determinar cómo se reparten los recursos disponibles por el servidor, concretamente el ancho de banda de acceso a la red. Este ancho de banda puede verse totalmente bloqueado por las reservas de recursos, haciendo que nuevas reservas por parte de otros clientes se vean rechazadas, provocando la percepción de un mal funcionamiento o una degradación en el grado de servicio del sistema. Para evitar esto, se propondrá un modo de servicio genérico que permitirá reservar recursos a aquellos clientes más necesitados, y obligará al resto a recibir los datos en modo *best-effort*. De esta forma de servicio simultáneo, se analizará mediante

simulación, la evolución del coste medio por cliente y la dinámica de la señalización extra necesaria para gestionarlo.

1.3 Desarrollo de la Tesis

El Capítulo 2 describirá los principales mecanismos propuestos para proporcionar QoS en las redes IP. Estas redes se dividen fundamentalmente en dos entornos: de área local (LAN, *Local Area Network*), controlada por ejemplo, por una empresa o una comunidad residencial; y de área extendida (WAN, *Wide Area Network*), controlada por un determinado proveedor de red, que se encarga de proporcionar conectividad entre extremos distantes de Internet. Tal y como se verá, los protocolos y mecanismos utilizados en estos entornos dependerán de su propia escalabilidad, ya que el número de flujos existente en un entorno LAN será en general mucho más pequeño que el existente en un entorno WAN, haciendo que la gestión sea menos costosa computacionalmente.

En el Capítulo 3, se definirán los distintos flujos de datos que pueden circular por una Internet en la que se ofrezca QoS extremo a extremo. Para realizar la clasificación se tiene en cuenta que los usuarios de una red con esta característica, tendrán unos requisitos de servicio diferentes a los de los actuales usuarios de redes *best-effort*. Es decir, se supone que la información necesaria, para que el usuario final perciba una calidad subjetiva notable, llegará a su destino antes, o a lo sumo en el mismo momento, de su consumo. Esto implica que las pérdidas deberán estar controladas para no afectar a la premisa de calidad subjetiva notable, así como el retardo en el caso de aplicaciones que igualmente lo requieran. Como se observará, los flujos se clasificarán según tres tipos, en función de cuales sean sus requerimientos de reservar recursos de la red: elásticos, inelásticos y semi-elásticos. De éstos, en este trabajo se contribuye a la definición de los semi-elásticos como tipo particular, ya que requieren una mayor o menor reserva de recursos en concordancia con el estado de la red *best-effort*, debido a que pueden utilizar este servicio para transmitir parte de la información hacia el cliente. Esta propiedad, permite plantear la minimización de estas reservas, para evitar que el coste de transmisión que debe soportar el proveedor que ofrece el servicio (el que posee la información originalmente), sea más elevado de lo necesario[ii]. En este capítulo, también se

[ii] Este coste afecta también al coste final que deberá pagar el cliente.

propondrá el modelo de coste a utilizar, así como el mecanismo utilizado para reducir el coste de la transmisión de flujos semi-elásticos.

En el Capítulo 4, se diseñará el sistema cliente como parte controladora del coste. Para ello, se presentará un modelo matemático sencillo que determine el coste de la transmisión en función de los diversos parámetros que afectan a la ocupación de la memoria del cliente. Además, se buscarán los umbrales máximos de ocupación que permiten minimizar el coste, de forma que un cliente pueda calcularlos en función del valor instantáneo de los diversos parámetros del sistema, especialmente de la tasa de llegada de paquetes, dependiente del estado de la red. Relacionado con esto, se propondrá un mecanismo para estimar esta tasa en el comienzo de la transmisión, necesaria para obtener inicialmente el valor del umbral máximo. Asimismo, se analizará cómo debe ser el valor del umbral mínimo que garantiza la disponibilidad de datos en la memoria del cliente, y cómo afecta la variabilidad de la tasa de llegada de datos en el funcionamiento del sistema, para proponer un mecanismo que permita minimizar el coste de forma precisa sin aumentar la señalización desproporcionadamente. Finalmente, en este capítulo, se mostrarán los resultados obtenidos mediante la simulación del sistema que se ha implementado en el simulador *Network Simulator* 2 (ns-2) [NS2].

En el Capítulo 5, se analizará el servicio simultáneo de flujos semi-elásticos, centrándose en la gestión del ancho de banda de acceso del servidor. Se propondrá un modo de servicio que permita reservar recursos a los clientes que necesiten con más urgencia llenar su memoria, mientras el resto se sirve en modo *best-effort*. De este mecanismo propuesto, se analizará, mediante simulación, la señalización necesaria para gestionar el sistema, así como la evolución del coste para casos donde se sirve a distinto número de clientes y donde se permite dar servicio con reserva de recursos a mayor o menor número de ellos. Respecto a la señalización, se utilizará como referencia el protocolo RSVP (*resource ReSerVation Protocol*), proponiendo la necesidad de un nuevo mensaje de reserva de recursos originado por el servidor, no incluido en el estándar y necesario para permitir a éste la correcta gestión de las reservas. Finalmente, se indicará cuál debe ser la elección óptima de parámetros para que el sistema consiga una minimización del coste correcta, con la menor señalización extra posible.

En el Capítulo 6 se expondrán las conclusiones más relevantes de este trabajo y se propondrán las líneas de actuación que quedan abiertas.

Finalmente, se adjuntan dos anexos presentados a continuación.

En el Anexo A se describe la implementación del sistema cliente-servidor de flujos semi-elásticos, realizada en el simulador ns-2 y utilizada para el estudio de los diversos mecanismos propuestos en este trabajo, así como para evaluar el comportamiento del sistema bajo distintas condiciones. Esta implementación se ha realizado íntegramente en código C++, añadido al propio del simulador, y los escenarios de simulación se han generado mediante OTcl.

En el Anexo B se presenta la implementación real sobre sistema operativo Linux de un sistema cliente-servidor de flujos semi-elásticos que permite dar servicio a un único cliente. Dicho sistema, se implementó para determinar la complejidad y viabilidad de implementación real del sistema cliente-servidor óptimo. Para ello, se utiliza una distribución libre del protocolo RSVP, y se implementan mecanismos de gestión de colas, ya incluidos en las últimas versiones del Kernel Linux. Sobre este sistema, se desarrolla en C el código del sistema cliente-servidor.

1.4 Artículos publicados

En este apartado, se enumeran las siguientes publicaciones motivadas por este trabajo de Tesis, así como algunas en proceso de revisión.

Nacionales

- M. Postigo-Boix, J. García-Haro, M. Aguilar-Igartua, *Transmisión Eficiente de Bloques en Tiempo Real sobre Redes IP*, XV Simposium Nacional de la Unión Científica Internacional de Radio (URSI 2000), Zaragoza, Spain, septiembre 2000, pp. 405-406

- M. Postigo-Boix, J. García-Haro, M. Aguilar-Igartua, *Análisis de Minimización de Costes en la Transmisión de Flujos Semi-Elásticos sobre Internet*, en las actas de las III Jornadas de Ingeniería Telemática (JITEL 2001), Barcelona, España, septiembre 2001, pp. 37-44.

Internacionales

- M. Postigo-Boix, J. García-Haro, M. Aguilar-Igartua, *Cost Minimization Study in the Client-Server Transmission of Semi-Elastic Flows Using Internet*, en Proceedings of the 2001 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, Victoria, B.C., Canada, agosto 2001, pp. 188-191.

- M. Postigo-Boix, J. García-Haro, M. Aguilar-Igartua, *Cost Minimization Study of Semi-Elastic Flows Using Internet*, en Proceedings of the IEEE

International Conference on Communications 2002 (ICC 2002), New York, USA, Abril 2002, pp. 2237-41.

- M. Postigo-Boix, J. García-Haro, M. Aguilar-Igartua, *A Cost Optimization Methodology for Internet Transmission of Semi-Elastic Traffic Flows*, en Proceedings of the 3rd International Conference on Internet Computing (IC 2002), Las Vegas, USA, junio 2002, pp. 311-16.

- M. Postigo-Boix, J. García-Haro, M. Aguilar-Igartua, *Simulation of a Reliable Client-Server System for Semi-Elastic Flows*, en Proceedings of the 2002 IASTED International Conference on Communication Systems and Networks, Malaga, España, septiembre 2002, pp. 332-37.

- M. Postigo-Boix, J. García-Haro, J. L. Melús-Moreno, *Transmission of Semi-Elastic Flows under High Variability Background Traffic*, en Proceedings of the 2002 IASTED International Conference on Applied Modelling and Simulation, Cambridge, USA, noviembre 2002, pp. 105-110.

Actualmente, en proceso de revisión:

- M. Postigo-Boix, J. García-Haro, J. L. Melús-Moreno, *Cost Optimization for Semi-Elastic Traffic Stream Transmission in an End-to-End QoS Enabled Internet*, enviado para revisión a IEEE Communications Letters.

- M. Postigo-Boix, J. García-Haro, J. L. Melús-Moreno, *Semi-Elastic Streaming Media Transmission: Resource Reservation Optimization Methodology*, enviado para revisión a IEEE Transactions on Multimedia, special issue on streaming media.

- M. Postigo-Boix, J. García-Haro, *Servicio Simultáneo de Flujos Semi-Elásticos en Internet. Caso Homogéneo*, enviado para revisión a JITEL 2003.

Capítulo 2
Calidad de servicio en redes IP

2.1 Introducción

Para poder implementar una red IP multiservicio totalmente integrada es necesaria una red que ofrezca a los usuarios (con requerimientos cada vez más estrictos) calidad de servicio (*Quality of Service*, QoS) extremo a extremo de forma diferenciada.

Algunos mecanismos necesarios para implantar esta red comienzan a estar maduros y disponibles comercialmente, mientras que otros están en fase de estandarización. Aún así, es posible definir un escenario que permita la implementación de QoS extremo a extremo como se puede observar en [STA99] [FIN02] [XIA02] [GIO03]. En las publicaciones actuales, se propone la implementación de QoS en todas las partes de la red: red de área local (*Local Area Network*, LAN) y red de área extendida (*Wide Area Network*, WAN).

En este capítulo se resumen los distintos mecanismos propuestos en la actualidad, que permiten ofrecer QoS en entornos LAN, así como en entornos WAN. Tal y como se verá, en entornos LAN es más fácil gestionar de forma individual la QoS para cada flujo de datos entre dos extremos, debido al menor volumen de información que en entornos WAN. En éstos, se gestiona la QoS por agregados de tráfico. Finalmente, se describirá cómo deben interoperar los distintos protocolos y mecanismos, permitiendo el traspaso de la información necesaria entre las distintas arquitecturas propuestas, para proporcionar QoS extremo a extremo.

2.2 Calidad de servicio en entornos LAN

En un escenario con QoS extremo a extremo, se supone que los usuarios de la red disponen de la tecnología adecuada, en sus redes locales, para poder acceder a estas prestaciones. En este apartado, se revisarán las tecnologías más comúnmente utilizadas para proveer de QoS a entornos LAN.

2.2.1 Calidad de servicio en Ethernet

Ethernet es la tecnología con más presencia en entornos LAN debido fundamentalmente a su alto ancho de banda y bajo precio. Realiza conmutación en la capa 2 (capa de enlace), creando conexiones punto a punto entre dos nodos de la red. El estándar IEEE 802.3x establece la forma de realizar esta conexión de forma bidireccional (*full-duplex*), es decir, permitiendo que los nodos envíen y reciban de forma simultánea. Las velocidades de transmisión que presenta esta red pueden ser de 10 Mb/s, 100 Mb/s (*Fast Ethernet*), 1 Gb/s (*Gigabit Ethernet*) y 10 Gb/s. Además, los estándares 802.1Q y D amplían la trama Ethernet en cuatro octetos para incluir etiquetado para LAN Virtual (VLAN) e información explícita de prioridad (mediante el campo *user_priority*) para los datos que circulan por la red Ethernet.

El campo *user_priority* del estándar 802.1D (definido formalmente en el IEEE 802.1p) utiliza tres bits, en la etiqueta VLAN del 802.1Q, para definir ocho tipos de tráfico Ethernet posibles. Para un determinado número de colas en un puerto de conmutación, se define qué tipo de tráfico usar y cómo asignarlo a las colas. El campo *user_priority* no especifica un comportamiento particular de los mecanismos de gestión de tráfico, aunque el 802.1D recomienda utilizar por defecto este campo para encolar tráfico con prioridades.

2.2.2 RSVP-E2E

Además de los estándares para proporcionar QoS en entornos LAN, los nodos de las redes locales pueden utilizar la arquitectura de Servicios Integrados (*IntServ*), utilizando el protocolo RSVP (*resource ReSerVation Protocol*) como protocolo de señalización. El uso de este protocolo es una hipótesis realista, ya que está incluido en las versiones más recientes de sistemas operativos ampliamente distribuidos como Microsoft Windows (2000, XP), y también existen versiones para LINUX. RSVP se utiliza para realizar dos funciones separadas: una de ellas es el RSVP-E2E (*End-to-end*, extremo a extremo) que es la explicada en este apartado y está basada en

[BRA97], mientras que la otra, RSVP-TE se describe en el apartado 2.3.6, y sirve como señalización en entornos de área extendida donde se utiliza MPLS (*Multi-Protocol Label Switching*).

La señalización RSVP-E2E se inicia en un determinado *host*[i] que solicita, de la red, un determinado nivel de QoS, que genera un flujo de datos originado por una aplicación. Los *routers* presentes en el trayecto del flujo de datos procesan los mensajes RSVP y establecen el estado para proveer el servicio requerido (Figura 2.1). A grandes rasgos, la señalización RSVP-E2E, es la comentada a continuación.

- El emisor caracteriza el tráfico que va a enviar y genera un mensaje *Path* que contiene dichas características (*Tspec*, *Traffic Specification*), y lo envía desde el emisor a la dirección de destino. Cada *router* dotado con capacidad para gestionar el protocolo RSVP-E2E a lo largo del trayecto, establece un determinado estado (*Path-state*) que incluye la dirección previa del mensaje *Path*[ii].
- Para realizar la reserva, el receptor envía la petición hacia el emisor mediante un mensaje *Resv*, por el camino por el cual ha llegado el mensaje *Path*. Este mensaje *Resv* incluye las especificaciones de la petición de reserva (*Rspec*, *Request Specification*), es decir, indica el tipo de clase de servicio *IntServ* deseado (carga controlada o garantizada) y también las especificaciones de filtro (*filter spec*), que caracterizan a los paquetes para los que se realiza la reserva. Los *routers* usan conjuntamente las especificaciones de reserva y de filtro para identificar a cada reserva, y a ambos parámetros juntos se denomina descriptor de flujo (*flow-descriptor*) de la sesión.
- Cuando cada *router* a lo largo del camino de vuelta hacia el emisor recibe un mensaje RSVP, se utiliza el proceso de control de admisión para autentificar la petición y reservar los recursos necesarios. Si la petición no se puede satisfacer (ya sea por falta de recursos disponibles o por fallo de autenticación), el *router* devuelve un mensaje de error hacia el receptor. En

[i] *Host* es un determinado dispositivo final de red.

[ii] Esta dirección será el próximo salto en el camino de vuelta hacia el emisor (*next hop upstream*).

caso contrario, si se acepta la petición se crea un estado de reserva (*Resv-state*), y el *router* envía el mensaje *Resv* al siguiente *router* en dirección al emisor.

- Cuando el último *router* recibe el mensaje *Resv* y acepta la petición, devuelve un mensaje de confirmación hacia el receptor.
- Cuando el emisor o el receptor finalizan una sesión RSVP, existe un proceso similar para liberar los recursos reservados (*tear-down process*).

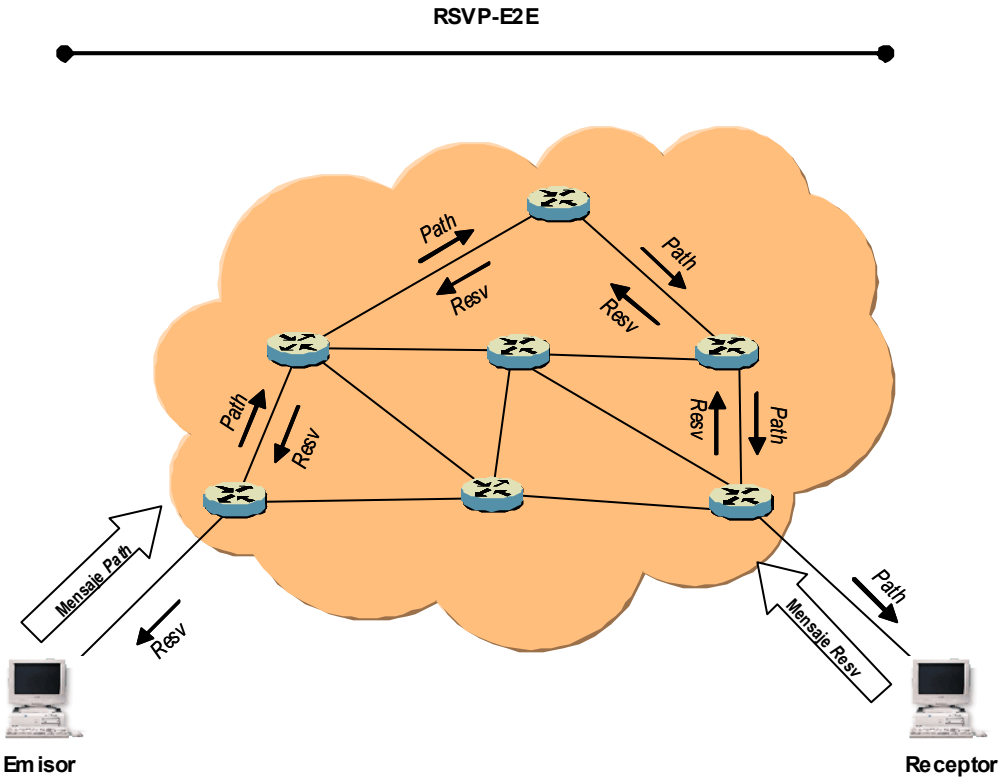

Figura 2.1: RSVP-E2E.

En la arquitectura *IntServ* que proporciona el protocolo RSVP, se definen dos clases de servicio distintas:

a) Servicio Garantizado (*Guaranteed Service*): es muy parecido a la emulación de un circuito virtual dedicado, de modo que asegura que los datos llegarán dentro de los límites de retardo previstos, así como la disponibilidad del ancho de banda especificado en los parámetros de tráfico y que los paquetes no serán descartados debido a sobrecargas en las colas de los nodos intermedios.

b) Servicio de Carga Controlada (*Controlled Load*): es el equivalente al servicio *best-effort*[iii] en condiciones de carga baja de la red, por lo que no el flujo de datos no se ve afectado por incrementos de carga en la red. Por tanto, es mejor que el servicio *best-effort*, aunque no puede asegurar ningún tipo de límite en el retardo extremo a extremo.

Como resumen del resto de características principales del protocolo RSVP-E2E, podemos enumerar las siguientes:

- Los estados en cada *router* son *soft*[iv], lo cual quiere decir que necesitan ser refrescados periódicamente por el emisor (*Path-state*) y por el receptor (*Resv-state*).
- RSVP no es un protocolo de transporte, sino un protocolo de control de red. Por si mismo, no transporta datos, pero si es cierto que trabaja en paralelo con los flujos de datos de TCP ó UDP, señalizando las reservas de recursos necesarias.
- Las aplicaciones necesitan APIs (API, *Application Programming Interface*) para especificar sus requerimientos de flujo, la inicialización de sus peticiones de reserva, y para la recepción de las notificaciones de reserva conseguida o fallada, después de la petición inicial y durante el transcurso de una sesión. Para que sean útiles, estas APIs también necesitan incluir información de errores de RSVP que describa cualquier fallo en la inicialización de la reserva o en cualquier momento en que cambien las condiciones negociadas.
- Aunque el tráfico RSVP pueda atravesar *routers* que no estén dotados de capacidad para gestionar RSVP, éstos se adaptan de forma transparente, es decir, en estos enlaces se trabajará en *best-effort*, como si la reserva a través de ellos fuera nula.

[iii] El servicio *best-effort*, es el servicio clásico de Internet, en el que no se asegura la QoS.

[iv] Para mantener un estado *soft* se necesita enviar mensajes de refresco periódicamente.

2.2.3 Mapeo entre RSVP y Ethernet

RSVP es un protocolo de Internet usado en entorno LAN para realizar reservas de recursos por flujo de datos[v]. De cualquier forma, aunque todos los dispositivos de capa 3 (capa de red) soporten RSVP, todavía no se podrá garantizar QoS en la capa 2. El grupo de trabajo ISSLL (IETF *Integrated Services over Specific Link Layers*) ha desarrollado el protocolo de señalización SBM (*Subnet Bandwidth Manager*) para el mapeo entre RSVP y las redes IEEE 802 [YAV00]. SBM define el control de admisión en entorno LAN y la gestión del ancho de banda, que combinado con el conformado de tráfico por flujo en los sistemas finales y el control de tráfico en la capa de enlace, permite una aproximación a los *Servicios Garantizados* y de C*arga Controlada* sobre redes LAN. La esencia de SBM es la introducción de una entidad de protocolo en cada segmento de capa 2, denominada DSBM (*Designated* SBM), que se coloca como nodo intermedio entre el emisor y receptor de cada segmento. Todos los mensajes RSVP pasan a través del DSBM, y éste se encarga de admitir los flujos en base a la disponibilidad de ancho de banda en la LAN. Los *hosts* que implementan SBM soportan RSVP y seleccionan una clase de tráfico para sus flujos de datos basándose en el campo *user_priority* del 802.1D.

2.2.4 Funciones de calidad de servicio en los *host*

Los *hosts* en un entorno LAN incluyen a clientes, servidores, PCs, portátiles, y otros ordenadores. En muchos casos son capaces de clasificar sus flujos de datos y utilizar un determinado tipo de servicio (*Type of Service*, ToS)[vi], y etiquetado 802.1D. Las tarjetas de red también pueden marcar las tramas y los paquetes basados en políticas predefinidas o dinámicamente distribuidas.

2.2.5 Funciones de calidad de servicio en los *routers* LAN

Uno de los *routers* LAN de interés especial es el de acceso a la red WAN (Figura 2.2). Éste debe clasificar, controlar y conformar de forma correcta todos los flujos de datos que entran en la red del proveedor de red, y comunicar las especificaciones de

[v] En entorno WAN también se utiliza como se verá en el apartado 2.3.6, pero en ese caso las reservas son por agregados de tráfico.

[vi] ToS es un campo definido para priorizar el tráfico en la arquitectura de Servicios Diferenciados (*DiffServ*), que se describirá en el apartado 2.3.2.

QoS requeridas a los *routers* WAN. Para este *router* de acceso, se recomienda el uso de colas basadas en clases (*Class-Based Queues*, CBQ)[vii], capaces de distinguir entre numerosas aplicaciones, usuarios, etc., para clasificar el tráfico con un alto nivel de granularidad en cientos de colas gestionadas mediante *software*. Cuando el tráfico está clasificado y encolado, el *router* de acceso realiza el conformado del tráfico en la entrada a la red WAN. Este conformado es muy importante para el tráfico que circula extremo a extremo, ya que un *router* de acceso procesa muchas clases de flujos, que en un *router* WAN pueden acabar agrupadas en un número relativamente pequeño de colas gestionadas mediante *hardware*.

El *router* de acceso utiliza además Servicios Diferenciados (*DiffServ*) para marcar los distintos flujos de datos que accederán a la red WAN. Estas marcas se pueden originar en los *hosts* y posteriormente conservadas o reemplazadas por el *router* de acceso. Además, el *router* puede utilizar estas marcas para señalizar a los *routers* WAN la clasificación del tráfico. En este caso, se pueden utilizar los ocho bits del campo ToS de forma que cumpla con la especificación *DiffServ* y permita pasar información adicional al *router* WAN. Esta funcionalidad se describirá en el apartado 2.4.

Un caso especial aparece cuando un *router* de acceso crea un túnel IPSec para transmitir datos de forma segura por la red WAN. En este caso, algunos identificadores de flujo originales (como las cabeceras TCP/UDP, y en modo túnel, la cabecera IP original) dejan de estar accesibles a los *routers* WAN.

Una posible solución para este caso, es la utilización de la cabecera IPSec para la clasificación de flujos en la red WAN. Uno de los estándares IPSec, ESP (*Encapsulating Security Payload*) [KEN98], utiliza en la cabecera, un campo obligatorio de cuatro octetos (Figura 2.2) denominado SPI (*Security Parameters Index*), como identificador de puerto generalizado (*Generalized Port Identifier*, GPI), para proporcionar al protocolo RSVP descripciones de la sesión cuando los números de puerto TCP/UDP están encriptados [BER97]. Por otra parte, el campo SPI no es constante durante toda la sesión, sino que cambia periódicamente cuando se

[vii] CBQ es un tipo de cola con prioridad donde se define la preferencia con la que se sirve cada cola y la cantidad de tráfico servido de cada una de ellas, en cada periodo de servicio.

recalcula la clave de sesión de un determinado flujo. Los intervalos utilizados para recalcular esta clave se fijan basándose en los niveles de tráfico, la longitud de la clave, entorno de amenazas, y el algoritmo criptográfico en uso. Estos periodos son controlados por los administradores locales, por lo que los *routers* WAN deben mantener dinámicamente actualizada esa información[viii]. Por tanto, hasta que no se resuelva este problema, sólo se puede utilizar la nueva cabecera IP que introduce IPSec para clasificar los distintos flujos de datos, lo que suele ser suficiente para definir redes privadas virtuales (*Virtual Private Networks*, VPNs), que es el servicio más común que utiliza este tipo de encapsulado.

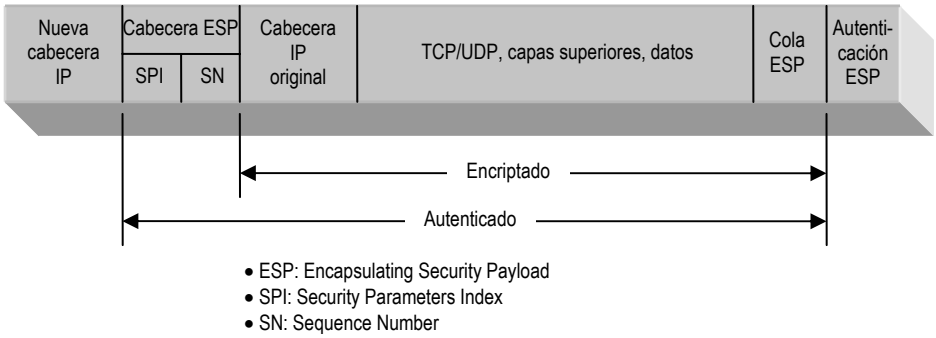

Figura 2.2: Tramado ESP en modo túnel.

2.3 Calidad de servicio en entornos WAN

Los proveedores de red implementan QoS en su red para poder identificar todo el tráfico para el que se ha negociado un servicio de QoS diferenciado basado en acuerdos de nivel de servicio (*Service Level Agreements*, SLAs). Así, se deberá asegurar que el tráfico recibe la QoS contratada, y además, el proveedor también deberá guardar información para facturar el servicio.

Los principales mecanismos para proporcionar QoS en entorno WAN son *DiffServ* y MPLS. *DiffServ* se utiliza para agregar tráfico y proporcionar un trato diferenciado a distintas clases de servicio, mientras que MPLS se usa principalmente para agregar tráfico y balancear la carga de red.

[viii] En RSVP, el valor del GPI puede actualizarse mediante los mensajes de señalización que periódicamente se generan para mantener los estados *soft*.

En las siguientes secciones se describirá cómo se implementan los mecanismos para proporcionar QoS en los *routers* WAN, y también se describirán los mecanismos *DiffServ* y MPLS.

2.3.1 Calidad de servicio en los *routers* WAN

La red WAN tiene tres tipos de *routers*: *routers* extremo (*edge routers*), *routers* troncales (*core routers*) y *routers* frontera (*border routers*).

Los *routers* extremo recogen el tráfico de los *routers* de acceso (entorno LAN) de los clientes y lo clasifican basándose en el interfaz por donde entra, direcciones IP origen y destino, protocolo IP, y puertos TCP/UDP de origen y destino. Puede utilizarse información adicional incluida en el campo *DiffServ* (ToS) o en la cabecera IPSec, tal y como se describió en el apartado 2.2.5. Cuando se clasifica un flujo, el *router* extremo realiza el control de policía y el conformado del tráfico para que pueda circular por la red WAN. Si se utiliza el protocolo MPLS[ix] [ROS01] en la red WAN, los *routers* extremos realizan la función de *routers* de etiquetado extremos (*Label Edge Routers*, LERs) y se emplean para iniciar el protocolo de distribución de etiquetas *(Label Distribution Protocol*, LDP) [THO99] y para añadir y quitar las cabeceras MPLS.

Los *routers* troncales no acostumbran a involucrarse en la clasificación del tráfico, sino que suelen leer las cabeceras IP o las etiquetas MPLS para reenviar los paquetes a velocidades muy elevadas.

Los *routers* frontera se encargan de realizar funciones de interfaz con otras redes.

Los *routers* WAN se supone que operan a velocidades mucho más altas que los *routers* en entorno LAN, ya que el tráfico que circula por ellos es mucho mayor al ser el resultado de agregar distintos flujos provenientes de diferentes entornos LAN. Así que normalmente implementan la clasificación, el encolado y la gestión del ancho de banda mediante mecanismos *hardware*. Es frecuente tener un número relativamente

[ix] Este protocolo se describe con más detalle en el apartado 2.3.3.

pequeño de colas (por ejemplo ocho) y usar WFQ[x] (*Weighted Fair Queuing*). Además, los *routers* WAN también suelen usar generalmente RED[xi] (*Random Early Detection*) como mecanismo de descarte de paquetes en las colas.

Una cantidad pequeña de colas significa que el *router* de acceso deberá gestionar un número comparativamente alto de flujos compartiendo unas pocas colas, por lo que diferentes tipos de flujos deberán ser agregados.

2.3.2 *DiffServ* y colas con prioridad

Los Servicios Diferenciados (*DiffServ*) son una arquitectura propuesta por el IETF basada en una serie de mejoras del protocolo IP que permiten una discriminación de servicio en redes IP, de forma escalable, sin la necesidad de mantener estados por flujo (como con RSVP), ni señalización en cada salto del camino. *DiffServ* proporciona nuevas interpretaciones para el octeto ToS de la cabecera IP y recomendaciones de cómo usarlo [NIC98][BLA98].

El campo *Precedence* del octeto ToS (Figura 2.3) consiste en tres bits que clasifican flujos desde 111 = Control de Red (más importante) hasta 000 = Rutina. Los siguientes campos son: D (*Delay*, retardo), T (*Throughput*), R (*Reliability*, fiabilidad) y C (Coste).

La parte DSCP (*DiffServ Code Point*) del campo *DiffServ*, consta de seis bits. Es compatible con el campo *Precedence* del ToS, pero no con los bits B/T/R/C. Todo el tráfico con el mismo DSCP se junta y se supone parte del mismo flujo. La implementación de *DiffServ* en una red define los denominados PHBs[xii] (*Per-Hop Behaviors*).

---

[x] WFQ es un mecanismo de colas con prioridad que ordena los paquetes por flujos basándose en el volumen de tráfico, intentando proporcionar tiempos de respuesta previsibles.

[xi] RED es un mecanismo de gestión de colas diseñado para prevenir el descarte de los últimos paquetes que entran a una cola llena, cuando se almacena tráfico a ráfagas. Esto puede provocar que muchas fuentes TCP reajusten sus tasas a la baja para luego aumentarla de forma simultánea, provocando oscilación en la utilización del enlace.

[xii] Una determinada manera de proceder al reenviar paquetes, aplicada en los nodos *DiffServ* a cada uno de los flujos determinados por el DSCP.

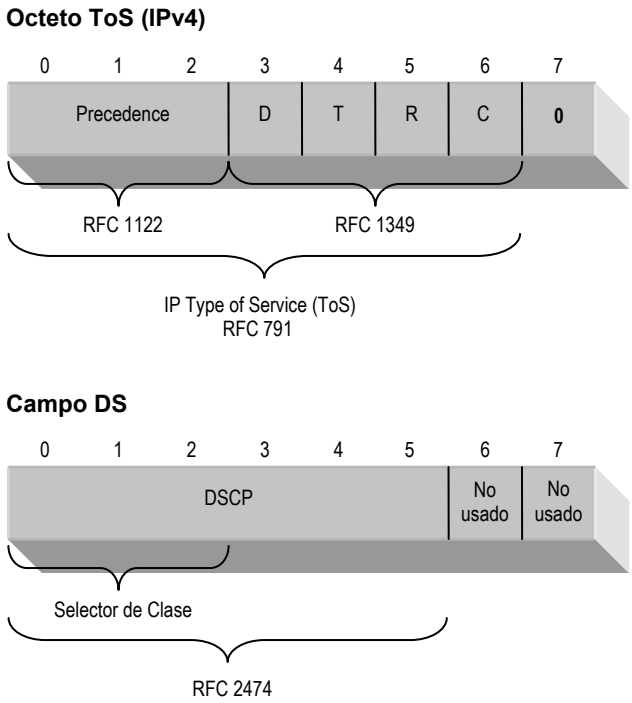

Figura 2.3: Relación entre el octeto ToS y el campo DiffServ.

Los PHBs estandarizados son EF (*Expedited Forwarding*, Reenvío Acelerado) y AF (*Assured Forwarding*, Reenvío Asegurado).

EF (DSCP = 101110) proporciona el mayor nivel de QoS al tráfico agregado. Garantiza el ancho de banda como una línea dedicada virtual, de modo que cualquier tráfico que exceda el perfil definido es descartado. Se utiliza para servicios en tiempo real con un *throughput* determinado.

AF define cuatro clases de tráfico para asignarlo a las colas de un *router* y tres prioridades de descarte de paquetes en cada clase (en total doce DSCPs), utilizadas generalmente con WRED[xiii] (*Weighted* RED) [WRED]. El tráfico excedente no se entrega con tanta probabilidad como el tráfico que entra dentro del perfil, por lo que puede ser descartado aunque no siempre.

En los *routers* WAN, los distintos agregados de tráfico se pueden asignar a colas con diferentes prioridades. El ancho de banda también puede asignarse a cada cola (de cada interfaz físico de los *routers*) basándose en medidas de tráfico, y

[xiii] WRED es un mecanismo de gestión de colas más avanzado que RED. Depende de mecanismos de clasificación para marcar paquetes con diferentes prioridades de descarte, que a su vez dependen de la ocupación media de las colas.

DiffServ puede usarse para asignar los flujos agregados a las diferentes colas. El conformado de tráfico en el *router* de acceso y en el *router* extremo permite la combinación en las colas de flujos pertenecientes a diferentes aplicaciones con características similares de QoS.

2.3.3 *Multi-Protocol Label Switching* (MPLS)

MPLS se introdujo en los entornos WAN como un mecanismo para facilitar el reenvío de paquetes permitiendo la conmutación mediante etiquetas, en lugar de utilizar el encaminamiento basado en direcciones IP, y proporcionar QoS diferenciada. Sin embargo, a medida que las nuevas tecnologías han ido evolucionando (procesado del orden de terabits en los *routers*, QoS mediante *DiffServ*, etc.), una de las aplicaciones más importantes que se ha dado a MPLS es la ingeniería de tráfico. Ésta se encarga de que el tráfico siga rutas que permitan encaminamientos alternativos, balanceo de carga y otros mecanismos para optimizar los recursos de la red. MPLS fuerza al tráfico a circular por estas rutas o LSPs (*Label Switched Paths*, caminos de conmutación de etiquetas). En la red MPLS, los *routers* se denominan LSRs (*Label Switching Routers*). Los LSRs extremos proporciona el interfaz entre la red IP externa y el LSP, mientras que los LSRs troncales proporcionan servicios de tránsito por la red MPLS. Cuando los paquetes IP entran a la red MPLS por el LSR extremo, se añaden las etiquetas MPLS. A la salida, otro LSR extremo, donde acaba el LSP, se encarga de eliminar la etiqueta y continuar con el reenvío normal del protocolo IP.

Las etiquetas MPLS son entendidas de forma local por la tabla de conmutación utilizada en los LSRs para el reenvío de paquetes. Cuando MPLS se implementa sobre redes ATM (*Asynchronous Transfer Mode*) o *Frame Relay*, algunas partes de la cabecera ATM o *Frame Relay* se usan como etiquetas MPLS externas. Todas las tecnologías de capa 2 incluyen una cabecera especial MPLS (Figura 2.4) que contiene una etiqueta, un campo experimental (Exp.), un indicador de etiquetas adicionales (S) y un campo TTL (*Time To Live*).

Figura 2.4: Cabecera MPLS.

Considerando que los LSRs no procesan las cabeceras IP, puede ser necesario transportar la información para clasificar los flujos en las cabeceras MPLS. Para realizar esto, existen dos mecanismos: E-LSP (*Experimental* LSP) y L-LSP (*Label* LSP). Con E-LSP, los 3 bits del campo experimental de la cabecera especial MPLS, se utiliza para transportar información acerca de la clase de servicio, que puede deducirse directamente del campo *Precedence* del octeto ToS o del DSCP. Por otra parte, con L-LSP, la información sobre la clase de servicio es transportada por la misma etiqueta, no siendo posible el mapeo directo como en el caso anterior.

2.3.4 Señalización en MPLS

MPLS puede implementarse de forma estática o usar un protocolo de señalización como RSVP-TE (RSVP *Tunneling Extensions*) [AWD99] o CR-LDP (*Constrained Routing Label Distribution Protocol*) [JAM02]. Cuando se utiliza un protocolo de señalización, es necesario implementar varias funciones, tales como: iniciar LSPs que atraviesen determinadas secuencias de LSRs, teniendo en cuenta el análisis de los *routers* troncales; crear estados en cada LSR para marcar el camino mediante la asignación, distribución y unión de etiquetas; reservar recursos en cada LSR, incluyendo ancho de banda, y cotas para el retardo y las pérdidas; reenrutar dinámicamente en periodos de congestión o fallos; monitorizar y mantener explícitamente los estados de las rutas LSP; proporcionar información de los LSPs a los sistemas de gestión de red; y otras funciones.

2.3.5 CR-LDP

El protocolo original para MPLS fue LDP, que soportaba LSPs basados en enrutado clásico IP (buscar el siguiente nodo del camino), pero no soportaba ingeniería de tráfico. Para soportarla, se extendió al CR-LDP. LDP intercambia información entre LSRs para concretar las etiquetas que utilizan para reenviarse el tráfico. LDP utiliza cuatro categorías de mensajes:

- Exploración: enviados periódicamente por los LSRs para anunciar su presencia.
- Sesión: para establecer, mantener y terminar una sesión entre dos entidades LDP.

- Anuncio: para crear, cambiar y eliminar mapeos de etiquetas para clases equivalentes de reenvío (FECs, *Forwarding Equivalence Classes*) mientras dura una sesión.
- Notificación: para señalizar y proporcionar información opcional.

CR-LDP señaliza la reserva de recursos en el trayecto de reenvío de paquetes, al contrario que RSVP que lo hace en el sentido opuesto. Además, a diferencia de RSVP, establece estados *hard* que no requieren refresco.

2.3.6 RSVP-TE

RSVP-TE se desarrolló como una extensión de RSVP-E2E al mismo tiempo que se definía CR-LDP. A continuación se describen las diferencias más destacables entre RSVP-E2E y RSVP-TE.

- RSVP-E2E proporciona señalización entre pares de *hosts*, mientras que RSVP-TE señaliza entre *routers*.
- RSVP-E2E se aplica para flujos individuales entre *hosts*. RSVP-TE crea un estado para un agregado de flujos entre los puntos de entrada y salida de un conjunto de tráfico. Un LSP agrega múltiples flujos entre *hosts* para reducir la cantidad de estados RSVP en la red.
- RSVP-E2E utiliza protocolos de enrutado que operan basándose en la dirección de destino de forma estática. RSVP-TE utiliza el enrutado de los *routers* troncales.
- RSVP-E2E utiliza un modelo de estados *soft*, que puede causar problemas de escalabilidad, retardo y *overhead*. RSVP-TE utiliza estados donde los mensajes *Path* y *Resv* se refrescan periódicamente, pero reduciendo considerablemente su volumen, usando el refresco de el estado de agregados y de forma incremental.

2.4 Calidad de servicio entre entornos LAN y WAN

Los mecanismos de QoS descritos en los apartados anteriores, se refieren a entornos LAN y a entornos WAN. No obstante, la implementación de QoS extremo a extremo requiere, además, que todas las partes de la red operen y colaboren conjuntamente para proporcionar QoS (Figura 2.5).

La operación entre flujos de datos consiste en el intercambio de información que permite la adecuada clasificación de dichos flujos.

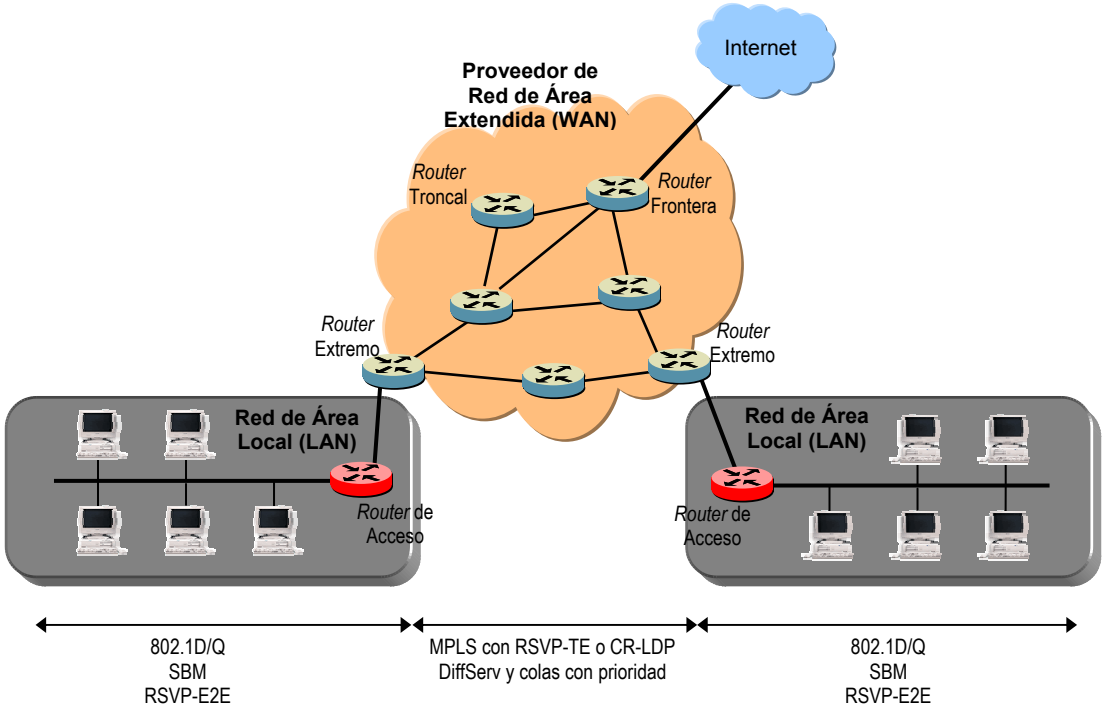

Figura 2.5: QoS extremo a extremo con distintos protocolos.

La operación de intercambio de señalización entre el entorno LAN y el WAN, incluye la señalización IP clásica y otros tipos de señalización específica, como la utilizada en VoIP (*Voice over IP*) para señalizar el inicio de sesión. También se debe incluir información acerca de los valores ToS/DSCP utilizados entre el *router* de acceso y el extremo, e información de cambios del campo SPI en IPSec. En un caso más general, será necesario permitir la operación entre diferentes arquitecturas IP. Por ejemplo, en un entorno LAN puede usarse la arquitectura *IntServ* con RSVP-E2E como protocolo de señalización, mientras que la red WAN puede utilizar una arquitectura *DiffServ*. Para preservar la QoS en estos tipos comunes de arquitectura, el grupo ISSLL ha desarrollado el estándar *IntServ* sobre *DiffServ* [BER00]. En este escenario, el *router* de acceso o el extremo realizan decisiones de admisión de peticiones de reserva mediante RSVP, basándose en el estado de la red *DiffServ*. El nodo que realiza la decisión debe poseer algunas características críticas como el soporte de RSVP y *DiffServ*, conocimiento de las especificaciones de nivel de servicio u otro tipo de limitaciones impuestas a un determinado tipo de tráfico, y también la habilidad de poder realizar el mapeo necesario entre RSVP y *DiffServ*. Con esto, los *routers* pueden procesar los mensajes RSVP y admitir o rechazar nuevos flujos.

La necesidad de regular la QoS mediante mecanismos de control de policía, hace que deba existir una buena coordinación entre el entorno LAN y los servidores de control de policía del proveedor de red WAN. El intercambio de información para realizar estas tareas incluye áreas de marcado de paquetes, decisiones de admisión de flujos, usuarios prioritarios, enrutado explícito, servicios de valor añadido mediante petición, reacción a la congestión de la red, etc.

2.5 Conclusiones

En este capítulo se han mostrado los diferentes mecanismos que permiten implementar una red IP con QoS extremo a extremo. Tal y como se ha descrito, es fundamental, proveer de QoS tanto al entorno LAN como al WAN, así como permitir que ambos puedan operar entre ellos.

En el entorno LAN se utilizan las nuevas versiones Ethernet que permiten proporcionar QoS entre dos nodos locales. Por otra parte, se plantea el uso del protocolo RSVP-E2E para señalizar la QoS necesaria. Para que estos protocolos puedan interactuar, se necesita realizar el mapeo entre ellos mediante SBM. Además, los *hosts* tienen capacidad para utilizar los protocolos y mecanismos adecuados para realizar peticiones de reserva de recursos. Asimismo, para acceder a la red WAN, se utiliza un *router* de acceso que se encarga de gestionar el tráfico y negociar con la red WAN la QoS necesaria.

En el entorno WAN, se aplica QoS en los *routers* extremo, realizando agregación de tráfico para reducir la necesidad de gestión de la QoS. Por su parte, los *routers* troncales suelen dedicarse a la conmutación a altas velocidades de los paquetes. Para facilitar esta conmutación se utiliza el protocolo MPLS, basado en la conmutación mediante etiquetas, que a su vez permite realizar ingeniería de tráfico para reorganizar las rutas dinámicamente.

Tal y como se ha visto, es importante no olvidar la necesidad de poder interrelacionar las diversas arquitecturas utilizadas en entorno LAN y WAN, y de permitir el correcto intercambio de información entre ellas para poder disponer de una red IP con QoS extremo a extremo.

Capítulo 3
Método de Minimización del Coste

3.1 Introducción

El rápido crecimiento de Internet en los últimos años se puede observar desde varios puntos de vista, aunque los aspectos más espectaculares se dan en cuanto a tamaño, velocidad y la aparición de nuevas aplicaciones y servicios en esta nueva red. Por tanto, cabe prestar una especial atención a dichas aplicaciones y servicios que utilizarán la red para orientar el diseño y la gestión de ésta a sus necesidades.

Las nuevas aplicaciones y servicios que se pretenden ofrecer en esta nueva red tienen requisitos muy concretos en cuanto al servicio que la red debe ofrecerles. Así, la red deberá disponer de suficientes recursos para que el tráfico generado por estas aplicaciones los utilice de forma adecuada. Por otra parte, como se ha visto en el Capítulo 2, son necesarios diversos mecanismos y protocolos para permitir a las redes IP la asignación de sus propios recursos a los flujos de datos que circulan por la red. Esto permite la provisión de niveles de servicio concretos que se negocian entre el usuario y la red mediante un determinado acuerdo de nivel de servicio (*Service Level Agreement*, SLA). De esta forma existe la posibilidad de diferenciar los distintos niveles de servicio que ofrece la red a las aplicaciones. Esta nueva posibilidad hace que sea vital el uso responsable por parte de las aplicaciones de esta diferenciación de servicios. Así, no será eficiente que una aplicación demande a la red un nivel de servicio de alta calidad si realmente no lo precisa, ya que ello implica

obtener y utilizar recursos de la red que pueden ser necesarios para otras aplicaciones.

Desde el punto de vista del reparto eficiente de recursos parece conveniente inducir a las aplicaciones hacia un uso apropiado de éstos. Para ello los mecanismos de provisión de calidad de servicio (*Quality of Service*, QoS,) se encargan de controlar, en la medida de lo posible, que no haya violaciones de los contratos establecidos con la red. Esto no es suficiente, ya que no violar el contrato es fácil si se reservan más recursos de los necesarios. Para evitar esta situación, se requiere un sistema de tarificación que permita facturar el menor precio posible a aquellos usuarios que utilicen de forma responsable los recursos de la red, y penalice a los que no lo hagan. De esta forma el usuario podrá hacer un uso incorrecto de la red pero a cambio, deberá pagar más por usarla.

Por tanto, es importante conocer los requisitos que tienen los distintos flujos de datos que circulan por la red, con respecto a la reserva de recursos que sería conveniente. Como se verá en este capítulo, hay flujos de datos con requisitos temporales muy estrictos (inelásticos), mientras que en el lado opuesto están los que no necesitan un nivel de servicio especial y pueden continuar usando el servicio clásico de *best-effort* (elásticos). Además, existen otros para los que la necesidad de reservar recursos depende del estado de la red. Para este tipo de flujos (semi-elásticos) se observará que es posible minimizar el coste de la transmisión si se reservan sólo los recursos imprescindibles. Para ello, se propondrá un sistema cliente-servidor capaz de realizar la transmisión del flujo de datos semi-elástico de forma que el coste asociado sea óptimo.

Este sistema servidor de flujos semi-elásticos, se encargará de proporcionar, como mínimo, la cantidad de información que el cliente necesita en cada momento. Por otro lado, el cliente controlará de forma adecuada la memoria donde se almacenarán los datos que llegan de la red. Analizando la ocupación de esta memoria, el cliente será capaz de minimizar el uso de la reserva de recursos. Para ello, si la memoria contiene poca información reservará recursos de red para evitar que se vacíe completamente, mientras que si la memoria llega a una determinada ocupación que garantice la disponibilidad de datos se podrá utilizar, mientras se cumpla esa condición, el servicio de *best-effort*. De forma tal que el coste se minimizará si se maximiza el uso del servicio *best-effort*. Así, se deberá garantizar un determinado criterio de minimización que como se comprobará más adelante,

pretende que al final de la transmisión la memoria del cliente quede lo más vacía posible.

3.2 Clasificación de Flujos de Datos en Internet

En Internet, cada flujo de datos tiene sus propios requerimientos de QoS. Las aplicaciones son las que fijan estos requisitos, basándose en la calidad que el usuario espera recibir. Por defecto, el usuario siempre espera recibir la máxima calidad posible, es decir, idealmente, el usuario desea que la red no afecte en absoluto a su consumo de información. Ante la ausencia de este caso ideal, el usuario se adapta a la tecnología disponible y soporta cierta degradación de esa calidad. Esta degradación se traduce fundamentalmente en pérdida de información, retardo y variabilidad de este retardo.

Según el punto de vista que se utilice, es posible clasificar los flujos de datos que circulan por Internet de varias formas. No es lo mismo verlos desde el punto de vista de la aplicación, que de la red. Así, en la red puede haber mucha pérdida de información, pero desde la aplicación no apreciarse de la misma forma o con la misma magnitud, debido a protocolos que garanticen la fiabilidad. La información puede sufrir mucho retardo en la transmisión, y sin embargo la aplicación ni siquiera percibirlo si no necesita consultarla hasta después de haberla recibido toda (no se percibe la transmisión). Y exactamente lo mismo ocurre con la variabilidad del retardo si la aplicación permite que haya mecanismos de sincronismo y reordenación de la información.

Indudablemente, la calidad de servicio no tiene sentido sin un usuario final, por lo que a continuación se planteará la clasificación de los flujos de datos desde el punto de vista del usuario, o lo que viene a ser lo mismo, de la aplicación.

La clasificación se basará en la necesidad de las aplicaciones de utilizar un nivel de servicio mejor que el servicio clásico de *best-effort*. Por otra parte, se analizará lo estrictos que son los requisitos de la aplicación en cuanto a pérdidas y retardo. La variabilidad del retardo no se tendrá en cuenta, ya que en el nivel de aplicación, ésta se puede traducir fácilmente en pérdidas o retardo.

3.2.1 Flujos Elásticos

Se considera que un flujo es elástico [ROB98] [STA99] cuando no tiene requerimientos estrictos en cuanto a pérdidas o retardo, y por lo tanto es suficiente el uso del servicio *best-effort*.

Generalmente, las aplicaciones no suelen aceptar que los datos que reciben lleguen con pérdidas y retardo, por lo que utilizan protocolos y mecanismos para reducir en la medida de lo posible uno de los dos casos o ambos. En la Internet actual, en la que la que se utiliza el servicio *best-effort*, se encuentran habitualmente dos grupos de aplicaciones de este tipo: las tolerantes al retardo (pero estrictas en cuanto a pérdidas) y las tolerantes a pérdidas (pero estrictas en cuanto a retardo). Las primeras utilizan protocolos fiables, tales como TCP (*Transmission Control Protocol*), que garantizan la integridad semántica de la información mediante la retransmisión de la información que se pierde. Este es el caso, por ejemplo, del servicio de correo electrónico o del servicio FTP (*File Transfer Protocol*). Las segundas evitan las retransmisiones que ralentizan la transmisión, por lo que utilizan protocolos como UDP (*User Datagram Protocol*) y RTP (*Real-time Transport Protocol*). Además, pueden utilizar técnicas adaptativas en las que se controla la cantidad de información que se envía en función del estado de la red, con tal de evitar las pérdidas en la medida de lo posible. Este el caso de las aplicaciones de tipo *streaming* [ROB98].

> En una Internet con garantías de QoS, las aplicaciones con requerimientos temporales estrictos harán uso de esas garantías para asegurar que la información llega a tiempo, aún permitiendo ciertas pérdidas en los datos que se envían. En este caso, los flujos no se considerarán elásticos, sino que como se verá más adelante, se podrán clasificar como inelásticos o semi-elásticos, ya que el servicio *best-effort* es insuficiente.

En este escenario, quedarán por tanto como flujos elásticos las aplicaciones tolerantes al retardo (tales como el correo electrónico o el FTP citados anteriormente). En la Figura 3.1, se puede observar la relación que existe en este caso entre el instante en que se requiere la información y el instante en que la información llega a su destino. Como se aprecia en la figura, son flujos totalmente tolerantes al retardo, ya que la información no se precisa en un instante concreto de tiempo.

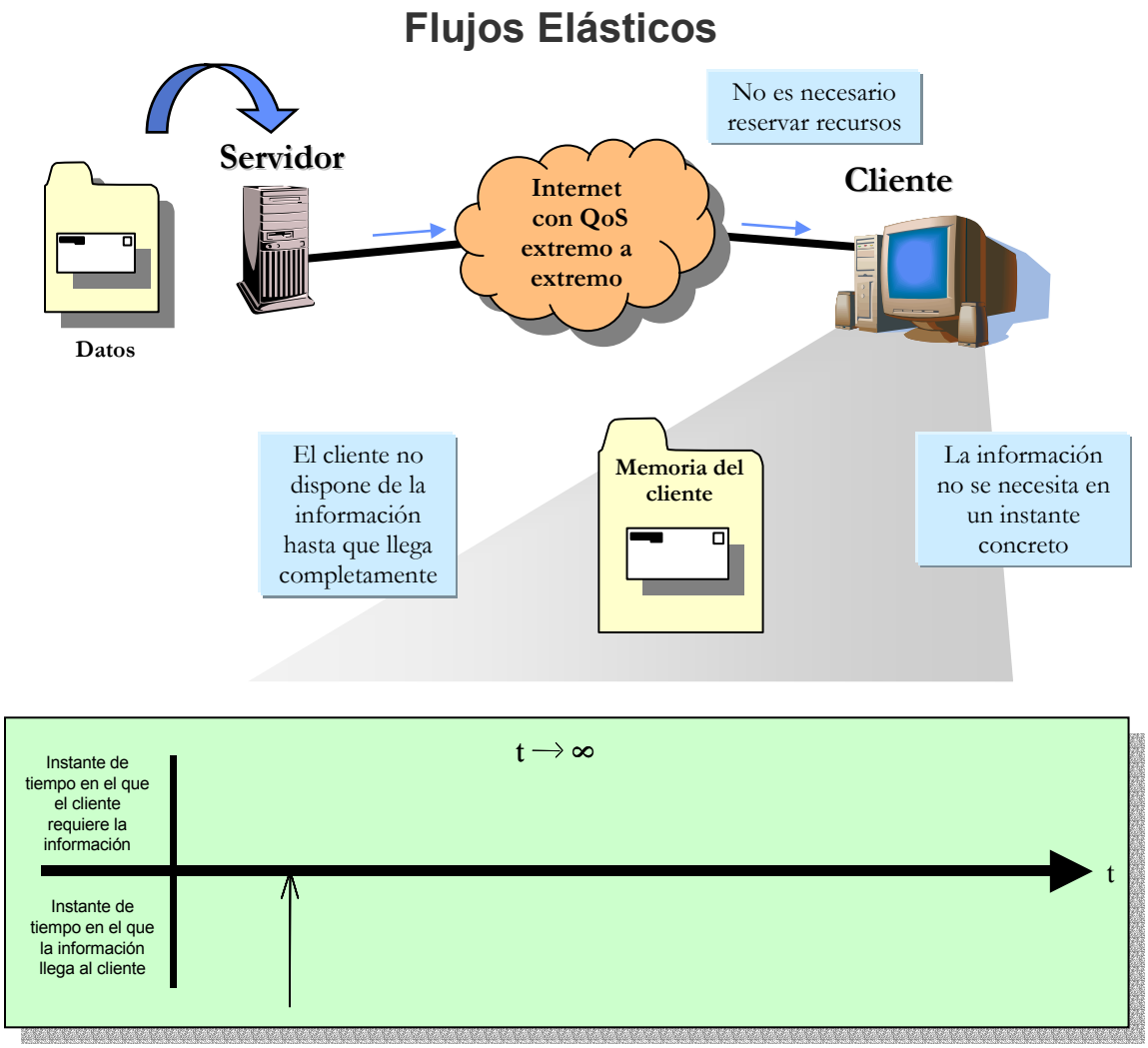

Figura 3.1: Flujos elásticos.

A continuación se examinarán los flujos con requisitos temporales estrictos que se caracterizan por necesitar un nivel de servicio superior al de *best-effort*.

3.2.2 Flujos Inelásticos

Los flujos con requerimientos temporales de QoS muy estrictos se definen como flujos inelásticos. En este caso, la información suele generarse en directo y es esencial la reserva de recursos de la red durante toda la transmisión para poder asegurar su entrega sin degradar la QoS requerida (Figura 3.2). Por otro lado, la información debe llegar al destino en un instante de tiempo específico (próximo a la generación) para poder ser consumida. Un ejemplo de aplicaciones que utilizan este tipo de flujo es la transmisión de vídeo en directo.

Lógicamente, es posible que la aplicación tenga algún grado de tolerancia a pérdidas de información y que por tanto, el nivel de QoS requerido sea algo menor.

De todas formas, a medida que la oferta de QoS mejora, la tendencia de cualquier usuario es a requerir el nivel máximo de QoS posible por lo que las redes deben orientarse a servir a aplicaciones con requerimientos temporales y semánticos fuertes.

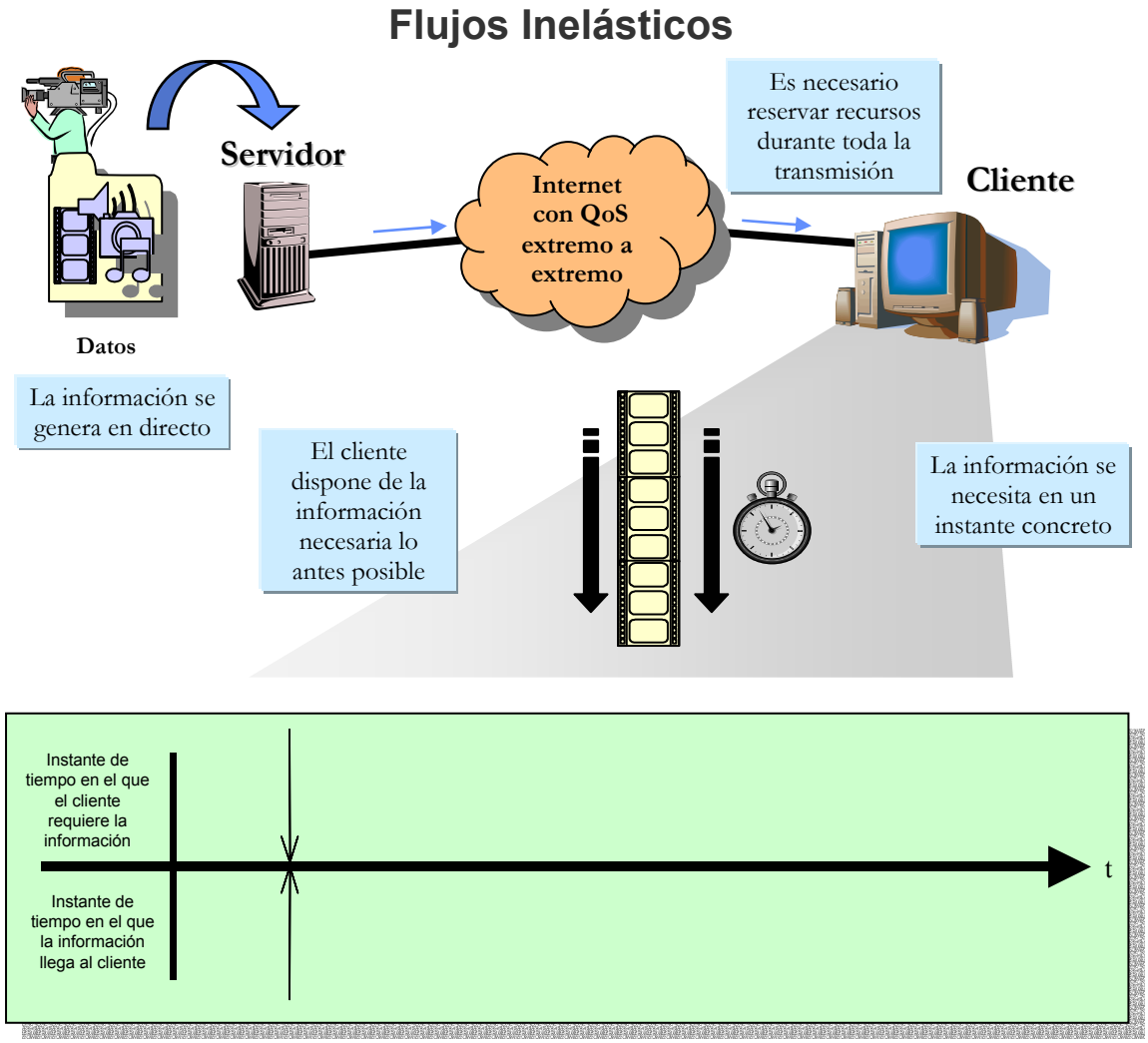

Figura 3.2: Flujos inelásticos.

3.2.3 Flujos Semi-elásticos

Finalmente, se puede definir otro tipo de flujo: el semi-elástico [POS00]. Este flujo requiere un cierto nivel de QoS, pero a menudo no es necesario mantenerlo durante toda la transmisión. En este contexto, la información necesita llegar al cliente en un instante muy específico, como pasa con los flujos inelásticos, pero la red puede entregarla antes (Figura 3.3). Esto es lo que sucede con la información prealmacenada, como es el caso de los servidores denominados *continuous media servers* [SHA00]. Estos flujos, necesitan que la información se almacene en el

extremo receptor en una memoria y se consume a una tasa de lectura concreta, por lo que lo único que hay que garantizar es la disponibilidad de suficiente información en esta memoria durante todo el tiempo.

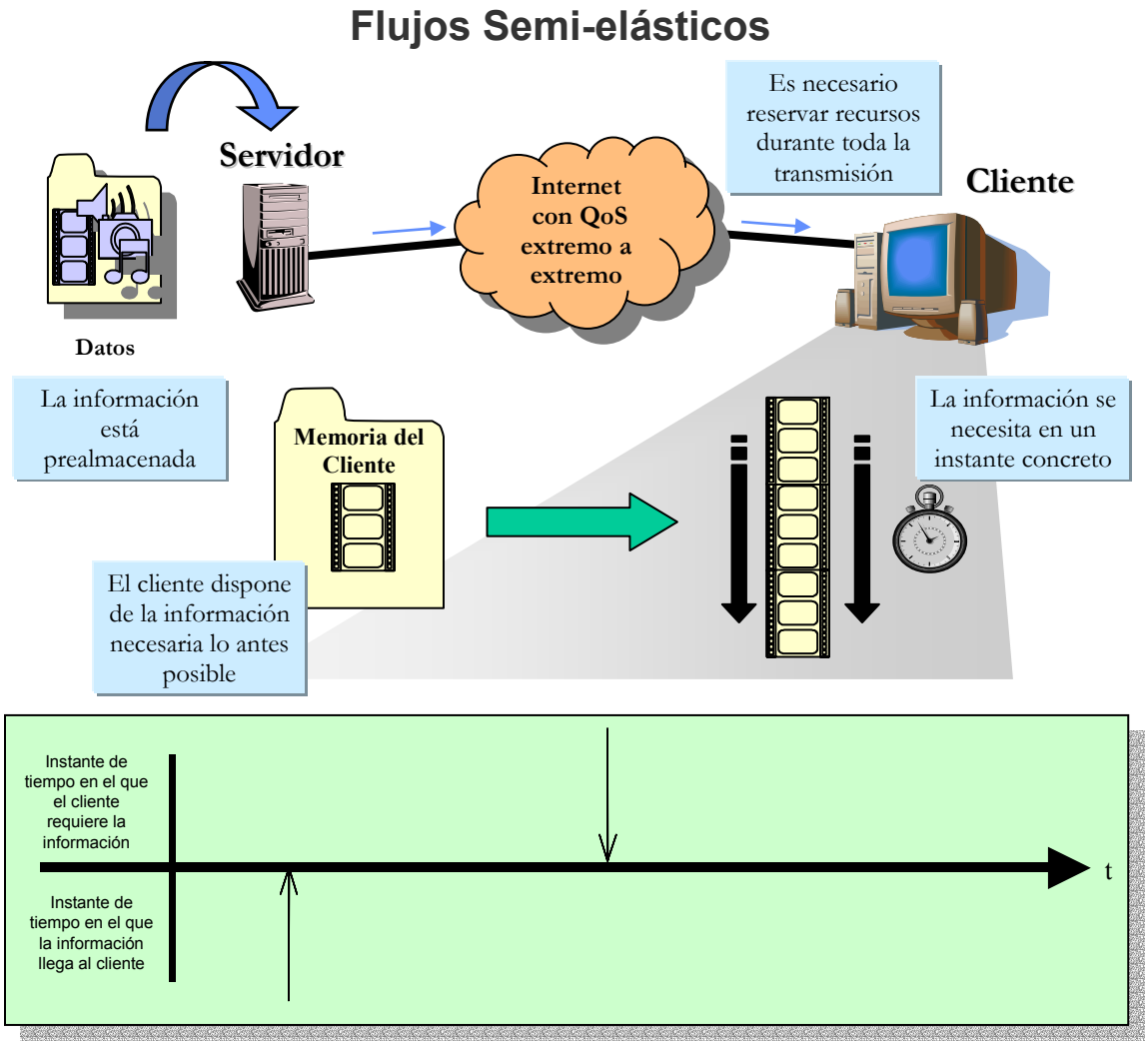

Figura 3.3: Flujos semi-elásticos.

Es importante resaltar, que la diferencia entre los flujos inelásticos y semi-elásticos radica en la naturaleza del origen de la información (en directo o prealmacenada), y por tanto, en la necesidad de reservar recursos de la red durante toda o parte de la transmisión. Esta cantidad de recursos a reservar para los flujos semi-elásticos dependerá del estado de la red, es decir, de la tasa con que llegue la información utilizando el servicio *best-effort*, ya que una tasa más alta supondrá una mayor disponibilidad de información.

En cuanto a la tolerancia a pérdidas de este tipo de flujos, cabe destacar que como siempre pueden tener una cierta tolerancia, pero en este caso y debido a la

posibilidad de entregar la información antes de que se necesite será habitual el uso de técnicas para garantizar la fiabilidad y evitar al máximo la pérdida de información.

3.3 Métodos de Minimización del Coste de la Transmisión de Flujos Semi-Elásticos

Como se ha visto, los flujos elásticos no requieren reserva de recursos, los inelásticos deben mantener la reserva de recursos durante toda la transmisión y finalmente, los flujos semi-elásticos son los que pueden variar el nivel de servicio que requieren en cada momento (en función de la disponibilidad de información en la memoria del cliente), ya que ello dependerá del estado de la red. El objetivo de este apartado es proponer un modelo de coste a utilizar, basándonos en las ideas propuestas en la literatura actual, orientadas al uso responsable de las reservas de recursos de red por parte de las aplicaciones. Finalmente, se definirá la eficiencia de los métodos de minimización para cuantificar el grado de reducción del coste que se obtiene al utilizar uno de estos métodos. En el apartado 3.4, se presentará un mecanismo basado en un sistema cliente-servidor, para conseguir renegociar el nivel de servicio en los instantes adecuados con el objetivo de reducir o minimizar el coste de la transmisión de estos flujos.

3.3.1 Definición de Coste

Para caracterizar el coste o el precio de una comunicación es necesario determinar tanto quién paga como por qué servicio paga. De este modo, se pueden distinguir tres tipos de entidades (cliente de servicio, proveedor de servicio y operador de red) que a su vez proporcionan o reciben un determinado servicio (Figura 3.4).

El cliente de servicio es aquel usuario que pretende acceder a cualquier tipo de información (audio, vídeo, datos, etc.) a través de la red. Esta información está localizada en un determinado lugar (servidor) y puede ser gratuita o de pago. Actualmente, existen en Internet los dos tipos de información, e indudablemente para que la información pueda ser de pago ha de ser lo suficientemente atractiva para el cliente del servicio. Cabe destacar que el cliente sólo paga por lo atractivo de la información que recibe y que en ningún caso valora los esfuerzos que pueda realizar el proveedor de servicios o el proveedor de red para que la transmisión de la

información sea la apropiada. Es más, el cliente valorará negativamente al proveedor de servicio o al operador de red en el caso de que esa transmisión no sea adecuada y por tanto, las expectativas de obtener una determinada información, con una determinada calidad (subjetiva), no se cumplan. También cabe destacar que en el esquema de la Figura 3.4 no se tiene en cuenta el coste de la conectividad a la red por parte del cliente ya que sólo se valora el coste de la transferencia de información desde el proveedor de servicio al cliente.

Si bien es importante la existencia de una buena red, no menos importante es tener buenos proveedores de servicios suficientemente atractivos para los clientes como para pagar por ellos. El proveedor de servicio posee información que desea el cliente. Por tanto, si la información es de pago, el cliente le pagará por ella. Con esa cantidad, debe conseguir beneficios y además pagar al operador de red para utilizar sus recursos de manera que las expectativas de calidad del cliente se vean cumplidas.

Finalmente, el operador de red al que accede el proveedor de servicio debe negociar con otros operadores de red para conseguir la conectividad adecuada hacia el cliente.

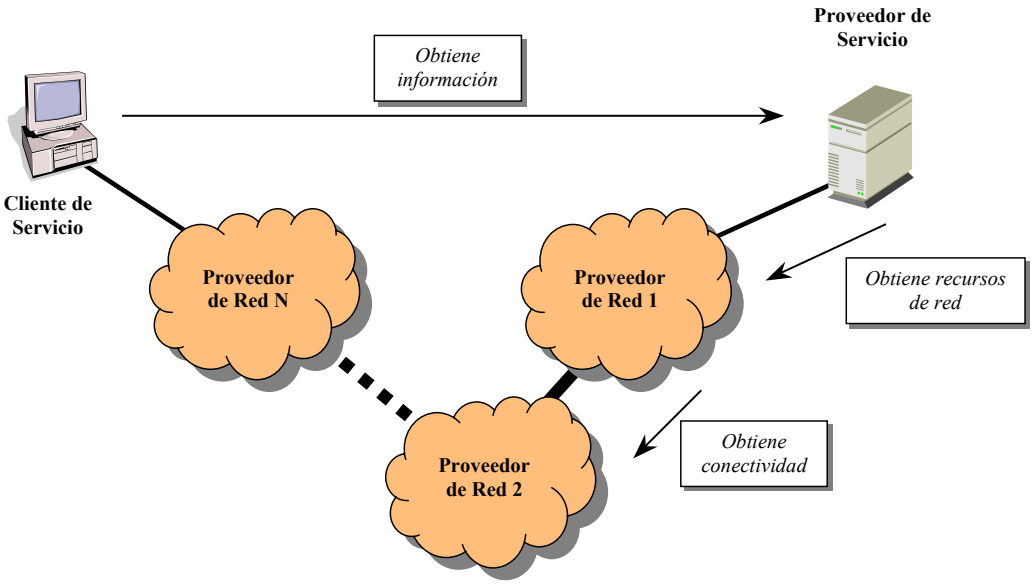

Figura 3.4: Entidades que proporcionan o reciben servicios.

Como ejemplo, supóngase cualquier tienda actual en Internet, donde se compran bienes reales (billetes de avión, libros, etc.). En este caso el proveedor del servicio es la propia empresa, que tiene, por ejemplo una página Web donde se

proporciona a los clientes del servicio la información necesaria para realizar la transacción (características de los productos, condiciones de la venta, etc.). Por otro lado, está el proveedor de red que en la red actual cobra únicamente por proporcionar transferencia de información y conectividad. Así, el precio de conectividad de un usuario normal no suele ser muy alto, mientras que el de una empresa grande en Internet es mayor porque se les puede ofrecer mejor conectividad para poder dar servicio a volúmenes grandes de tráfico.

A partir de este momento, el estudio se centrará en la definición del coste desde el punto de vista del proveedor de servicio, en cuanto a la obtención de recursos de red que le proporciona un operador de red con garantías de QoS extremo a extremo. Éste es quizás uno de los costes que más repercute en la capacidad de negocio de esta arquitectura, ya que incide directamente en los beneficios del proveedor de servicio. Actualmente, en el ámbito técnico, se están proponiendo y discutiendo los mecanismos que deben utilizar los operadores de red para tarificar los servicios que ofrecen tanto a otros operadores de red, como a los clientes a los que proporcionan accesibilidad [BUR01] [ROB98].

Fundamentalmente, se pueden distinguir tres tipos de filosofías de sistema de tarificación (Tabla 3.1) [ROB98]: tarifa plana, tarifa por congestión [MAC95] [SON97] y tarifa por transacción.

Con la tarifa plana, los usuarios pagan una tasa fija independientemente de la cantidad de tráfico que generen. Generalmente, esta tasa depende de la capacidad de conectividad a la red, es decir, de la tasa de acceso a dicha red. Este tipo de tarifa es la más simple de todas ya que no implica ningún tipo de control por parte del operador de red. Por otra parte, también es la menos justa, ya que se paga lo mismo independientemente del tráfico que se inyecte a la red. Además incita a la congestión, ya que un usuario que usa poco la red puede muy fácilmente sentirse libre de utilizarla en un mayor grado, ya que no hay ninguna restricción a parte de la velocidad del acceso.

La tarifa por congestión se basa en la de tarifa plana, pero además añade un coste que depende del grado de calidad de servicio que se requiere. Así, si un usuario requiere, en caso de congestión, que su transmisión tenga unos determinados niveles de calidad de servicio, deberá pagar por ello. En caso de que no haya congestión se paga por la tarifa plana. Es fácil deducir que este tipo de tarificación tiene el

problema de que el precio de la conexión depende de lo saturada que esté la red. Por tanto, el usuario debe confiar en que el operador de red hace todo lo posible por reducir la congestión.

Finalmente, la tarifa por transacción se basa en tarificar el volumen de información que se inyecta a la red y el nivel de servicio que se asocia [SON97]. Este tipo de tarificación permite que el tráfico que se envía mediante un servicio *best-effort* sea más barato que el que se envía haciendo una reserva de recursos. Por otro lado, se tarifica en función de la información que se quiere transmitir por la red. Esto implica que el operador de red debe asumir el coste de la transmisión independientemente del estado de congestión. De esta forma, el usuario paga por transmitir un determinado volumen de transmisión con una cierta calidad de servicio y es el operador de red el que debe encargarse de proveer un buen servicio (suficiente ancho de banda, pérdidas y retardo controlados) para maximizar sus beneficios.

Para tarificar correctamente la información transmitida, se debe definir una determinada función de coste. Esta función de coste se divide en cuatro términos (Tabla 3.2) orientados a conseguir un uso responsable de las reservas de recursos [WAN00]: carga por uso, carga por desaprovechamiento del servicio, carga por congestión y carga por señalización.

Tipo de tarifa	Descripción
Tarifa plana	Tasa fija e independiente de la cantidad de tráfico generado. Depende de la velocidad del acceso a la red.
Tarifa por congestión	Basada en la tarifa plana, pero añadiendo un coste dependiente de la calidad de servicio requerida en caso de congestión.
Tarifa por transacción	Tarifica el volumen de datos que se introduce en la red en función del nivel de servicio requerido por el cliente.

Tabla 3.1: Sistemas genéricos de tarificación.

La carga por uso (C_u) depende del volumen de información que usa la red (V_u *bytes*) y del nivel de calidad de servicio y se mide en unidades de tarificación (*ut.*). Así, para un determinado contrato de nivel de servicio (SLA) es posible obtener una

tarifa determinada (p_u, *ut./byte*) para la transmisión de un volumen de V_u *bytes* de información.

$$C_u = p_u \cdot V_u \tag{3.1}$$

Fuera de nuestro ámbito de estudio está la caracterización de p_u en función de los diversos parámetros que intervienen en la definición del nivel de servicio requerido por el usuario [REI01] [STI01]. Para el estudio del coste de la transmisión de flujos semi-elásticos se simplificará C_u. En este tipo de flujos, se estudiará el coste debido a la reserva de recursos (ReR) dando por supuesto que el precio mínimo es el relacionado con la transmisión de toda la información mediante el servicio convencional de *best-effort* (BE). Así C_u quedará como en 0 si se normaliza p_u, es decir si se supone como tarifa un valor de 1. Para el caso de flujos semi-elásticos, donde la información se envía a la misma tasa que se reserva para aprovechar al máximo dicha reserva, el coste es el expresado en (3.2) donde T_{ReR} tiempo en que se reservan recursos) dependerá de la tasa α_{ReR} (tasa de reserva de recursos en *bits/s*).

$$C_u = V_u = V_{ReR} = \alpha_{ReR} T_{ReR} \tag{3.2}$$

La carga por desaprovechamiento del servicio (C_d) (3.3), donde se desaprovecha un ancho de banda suficiente para enviar V_d *bytes*, se justifica si se tiene en cuenta que los recursos que reserva un flujo y que no se utilizan se pueden reaprovechar para otra conexión a la que se le ofrezca un menor nivel de servicio. Por ejemplo, si una aplicación reserva un determinado ancho de banda pero utiliza sólo la mitad, la otra mitad se puede permitir que la utilice otra conexión mediante el servicio de *best-effort* (ya que si la aplicación que tiene hecha la reserva lo requiere, el ancho de banda será para ella). Sin esta carga adicional, el proveedor de red podría ver reducidos sus beneficios, en el caso de que se realizara un mal aprovechamiento de los recursos reservados.

$$C_d = p_d \cdot \left(p_{u_1} - p_{u_0}\right) \cdot V_d \tag{3.3}$$

Es destacable que si se reservan recursos que permiten transmitir un volumen de información $V = V_u + V_d$, pero sólo se usa V_u (V_d no se desaprovecha) el proveedor de red puede recibir lo mismo que si cobrara la tasa más cara (p_{u_1}) y se transmitiera un volumen V, si vende V_d con un precio p_{u_0}. El parámetro p_d permite escalar el coste C_d de forma adecuada. Como se observa, esta parte de la función de coste tiene

como propósito el correcto aprovechamiento de las reservas, de esta manera se evita que se reserven recursos de red de forma abusiva.

La carga por congestión (C_c) (3.4) tiene como objetivo evitar que un flujo de datos no cumpla el contrato adquirido con el proveedor de red, sobrepasándolo y causando congestión no prevista en la red. La tarifa p_c que se aplicará a la información que cause congestión (V_c) puede depender del estado de la red o ser estática [WAN00].

$$C_c = p_c \cdot V_c \tag{3.4}$$

Finalmente, la carga por señalización (C_s) (3.5) permite controlar la cantidad de señalización que una aplicación utiliza (V_s) para negociar varios niveles de calidad de servicio en una misma transmisión. Como se ha comentado anteriormente, los flujos semi-elásticos requieren reservar recursos si la memoria del cliente no está suficientemente llena, mientras que pueden utilizar un servicio *best-effort* si ésta tiene datos almacenados suficientes. Parece lógico que si el proveedor de red da permiso a una aplicación para renegociar dinámicamente el grado de servicio requerido, exista una tarifa (p_s) que penalice el uso excesivo de esta señalización.

$$C_s = p_s \cdot V_s \tag{3.5}$$

Términos	Descripción
Carga por uso (C_u)	Penaliza el uso de los recursos de la red en función del volumen de información a enviar.
Carga por desaprovechamiento del servicio (C_d)	Penaliza el desaprovechamiento de las reservas de recursos de red, ya que aunque el proveedor de red puede reaprovechar los recursos no usados, lo hace a un menor precio (servicio *best-effort*).
Carga por congestión (C_c)	Penaliza el incumplimiento del contrato por parte del cliente causando congestión.
Carga por señalización (C_s)	Penaliza el uso de señalización debida a cambios en las reservas de recursos solicitados de forma dinámica por el cliente.

Tabla 3.2: Función de Coste.

Por tanto, para una determinada transacción el coste de ésta se podrá expresar como la suma de los costes anteriores.

$$C = C_u + C_d + C_c + C_s \tag{3.6}$$

Para el caso de los flujos semi-elásticos se puede simplificar la expresión (3.6) realizando algunas suposiciones. En primer lugar se supone que C_d es nulo ya que el proveedor de servicio (servidor) utilizará todos los recursos reservados para enviar lo más rápidamente posible la información hacia el cliente. Por otro lado, se el servidor no ocupa más recursos de los reservados (disponibles para la transmisión) y por tanto, C_c será nulo. Finalmente, la expresión del coste quedará como en (3.7) suponiendo $p_u = p_s = 1$ *ut./byte*. Esta suposición se realiza para normalizar las tarifas, dando la misma importancia a la señalización que a la información que quiere enviar el usuario.

$$C = C_u + C_s = p_u \cdot V_u + p_s \cdot V_s = V_u + V_s = V_{ReR} + V_s \tag{3.7}$$

3.3.2 Eficiencia del Método de Minimización

Los flujos semi-elásticos se caracterizan por no necesitar una reserva de recursos (ReR) durante toda la transmisión, pudiendo realizar parte de ella mediante el servicio *best-effort*. En los siguientes apartados se verá cómo gracias a esta característica es posible reducir el coste de la transmisión utilizando una cierta metodología.

Parece interesante, por tanto, medir la reducción del coste que se obtendrá con estos métodos respecto al coste de transmitir toda la información con reserva de recursos, como en el caso de los flujos inelásticos. Para determinar esta reducción se define la eficiencia del método η para una transmisión semi-elástica de *V bytes* como la reducción del coste conseguida con el método respecto del coste máximo que se obtiene reservando recursos durante toda la duración de la comunicación. En (3.8) se muestra la expresión resultante, de donde se puede observar que para que el método sea útil ($\eta > 0$) es necesario que se cumpla que $V_s < V - V_{ReR}$. Así una eficiencia igual a 1 indica que el coste ha sido 0, mientras que una eficiencia igual a 0 implica que todos los datos se han enviado usando el modo ReR.

$$\eta = 1 - \frac{C}{C_{MAX}} = 1 - \frac{V_{ReR} + V_s}{V} \tag{3.8}$$

3.4 Sistema Cliente-Servidor de Flujos Semi-Elásticos

En un sistema cliente-servidor, los clientes generan peticiones de información al servidor. En nuestro estudio, se supone que la información que se pide se puede transmitir con un flujo semi-elástico. En una Internet con QoS extremo a extremo es posible reservar recursos permitiendo la correcta transmisión de estos flujos semi-elásticos. Por tanto, las aplicaciones cliente-servidor podrán usar mecanismos de provisión de QoS para reservar ancho de banda y garantizar un retardo mínimo que asegure la entrega correcta de la información. Si la tasa de entrega de la información a la aplicación cliente, durante la reserva de recursos, es igual a su tasa de lectura, la reserva se mantendrá durante toda la transmisión y la memoria del cliente se mantendrá vacía. No obstante, si es mayor que la tasa de lectura del cliente, la información llenará su memoria. Así, sería posible enviar toda la información al cliente si la memoria fuese suficientemente grande. Esta opción no es muy útil cuando no toda la información es necesaria al mismo tiempo[i], ya que puede que parte de esa información no se utilice, y además se requiere reservar ancho de banda durante toda la transmisión incrementando innecesariamente el coste de la comunicación. Por otra parte, comporta el almacenamiento de toda la información en la memoria del cliente, por lo que el aprovisionamiento de ésta dependerá de la cantidad de información a enviar (en ocasiones muy grande, o no conocida).

Es posible mejorar el coste de la transmisión fijando un umbral máximo en la memoria del cliente, en vez de llenarla hasta que todos los datos se hayan enviado. Cuando los datos almacenados en la memoria del cliente llegan al umbral máximo, el cliente informa a la red para que cambie el modo de transmisión a *best-effort*. Durante la transmisión en modo *best-effort*, los datos llegan a la memoria del cliente con una tasa que depende del estado de la red. Esta tasa puede ser menor o igual que la tasa de lectura del cliente. En el caso de ser igual, la cantidad de datos almacenados se mantendrá, mientras que si la tasa es menor, la cantidad de datos decrecerá. Obsérvese que si la tasa de *best-effort* es mayor que la de lectura, no es necesaria la reserva de recursos, y simplemente con mecanismos tradicionales de control de flujo (como en TCP) es posible la transmisión. Por otra parte, cabe resaltar, que cuanto mayor sea la cantidad de datos que llegan durante el periodo de

[i] Como en el caso de aplicaciones tipo *streaming*.

best-effort (V_{BE}), menor será el coste de la transmisión, ya que éstos no se entregan mediante reserva de recursos.

A continuación se definirán todos los parámetros de la memoria del cliente que permiten controlar su ocupación. Seguidamente se especificará la dinámica de estos datos cuando se utiliza el método que se propone en este trabajo para reducir el coste de la transmisión de flujos semi-elásticos. Finalmente, se planteará un criterio, basado en la ocupación de la memoria del cliente, que permitirá minimizar el coste de la transmisión y que será el objetivo a alcanzar por cualquier método que pretenda una eficiencia máxima.

3.4.1 Memoria del Cliente

En la Figura 3.5 se muestra la memoria del cliente. Como se puede observar, hay tres parámetros principales que la definen: tamaño de la memoria (*M*), umbral máximo (*Max*) y umbral mínimo (*Min*).

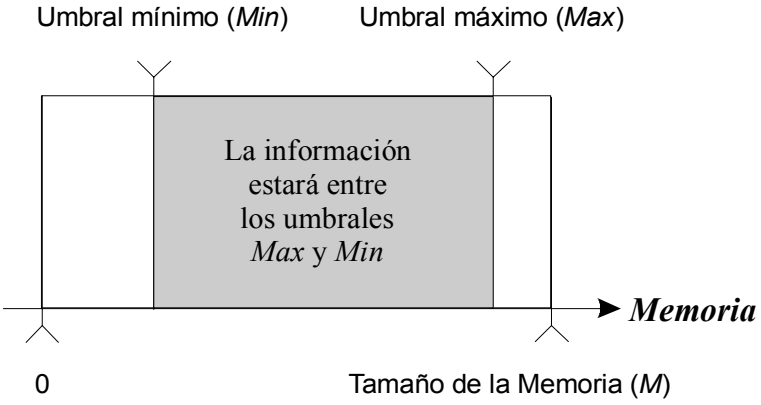

Figura 3.5: Memoria del cliente.

M es el espacio de memoria reservado en el cliente para almacenar los datos que llegan del servidor. Así, la cantidad de datos guardados debe estar siempre por debajo de este límite.

Como se ha explicado anteriormente, el umbral máximo (*Max*) se utiliza para cambiar entre el modo de transmisión con reserva de recursos y el de *best-effort*. Como se verá en el Capítulo 4, éste es uno de los parámetros más relevantes ya que es la clave para minimizar el coste.

Finalmente, el umbral *Min* se usa para cambiar del modo de *best-effort* al modo de reserva de recursos. Este valor debe ser definido por el cliente para garantizar la disponibilidad de datos durante todo el tiempo. También permite garantizar un

tiempo de guarda para realizar las reservas de recursos de forma correcta, ya que habrá un retardo en la señalización y en la reserva debido a posibles bloqueos.

3.4.2 Dinámica de los Datos Almacenados

La cantidad de información almacenada en la memoria del cliente variará a través del tiempo dependiendo del modo de entrega de datos (modo ReR o modo BE). Cuando el modo ReR es el seleccionado, la tasa de entrega será α_{ReR} ($bits/s$). De forma similar, se define la tasa de entrega en el modo BE como α_{BE} ($bits/s$). Es importante remarcar la dependencia de α_{BE} con la carga de la red. Obviamente, una red muy cargada entregará datos al cliente a una tasa menor que una red infrautilizada. Aquí, α_{BE} se define a nivel de la memoria del cliente (*bits*/s efectivos que entran en la memoria), no a nivel de transporte, por lo que se ve afectada por las pérdidas debidas a errores en la transmisión, descartes de paquetes en los *routers* y por las posibles retransmisiones y reordenación de paquetes. Cuanto menor sean éstos mayor será α_{BE}.

La Figura 3.6 muestra la variación en el tiempo de la cantidad de datos en la memoria del cliente. Como se observa, de 0 a t_0 es el primer periodo de tiempo en el que se usa reserva de recursos (al principio de la transmisión) y ($t_1 - t_0$) es el primer periodo de tiempo usando el modo BE. Inicialmente, la memoria está vacía, por lo que parece lógico enviar los primeros paquetes con reserva de recursos para proporcionar de forma rápida la información al cliente. En este tiempo, la cantidad de información almacenada en la memoria del cliente aumentará como si se llenara con una tasa de entrada α_i ($bits/s$). Esta tasa (3.9) es la diferencia entre la tasa de entrega cuando se reservan recursos (α_{ReR}) y la tasa de lectura del cliente (α_r, $bits/s$).

$$\alpha_i = \alpha_{ReR} - \alpha_r \tag{3.9}$$

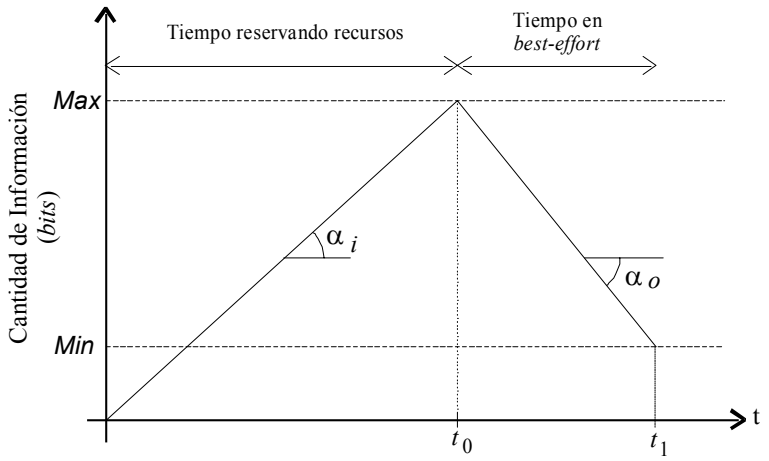

Figura 3.6: Dinámica de los datos almacenados en la memoria del cliente

Cuando los datos almacenados alcanzan *Max*, el modo de entrega de datos pasa a BE. Similarmente, en este periodo, la cantidad de información decrecerá si $\alpha_{BE} < \alpha_r$, por lo que la memoria se vaciará con una tasa α_o (*bits/s*). De nuevo, esta tasa (3.10) es la diferencia entre la tasa de entrega en *best-effort* y la tasa de lectura del cliente. Más tarde, cuando el nivel de datos almacenados llegue a *Min*, el modo de entrega cambiaría otra vez a ReR si la transmisión no se ha completado.

$$\alpha_o = \alpha_{BE} - \alpha_r \tag{3.10}$$

Para determinar como afecta la tasa α_{BE} [ii] al comportamiento de la memoria, se define la carga apreciada por el cliente (ρ) como,

$$\begin{cases} \rho = 1 - \dfrac{\alpha_{BE}}{\alpha_r} & \alpha_{BE} \le \alpha_r \\ \rho = 0 & \alpha_{BE} \ge \alpha_r \end{cases} \tag{3.11}$$

La ecuación (3.11) define una carga percibida de 0 cuando el modo BE es suficiente para mantener la memoria llena. Por el contrario, una carga percibida de 1 indica que no llega ningún paquete durante este periodo. Por tanto, indica el grado de diferencia que aparentemente produce la red sobre la tasa α_{BE} con respecto a la tasa de lectura α_r de un cliente.

[ii] Esta tasa está relacionada con la carga de la red, ya que una tasa pequeña indicará una carga alta, y una tasa grande una carga baja.

3.4.3 Criterio de Minimización del Coste

Como se ha visto, el método propuesto para reducir el coste de una transmisión de flujos semi-elásticos se basa en enviar todos los datos posibles mediante el servicio de *best-effort*, garantizando en todo momento la disponibilidad de información en la memoria del cliente. Esta disponibilidad está controlada por el nivel *Min*, de manera que la información en la memoria siempre debe estar por encima. Para garantizar que el coste es mínimo, se propone como criterio de minimización que el nivel de datos en la memoria al acabar la transmisión, esté en el nivel *Min*. Si esto ocurre se garantiza que la utilización de la reserva de recursos de forma mínima. Esto se debe a que, en el modelo propuesto, la ocupación de la memoria crece si se utiliza el modo ReR y decrece si se utiliza el modo BE. Si al finalizar la transmisión el nivel de datos está por encima de *Min*, quiere decir que el modo ReR se ha utilizado más tiempo que si no lo estuviera y por tanto la reserva de recursos ha sido mayor. En el Capítulo 4 se demostrará que efectivamente, este criterio se puede utilizar para minimizar el coste de una transmisión de flujos semi-elásticos.

3.5 Conclusiones

En este capítulo se han clasificado los flujos de datos en tres tipos: elásticos, inelásticos y semi-elásticos. Los flujos elásticos son aquellos que requieren un nivel de servicio igual al clásico de *best-effort*. Son flujos que no tienen ningún tipo de requerimiento temporal y por tanto los datos pueden llegar en cualquier momento al extremo cliente. Por otro lado, los flujos inelásticos tienen requerimientos temporales muy estrictos. En este caso, la información debe llegar en un instante muy concreto para ser consumida correctamente por el cliente. Además, la información se genera en directo por lo que no es posible avanzar la transmisión. Este tipo de flujos necesita reservar recursos durante toda la transmisión para evitar que otros flujos de datos entorpezcan en la transmisión, haciendo que aparezcan pérdidas o retardo inesperado. Finalmente, los flujos de datos semi-elásticos se caracterizan por tener requisitos temporales muy estrictos, pero a diferencia de los flujos inelásticos, aquí la información está prealmacenada. Esto permite que los datos se puedan enviar al extremo cliente antes de que sean necesarios, donde se almacenan en una memoria hasta el momento de su consumo. Asimismo, los recursos que se necesitan reservar con este tipo de flujos dependen del estado de la red, que a su vez influye en la ocupación de la memoria del cliente. La transmisión será correcta si la memoria

tiene los datos necesarios en el momento de su consumo, y por tanto, si su ocupación está entre unos determinados límites.

Para que las reservas de recursos de red se realicen de forma responsable es necesario utilizar un sistema de tarificación que sea justo, es decir, que aplique un coste mínimo a aquellos flujos de datos que utilicen los mínimos recursos de red posibles, y penalice a aquellos que no lo realizan así. En este capítulo, se ha descrito un modelo de coste basado en el uso de los recursos de la red por parte de un supuesto proveedor de servicio. Este modelo se divide en cuatro partes: carga por uso, carga por desaprovechamiento del servicio, carga por congestión y carga por señalización. La carga por uso tarifica la cantidad de datos que el usuario envía por la red teniendo en cuenta el nivel de servicio que se le ofrece. La carga por desaprovechamiento del servicio penaliza a aquellos usuarios que reservan recursos en exceso. La carga por congestión penaliza a aquellos flujos que incumplen el contrato de nivel de servicio con la red y que por tanto, sobrecargan la red de forma imprevisible. La carga por señalización se aplica al usuario en concepto de gestión de las reservas, de forma que una aplicación que renegocie su contrato muchas veces causando cambios en el estado de las reservas de la red esté más penalizada que otra que señalice menos.

Para el caso de los flujos semi-elásticos se ha planteado la posible minimización del coste si sólo se reservan los recursos necesarios. Como se ha mostrado, se propone un sistema cliente-servidor para la transmisión de estos flujos, donde el cliente gestiona la ocupación de su memoria en la que se almacenan los datos que provienen del servidor. Esta memoria tiene dos niveles entre los que debe estar su ocupación. El nivel mínimo garantiza que el cliente no se quede sin datos y permite controlar el retardo que se admite para que una reserva de recursos esté activa. Por otra parte, el nivel máximo, que dependerá del tamaño máximo de la memoria, permitirá controlar que los recursos que se reservan de la red son los mínimos para tener un coste óptimo. De esta forma la ocupación de la memoria del cliente aumenta cuando hay reserva de recursos hasta alcanzar el nivel máximo. Cuando lo alcanza, la transmisión se realiza mediante el servicio de *best-effort*, lo que reduce el coste de la transmisión. Si la ocupación llega al nivel mínimo, se cambia a modo con reserva de recursos y así sucesivamente. Por tanto, el coste será mínimo si se garantiza que al final de la transmisión la ocupación de la memoria está en el nivel mínimo, es decir, se ha reservado recursos durante el menor tiempo posible.

Capítulo 4

Diseño del Cliente como Parte Controladora del Coste

4.1 Introducción

La transmisión de flujos de datos semi-elásticos se realiza mediante un sistema cliente-servidor en el que el cliente controla la ocupación de su memoria para determinar si es necesario o no reservar recursos de la red. El servidor utiliza dos modos de transmisión, uno en el que se realiza Reserva de Recursos (ReR) y otro en el que se utiliza el servicio tradicional de *Best-Effort* (BE). Por su parte, el cliente dispone de una memoria de la que se monitoriza su ocupación, controlando que la información almacenada permanezca entre dos umbrales, uno máximo (*Max*) y otro mínimo (*Min*). El umbral mínimo debe garantizar la disponibilidad de datos durante toda la transmisión, permitiendo que los datos recibidos se puedan utilizar lo antes posible. El umbral máximo se encarga de determinar cuándo se debe cambiar el modo de transmisión a BE. Así, si la ocupación llega a este umbral, se dejarán de reservar recursos, lo cual permite que disminuya el coste. Además, mediante el diseño óptimo de este umbral se debe conseguir la minimización o reducción del coste de la transmisión. De forma tal, que dependiendo de la información que se reciba en los periodos BE, se utilizará más o menos el modo de transmisión ReR, pero siempre se procurará garantizar que se maximice el uso del servicio BE (supuesto más económico).

En este capítulo, se analiza el uso del umbral *Max* en la minimización del coste total de la transmisión. Para ello, se examina el comportamiento de la ocupación de la memoria del cliente y como afectan los diversos parámetros involucrados en el coste. Del análisis se obtiene una serie umbrales que garantizan que la información enviada mediante el modo ReR sea mínima. Por otra parte, se plantea un problema en el cálculo del umbral óptimo, ya que es necesaria una estimación de la tasa de datos que recibe el cliente durante la transmisión en modo BE. Esta tasa no se conoce a priori, por lo que es necesario estimarla previamente al cálculo del umbral *Max*.

También se investiga el dimensionado del umbral *Min*, que debe garantizar que en el proceso de señalización, la memoria no se quede vacía en caso de que la señalización de cambio de modo BE a modo ReR se demore demasiado o sea rechazada.

En el análisis, se supone el caso ideal en que la ocupación de la memoria del cliente varia de forma uniforme. En un caso real, esta ocupación se ve afectada por las circunstancias reales y dinámicas de carga de tráfico y su variabilidad en las redes. Es decir, la ocupación se incrementa de forma escalonada, cada vez que se recibe un paquete de datos, y se consume de la misma forma. Por otro lado, estos paquetes de datos, aunque se envíen uniformemente espaciados en el tiempo, llegan al cliente retardados de forma aleatoria dependiendo del tráfico de datos que circula por la red en modo BE. Esto implica que el cliente no puede confiar plenamente en las estimaciones que realiza, por lo que se plantea la necesidad de calcular el umbral *Max*, de forma que exista una cierta protección ante este tipo de eventos.

Finalmente, se mostrarán los resultados de simulación más relevantes desarrollados sobre, y obtenidos mediante el simulador de red Network Simulator 2 (ns-2) [NS2], que se utilizan para evaluar el correcto funcionamiento del sistema cliente-servidor de flujos semi-elásticos.

4.2 Umbral máximo de ocupación de la memoria

Tal y como se vio en el Capítulo 3, el sistema propuesto para la transmisión de flujos semi-elásticos está basado en un cliente y un servidor. De estos dos elementos, el cliente es el encargado de controlar el coste de la transmisión mediante la monitorización de la ocupación de la memoria donde se almacenan los datos recibidos

del servidor. En esta memoria se definen dos umbrales (mínimo y máximo) que permiten controlar dicha ocupación. El umbral mínimo (*Min*) se encarga de garantizar que la memoria dispone de datos suficientes durante toda la transmisión, de manera que el cliente pueda consumirlos a medida que los recibe. El umbral máximo (*Max*) permite controlar el coste de la transmisión indicando el momento adecuado para cambiar el modo de transmisión a *Best-Effort* (BE), tal y como se verá en este apartado.

El coste de la transmisión se definió como la cantidad de información transmitida en modo de Reserva de Recursos (ReR). Por tanto, para minimizar este valor se deberá maximizar el uso del modo BE. En este apartado, se calculará el modelo de coste correspondiente a la dinámica de los datos en la memoria del cliente. Este modelo formulará el coste en función de todos los parámetros que intervienen en el sistema y que afectan a la ocupación de la memoria del cliente. El modelo se analizará debidamente, para observar el comportamiento del coste a variaciones de los distintos parámetros. De todos los parámetros se incidirá especialmente en *Max* ya que es el elegido para minimizar la función de coste.

4.2.1 Cálculo de los umbrales *Max* óptimos

El criterio para minimizar el coste se basa en enviar datos mediante los modos ReR y BE, maximizando en la medida de lo posible la utilización del modo BE. Como es lógico, si la red está poco cargada se reservarán recursos durante un tiempo menor que en el caso de haber más carga en la red.

El método propuesto inicia la transmisión en modo ReR, con el objetivo de llenar la memoria hasta el umbral *Max*. Esto garantiza que el cliente pueda empezar a leer la información inmediatamente. Cuando se alcanza el umbral *Max*, el modo de transferencia pasa a BE, de manera que la ocupación de la memoria decrecerá si la tasa de llegada de datos es menor que la tasa de lectura del cliente. Para asegurar que el cliente no se quede sin datos en la memoria, se define el umbral *Min*, de forma que si el nivel de datos almacenados llega a *Min*, se pasa a modo ReR. Por tanto, el método se basa en ir conmutando de un modo a otro garantizando que se cumpla un determinado criterio de minimización.

En el Capítulo 3, se mostró el comportamiento teórico de los datos almacenados en la memoria del cliente. En el modo de transferencia ReR la información llega con

tasa $\alpha_i = \alpha_{ReR} - \alpha_r$, mientras que en el modo BE la memoria se vacía con tasa $\alpha_o = \alpha_{BE} - \alpha_r$.

Tal y como se definió el coste de la transmisión, es posible encontrar los valores de *Max* que lo minimizan. Para un valor *Max* concreto, el coste de la transmisión de un flujo semi-elástico depende por tanto, del número de periodos ReR que se utilicen así como de la duración de éstos. Dependiendo del volumen de información (*V*) que reciba el cliente, se completará un número determinado de periodos ReR (N_{ReR}) y BE (N_{BE})[i]. En el caso de que una aplicación servidor transmita un volumen de información[ii] de V_{serv} *bytes* a un cliente mediante un flujo semi-elástico, utilizando un mecanismo de transporte fiable[iii], se cumplirá $V = V_{serv}$ y por tanto, será posible informar al cliente de la cantidad de información que recibirá en el transcurso de la transmisión (conocida a priori).

N_{ReR} depende de *V* y de *Max* de manera que si $V < (Max / \alpha_i)\alpha_{ReR}$ el valor de N_{ReR} será 1, ya que la transmisión acabará antes de que se alcance el umbral *Max*[iv]. En este caso, toda la información se habrá enviado utilizando el modo de transmisión ReR y por tanto el coste será máximo e igual a *V*. En cualquier otro caso, N_{ReR} valdrá como mínimo 1. Para calcular el número de periodos ReR de una transmisión se debe tener en cuenta que hay un periodo ReR por cada periodo BE y que el primer periodo ReR es mayor que los demás, ya que contiene el tiempo que se tarda en llenar la memoria del cliente desde 0 a *Min* (4.1). Por tanto, y teniendo en cuenta que como mínimo habrá un periodo ReR, N_{ReR} se expresará como,

---

[i] Se considera un periodo completo cuando la ocupación de la memoria del cliente varía desde un umbral hasta el otro, es decir, la transmisión no acaba en medio del intervalo en cuestión.

[ii] Este volumen de datos se refiere a los datos almacenados en el servidor. Por tanto, son datos a nivel de aplicación, que no necesariamente deben coincidir con los enviados a nivel de transporte.

[iii] Como ejemplo, supongamos el caso de usar el protocolo TCP (*Transmission Control Protocol*).

[iv] Obsérvese que Max/α_i es el tiempo que se tarda en alcanzar el umbral *Max*.

$$N_{ReR} = \left\lfloor \frac{V - \frac{Max}{\alpha_i}\alpha_{ReR}}{\frac{Max - Min}{\alpha_i}\alpha_{ReR} + \frac{Min - Max}{\alpha_o}\alpha_{BE}} \right\rfloor + 1 \qquad V \geq \frac{Max}{\alpha_i}\alpha_{ReR} \qquad (4.1)$$

En la ecuación (4.1) el numerador es la cantidad de información que se recibe en el cliente después del primer periodo ReR; y el denominador es la cantidad de datos que se reciben en la suma de un periodo de transmisión BE y un ReR. Nótese, que la función $\lfloor . \rfloor$ representa la parte entera del cociente.

Para el cálculo de N_{BE} se debe tener en cuenta que *Max* nunca puede ser menor que *Min*, y consecuentemente, se deberá cumplir $V \geq (Min/\alpha_i)\alpha_{ReR}$ para que pueda haber un periodo BE. Considerando esto, y de forma similar al cálculo de N_{ReR} se puede expresar el número de periodos N_{BE} como,

$$N_{BE} = \left\lfloor \frac{V - \frac{Min}{\alpha_i}\alpha_{ReR}}{\frac{Max - Min}{\alpha_i}\alpha_{ReR} + \frac{Min - Max}{\alpha_o}\alpha_{BE}} \right\rfloor \qquad V \geq \frac{Min}{\alpha_i}\alpha_{ReR} \qquad (4.2)$$

Una vez conocidos N_{ReR} y N_{BE} es posible deducir el coste por uso (C_u) de la transmisión, en función de *Max*, para un valor concreto de *V*. Así, tal y como se ha recordado en el cálculo de N_{ReR}, si *Max* es demasiado grande puede que la transmisión acabe antes de alcanzarlo y por lo tanto el coste sea *V* (4.3). Asimismo, en el caso de que *Max* sea menor, la transmisión puede finalizar en un periodo ReR o en un periodo BE. Si acaba en un periodo BE se cumplirá $(N_{ReR} - N_{BE}) = 1$ (téngase en cuenta que tanto N_{ReR} como N_{BE} representan periodos completos), y por tanto se puede expresar el coste como la información recibida en el primer periodo ReR $(Max/\alpha_i)\alpha_{ReR}$, más la información recibida en el resto de periodos ReR $((Max - Min)/\alpha_i)\alpha_{ReR}(N_{ReR} - 1)$. De forma similar, cuando la finalización de la transmisión ocurre en un periodo ReR se cumple $(N_{ReR} - N_{BE}) = 0$ y se puede expresar el coste en función de los datos recibidos, ya que, la información recibida mediante el modo ReR es la diferencia entre el total de datos recibidos y los recibidos en modo BE (4.1).

Por tanto, se puede definir la función de coste por uso de la transmisión de flujos semi-elásticos mediante este sistema como,

$$C_u = \begin{cases} V & Max \geq \dfrac{V}{\alpha_{ReR}}\alpha_i \\ \dfrac{Max}{\alpha_i}\alpha_{ReR} + \dfrac{Max - Min}{\alpha_i}\alpha_{ReR}\left(N_{ReR} - 1\right) & \left(N_{ReR} - N_{BE}\right) = 1 \\ V - \dfrac{Min - Max}{\alpha_o}\alpha_{BE}N_{BE} & \left(N_{ReR} - N_{BE}\right) = 0 \end{cases} \quad (4.3)$$

Por otra parte, queda cuantificar el coste de señalización normalizado $\overline{C}_s$ (4.4) que dependerá de N_{ReR} y que indica el número de mensajes de señalización necesario. En el primer caso, en que $C_u = V$, sólo hay dos negociaciones (inicio de sesión en el que se pasa a modo ReR y finalización de la reserva)[v] por lo que el coste de señalización normalizado será 2. En el caso de que la transmisión acabe en un periodo BE se realizarán 2 negociaciones por periodo ReR más la última de finalización. Finalmente, en el caso de terminar en un periodo ReR, se realizarán 2 negociaciones por periodo BE más la del inicio y la del final[vi]. El coste de señalización será $C_s = \overline{C}_s \cdot \overline{V}_s$, donde $\overline{V}_s$ es el volumen de datos que ocupa un mensaje de señalización.

$$\overline{C}_s = \begin{cases} 2 & Max \geq \dfrac{V}{\alpha_{ReR}}\alpha_i \\ 2N_{ReR} + 1 & \left(N_{ReR} - N_{BE}\right) = 1 \\ 2N_{ReR} + 2 & \left(N_{ReR} - N_{BE}\right) = 0 \end{cases} \quad (4.4)$$

Por tanto, para determinar los valores óptimos de *Max* es necesario minimizar la función de coste $C = C_u + C_s$ en función de *Max*. Para ello, se debe resolver la ecuación (4.5) para analizar los mínimos de la función.

$$C' = \frac{\partial C}{\partial Max} = C'_u + C'_s = \frac{\partial C_u}{\partial Max} + \frac{\partial C_s}{\partial Max} = 0 \quad (4.5)$$

[v] El estudio se realiza de forma genérica (sin especificar el protocolo de señalización) y por tanto puede que existan variaciones en la cantidad de mensajes de señalización necesarios, para mecanismos concretos. De cualquier forma, el coste de señalización será siempre dependiente de N_{ReR}.

[vi] En (4.4) se expresa en función de N_{ReR} que es igual a N_{BE}.

$$C'_u = \begin{cases} 0 & Max \geq \dfrac{V}{\alpha_{ReR}}\alpha_i \\ \dfrac{\alpha_{ReR}}{\alpha_i} N_{ReR} & \left(N_{ReR} - N_{BE}\right) = 1 \\ \dfrac{\alpha_{BE}}{\alpha_o} N_{BE} & \left(N_{ReR} - N_{BE}\right) = 0 \end{cases} \tag{4.6}$$

$$C'_s = \begin{cases} 0 & Max \geq \dfrac{V}{\alpha_{ReR}}\alpha_i \\ 0 & \left(N_{ReR} - N_{BE}\right) = 1 \\ 0 & \left(N_{ReR} - N_{BE}\right) = 0 \end{cases} \tag{4.7}$$

De las ecuaciones (4.3), (4.4), (4.6) y (4.7) se deduce que para el intervalo donde $Max \geq \left(V / \alpha_{ReR}\right)\alpha_i$ la función de coste *C* no se puede minimizar (es constante) y tiene el máximo valor posible[vii]. Para el resto de intervalos, se observa que (4.5) no se cumple, por lo que es necesario explorar los intervalos de monotonía de la función *C*.

De (4.6) y (4.7) se deduce que *C* es creciente con respecto a *Max* en los intervalos donde $\left(N_{ReR} - N_{BE}\right) = 1$, por tanto, en estos intervalos, *C* será mínimo para el menor valor *Max* posible. Sin embargo, *C* es decreciente cuando $\left(N_{ReR} - N_{BE}\right) = 0$, ya que α_o es de valor negativo, por lo que en estos intervalos el mínimo coste se obtendrá para el mayor valor de *Max* posible.

Como $\left(N_{ReR} - N_{BE}\right) = 1$ implica que la transmisión acaba en un periodo BE, si *Max* disminuye, esta situación se prolonga hasta que la transmisión acaba cuando la ocupación de la memoria del cliente está en el umbral *Min*, es decir, justo en el momento en que $\left(N_{ReR} - N_{BE}\right) = 0$. A partir de ese instante, tal y como se ha demostrado, el coste aumenta si *Max* disminuye, ya que se alarga el periodo ReR en el que acaba la transmisión. De esta forma, y volviendo a las expresiones de N_{ReR} (4.1) y N_{BE} (4.2) se debe cumplir un determinado criterio de minimización (Figura 4.1), es decir, $N_{ReR} = N_{BE}$ y la transmisión debe acabar cuando la ocupación de la memoria del cliente es igual a *Min*.

[vii] Toda la información se envía utilizando el modo de transferencia ReR.

$$N_{ReR} = \frac{V - \frac{Max}{\alpha_i}\alpha_{ReR}}{\frac{Max - Min}{\alpha_i}\alpha_{ReR} + \frac{Min - Max}{\alpha_o}\alpha_{BE}} + 1 = \\ = N_{BE} = \frac{V - \frac{Min}{\alpha_i}\alpha_{ReR}}{\frac{Max - Min}{\alpha_i}\alpha_{ReR} + \frac{Min - Max}{\alpha_o}\alpha_{BE}} \tag{4.8}$$

Así, el valor más grande de *Max* (*Max*$_0$) que minimiza el coste *C* será para $N_{ReR} = N_{BE} = 1$,

$$Max_0 = \frac{V\alpha_O - Min\alpha_{BE}}{\alpha_{ReR}\alpha_o - \alpha_{BE}\alpha_i}\alpha_i = \frac{\alpha_r V - \alpha_{BE}(V - Min)}{\alpha_r(\alpha_{ReR} - \alpha_{BE})}(\alpha_{ReR} - \alpha_r) \tag{4.9}$$

En general, para $N_{ReR} = N_{BE} = n+1$,

$$Max_n = Min + \frac{Max_0 - Min}{n+1} \tag{4.10}$$

Por tanto, es posible controlar la ocupación máxima de la memoria del cliente, por lo que el umbral *Max* se podrá escoger por debajo del tamaño máximo de memoria (*M*) disponible en el cliente para la transmisión, teniendo en cuenta que cuanto mayor sea *n* más elevado será el número de periodos en el que se reservarán recursos (así como el número de renegociaciones y señalización asociada).

De esta forma, se puede calcular el coste mínimo de la transmisión (C_{min}) que será aquel en el que se usa un valor $Max = Max_n$ (4.11).

$$C_{min} = C\Big|_{Max=Max_n} = \frac{\alpha_r V - \alpha_{BE}(V - Min)}{\alpha_r(\alpha_{ReR} - \alpha_{BE})}\alpha_{ReR} + (2n+3)\cdot\overline{V}_s \tag{4.11}$$

Como se observa en (4.11) el coste depende directamente de *n* por lo que deberá usarse un el valor más pequeño posible. Recuérdese, que *n* indica el tamaño de *Max*, así un valor de *n* pequeño indica *Max* grande, mientras que una *n* grande indica *Max* pequeño. Asimismo, el valor de *Max* que se deberá utilizar en una transmisión de flujos semi-elásticos dependerá de la memoria del cliente disponible (*M*). En (4.12) [viii] se expresa la condición que debe cumplir *n* para que *Max* sea inferior, donde el valor óptimo de *n* corresponde a la igualdad.

[viii] La función $\lceil . \rceil$ corresponde a la parte entera superior.

$$Max_n \le M \Rightarrow n \ge \left\lceil \frac{Max_0 - Min}{M - Min} - 1 \right\rceil \tag{4.12}$$

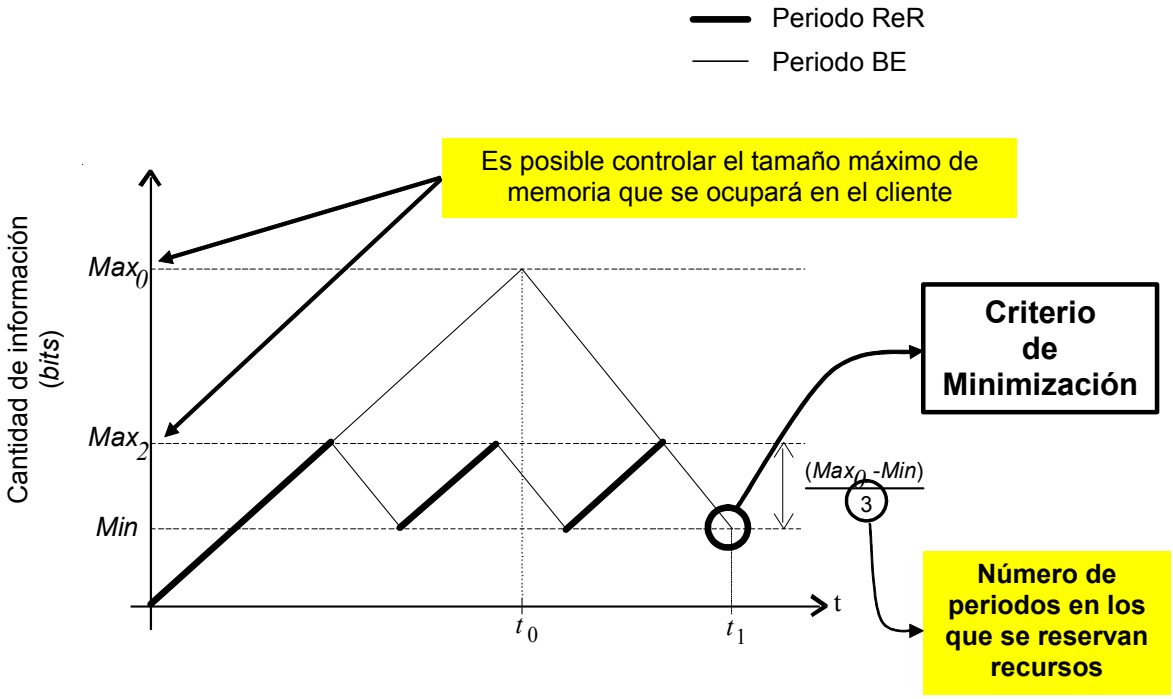

Figura 4.1: Criterio de Minimización del coste de transmisión.

Pero, *Max* no sólo tiene una cota superior, sino también una cota inferior, ya que el coste no deberá ser nunca superior al obtenido sin reservar recursos de red (C_{max}).

$$C_{min} = C_{Max} = V \Rightarrow n < \frac{V}{2} - \frac{\alpha_r V - \alpha_{BE}(V - Min)}{2\alpha_r(\alpha_{ReR} - \alpha_{BE})}\alpha_{ReR} + \frac{3}{2} \tag{4.13}$$

Continuando con la exploración de la ecuación del coste mínimo (4.11), se puede advertir que una vez fijo el umbral *Max* adecuado (supóngase un valor de n lo más pequeño posible) el método propuesto para la transmisión de flujos semi-elásticos presenta un problema a tener en cuenta: de nuevo es posible superar C_{max} si la tasa de llegada de datos en el periodo BE es demasiado pequeña (o lo que es lo mismo, la carga de la red que aprecia el cliente es demasiado grande). De aquí se deduce que el método será válido si se cumple la ecuación (4.14).

$$\rho < \frac{(V - C_s)\alpha_i - \alpha_{ReR} Min}{(V - C_s)\alpha_i - \alpha_{ReR} Min + C_s \alpha_{ReR}} \tag{4.14}$$

Pero cabe resaltar, que incluso en este caso, falta considerar un tercer parámetro con el que poder evitar de nuevo esta circunstancia: la tasa de llegada de datos en modo ReR. De esta forma, si el cliente detecta una situación de ρ demasiado alta teniendo en cuenta la ecuación (4.14), deberá reducir α_{ReR}. Esta forma de actuar, permite reducir la señalización hasta el punto crítico de tener un único periodo de transferencia ReR y que toda la información se envíe en él, lo cual comporta no usar el modo BE.

En todo este estudio, se ha supuesto una transmisión fiable en la que toda la información enviada por el servidor llega de forma fiable al extremo cliente. En las transmisiones no fiables, donde la información que envía el servidor puede no llegar completa al cliente, el volumen total de información que le llegará será en general menor que *V*. Esta situación dificulta el cálculo del umbral *Max* para minimizar el coste de la transmisión, ya que *V* es desconocido. Por otra parte, las pérdidas deben mantenerse controladas, ya que en los periodos ReR los datos además de llegar a tiempo, llegarán con probabilidad de pérdida menor (o al menos controlada) que en los periodos BE. Por tanto, para igualar las pérdidas y que de esta forma el usuario no perciba diferencias de calidad entre modos de transferencia, será necesario controlar las pérdidas mediante protocolos adecuados. Así, en este tipo de entorno, tal y como puede ser una transmisión mediante el protocolo UDP (*User Datagram Protocol*), es necesario controlar la fiabilidad mediante el uso de algún mecanismo para controlar las pérdidas que afectan al total de información recibida por el cliente. Las pérdidas pueden medirse utilizando protocolos como RTP (*Real Time Protocol*), y se puede añadir cierto grado de fiabilidad en los periodos BE usando protocolos como VDP (*Video Datagram Protocol*) [CHE96] o RTSP (*Real Time Streaming Protocol*) [SCH98] usados en la distribución BE de vídeo pre-almacenado. Por consiguiente, las transmisiones de flujos semi-elásticos necesitan el control de pérdidas e incluir mecanismos de fiabilidad para mantener la calidad de los datos durante los periodos BE, similar a los periodos ReR.

En esta situación, la aplicación o extensión de las expresiones anteriores para calcular *Max* e intentar la minimización del coste es sencilla. Si el cliente permite pérdidas de información durante la transmisión, y se observa una probabilidad de pérdida p_{perd}, la cantidad de datos que se esperará recibir será,

$$V' \approx V \cdot \left(1 - p_{perd}\right) \tag{4.15}$$

Además, tanto α_{ReR} como α_{BE} se pueden estimar y por tanto, el cálculo de *Max* se puede obtener con dichas estimaciones y la ecuación (4.10).

Finalmente, es posible que la transmisión acabe de forma inesperada[ix]. En este caso, es muy probable que el coste no se minimice, pero sí que se verá reducido si se usa también el modo BE. Como se deduce de (4.3) si $Max \geq (V / \alpha_{ReR})\alpha_i$, el coste de la transmisión se reducirá con respecto a *V*, ya que parte del volumen de la información se habrá enviado usando el modo BE.

4.2.2 Estimación de la tasa de llegada en modo BE para el cálculo del umbral máximo

Tal y como se deduce de la ecuación (4.9), es necesaria una estimación de la tasa de llegada de datos en modo BE (α_{BE}) para poder calcular el umbral óptimo Max_0 y a partir de éste el Max_n que permita la memoria del cliente. El método propuesto para la transmisión de flujos semi-elásticos plantea el inicio de la transmisión en modo ReR. Se supone que la información debe llegar pronto al cliente y la tasa de llegada de datos debe estar garantizada para poder empezar a leer la información almacenada a partir del instante en que llegan los primeros datos a su memoria. Por otra parte, para cambiar el modo de transferencia se debe llegar al umbral *Max* escogido de forma óptima para minimizar el coste de la transmisión. Por tanto, se plantea la necesidad de obtener una estimación de la tasa α_{BE} para poder calcular el umbral *Max*.

El caso ideal sería realizar una estimación continua de la tasa α_{BE} durante todo el tiempo (incluso antes de iniciar la transmisión). Esta situación supondría en el extremo la necesidad cliente de tener un proceso, funcionando simultáneamente a los procesos cliente y servidor, que se encargara de realizar el cálculo [STE00] [PAX97]. Además, también implica cargar la red con paquetes de control adicionales para acometer la estimación continua de la tasa de entrega de datos en modo BE.

Para evitar esta carga extra de la red y la necesidad de otra aplicación que estime la tasa α_{BE} se plantean otras dos soluciones que utilizan el propio cliente para realizar la estimación. La primera consiste en iniciar la entrega de datos al cliente en

[ix] Por ejemplo, en el caso de una transmisión interactiva donde el usuario puede parar la transmisión en cualquier momento.

modo BE en vez de en modo ReR. Esto significa que el cliente deberá esperar un tiempo inicial (T_{est}) antes de empezar a leer datos de su memoria, para que se pueda llevar a cabo la estimación de α_{BE}. Pasado este tiempo, se podrá iniciar la transmisión de datos en modo ReR hasta alcanzar el umbral *Max*, ahora ya calculado. Para calcular *Max* hay que tener en cuenta que con respecto al modelo inicialmente propuesto, en este caso, la cantidad de información a recibir V se cuenta a partir del momento en que empieza el modo ReR, y además el nivel de ocupación con el que se inicia este modo ya no es 0. Teniendo esto en consideración, el nuevo volumen de datos a recibir V' será,

$$V' = V - \alpha_{BE} \cdot T_{est} \tag{4.16}$$

Y la expresión de Max_0 nueva (Max_0'),

$$Max_0' = Max_0 - \alpha_{BE} \cdot T_{est} \frac{\alpha_{ReR}\left(\alpha_{BE} - \alpha_r\right)}{\alpha_r\left(\alpha_{ReR} - \alpha_{BE}\right)} \tag{4.17}$$

Dado que este método supone que el cliente puede esperar a que se realice una estimación suficientemente precisa y esto no es lo deseable en una red con QoS, se plantea un tercer método. En este caso, el cliente escoge un umbral *Max* inicial (Max_{ini}) que le permitirá pasar a modo BE, donde se realizará la estimación de α_{BE}. Esta nueva manera de obtener la estimación, permite que el cliente empiece a leer los datos de forma rápida, ya que la transferencia se inicia en modo ReR. Como inconveniente, se obliga al servidor a enviar una cierta cantidad de información usando reserva de recursos. Esto implica que si la carga es baja, puede que se envíen más datos de los necesarios en modo de transferencia ReR y el coste sea mayor que el coste mínimo (C_{min}), por lo que para que el método sea válido se debe cumplir la ecuación (4.18) que implica que dependiendo de la carga ρ, Max_{ini} deberá ser menor a medida que la carga sea menor (4.19).

$$C_{min} \geq \frac{Max_{ini}}{\alpha_i}\alpha_{ReR} \tag{4.18}$$

$$Max_{ini} \leq \left[V - (1-\rho)(V - Min)\right]\frac{\alpha_{ReR} - \alpha_r}{\alpha_{ReR} - \alpha_r(1-\rho)} \tag{4.19}$$

De la ecuación (4.19) se deduce que Max_{ini} debe ser lo más pequeño posible, sin embargo, es necesario que, a su vez, sea suficientemente grande como para obtener una estimación aproximada de α_{BE}. Si Max_{ini} es demasiado pequeño y la estimación

de α_{BE} es menor que el valor real, se verá en el apartado 4.4, que el coste puede verse incrementado debido a que el cliente detecta una necesidad de reservar recursos mayor que la real. Para evitar este fenómeno se recomienda utilizar como mayor valor de *Max*, el valor de Max_1. Esto garantiza que habrá al menos otro periodo BE, en el que hacer otra estimación, suficientemente grande como para realizar una estimación que permita calcular de nuevo el umbral *Max* que minimice el coste (Figura 4.2).

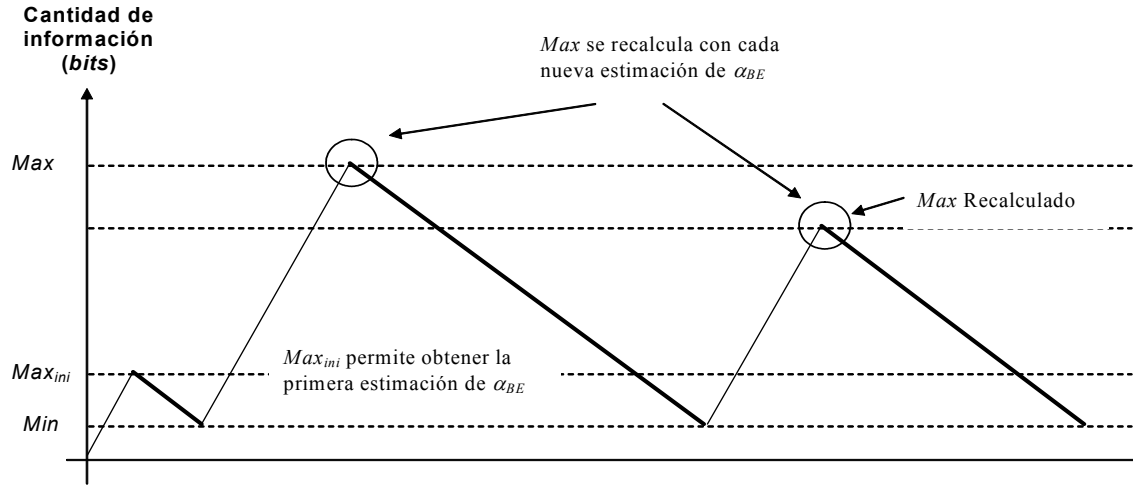

Figura 4.2: Uso de Max_{ini} para estimar la tasa de llegada de paquetes en el modo BE.

Igualmente en este caso, al utilizar el umbral Max_{ini} se debe calcular Max_n teniendo en cuenta que ya se ha recibido parte de la información, por lo que el volumen de información a recibir restante (V') será en este caso,

$$V' = V - \frac{Max_{ini}}{\alpha_i}\alpha_{ReR} - \frac{Max_{ini} - Min}{\alpha_o}\alpha_{BE} \tag{4.20}$$

4.3 Umbral mínimo de ocupación de la memoria

En la memoria del cliente se definen dos umbrales: uno máximo que permite minimizar el coste de la transmisión o al menos reducirlo, y uno mínimo que debe garantizar la disponibilidad de datos durante toda la transmisión, teniendo en cuenta que se permite al cliente iniciar el consumo de la información tan pronto como le sea posible.

La memoria del cliente puede llegar a vaciarse si la señalización para cambiar de modo de transferencia a ReR se retarda debido a la congestión[x] y a la propagación de los mensajes, o si la petición de reserva es rechazada por la red[xi]. Estas situaciones pueden controlarse con un diseño adecuado del umbral *Min*.

El máximo retardo en la reserva (T_R) (en un protocolo de reserva de recursos controlado por el cliente[xii]) es igual el tiempo de ida y vuelta[xiii] (*RTT*, *Round Trip Time*) entre el cliente y el servidor, multiplicado por el número de veces que la reserva es rechazada por la red (N_r)[xiv] más uno[xv] (4.21). Por esta razón, *Min* debe cumplir la ecuación (4.22) para prevenir N_r rechazos en el peor de los casos, es decir cuando $\alpha_{BE} = 0$.

$$T_R = (N_r + 1) \cdot RTT \tag{4.21}$$

$$Min \geq T_R \cdot \alpha_r = (N_r + 1) \cdot RTT \cdot \alpha_r \tag{4.22}$$

4.4 Protección frente a variaciones de la tasa de llegada de datos en modo BE

Utilizando el método de minimización propuesto, la exactitud con la que se alcanza el coste mínimo depende de la precisión con que se obtienen las estimaciones de las tasas de llegada de datos. Por este motivo, es necesario cuantificar las posibles diferencias entre el coste esperado usando el método (C_{min}) y el coste real obtenido.

---

[x] Este efecto se puede reducir si se reserva un canal con retardo controlado para el protocolo de señalización.

[xi] Esto puede reducir el grado de servicio que el proveedor de red ofrece, por lo que será conveniente que se redimensione la red en términos de ancho de banda, o bien, que se replanifique la topología o se mejoren los mecanismos de enrutado.

[xii] Como por ejemplo RSVP (*resource ReSerVation Protocol*).

[xiii] El tiempo de ida y vuelta, se verá afectado por el retardo de transmisión y por la congestión de la red.

[xiv] En el peor caso, la reserva será rechazada en el extremo servidor, por lo que el cliente detectará el rechazo pasado un tiempo *RTT*.

[xv] En el mejor caso, el cliente percibe la reserva como realizada pasado un tiempo *RTT*.

Este apartado se centra en analizar cómo afecta la variabilidad de α_{BE} al cálculo de *Max*.

El umbral *Max* óptimo se calcula mediante una estimación de α_{BE}. Esta estimación puede variar de un periodo de transmisión BE a otro, por lo que la correcta minimización del coste puede resultar dificultosa. Supongamos que se realiza un estimación α_{BE}' calculada en un periodo BE. En el siguiente periodo BE, pueden ocurrir tres situaciones distintas:

1. $\alpha_{BE} = \alpha_{BE}$': en este caso es posible minimizar el coste.
2. $\alpha_{BE} < \alpha_{BE}$': la ocupación de la memoria del cliente decrece más rápido de lo esperado y por tanto, se alcanza el umbral *Min* más rápidamente. Esta situación provoca que el sistema cambie de nuevo a modo ReR después de haber calculado un nuevo valor *Max* con la nueva estimación de α_{BE} obtenida (Figura 4.3).
3. $\alpha_{BE} > \alpha_{BE}$': la ocupación de la memoria del cliente decrece más lentamente de lo esperado. Por tanto, en caso de estar en el último periodo BE, la transmisión acabará sin cumplir el criterio de minimización, es decir, sin que la ocupación quede en el umbral *Min*, lo que implica un uso inadecuado de la reserva de recursos. Esta situación es crítica, y debe prevenirse para reducir el coste de la transmisión (Figura 4.4).

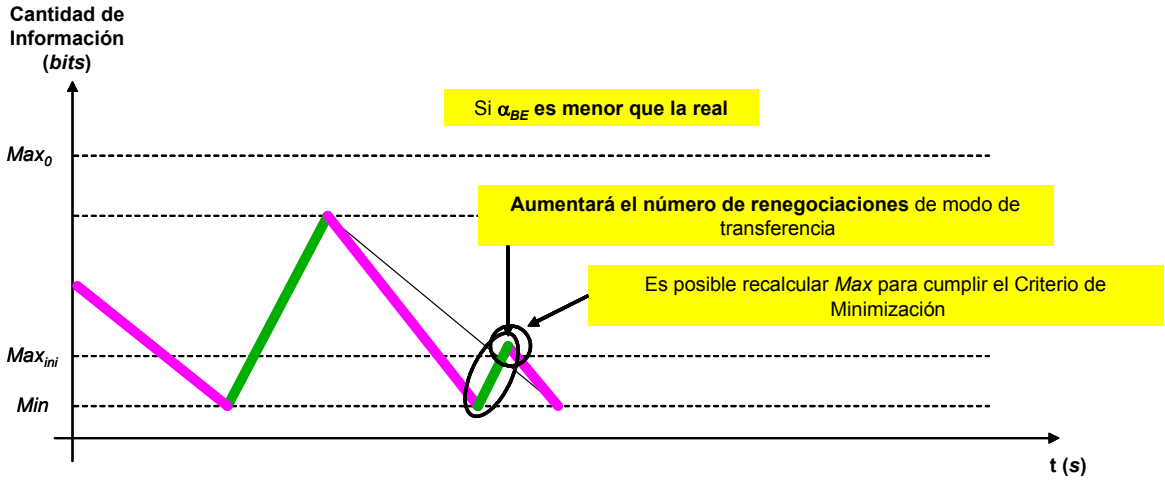

Figura 4.3: Caso $\alpha_{BE} < \alpha_{BE}$'.

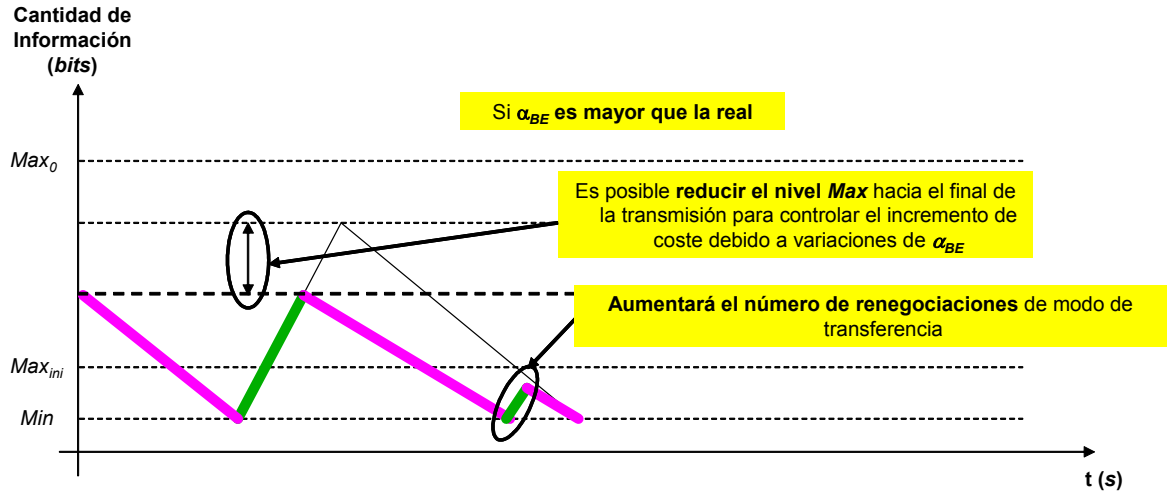

Figura 4.4: Caso $\alpha_{BE} > \alpha_{BE}$'.

Para evitar la situación 3, es posible reducir el umbral *Max* de tal forma que el máximo incremento del coste (4.23) no supere un determinado valor. Este incremento máximo tendrá lugar cuando ρ=0, ya que la ocupación de la memoria no decrecerá, debido a que se cumple $\alpha_{BE} = \alpha_r$.

$$\Delta C\big|_{\max} = \frac{\alpha_{ReR}}{\alpha_{ReR} - \alpha_r}(Max - Min) \tag{4.23}$$

Además, es posible controlar el error relativo máximo (4.24) que se puede cometer cuando se utiliza el método. Este error está determinado por $\Delta C\big|_{\max}$ y el coste parcial ($\hat{C}$) hasta el momento de cambiar al modo BE.

$$\varepsilon\big|_{\max} = \frac{\Delta C\big|_{\max}}{\hat{C}} \tag{4.24}$$

El error relativo depende del número de periodos ReR en un momento concreto de la transmisión (*N*) y del periodo Max_n usado. En general, dependiendo de *Max* y *N*, $\varepsilon\big|_{\max}$ se expresa como,

$$\varepsilon\big|_{\max}(N) = \frac{Max - Min}{(N+1)Max - (N \cdot Min)} \tag{4.25}$$

Substituyendo *Max* con Max_n, se obtiene (4.26). Por tanto, el error relativo decrece a medida que *n* ó *N* aumentan. Esto significa que la minimización será más precisa si Max_n es pequeño, y que las estimaciones inexactas afectarán menos al coste total en los últimos periodos.

$$\varepsilon\big|_{\max} = \frac{Max_0 - Min}{(N+1)Max_0 + (n-N)Min} \tag{4.26}$$

Así, el error relativo puede delimitarse disminuyendo Max_n en las últimas renegociaciones (utilizando una n mayor). Esto es debido a que para un umbral Max_n con $N < n$, el sistema está protegido frente a ciertas variaciones en la tasa α_{BE}, ya que es posible alcanzar el umbral Min antes de acabar la transmisión. En el caso de Max_0, no hay protección, ya que si se cumple la situación 3 comentada anteriormente, el método no cumplirá el criterio de minimización. Para el resto de umbrales Max_n, el máximo incremento de α_{BE} ($\Delta\alpha_{BE}\big|_{\max}$) sin incrementar el coste, se puede expresar en función de N (4.27).

$$\Delta\alpha_{BE}\big|_{\max} = \frac{(\alpha_{BE} - \alpha_r)(\alpha_{ReR} - \alpha_{BE})N}{(\alpha_{BE} - \alpha_r) + (\alpha_{ReR} - \alpha_{BE})N} \tag{4.27}$$

De acuerdo con esto, los posibles errores en la minimización del coste pueden verse reducidos, si se emplea un valor Max menor incrementando n. No obstante, reducir Max implica aumentar el número de renegociaciones necesarias, por lo que tampoco se debe incrementar el coste de señalización de forma incontrolada, sino que debe tenerse en cuenta la variabilidad de la tasa.

De entre las muchas posibles acciones para proteger al sistema frente a variaciones de α_{BE} sin incrementar el coste de señalización de forma excesiva proponemos el siguiente mecanismo:

- Inicialmente, el método de minimización es el que se ha descrito, pero se selecciona un valor $\varepsilon\big|_{\max}$.
- Max_{ini} permite la primera estimación de α_{BE} .
- Se utiliza un valor de n>0 para tener una cierta protección frente a variaciones de α_{BE} . El valor mínimo de n dependerá del total de memoria M del cliente.
- En las siguientes estimaciones de α_{BE}, se calcula el $\Delta\alpha_{BE}\big|_{\max}$ entre dos estimaciones. El cliente calcula n para protegerse ante $\Delta\alpha_{BE}\big|_{\max}$ y se compara con el necesario para garantizar $\varepsilon\big|_{\max}$. La n resultante más pequeña se selecciona para reducir, de esta forma, el número de

renegociaciones extra necesarias. Además si n es menor que el mínimo determinado por la memoria del cliente M, se selecciona este último valor.

- Cuando el valor de Max_n satisface $\varepsilon\big|_{\max}$, n se mantiene hasta el final de la transmisión, ya que ésta es la situación deseada (el error relativo es menor que el máximo escogido).

4.5 Experimentos de simulación para la comprobación de los resultados teóricos

En este apartado, se enumeran los distintos experimentos de simulación que se han llevado a cabo, para la comprobación de resultados de las teorías expuestas. Para la realización de estos experimentos de simulación se ha utilizado el simulador ns-2 [NS2], ampliamente utilizado en el campo de la simulación de redes IP. La implementación del sistema cliente-servidor de flujos semi-elásticos en el simulador se detalla en el Anexo A. En el Anexo B, se puede encontrar la descripción de una implementación simple en C++ utilizada para evaluar la complejidad de la implementación real del sistema. Para la comprobación de resultados de este apartado, el estudio se centra en la implementación realizada en el simulador ya que permite una mayor flexibilidad para recrear distintos escenarios.

Los experimentos que se enumeran a continuación revelan el comportamiento del sistema cliente-servidor de flujos semi-elásticos en diversas situaciones. Las primeras simulaciones pretenden ilustrar el funcionamiento del sistema cliente-servidor, mientras que las últimas muestran medidas del sistema bajo distintas situaciones de tráfico.

Todos los experimentos utilizan como protocolo de transmisión el TCP (*Transmission Control Protocol*)[xvi]. Este protocolo garantiza la fiabilidad de la información mediante retransmisiones y la secuencia en la transmisión haciendo uso de ordenación de los segmentos de datos recibidos cuando así sea precisa. Dado que se pretende que la transmisión sea en tiempo real, se supone éste como el peor caso, respecto al uso de protocolos no fiables.

[xvi] Concretamente, se utiliza la versión Tahoe disponible en el simulador ns-2. El efecto que observa al utilizar varias versiones de TCP es la diferencia entre las tasas de entrega de datos al cliente, siendo para el caso Tahoe la más pequeña, por lo que se estudia un peor caso.

4.5.1 Comportamiento genérico

El objetivo de estas simulaciones es el ilustrar el comportamiento del sistema cliente-servidor de forma genérica. Estos experimentos se realizaron para probar que la implementación en el simulador se comporta de manera similar al modelo teórico. Por otro lado, permitieron mostrar el comportamiento de la función de coste mediante simulación.

En la Figura 4.5 se pueden ver los distintos cambios de modo de transmisión que se observan en una simulación, donde el tráfico de *background* es CBR[xvii]. Como se aprecia, el umbral *Max* está prefijado a 1 M*byte*.

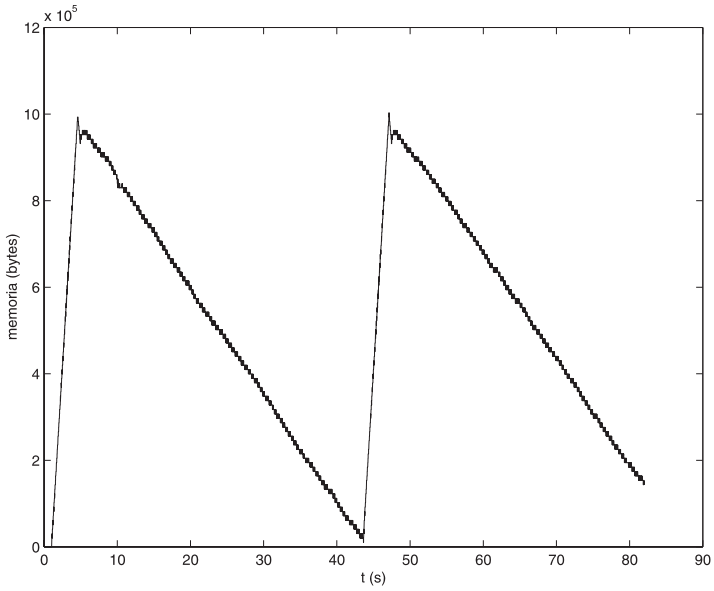

Figura 4.5: Cambios de modo de transmisión.

La Figura 4.6 muestra el coste por uso en función de *Max* obtenido por simulación para un tráfico CBR. En esta gráfica se observa claramente como a medida que el umbral *Max* disminuye, el coste aumenta debido a que la estimación

[xvii] Este tipo de tráfico genera paquetes de longitud constante con una periodicidad constante. En la simulación la generación de datos de *background* es la suficiente para conseguir que la tasa de llegada en el modo BE sea menor que la tasa de lectura del cliente.

de α_{BE} tiende a ser más pequeña[xviii]. Por otra parte, en la figura se superponen los resultados del cálculo analítico del coste utilizando la ecuación (4.3) y la estimación que realiza el cliente de la tasa α_{BE}. Nótese que está superpuesta debido a que el modelo analítico coincide prácticamente con el comportamiento real del sistema, cuando el tráfico de *background* es uniforme.

Finalmente, en la Figura 4.7 se amplia la zona del umbral *Min* en la finalización de una transmisión, para observar que el sistema cumple de forma bastante correcta el criterio de minimización. En esta figura, se puede apreciar claramente como evoluciona la ocupación de la memoria del cliente mediante incrementos iguales al tamaño del paquete utilizado para la transmisión[xix].

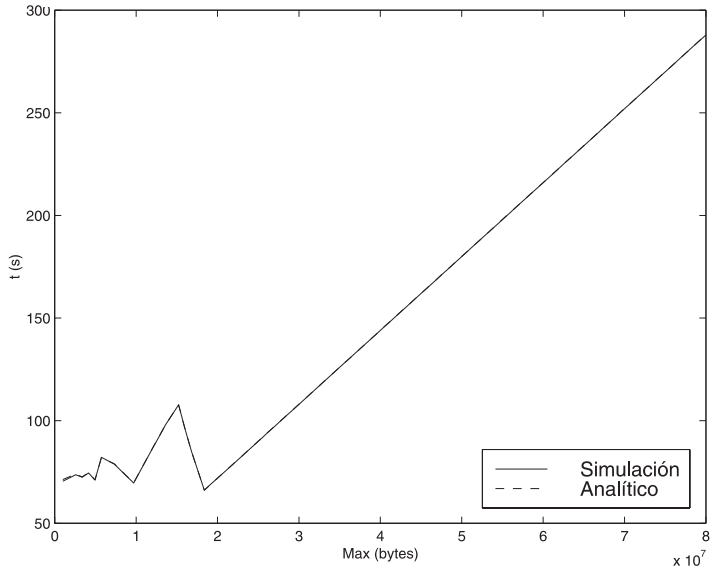

Figura 4.6: Coste por uso en función de Max.

[xviii] Obsérvese como el coste mínimo para los diferentes Max_n se incrementa al reducir *Max*, debido a que la estimación de α_{BE} es cada vez peor al reducirse el periodo en que se realiza dicha estimación.

[xix] En el siguiente apartado se explica la necesidad de utilizar estos paquetes para controlar la tasa de emisión del servidor.

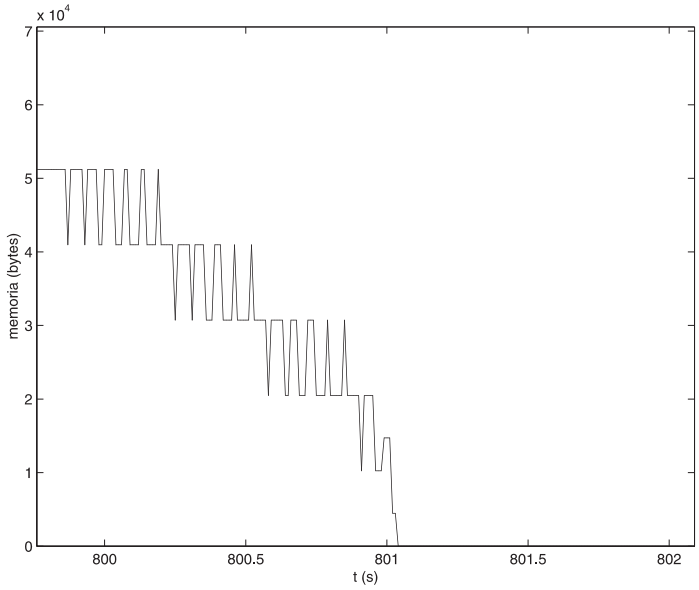

Figura 4.7: Criterio de Minimización (Min=10 Kbytes).

4.5.2 Control de Tasa

El sistema servidor de flujos semi-elásticos debe enviar la información en forma de paquetes para poder controlar la tasa de envío. Es decir, el volumen de información a enviar V_{serv} se dividirá en paquetes de longitud fija, para enviarlo al cliente. De esta manera, el servidor puede controlar la tasa de entrega de paquetes al nivel TCP que siempre será mayor que la de entrega de TCP a la memoria del cliente, ya que puede haber pérdidas de datos en la red.

Cuando el servidor envía los diversos paquetes mediante el protocolo TCP, éstos se almacenan en una determinada memoria, donde se segmentarán convenientemente para ser enviados, de forma fiable al cliente. Si uno de estos segmentos se pierde por el camino, se reenviará para asegurar que la información llega correctamente al extremo receptor.

El protocolo TCP intenta aprovechar todo el ancho de banda disponible, de manera que envía segmentos, siempre que éstos estén preparados para ser enviados (haya datos que enviar) y que la red lo permita (hay ancho de banda disponible sin congestión). Si la aplicación servidor indica el envío de toda la información requerida por el cliente, ésta se segmentará y se enviará lo más rápidamente que permita la red. En este caso, si la red presenta ρ=0 la información puede llegar más rápida que la velocidad de lectura del cliente y, por tanto, su memoria tenderá a llenarse. Esto puede provocar que esta memoria se desborde, o en algunos casos, donde pueda

existir interactividad entre cliente y servidor (y por tanto, se puedan pedir otros datos diferentes a los ya enviados), que la información enviada acabe por no utilizarse (habiéndose enviado por la red información innecesaria). Otro de los efectos nocivos que presenta el hecho de enviar un flujo de datos a una tasa mayor de la necesaria para la transmisión eficiente, es el exceso de tráfico en la red. Este exceso de tráfico, hace que el resto de conexiones presentes en el camino entre cliente y servidor, se puedan ver afectadas de tal forma que deban ajustar su tasa al ancho de banda disponible, aumentando el número de retransmisiones, si se quiere mantener la fiabilidad. Si el servidor no excede la tasa necesaria para enviar el flujo de datos al cliente, se minimizará este efecto.

Por otra parte, en el modo con reserva de recursos es necesario no sobrepasar la tasa que se ha negociado, ya que en este caso, el tráfico sobrante o bien se pierde, o se transmite con el modo BE[xx]. Esto implica de nuevo, afectar al resto del tráfico que usa el modo BE, sin que ello sea necesario.

El objetivo, por tanto, es conseguir que la tasa α_{BE} no supere la tasa de lectura del cliente α_r, y que en el modo con QoS no se supere la tasa reservada.

Para conseguir controlar estas tasas a nivel de aplicación, ha sido necesario implementar en el simulador ns-2 una interfaz adecuada con el protocolo TCP. Esta interfaz cumple los siguientes requisitos:

- La aplicación envía al TCP sólo un paquete[xxi] cada vez.
- El TCP controla el instante en que se permite a la aplicación enviar un paquete nuevo[xxii].

[xx] En las simulaciones, el tráfico que sobrepasa la tasa negociada se envía mediante el modo BE. Esto implica que para enviar tráfico en este modo, sólo es necesario negociar una tasa nula, sin necesidad de finalizar la sesión, reduciendo la señalización del RSVP (*resource ReSerVation Protocol*), que es el protocolo empleado para reservar recursos en el entorno simulado.

[xxi] Se entiende como paquete a los datos que envía la aplicación hacia el nivel TCP. Estos datos son partes (paquetes) del archivo (o mensaje) que se envía al cliente.

- Cuando es posible aceptar un paquete nuevo, la interfaz lo indica a la aplicación. La aplicación comprueba si puede enviar un paquete nuevo y si es posible lo envía[xxiii]. En caso de que no sea posible enviarlo calcula el instante en que se enviará el siguiente paquete y lo envía en ese instante.

En el caso de controlar α_{BE} el servidor lo puede hacer directamente, haciendo que no supere α_r. De esta forma no se superará la tasa de lectura del cliente y el tamaño de la memoria estará controlado. En el caso de controlar α_{ReR}, es necesario tener en cuenta la sobrecarga, o en términos anglosajones el *overhead*, que introduce el TCP, para conocer la tasa a nivel de aplicación. La primera suposición que se realiza es que en el modo con QoS no se pierden segmentos (o al menos, las pérdidas son muy pequeñas), es decir, no habrán retransmisiones, por lo tanto la información que circulará a nivel IP, será la misma que a nivel de aplicación más el *overhead* producido por las cabeceras de los protocolos TCP/IP.

El protocolo IP introduce una cabecera de 20 *bytes* en IPv4 y de 40 *bytes* en IPv6. También introduce cabeceras de extensión, que se añaden en caso de utilizar ciertas opciones del protocolo. Por su parte el protocolo TCP introduce una cabecera de 20 *bytes* a la que hay que sumarle las posibles opciones que utilice TCP. Una vez se conoce el tamaño total de las cabeceras introducido por TCP/IP es necesario conocer el tamaño del segmento que se utiliza. Por defecto se utiliza un tamaño de 536 *bytes* o tal que el datagrama IP tenga el tamaño de la mínima MTU (*Maximum Transfer Unit*) presente entre cliente y servidor [COM95].

Cuando se conocen estos parámetros es posible obtener el *overhead*, de manera que se pueda emitir información a nivel de aplicación sin sobrepasar la tasa reservada (α_{Res}). En las simulaciones realizadas se utiliza una cabecera TCP/IP de 40 *bytes* (IPv4) y un tamaño de segmento máximo de 536 *bytes*. Esto implica un *overhead* de 40/536 *bytes*, por lo que la tasa a nivel de aplicación (α_{ReR}) deberá ser:

[xxii] Este instante es el momento en que la ventana de transmisión del TCP alcanza el final del paquete. A partir de este momento, el próximo segmento que se pueda enviar debe ser del siguiente paquete (Véase el Anexo B).

[xxiii] Se podrá enviar un nuevo paquete si no se supera la tasa máxima a la que la aplicación puede enviar datos.

$$\alpha_{ReR} = \frac{536}{576}\alpha_{Res} \tag{4.28}$$

En un sistema real, sería posible implementar algún tipo de mecanismo que permitiera calcular de forma automática el *overhead* para que este dato pudiese ser utilizado por el servidor.

Para comprobar que el funcionamiento del sistema es el previsto cuando la carga es 0, se realizaron simulaciones con tráfico de relleno (*background*) CBR (*Constant Bit Rate*). Este tráfico permite que los segmentos que envía el servidor lleguen de forma bastante uniforme y que por tanto, las tasas de llegada de paquetes sean también suficientemente uniformes. En la Figura 4.8, se muestra una simulación en la que se comprueba que la tasa de llegada de paquetes en modo ReR es igual a la tasa de entrega del servidor (4 *Mbit/s*). Además, en el periodo BE, aunque la red permite enviar datos más rápidamente que la tasa de lectura del cliente, el servidor evita que la tasa supere ese valor para no saturar la memoria del cliente.

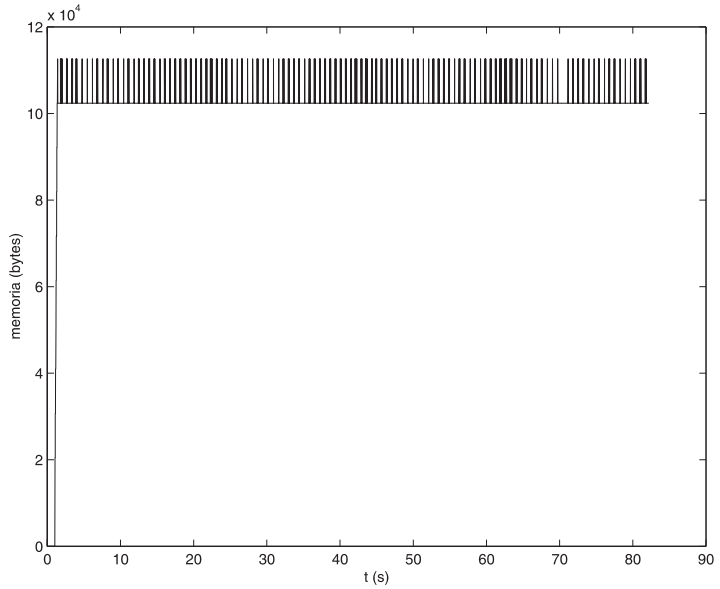

Figura 4.8: Control de Tasa.

4.5.3 Uso de Max_{ini}

El cliente de flujos semi-elásticos propuesto, utiliza un umbral *Max* inicial para poder calcular la tasa α_{BE} en el siguiente periodo BE, tal y como se explicó en el apartado 4.2.2. Este umbral puede afectar a la estimación que se realiza, de tal

forma que si el subsiguiente periodo BE es demasiado pequeño, la estimación de la tasa α_{BE} puede ser también demasiado pequeña. Esto origina que el umbral *Max* estimado sea demasiado grande, es decir, el cliente interpreta que la red está más cargada y eleva el tiempo en modo ReR, con el consecuente incremento de coste. En las simulaciones realizadas en este apartado, se muestra como influye el umbral Max_{ini} a la estimación de la tasa α_{BE} en diversas situaciones.

En la Figura 4.9 se observa como para un Max_{ini} = 10 *Kbytes*, el primer umbral *Max* que se estima es mayor que el segundo debido a que la tasa α_{BE} estimada es menor que la real. Para solventar este problema, el sistema reduce *Max* en la segunda estimación. Como se observa en la Figura 4.10, el segundo umbral *Max* es más parecido al primero cuando las estimaciones son más semejantes.

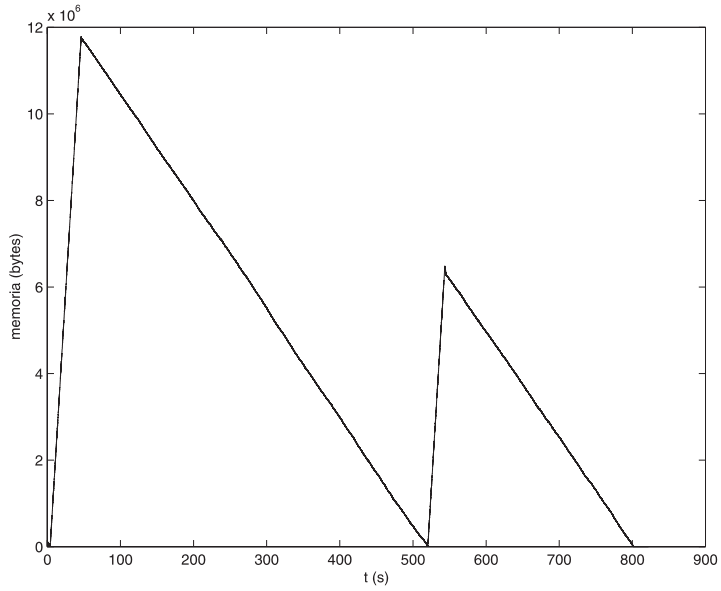

Figura 4.9: Umbral Max_{ini} pequeño (10 Kbytes).

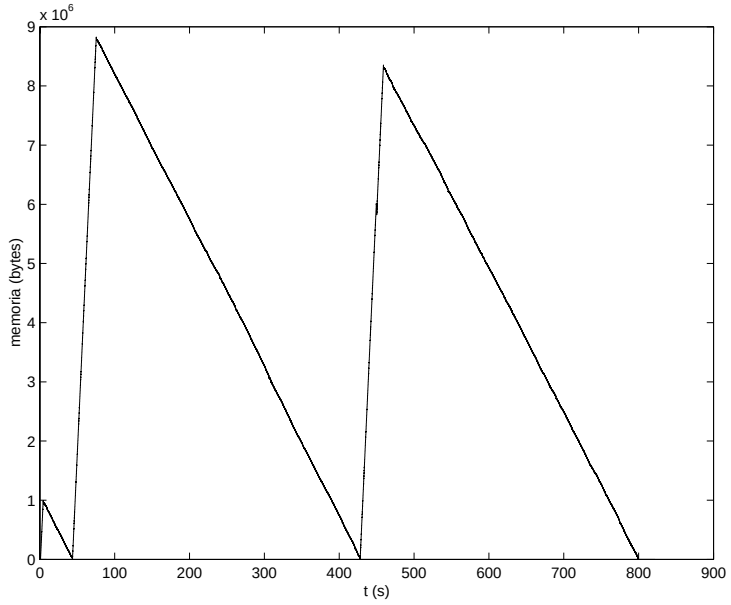

Figura 4.10: Umbral Max_{ini} mayor (100 Kbytes).

Tal y como se observa, tanto en la Figura 4.9 como en la Figura 4.10, el criterio de minimización se cumple gracias que no se utiliza Max_0, lo que permite proteger el sistema frente a variaciones de α_{BE} (4.27).

4.5.4 Tamaño del paquete y de la memoria del cliente

De forma similar a lo visto en el apartado anterior, las estimaciones de las tasas de llegada de paquetes dependen del tamaño o longitud del paquete (L) que se utiliza para enviar información entre servidor y cliente y del tamaño total de la memoria del cliente (M).

Supongamos el caso extremo en que L=M (Figura 4.11) donde se cambia de modo cada vez que llega o se consume un paquete. En este caso la información llega o se consume en el mismo instante en que se cambia de periodo, por lo que la estimación de las tasas α_i y α_o es inadecuada ya que en los periodos ReR y BE se recibe o se consume sólo un paquete.

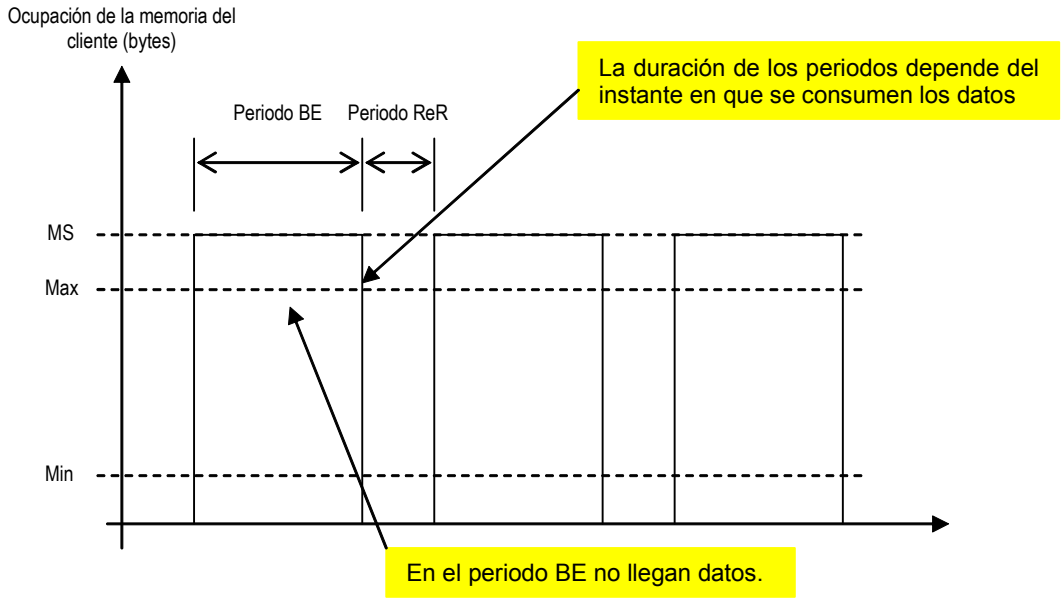

Figura 4.11: Ejemplo donde la longitud de un paquete (L) es igual a la memoria del cliente (M).

El otro caso extremo es aquel en el que $L=dL$ (diferencial de L)[xxiv]. En este caso el tamaño de paquete es infinitésimo, y por tanto, la estimación de las tasas será perfecta.

Llegado a este punto, también es importante remarcar el efecto que puede tener el tráfico de *background* en las estimaciones. Tal y como se vio en el apartado 4.4, si la llegada de datos en el periodo BE presenta variabilidad, el cliente puede tener dificultades para minimizar el coste de la transmisión. Esta variabilidad, dependerá de la carga de red y de las características del tráfico de *background* que circule por la red. Fuera del ámbito de este trabajo queda el valorar la exactitud de los diversos modelos de tráfico que caracterizan el tráfico en Internet.

Para la realización de este estudio, se recurre al uso de diversos patrones de tráfico que provocan que la tasa de llegada de paquetes en modo BE presente diversos grados de variabilidad, independientemente de la validez del modelo como patrón de tráfico de Internet. En particular, se recurre a cuatro patrones de tráfico ya implementados para las distribuciones del simulador ns-2.

[xxiv] Como en el ejemplo de la Figura 4.2 y en otros, donde la ocupación de la memoria del cliente es continua.

El primer tipo de tráfico es CBR y se incluye con la distribución estándar del simulador ns-2. Este tipo de tráfico genera paquetes de forma uniformemente distribuida en el tiempo, lo cual proporciona la mínima variabilidad posible para la tasa de llegada de paquetes en modo BE.

El segundo patrón es un modelo de tráfico de Pareto, también incluido en la distribución estándar del ns-2. Éste es un patrón de tráfico bien conocido por modelar la longitud de los documentos Web [CRO95]. Este tipo de tráfico presenta una mayor variabilidad que el tráfico CBR.

Otro patrón utilizado en estas simulaciones es un modelo para tráfico HTTP 1.1 (*Web Traffic*), que simula uno de los protocolos más usuales en Internet. Este modelo está basado en el generador de tráfico Web SURGE [BAR98], y el código para ns-2 se puede encontrar en [NS2]. Como se verá a continuación, la variabilidad que provoca este tráfico es similar al de Pareto.

Finalmente, se propone el uso de una traza de vídeo MPEG-4 (*Star Wars IV*) [MPEG]. Las trazas de vídeo MPEG-4 generan tráfico a ráfagas, ya que el proceso de codificación produce tasas de *bit* variables dependiendo de la actividad y cambios en las escenas. Este tráfico se utiliza para conseguir alta variabilidad en la tasa de llegada de paquetes. En la Tabla 4.1 se muestran los diferentes parámetros usados para las distintas fuentes de tráfico de *background*.

Para comprobar cómo afectan los parámetros L y M al método de minimización, el estudio se centra en la eficiencia del método (η). Se supone un coste de señalización $C_s = 0$ para centrar el estudio en el efecto de L y M sobre la estimación de Max_n dependiente de las estimaciones de la tasa α_{BE}. Por tanto, la expresión de la eficiencia máxima, en este caso, será cuando el coste obtenido sea el coste mínimo (4.29).

$$\eta = 1 - \frac{C_{\min}}{V} = 1 - \frac{\alpha_r \rho + \alpha_r (1-\rho)\dfrac{Min}{V}}{\alpha_r (\alpha_{ReR} - \alpha_r (1-\rho))}\alpha_{ReR} \tag{4.29}$$

Por otra parte, se realiza el experimento para todos los cuatro tipos de tráfico comentados anteriormente, con el objetivo de ver como afecta la variabilidad al método de minimización.

Se realizan dos experimentos. En el primero, se muestra como evoluciona η para diferentes tamaños de paquete (L) y diferentes tamaños de memoria del cliente

(*M*). El tráfico de *background* hace que el cliente vea una carga ρ diferente para cada caso, ya que es difícil, a priori, conseguir la misma carga, puesto que ésta se ve afectada por las pérdidas y las retransmisiones del TCP. De cualquier forma, la eficiencia obtenida en todas las simulaciones se sitúa en el margen de 0.05 a 0.25, para un tamaño de archivo (*V*) de 300 *Mbytes* y cargas que oscilan entre el 80% y el 90%[xxv].

El tamaño de la memoria del cliente (*M)* queda fijado en el extremo cliente, que es el que decide la cantidad de memoria que se utiliza para la aplicación, en función de la disponibilidad de la misma y de sus recursos. Se escoge para las simulaciones un rango bastante amplio de tamaño de memoria: desde 200 *Kbytes* hasta 200 *Mbytes*. En este experimento, la variación de *M* afecta a la *n* escogida por el sistema para el cálculo de Max_n. A medida que *M* decrece, *Max* tiene que ser más pequeño, incrementando el número de renegociaciones, y reduciendo el periodo BE utilizado para realizar la estimación de α_{BE}. Los resultados obtenidos muestran que exiten rangos de *M*, en los que la eficiencia es prácticamente constante, ya que el cliente escoge el mismo umbral *Max* en todo el rango, debido a que *M* todavía no es suficientemente grande como para permitir un Max_n superior. Esto muestra que es innecesario para el cliente el dimensionar *M* en un valor entre Max_n y Max_{n-1}.

El tamaño del paquete (*L*) refleja cuan grandes son los incrementos y decrementos de ocupación en la memoria del cliente, y por tanto, cuanto mayor sea, peor serán las estimaciones. *L* es un parámetro que controla el servidor, ya que es el tamaño fijo que utiliza para ir enviando paquetes a la red destinados al cliente. El rango de valores escogido para realizar las simulaciones va desde paquetes pequeños, en torno a 500 *bytes*, hasta paquetes considerablemente grandes, del orden de 10000 *bytes*.

[xxv] Obsérvese que este caso es interesante desde el punto de vista del proveedor de red, ya que la red está muy utilizada. Sin embargo, desde el punto de vista del proveedor de servicio es una mala situación ya que el coste puede verse poco reducido.

Parámetros de la fuente CBR	**Distribución**	**Valor medio**
Tamaño de paquete	Constante	500 *bytes*
Tasa	Constante	Se incrementa con ρ
Parámetros de la fuente Pareto	**Distribución**	**Valor medio**
Tamaño de paquete	Constante	500 *bytes*
Tiempo de ráfaga	Pareto	1000 ms
Tiempo de inactividad	Pareto	500 ms
Tasa	Constante	Se incrementa con ρ
Parámetros de la fuente HTTP 1.1	**Distribución**	**Valor medio**
Tiempo de inactividad (OFF)	Pareto (α= 5, β = 2)	10 s
Número de componentes por página	Pareto (α = 1, β = 1.5)	3
Tiempo de actividad (ON)	Pareto (α = 0.167, β = 1.5)	0.5 s
Tamaño de los componentes de la página	Pareto (α = 2, β = 1.2)	12 *Kbytes*

Parámetros de la fuente MPEG-4 (*Star Wars IV*)	**Valor**
Tamaño de archivo	$120 \cdot 10^6$ *bytes*
Duración del vídeo	1 hora
Número de cuadros	89998
Tamaño medio de cuadro	$1.4 \cdot 10^3$ *bytes*
Varianza del tamaño de cuadro	$8.2 \cdot 10^5$
Covarianza del tamaño de cuadro	0.66
Tamaño de cuadro Mínimo	26 *bytes*
Tamaño de cuadro Máximo	9370 *bytes*
Tasa de *bit* media	$2.8 \cdot 10^5$ *bits*/s
Tasa de *bit* de pico	$1.9 \cdot 10^6$ *bits*/s

Tabla 4.1: Parámetros del tráfico de background.

Una forma de observar la evolución de la eficiencia del sistema en función de estos dos parámetros L y M, es la representación gráfica en 3 dimensiones de los resultados obtenidos en cada caso[xxvi]. Estas gráficas se han obtenido mediante representación de los datos obtenidos con el simulador ns-2 y convertidos a matrices de valores, representándose con el programa MATLAB (versión 6.1).

La Figura 4.12 muestra los resultados para tráfico de *background* CBR. Este tráfico presenta la menor variabilidad y consecuentemente, los resultados son muy similares para todas las distintas situaciones simuladas. La eficiencia es mayor

[xxvi] En las gráficas se representan aproximadamente 1600 puntos por gráfica.

cuando el tamaño de paquete usado es pequeño y se puede apreciar una mayor variación a medida que *L* se hace comparable a *M*.

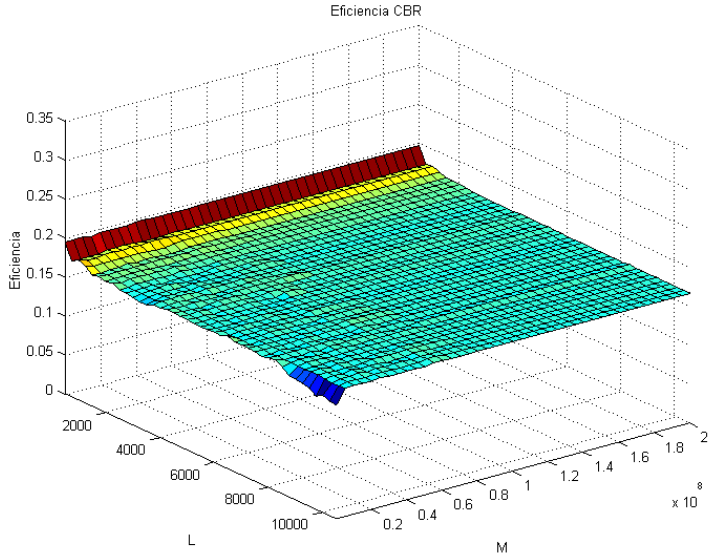

Figura 4.12: Eficiencia vs. M y L para tráfico de background CBR.

En la Figura 4.13 (tráfico de Pareto), los resultados muestran una mayor variabilidad que en el caso CBR. Este tráfico de *background* presenta mayor variación a medida que cambia el tamaño del paquete. De nuevo, para tamaños pequeños la eficiencia alcanza valores mayores. Sin embargo, tal y como muestra la Figura 4.13, el valor obtenido para tamaños de paquete similares varía más que en el caso CBR. En la dimensión *M*, se aprecia que para diferentes valores de *Max*, la eficiencia es casi la misma pero con variación un poco mayor que para el caso CBR. No obstante, esta variación es muy pequeña (2% de eficiencia).

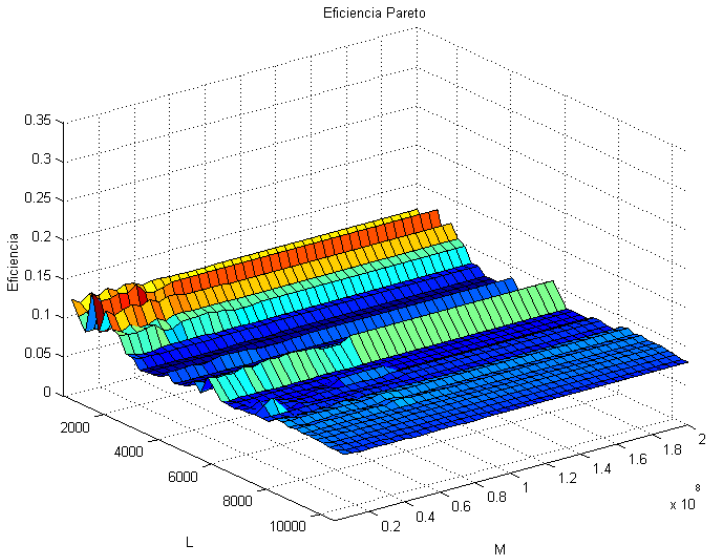

Figura 4.13: Eficiencia vs. M y L para tráfico de background Pareto.

Cabe reseñar que las simulaciones con tráfico HTTP 1.1 se han realizado con un tamaño de archivo a enviar V = 20 *Mbytes* (en lugar de 300 *Mbytes*). Ello es debido a los elevados requerimientos computacionales que estas simulaciones requieren[xxvii] y que provocaban errores en el simulador. Además, el coste temporal de dichas simulaciones era inviable, ya que cada simulación con 300 *Mbytes* requería un tiempo real de simulación del orden de 5 días[xxviii], utilizando un PC con 512 *Mbytes* de memoria y un procesador a 800 Mhz. Con 20 *Mbytes* de tamaño de archivo, cada simulación tardaba unos 50 minutos de tiempo de *CPU*, lo que equivalía en tiempo real a unas 2.5 horas, lo cual es considerablemente menor que el caso anterior[xxix].

[xxvii] Estas fuentes de tráfico están programadas en oTCL. Al ser este un lenguaje interpretado los tiempos de simulación se incrementan considerablemente, a diferencia del resto de fuentes implementadas en C++.

[xxviii] Obsérvese que este tiempo es estimado y teórico, ya que ninguna de estas simulaciones logró terminar debido a la sobresaturación del sistema y el consiguiente final del proceso.

[xxix] Cada gráfica contiene 1600 puntos por lo que se necesita del orden de medio año para finalizar. Para reducir este tiempo, se utilizaron sólo 400 puntos y 3 procesadores.

Efectuada esta aclaración, cabe comentar que para obtener una carga del orden del 80% debida al tráfico de *background* de la red se necesitaron lanzar 300 sesiones HTTP en paralelo, con los parámetros de la Tabla 4.1.

Como se puede apreciar en la Figura 4.14, la eficiencia obtenida no es tan plana como en los casos anteriores, la variabilidad que se aprecia no es demasiado considerable. Por tanto, se puede observar cómo el sistema responde correctamente para diversos tamaños de memoria o distintas longitudes de paquete, pudiendo así adaptarse a las circunstancias particulares de cada aplicación y cliente.

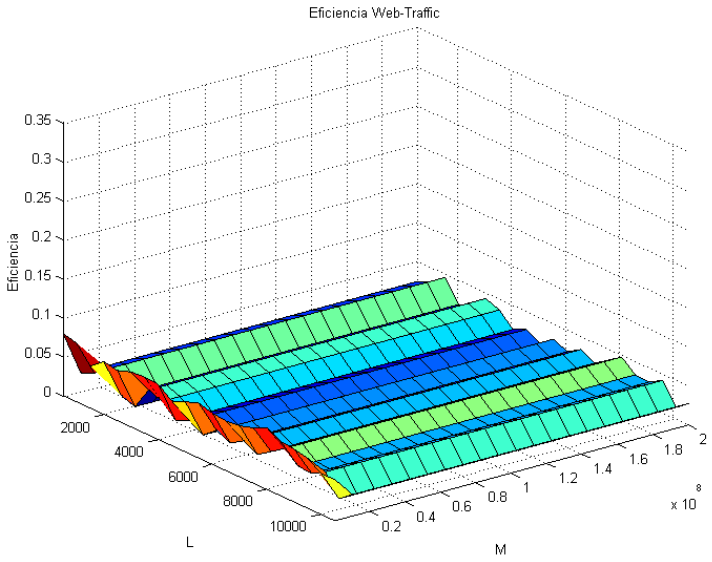

Figura 4.14: Eficiencia vs. M y L para tráfico de background HTTP 1.1.

La Figura 4.15 representa los resultados para tráfico de *background* MPEG-4. Se necesitaron 12 fuentes de vídeo individuales para obtener un 80% de carga. En este caso, el tráfico es muy variable debido a las características del tráfico MPEG-4. Esta característica incide en la eficiencia del método, ya que es más difícil conseguir estimaciones adecuadas de la tasa para calcular un umbral *Max* apropiado para cumplir el criterio de minimización. En este caso la diferencia entre usar un paquete pequeño o grande es menor, ya que el principal problema es la variabilidad en las pérdidas y retransmisiones de los datos debido a la influencia del tráfico de *background*. Esto provoca estimaciones similares para varios tamaños de paquete. Observando la dimensión *M,* está claro que la eficiencia obtenida para diferentes valores de *Max* es más variable que para los casos anteriores. Una vez más, esto es debido a la variabilidad en la llegada de paquetes al cliente y las estimaciones resultantes. Nótese que cuando *M* es pequeño, la variabilidad decrece. Este efecto

ocurre para valores de *Max* pequeños, ya que la probabilidad de cumplir el criterio de minimización es mayor[xxx].

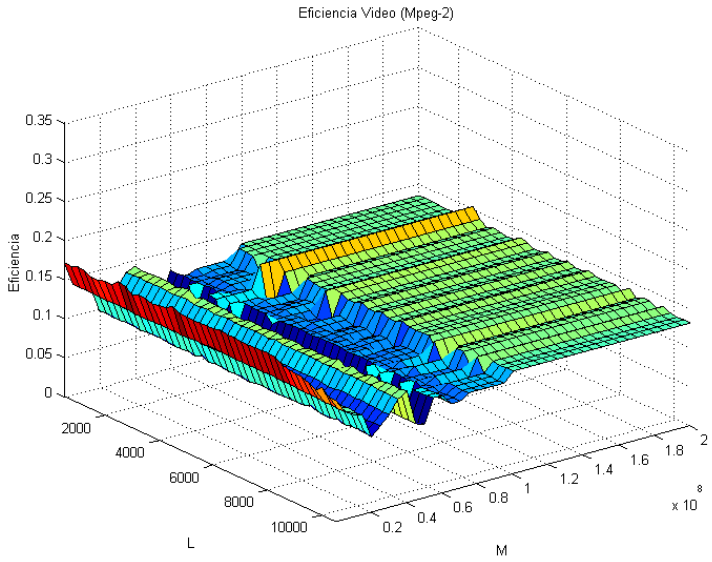

Figura 4.15: Eficiencia vs. M y L para tráfico de background MPEG-4.

El segundo experimento, pretende comparar la evolución de η en función de la carga ρ para los diferentes tipos de tráfico de *background*. Para el tráfico CBR y Pareto, se varía ρ ajustando la tasa de paquetes por unidad de tiempo, mientras que para los tráficos MPEG-4 y HTTP 1.1 se varía el número de fuentes simultáneas y desincronizadas entre ellas. La Figura 4.16 muestra los resultados de simulación. Los puntos representan los resultados obtenidos de forma analítica mediante la ecuación (4.29) y como se puede observar son los mejores valores de eficiencia posibles. Todos los resultados de simulación están por debajo de los analíticos debido al efecto de la variabilidad del tráfico. Como se observa en la Figura 4.16, la diferencia entre los valores analíticos y los de simulación para todos los tráficos son bastante parecidos para rangos de carga entre 0.4 y 1, mientras que para cargas bajas hay mayor diferencia, especialmente cuando el tráfico de *background* es muy variable. En realidad, el caso de carga baja es el mejor desde el punto de vista del proveedor de servicio, ya que la reducción de coste es mayor que para cargas altas, pero para los operadores de red es más interesante ofrecer el servicio BE para el

[xxx] Los datos almacenados cuando acabe la transmisión estarán más cerca del umbral *Min*.

mayor número posible de usuarios, lo que implica cargas altas[xxxi] que harán que el servicio ofrecido sea peor que el ofrecido mediante reserva de recursos. Por tanto, para cargas altas (0.4-1) el método presenta un comportamiento bastante correcto.

Para cargas bajas, en particular con tráfico MPEG-4, la máxima diferencia con respecto a los resultados analíticos es de un 10% de eficiencia. Esto significa que para tráfico de *background* muy variable, la reducción del coste puede verse reducida de forma notable, con respecto a tráficos de *background* más uniformes, por lo que será necesario aplicar técnicas de control para mejorar la reducción del coste como se verá a continuación. Para este caso es importante resaltar, que esta situación, no será habitual en redes de área extendida, ya que en la red troncal, la agregación de tráfico tiende a suavizar el patrón.

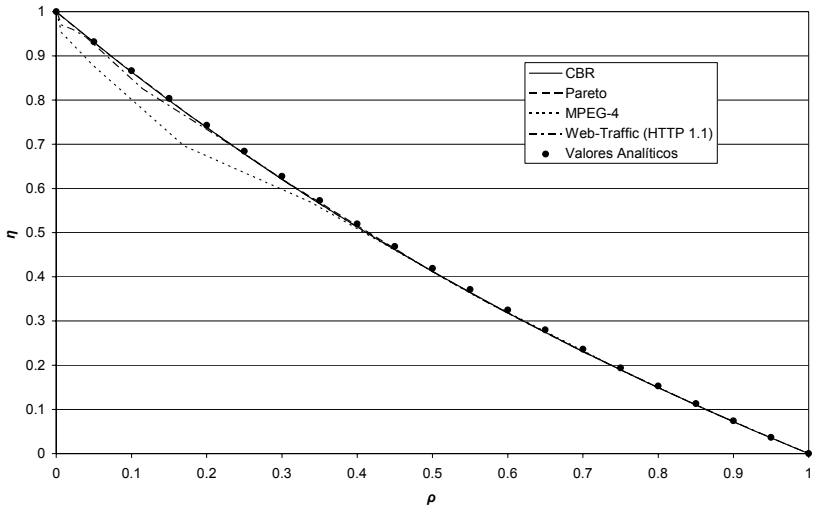

Figura 4.16: Eficiencia vs. ρ para diferentes tipos de tráfico de background.

4.5.5 Variaciones de α_{BE} (distintos tipos de tráfico): Ocupación de la memoria, Efectividad, Exactitud, N_{ReR}.

Como se vio en el apartado 4.4, es posible reducir el umbral *Max* hacia el final de la transmisión para incrementar la exactitud y eficiencia del método de minimización. En este experimento, se pretende evaluar el comportamiento del método de minimización usando el algoritmo del apartado 4.4 para optimizarlo en esta situación. Para ello, se define la exactitud (E) (4.30) como la efectividad del

[xxxi] En las simulaciones una carga de 0.9 equivale a una tasa de llegada de paquetes igual a 50 kb/s.

método para conseguir un coste igual a C_{min}, debido al cálculo de Max_n, es decir no se tiene en cuenta el coste de señalización. Por tanto, E=1 indica que el cliente ha conseguido un coste mínimo, mientras que E=0 equivale a un coste máximo.

$$E = \frac{C_{\max} - C}{C_{\max} - C_{\min}} = \frac{V - C}{V - C_{\min}} \tag{4.30}$$

Para evaluar el método se emplean, en las simulaciones, los tipos de tráfico de *background* presentados en el apartado 4.5.4, y un tamaño de archivo a enviar por el servidor de 100 *Mbytes*. Por otra parte, el comportamiento se evalúa con dos tamaños de memoria del cliente distintos (200 *Kbytes* y 20 *Mbytes*).

La Figura 4.17 y la Figura 4.18 muestran como en un cliente con una memoria M=200 *Kbytes, E* evoluciona para diferentes valores de ρ y para cada tipo de tráfico de *background*. En la Figura 4.17 se presenta el caso en el que se utiliza el método original, mientras que en la Figura 4.18 el método utiliza el algoritmo del apartado 4.4, $\text{con}\,\varepsilon\big|_{\max} = 0.01$. La Figura 4.19 y la Figura 4.20 representan el número de periodos ReR (N_{ReR}), que se necesitan en cada situación para observar el incremento de renegociaciones necesario al utilizar el nuevo método. La Figura 4.21, la Figura 4.22, la Figura 4.23 y la Figura 4.24 muestran el caso con M=20 *Mbytes*.

Como se puede observar, el nuevo método se comporta de forma muy similar al método original y el número de periodos ReR son prácticamente los mismos en los dos casos. Nótese que en esta situación *Max* debe ser menor de 200 *Kbytes*, y por tanto, el número de periodos ReR será mayor que en el caso con memoria de 20 *Mbytes* (Figura 4.23 y Figura 4.24). Asimismo, se aprecia que E siempre permanece por encima del 94%, ya que la protección contra variaciones de α_{BE} con este umbral *Max* es suficientemente buena (Ecuación (4.25)), por lo que el nuevo método no introduce nada más que unas pocas renegociaciones en algunos casos.

Para el caso de M=20 *Mbytes*, el método original se muestra muy inexacto, especialmente cuando el tráfico es muy variable[xxxii]. Usando el nuevo método, la exactitud crece hasta el 95% para todos los casos, incrementando el número de periodos ReR en aproximadamente 5.

[xxxii] Para tráfico MPEG-4 llega al 50% y para el resto, al 70%.

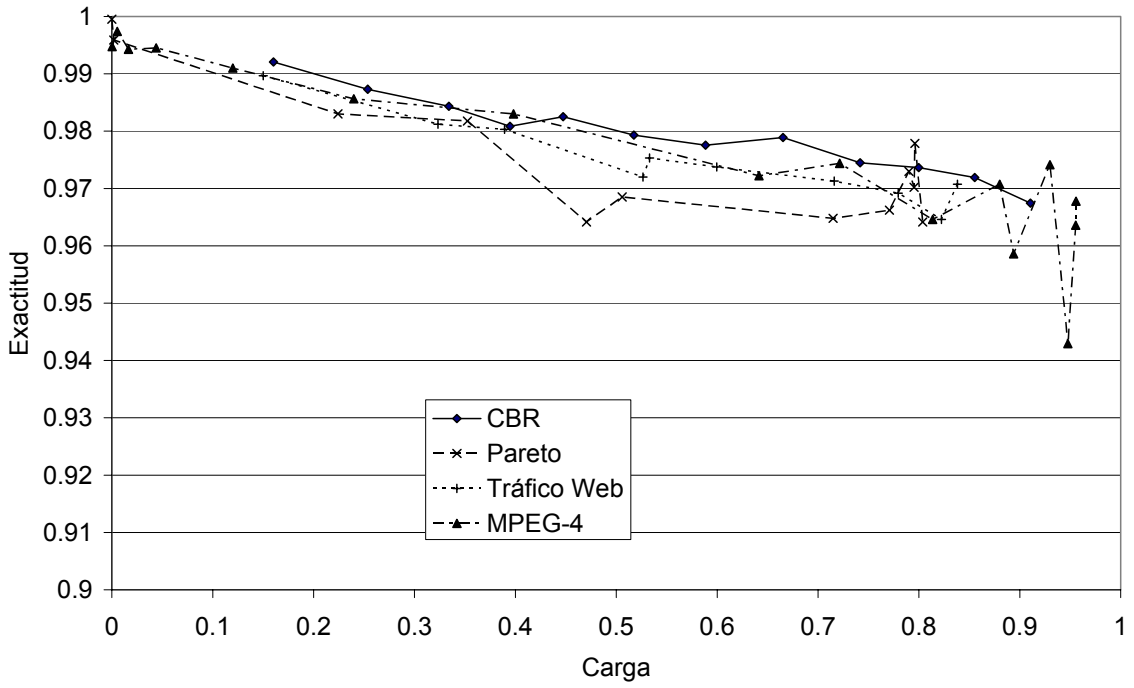

Figura 4.17: Exactitud del método (M=200 Kbytes, método original).

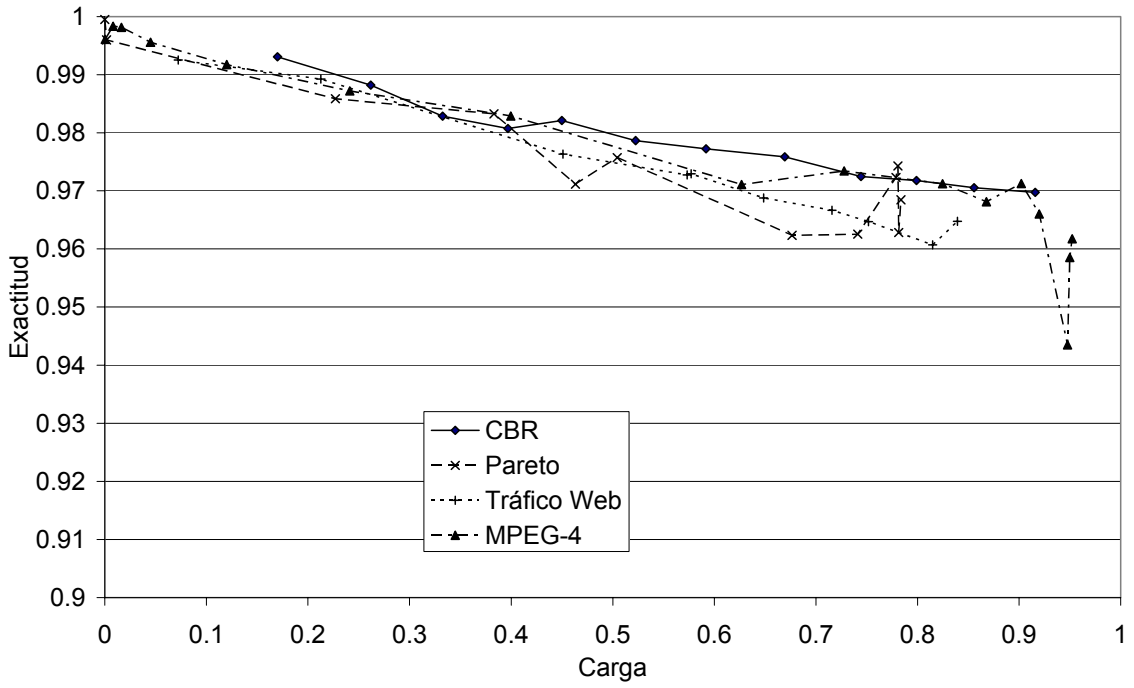

Figura 4.18: Exactitud del método (M=200 Kbytes, método nuevo).

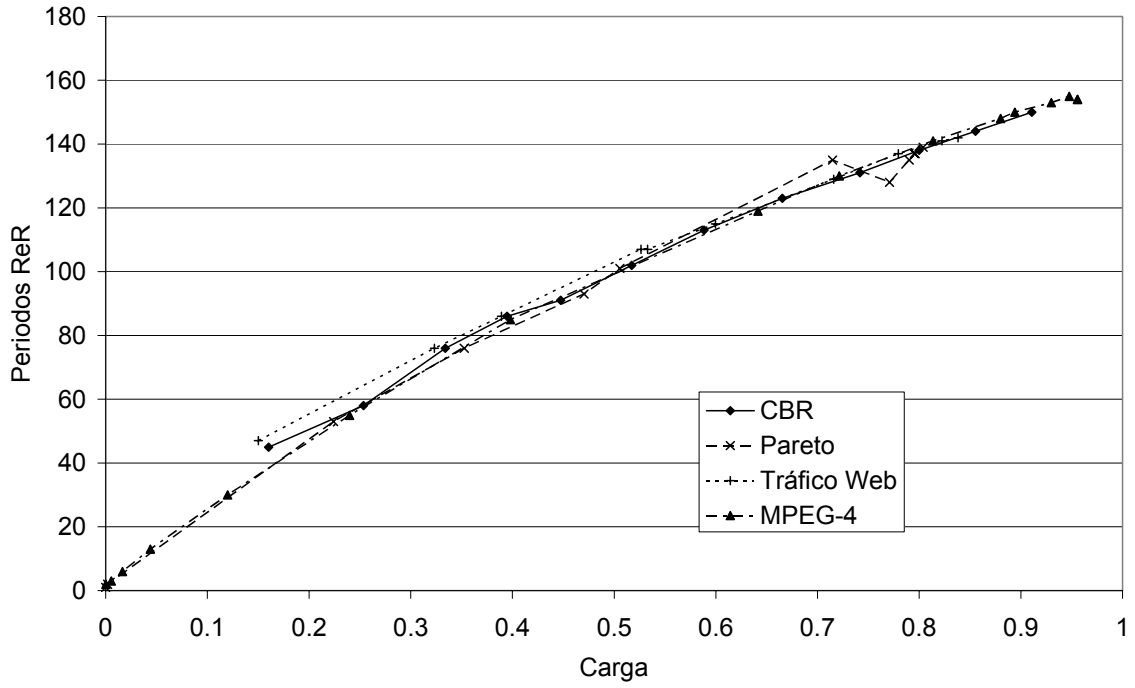

Figura 4.19: Número de periodos ReR (M=200 Kbytes, método original).

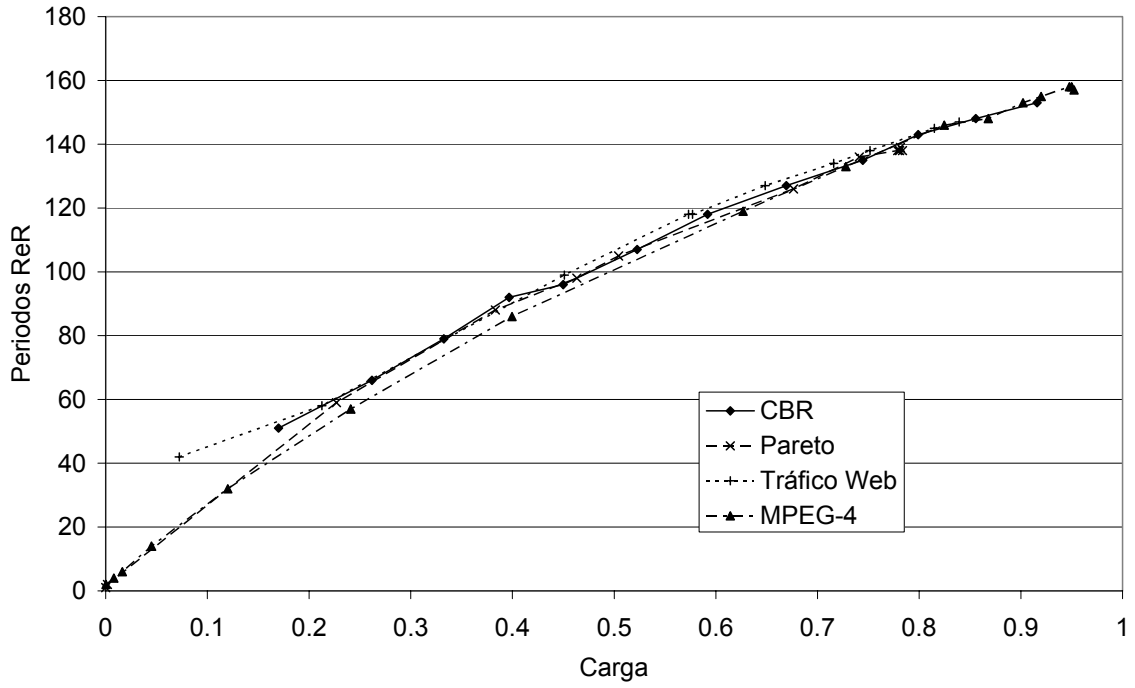

Figura 4.20: Número de periodos ReR (M=200 Kbytes, método nuevo).

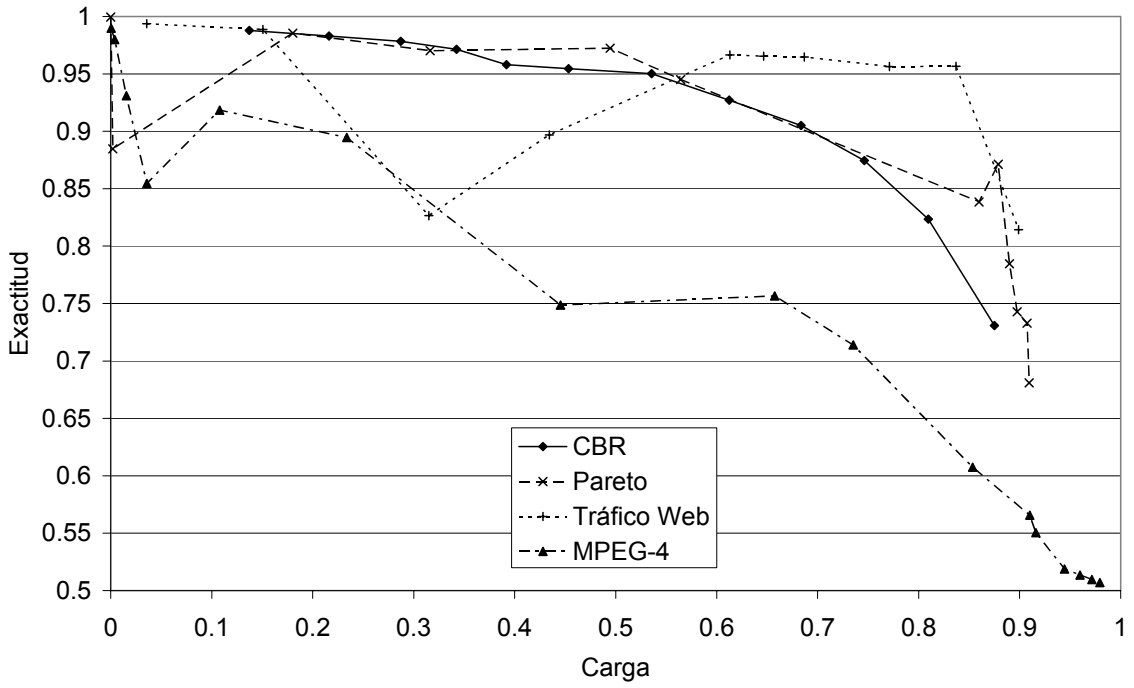

Figura 4.21: Exactitud del método (M=20 Mbytes, método original).

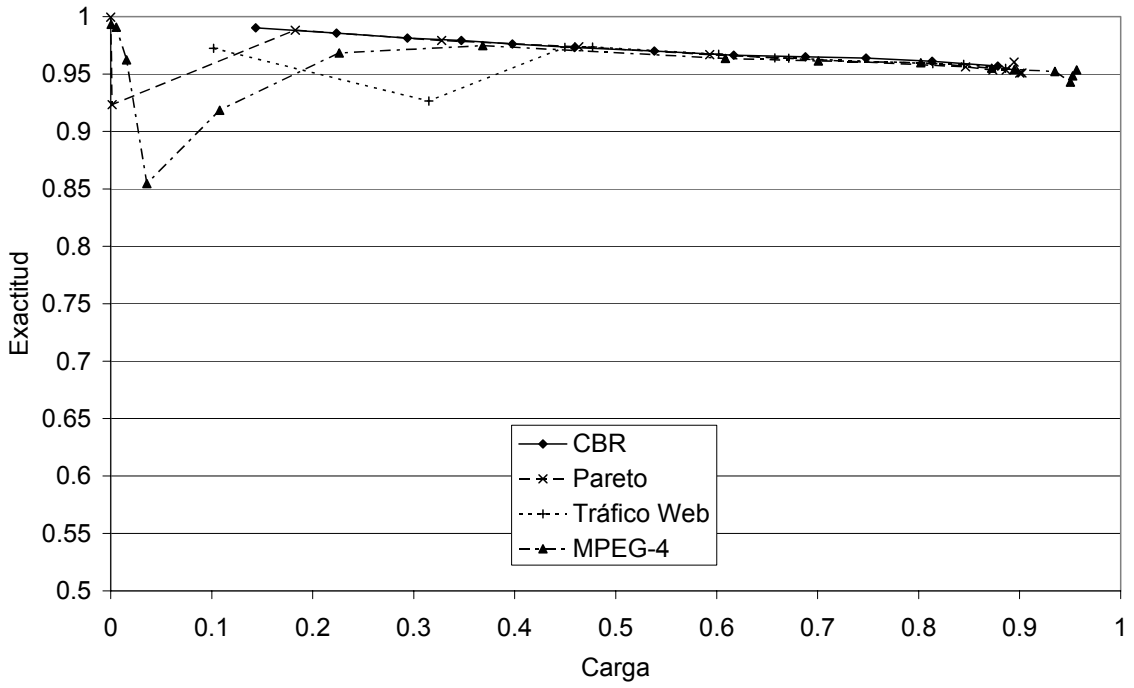

Figura 4.22: Exactitud del método (M=20 Mbytes, método nuevo).

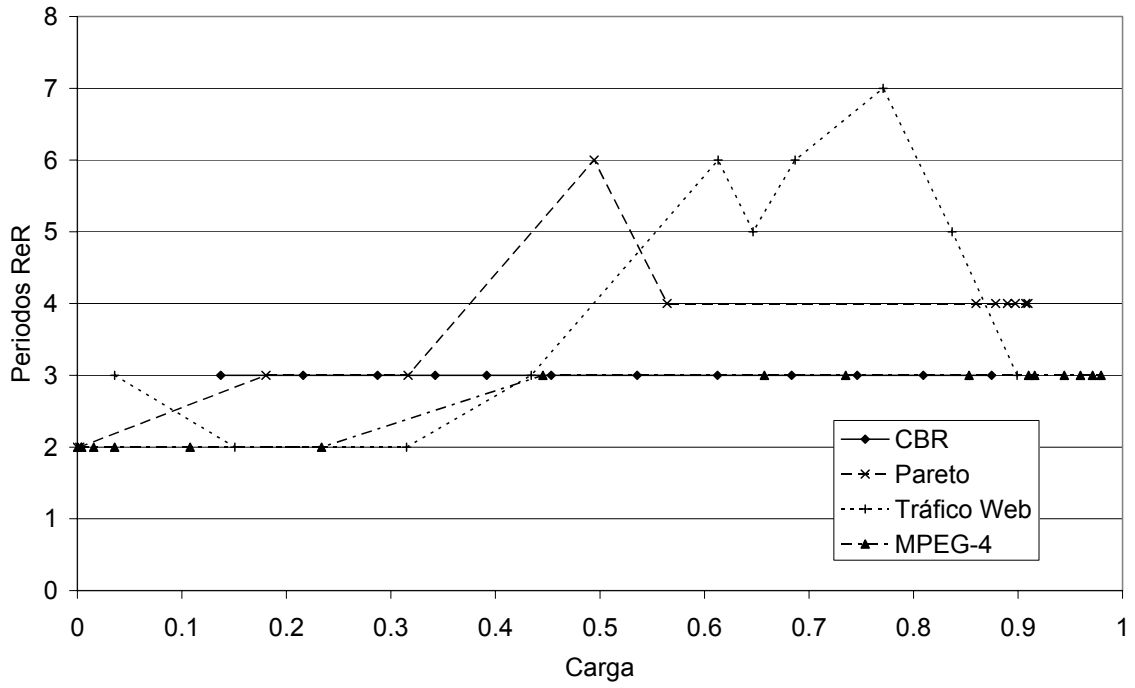

Figura 4.23: Número de periodos ReR (M=20 Mbytes, método original).

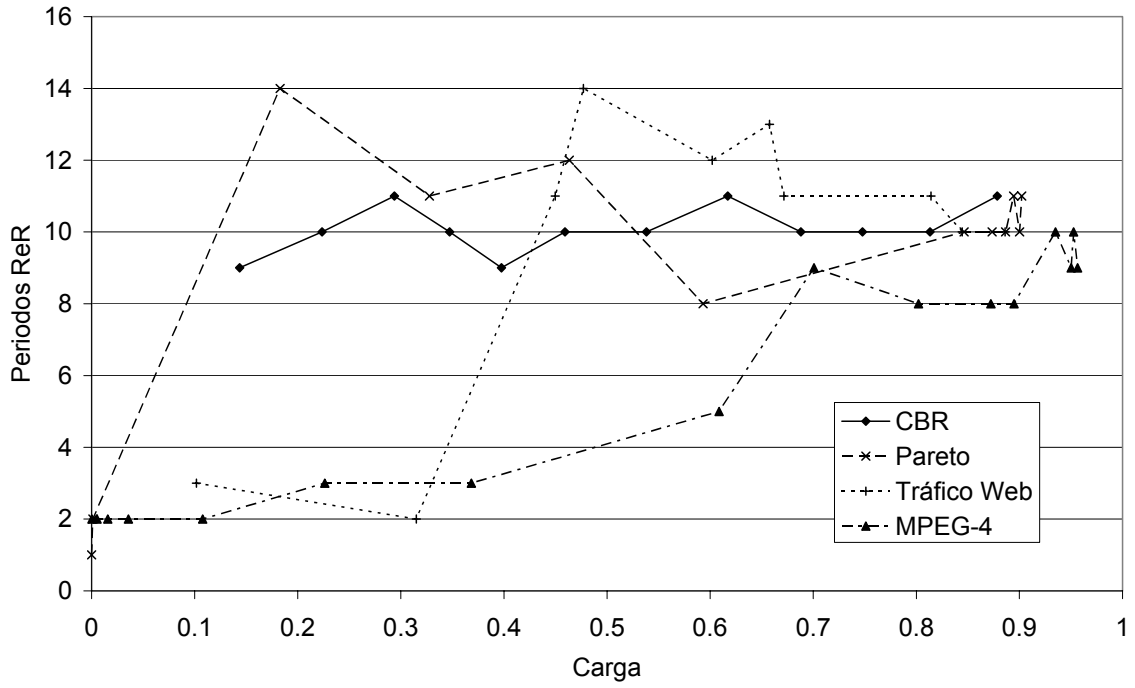

Figura 4.24: Número de periodos ReR (M=20 Mbytes, método nuevo).

Para mostrar como actúa el nuevo método se muestran tres ejemplos de la evolución de la ocupación de la memoria del cliente. En el primer ejemplo (Figura 4.25), se observa como con un tráfico de *background* CBR, el nuevo método reduce el

umbral *Max* hasta que se cumple $\varepsilon\big|_{\max} < 0.01$. Como se advierte, en la última renegociación del método original (≈ 690 s), en la que se usa el umbral *Max* más grande posible en el cliente (≈ 2.6 10^6 *bytes*), es cuando se reduce éste en los periodos menores para protegerse de $\Delta\alpha_{BE}\big|_{\max}$. El coste debido al uso del modo ReR es de 35.37 10^6 *bytes*.

En la Figura 4.26, se representa la misma transmisión pero con un tráfico de *background* de Pareto, en el que el coste debido a la transmisión mediante el modo ReR es de 36.36 10^6 *bytes*. Aproximadamente 10^6 *bytes* de diferencia que en el caso de tráfico CBR.

Finalmente, en la Figura 4.27, se muestra el mismo experimento que en la Figura 4.26 pero con el método original, sin protección frente a variaciones de α_{BE}. El número de periodos ReR se reduce en 2, pero por otro lado, el coste aumenta a 37.66 10^6 *bytes*. Por tanto, el método es válido si el coste de renegociar dos periodos ReR más es menor que el de enviar 10^6 *bytes* de información utilizando reserva de recursos.

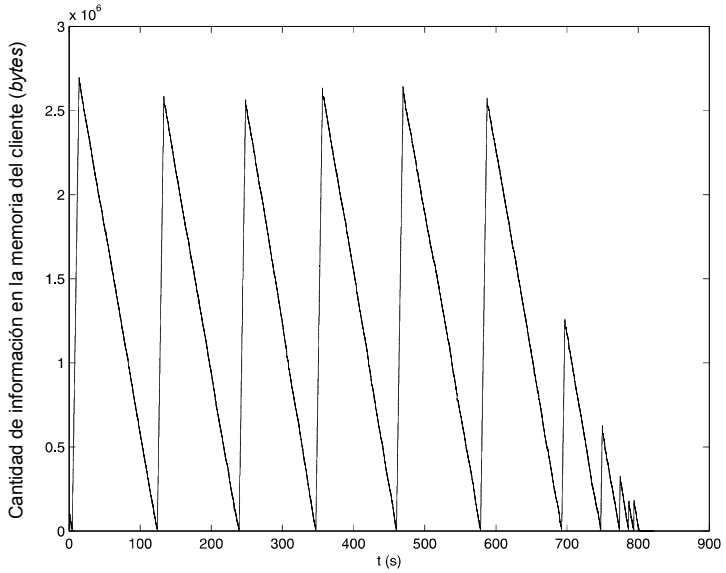

Figura 4.25: Protección frente a variaciones de α_{BE} reduciendo Max para tráfico de background CBR (ρ = 0.25). $\varepsilon\big|_{\max} = 0.01$. FS = 100 Mbytes.

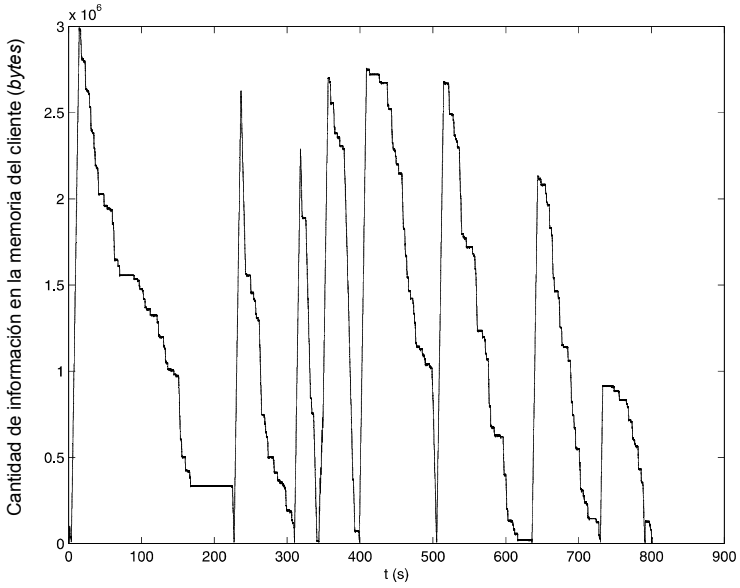

Figura 4.26: Protección frente a variaciones de α_{BE} reduciendo Max para tráfico de background Pareto (ρ = 0.25). $\varepsilon\big|_{\max} = 0.01$. FS = 100 Mbytes.

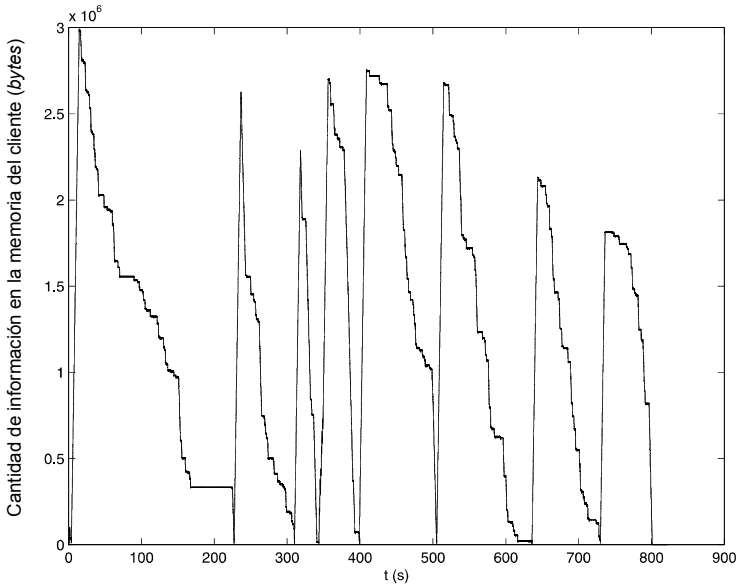

Figura 4.27: Variaciones de α_{BE} sin protección con tráfico de background Pareto (ρ = 0.25). $\varepsilon\big|_{\max} = 0.01$. FS = 100 Mbytes.

4.6 Conclusiones

En este capítulo se ha deducido el coste de una transmisión de flujos semi-elásticos en función de los parámetros que influyen en la ocupación de la memoria del cliente. Este coste depende de forma especial del umbral máximo definido en dicha memoria. Del estudio de la función de coste, se observa que es posible

minimizarla si se utilizan ciertos umbrales óptimos (Max_n), que consiguen que el coste debido al uso del modo de transmisión ReR sea mínimo.

Para poder calcular estos umbrales óptimos es preciso obtener una estimación de la tasa de llegada de datos a la memoria del cliente en el periodo BE. Esta estimación se puede realizar de varias formas posibles, pero se ha propuesto el uso de un umbral *Max* inicial (Max_{ini}) que obligue al cliente a cambiar la transmisión a modo BE, donde se realizará la estimación. Este mecanismo permite que sea el cliente el que realice la estimación y no sea necesaria otra aplicación que realice la estimación utilizando paquetes extra, y por otro lado, deja que se inicie la transmisión en modo ReR, haciendo que los datos puedan ser consumidos por el cliente de forma inmediata.

También se ha analizado el dimensionado del umbral *Min* en la memoria del cliente. Este umbral debe garantizar la disponibilidad de datos en esta memoria, de forma que en caso de que haya retardo en la señalización de cambio de modo de transmisión, la memoria no se quede vacía. Para ello el cliente debe tener en cuenta tanto el retardo de transmisión, como los posibles rechazos de reservas, que pueden producirse en la red.

Las estimaciones que realiza el cliente de la tasa de llegada de paquetes, para minimizar el coste de la transmisión, pueden verse afectadas por el tráfico de *background* que circula por la red en modo BE. Así, es posible que las estimaciones varíen de unos periodos a otros, haciendo que la minimización del coste se vea afectada. Para protegerse frente a estas variaciones, se ha analizado y propuesto un mecanismo, basado en reducir el umbral *Max*, que permite al cliente recalcular este umbral, de forma que se aumente la exactitud con que se alcanza el coste mínimo, incrementando de forma mínima el número de renegociaciones necesarias entre modos de transferencia.

Finalmente, en este capítulo, se han mostrado los resultados obtenidos por simulación, que permiten observar el funcionamiento del cliente como controlador del coste de la transmisión, así como de los mecanismos propuestos para mejorar las prestaciones del sistema en cuanto a minimización del coste se refiere.

Capítulo 5

Servicio simultáneo de flujos semi-elásticos

5.1 Introducción

La transmisión eficiente de flujos semi-elásticos permite minimizar el coste que debe pagar un proveedor de servicios (y en consecuencia el cliente) a su proveedor u operador de red por el uso de reserva de recursos, necesaria para realizar una transmisión rápida y fiable.

El cliente controla el modo de transmisión, indicando al servidor si debe enviar los datos mediante reserva de recursos (ReR) o mediante el modo *best-effort* (BE). Para minimizar el coste, el cliente debe maximizar el uso del modo BE, teniendo en cuenta la carga existente en la red en este modo. Para evaluar esta carga y determinar cuándo reservar recursos, el cliente observa la ocupación de su memoria, asegurando que se mantiene entre dos umbrales. Estos umbrales, tal y como se vio en el Capítulo 4, se escogen convenientemente para que el cliente disponga de datos durante toda la transmisión y para que no se reserven más recursos de los necesarios. Cuando la ocupación de la memoria del cliente llega a uno de estos umbrales, se pide al servidor un cambio de modo de transferencia. Así, si se alcanza el umbral mínimo (*Min*), el cliente pedirá un cambio a modo ReR para garantizar la llegada de datos, mientras que si se alcanza el umbral máximo (*Max*), se solicitará la transmisión en modo BE para reducir el coste de la transmisión. Seleccionado de

forma conveniente, el umbral *Max* permite minimizar el coste de transmisión debido a reservas de recursos.

El servidor, por su parte, envía los datos a la tasa adecuada según se utilice el modo ReR o BE. Además, debe intentar servir el máximo número de clientes de forma simultánea y eficiente, es decir, controlando que la minimización del coste no se degrade demasiado, y que no se incremente en exceso la señalización extra necesaria para controlar el servicio simultáneo.

Suponiendo que los clientes son independientes entre sí, es posible que varios de ellos realicen una petición de reserva de recursos al mismo tiempo, pudiendo llegar a reservar todo el ancho de banda de acceso del servidor, si éste no se gestiona correctamente, y por tanto, producir bloqueo a posibles reservas de otros clientes, afectando seriamente a la transmisión.

En este capítulo, se analiza un mecanismo de gestión de ancho de banda de acceso del servidor a la red, particularizando dos casos extremos, que como se verá no son los más óptimos, pero sí los más sencillos de implementar. Asimismo, se examinarán las implicaciones que supone esta gestión en los protocolos de señalización de reserva de recursos, particularizando para el protocolo RSVP (*resource ReSerVation Protocol*) [BRA97], que es el utilizado en la implementación del sistema mediante el simulador ns-2 (Anexo A).

5.2 Servicio simultáneo de flujos semi-elásticos

El servicio de flujos semi-elásticos en redes con calidad de servicio extremo a extremo, está basado en dos modos de servicio: *best-effort* (BE) y reserva de recursos (ReR). El servidor utiliza los dos modos de transmisión según sean las necesidades del cliente. Así, si el cliente tiene suficientes datos almacenados en su memoria, se podrá utilizar el modo BE, mientras que si la memoria alcanza un determinado umbral mínimo de ocupación, se utilizará el modo ReR.

Cuando se debe servir a varios clientes que demandan simultáneamente la transmisión de flujos semi-elásticos, es necesario analizar cómo el servidor debe gestionar su ancho de banda de acceso a la red, ya que es posible que varios clientes soliciten reservar recursos de forma simultánea, lo que puede implicar que fácilmente se agote el ancho de banda disponible.

En este apartado, se investigarán tres modelos de gestión de ancho de banda para dar servicio simultáneo: reserva de recursos por cliente, reserva de recursos para un cliente y reserva de recursos para m clientes. De estos tres modelos, el tercero es el caso genérico, mientras que el primero y el segundo son casos particulares. El modelo se basa en dividir el ancho de banda de acceso existente en dos partes: una para garantizar el servicio en modo BE y otra para el modo ReR. La parte que se gestiona es la reservada para el modo ReR de forma que se garantice el servicio para todos los clientes de forma simultánea. Así, si se da servicio a $\mathcal{C}$ clientes, $\mathcal{C}_{ReR}$ se servirán en modo ReR y $\mathcal{C}_{BE}$ en modo BE (5.1).

$$\mathcal{C} = \mathcal{C}_{ReR} + \mathcal{C}_{BE} \tag{5.1}$$

Este estudio es una primera aproximación a la gestión del ancho de banda de acceso que debe realizar el servidor, por lo que se supondrá que todos los clientes son homogéneos, es decir, todos leen la información a la misma velocidad, reservan el mismo ancho de banda, y la oferta de ancho de banda disponible en el camino entre cliente y servidor es la misma y sin restricciones. Este no es un caso real, pero permitirá analizar los distintos modos de servicio en condiciones similares.

5.2.1 Número máximo de clientes

Primeramente, cabe examinar cuál es el número máximo de clientes ($\mathcal{C}_{\max}$) a los que podrá dar servicio un servidor en la situación supuesta. Para ello, en primera instancia se descartan los efectos que puede tener la señalización de reserva de recursos en el sistema[i].

En el caso en que la carga apreciada por los clientes (véase el Capítulo 3) sea nula durante toda la transmisión, se dará servicio en modo de transmisión BE, a una tasa igual a la tasa de lectura del cliente (α_r). Por tanto, el valor que limitará el número de clientes es el ancho de banda disponible para el modo BE, que en este caso será igual al ancho de banda de acceso del servidor (B_s), ya que se podrá utilizar

[i] La señalización de reservas de recursos se puede descartar si se supone pequeña respecto a la cantidad de datos que envía el servidor. Debe tenerse en cuenta si esta señalización es tan grande como para ocupar el ancho de banda necesario para el servicio de un cliente. Como se verá más adelante, la cantidad de señalización necesaria dependerá de la forma en que se de servicio a los clientes.

todo para este modo, al no ser necesario utilizar el modo ReR. Así, el número máximo de clientes a los que se podrá dar servicio será,

$$\mathcal{C}_{\max} = \frac{B_s}{\alpha_r} \tag{5.2}$$

El otro caso extremo, es aquél en que todos los clientes precisan recibir toda la información mediante el modo ReR. En esta situación, si se supone que el servidor dispone de un ancho de banda de acceso $B_s = B_{BE} + B_{ReR}$ $(bits/s)$, donde B_{BE} es el ancho de banda mínimo disponible para tráfico BE, y B_{ReR} es el ancho de banda máximo disponible para tráfico ReR, el servicio estará limitado por B_{ReR} .

$$\mathcal{C}_{\max} = \frac{B_{ReR}}{\alpha_r} \tag{5.3}$$

Tal y como se deduce de lo anterior, en casos intermedios, el mayor número de clientes al que se puede dar servicio estará entre los valores (5.2) y (5.3) anteriores, por tanto, para evitar que este número dependa de la carga de la red, se supondrá el número mayor de clientes a los que se puede dar servicio al valor de la parte entera de (5.3), ya que es el más restrictivo.

5.2.2 Reserva de recursos por cliente

En este modo de servicio, el servidor debe garantizar que todos los clientes puedan reservar recursos en el momento que así lo requieran, controlando los recursos máximos que puede reservar cada cliente.

Como se vio en el Capítulo 4, el coste de la transmisión depende de la tasa de llegada de paquetes en modo BE y en modo ReR. Cuanto mayores son estas tasas, menor es el coste[ii], por lo que se deberá utilizar siempre la mayor cantidad posible de ancho de banda de B_{ReR} para cada cliente en modo ReR. Por tanto, si se sirve a un solo cliente, parece lógico pensar que se podrá utilizar todo B_{ReR} [iii], mientras que si

[ii] Recuérdese que si la tasa que se reserva en modo ReR es igual a la tasa de lectura del cliente, no hay reducción del coste; y si la tasa en modo BE es nula, sucede lo mismo.

[iii] El servidor debe conocer cuál es el ancho de banda máximo que se puede reservar en el camino hacia el cliente, para no modificar reservas de forma errónea y evitar que se rechacen las modificaciones. Este ancho de banda se puede deducir controlando las

son dos clientes se podrá utilizar $B_{ReR}/2$ para cada uno, y en general para un número de clientes $\mathcal{C} = k$, se podrá reservar, para cada uno, el ancho de banda expresado en (5.4).

$$\alpha_{Res}(k) = \frac{B_{ReR}}{k} \tag{5.4}$$

Cabe resaltar, que k no puede ser cualquier valor, ya que siempre se debe cumplir que α_{Res} sea mayor que la tasa de lectura del cliente α_r, ya que en caso contrario es imposible realizar la transmisión correctamente, es decir, no se debe superar el número de clientes máximo ($\mathcal{C}_{\max}$) al que se puede dar servicio. Por otro lado, cuanto mayor sea α_{Res} respecto de α_r, mayor será la eficiencia media conseguida ($\overline{\eta}$) (5.5), ya que se incrementarán las eficiencias de transmisión de cada cliente (η_i).

$$\overline{\eta} = \frac{\sum_{i=1}^{\mathcal{C}} \eta_i}{\mathcal{C}} \tag{5.5}$$

Analizando la señalización relacionada con las reservas de recursos, se advierte la necesidad de cambiar el ancho de banda que reserva cada cliente a medida que varía el número de clientes que se está sirviendo en cada momento (varía k). Por otra parte, se precisa que el protocolo de señalización de reserva de recursos pueda estar controlado por el servidor, con el fin de controlar de forma inmediata el ancho de banda reservado para cada cliente que demande el uso del modo ReR en ese momento. Por consiguiente, si se utiliza el protocolo RSVP será necesario que la interfaz con el servidor le informe de la llegada de mensajes RSVP antes de realizar alguna acción (rechazo de reserva, actualización de reservas, etc.) para poder modificar el estado de las reservas actuales, si ello es adecuado, a fin de evitar rechazos innecesarios[iv].

---

características del camino mediante objetos ADSPEC (*ADvertisement SPECification*) utilizados en RSVP.

[iv] En el modelo implementado en el simulador ns-2, se propone un interfaz con estas características, de forma que el protocolo RSVP indica la llegada de una nueva reserva al servidor para que actúe antes de continuar con el proceso de reserva (véase el Anexo A).

5.2.3 Reserva de recursos para un cliente

En este modelo sólo reserva recursos un cliente y el resto recibe datos en modo BE. En este caso se simplifica al máximo el control de cuáles son los recursos disponibles por cliente, ya que todos los recursos reservables se asignan a un único cliente. El cliente al que se asignan estos recursos es el que necesita con mayor prioridad que su memoria se llene, para así evitar quedarse sin datos que leer[v]. En este caso el ancho de banda a reservar es siempre el mismo (5.6).

$$\alpha_{Res}(k) = B_{ReR} \tag{5.6}$$

En cuanto a la eficiencia media para este modo de servicio, será la máxima posible, ya que se reserva la mayor cantidad de recursos posible. Por otro lado, esta eficiencia se degradará progresivamente a medida que aumente el número de clientes a los que se da servicio, ya que también aumentará el tráfico en modo BE, pudiendo llegar a ser el enlace de acceso el más restrictivo, limitando la tasa de BE que se ofrece a los clientes.

5.2.4 Reserva de recursos para *m* clientes

En este caso se permite dar servicio simultáneo en modo ReR a m clientes, repartiéndose los recursos disponibles para reservar[vi]. Así, la reserva de recursos por cliente es igual a éste pero con $m = \mathcal{C}_{\max}$, mientras que la reserva de recursos para un cliente ocurre para $m = 1$.

El funcionamiento general de esta metodología es idéntico al del primer caso (apartado 5.2.2) para $k \le m$ (el número de transmisiones permitidas en modo ReR es menor o igual que m), mientras que para $k > m$ es igual que el segundo (apartado 5.2.3), teniendo en cuenta que una nueva petición de paso a modo ReR, cuando el número de clientes que se sirven en modo ReR en un momento concreto ($\mathcal{C}_{ReR}$) es igual a m, implica que el cliente que lleva más tiempo en modo ReR pase a modo BE.

[v] Será el último cliente cuya memoria haya alcanzado el umbral mínimo.

[vi] De nuevo se debe cumplir que m sea menor o igual que el número máximo de clientes a los que se puede dar servicio.

5.3 Servicio mediante reserva de recursos para *m* clientes

En esta sección, se analizará el servicio genérico basado en la reserva de recursos para *m* clientes. En la primera parte, se describe el funcionamiento de la gestión del ancho de banda de acceso del servidor. Esta gestión permite repartir de forma apropiada el ancho de banda entre todos los clientes, y controlar cuáles se sirven en modo ReR y cuáles en modo BE. En la segunda parte, se realiza un análisis de la señalización necesaria para este tipo de servicio, particularizando también para los otros dos casos extremos, comentados anteriormente: reserva de recursos por cliente y reserva de recursos para un cliente. A modo de resumen, se muestra en la Tabla 5.1 las variables relacionadas con el servicio simultáneo de flujos semi-elásticos presentadas en apartados anteriores.

5.3.1 Gestión del ancho de banda de acceso del servidor

Tal y como se comentó en el apartado 5.2.4, el ancho de banda que se reservará en el modo ReR depende del número de clientes a servir. Esta gestión varía según el número de clientes que tienen iniciada una sesión en el servidor, por lo que se analizarán las acciones a realizar en el inicio de una sesión y al final de ésta. Por otra parte, se verá cuál es la gestión de este ancho de banda a medida que se recibe por parte de los clientes, nuevas peticiones de paso a modo ReR.

Inicio de Sesión

Inicialmente, el servidor no da servicio a ningún cliente hasta que llega la primera petición. Se aceptarán nuevas peticiones siempre que no se supere el número máximo de clientes que acepta el servidor. El número de peticiones recibidas determina el ancho de banda disponible para el uso de cada cliente en el modo ReR (Figura 5.1). Así, si el número de clientes a los que se puede dar servicio en modo ReR en ese momento[vii] (k) es menor que m, se incrementará en uno ese número para controlar el valor instantáneo, se cambiará el ancho de banda a utilizar por cada cliente en modo ReR (α_{Res}), y se informará a todos los clientes de éste nuevo valor[viii].

[vii] Este número depende del número de sesiones abiertas y nunca puede superar m.

[viii] El protocolo RSVP puede informar a los clientes de este valor, mediante el uso de mensajes PATH que indican el camino que siguen los mensajes, así como la tasa en modo ReR que permite utilizar el servidor.

En caso de que k sea igual a m (hay un número de clientes con sesión abierta mayor o igual al máximo permitido en modo ReR), sólo se informará al cliente del valor del ancho de banda que deberá reservar en modo ReR.

Variable	Descripción
B_s	Ancho de banda de acceso del servidor ($bits/s$)
B_{BE}	Ancho de banda mínimo disponible para modo BE ($bits/s$)
B_{ReR}	Ancho de banda máximo disponible para modo ReR ($bits/s$)
α_{Res}	Tasa de reserva de recursos ($bits/s$)
α_r	Tasa de lectura del cliente ($bits/s$)
C	Número de clientes (sesiones) a los que se está dando servicio
C_{ReR}	Número de clientes sirviéndose en modo ReR
C_{BE}	Número de clientes sirviéndose en modo BE
$C_{\max}$	Número máximo de clientes (de ambas clases) a los que se puede dar servicio
k	Número de sesiones que se permiten en modo ReR
m	Número de sesiones máximo que se permiten en modo ReR

Tabla 5.1: Variables relacionadas con el servicio simultáneo de flujos semi-elásticos.

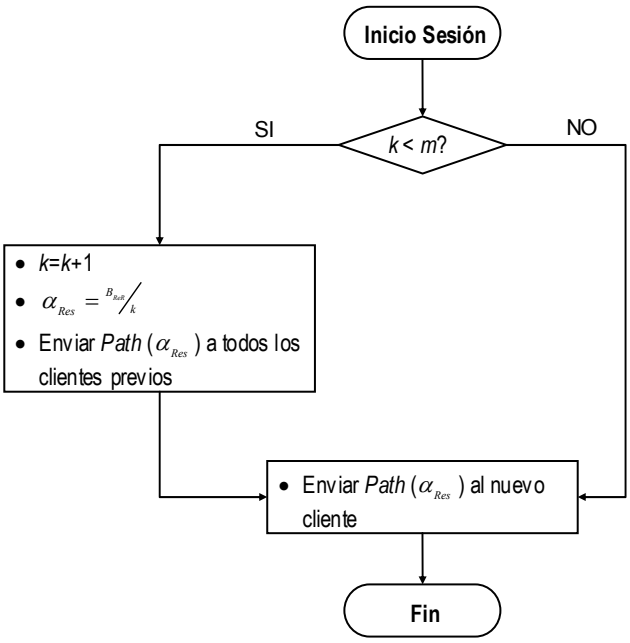

Figura 5.1: Diagrama de flujo para la gestión del ancho de banda de acceso del servidor (llega una nueva petición de servicio).

Fin de sesión

Cuando se finaliza una sesión, el ancho de banda disponible se deberá repartir entre el resto de clientes si es posible (Figura 5.2). De esta forma, si el número de clientes a los que se da servicio (sin contar el cliente que acaba de finalizar) $\mathcal{C}$, es menor a m, se reduce k y se aumenta el ancho de banda máximo a reservar por cada cliente en modo ReR, además de informar al resto de clientes del nuevo ancho de banda disponible. Si no es así, no se realiza ninguna acción.

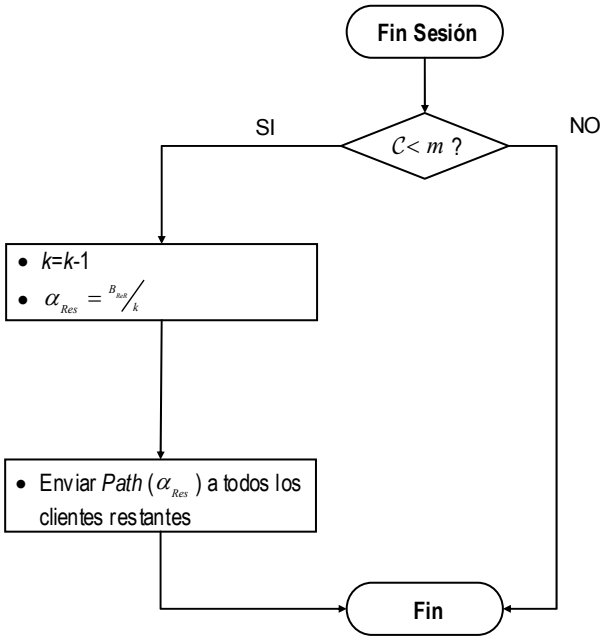

Figura 5.2: Diagrama de flujo para la gestión del ancho de banda de acceso del servidor (finaliza una sesión).

Peticiones de paso a modo ReR

Por otro lado, el servidor recibe peticiones de paso a modo de transferencia ReR. Según el número de clientes que se sirvan en ese momento, el servidor actuará de una u otra manera (Figura 5.3). Si llega una petición de paso a modo ReR y no se está sirviendo a ningún cliente en ese modo ($\mathcal{C}_{ReR} = 0$), no se realiza ninguna acción, ya que todo el ancho de banda para modo ReR está disponible y se supone que la petición de reserva de recursos es correcta.

Si se está sirviendo a algún cliente en modo ReR, pero $\mathcal{C}_{ReR} < m$,[ix] se ajustarán las tasas reservadas[x] para todos los clientes que todavía no hayan ajustado su tasa a la máxima especificada y se aceptará la petición de paso a modo ReR para ese

[ix] Se sirve en modo ReR a un número menor al número máximo de clientes permitidos en ese modo.

[x] Esto implica que el servidor puede requerir un cambio en las reservas existentes. El protocolo RSVP no permite realizar este cambio, ya que debe ser el cliente quien lo realice. Para implementar esta nueva habilidad, se propone el uso de un nuevo mensaje RESV_SENDER similar al mensaje RESV de RSVP, pero enviado por el servidor.

cliente. En el caso de que $\mathcal{C}_{ReR} = m$,[xi] se pasará a modo BE al cliente que lleve en modo ReR más tiempo, para permitir que el cliente que ha realizado la petición (y que se supone con una ocupación de memoria más crítica) pase a modo ReR.

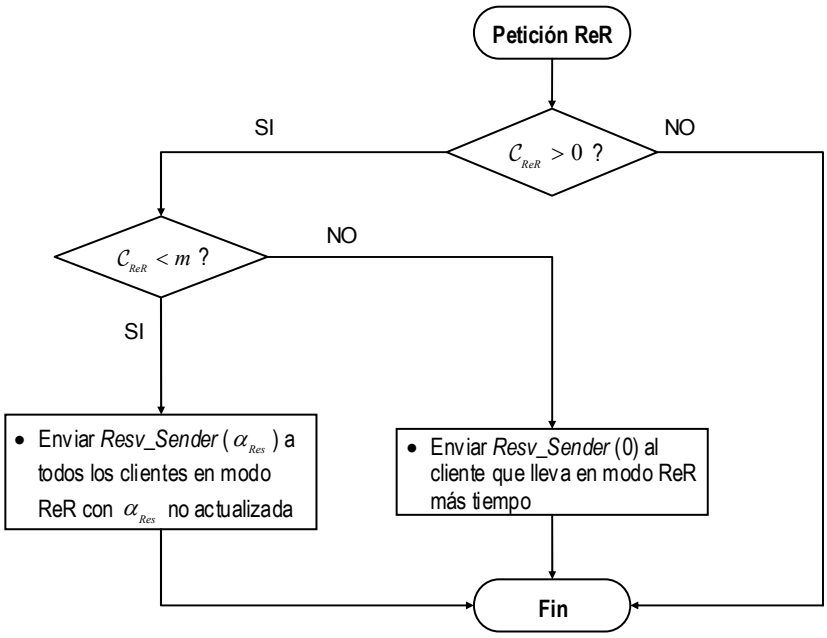

Figura 5.3: Diagrama de flujo para la gestión del ancho de banda de acceso del servidor (peticiones de paso a modo ReR).

Como se observa, el servidor puede decidir el paso a modo BE de una de las transmisiones. Esto puede hacer que el cliente pase a modo de transmisión BE, cuando su memoria está cerca del umbral mínimo[xii] o incluso estando por debajo si es al inicio de la recepción de datos (Figura 5.4), provocando en el primer caso que se repita la petición de reserva de recursos rápidamente, o que no se pueda iniciar correctamente la lectura en el cliente, en el segundo.

[xi] Se da servicio en modo ReR a un número igual al número permitido en ese modo.

[xii] En este caso, en un breve instante de tiempo el cliente realizará otra petición de paso a modo ReR.

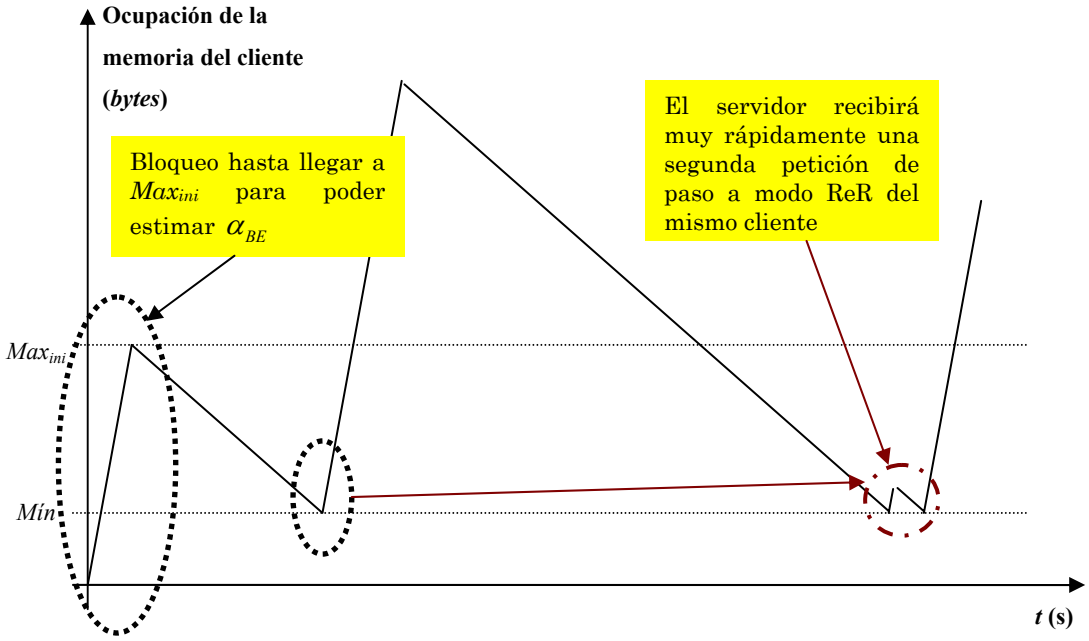

Figura 5.4: Zonas de ocupación de la memoria del cliente en las que no se debe pasar a modo BE.

Para evitar que se cambie el modo de transmisión a modo BE antes de que la memoria del cliente llegue al nivel mínimo, es necesario bloquear la conexión, de forma que no se permita que esa conexión pase a modo BE hasta que haya alcanzado el nivel mínimo de ocupación[xiii]. En caso de que el servidor esté totalmente bloqueado[xiv], la petición inicial se almacena en una cola hasta que sea posible iniciar la transmisión.

Por otro lado, también se debe evitar que se pase a modo BE si la ocupación de la memoria del cliente está cerca del nivel mínimo. Para evitar esto, es posible aumentar el nivel mínimo del cliente.

5.3.2 Análisis de señalización

Es importante analizar la cantidad necesaria de señalización en el servidor para poder gestionar el ancho de banda de forma correcta, ya que es un factor muy

[xiii] Al inicio conviene que este nivel sea Max_{ini} (véase el Capítulo 4) para asegurar una estimación precisa de la tasa de BE.

[xiv] El servidor estará totalmente bloqueado si se está dando servicio en modo ReR, por primera vez, al número máximo de clientes en ese modo.

importante que permitirá comparar las distintas formas propuestas de servicio. Por tanto, es posible contabilizar el número de mensajes PATH necesarios en cada caso (M_{Path}), así como el de mensajes RESV_SENDER (M_{Resv_Sender}).

El número de mensajes que se precisa depende de las distintas fases descritas en la Figura 5.1, la Figura 5.2 y la Figura 5.3, así como del número de clientes que se sirva en ese momento $\mathcal{C}$ y del valor máximo de clientes en modo ReR aceptado (m).

Inicio de Sesión

En este caso y en base a la Figura 5.1, el número de mensajes de señalización extra necesarios[xv] en esta fase cada vez que se inicie una sesión, será:

$$M_{inicio} = M_{Path} = \begin{cases} \mathcal{C} & \mathcal{C} < m \\ 0 & \mathcal{C} \geq m \end{cases} \tag{5.7}$$

Para el caso de reserva de recursos para cada cliente, el número de mensajes será más crítico a medida que se incremente $\mathcal{C}$. Esto implica que el número de mensajes necesario dependa del número de clientes a los que se esté dando servicio, lo que puede suponer un problema de escalabilidad en el sistema cuando $\mathcal{C}$ es grande. Por otro lado, disminuyendo m es posible controlar este problema.

En el caso de reserva de recursos para un cliente, se cumple $m = 1$, por lo que no se necesitan mensajes de señalización extra al inicio de la sesión, independientemente del número de clientes en el sistema.

Fin de sesión

De la Figura 5.2, se puede deducir el número de mensajes necesarios en esta fase si $\mathcal{C}$ es el número de clientes que restan sin contar el que abandona la sesión:

$$M_{final} = M_{Path} = \begin{cases} \mathcal{C} & \mathcal{C} < m \\ 0 & \mathcal{C} \geq m \end{cases} \tag{5.8}$$

Como se advierte, de nuevo para el servicio con reserva de recursos para cada cliente, la señalización necesaria será $\mathcal{C}$ mientras que para el caso de reserva de recursos por cliente no será necesario ningún mensaje extra.

[xv] Mensajes de señalización extra en relación al número de mensajes necesario para el caso de un solo cliente.

Peticiones de paso a modo ReR

Observando la Figura 5.3, se deducen los mensajes necesarios para esta fase, teniendo en cuenta los que se envían para cambiar las tasas reservadas no actualizadas (5.9) y los necesarios para liberar la reserva al cliente con mayor tiempo continuo reservando recursos, cuando al llegar la petición se están sirviendo en modo ReR $\mathcal{C}_{ReR}$ clientes (5.10).

$$M_{ReR\_cambio\_tasa} = M_{Resv\_Sender} = \begin{cases} 0 & \mathcal{C}_{ReR} = 0 \\ \mathcal{C}_{ReR} & 0 < \mathcal{C}_{ReR} < m \\ 0 & \mathcal{C}_{ReR} = m \end{cases} \tag{5.9}$$

$$M_{ReR\_quitar\_reserva} = M_{Resv\_Sender} = \begin{cases} 0 & \mathcal{C}_{ReR} < m \\ 1 & \mathcal{C}_{ReR} = m \end{cases} \tag{5.10}$$

En el caso de $\mathcal{C}_{ReR} = 0$, no es necesaria señalización extra, ya que no hay ningún problema en realizar la reserva. Cuando se cumple $0 < \mathcal{C}_{ReR} < m$ se supone el peor caso, es decir, se considera necesario enviar a todos los clientes un cambio de reserva de recursos. Esto será en general relativamente habitual para $\mathcal{C} < m$, ya que en este caso, se deberán ajustar las tasas inmediatamente. Para el servicio mediante reserva de recursos por cliente, esta situación se produce durante el máximo tiempo posible ya que depende de m que en este caso es máxima. En el caso extremo con reserva de recursos para un cliente, no se requieren los mensajes para cambiar la reserva ya que se utiliza todo el ancho de banda para dicha reserva.

Por otro lado, la señalización precisa para liberar la reserva que lleva más tiempo iniciada, es únicamente necesaria si $\mathcal{C}_{ReR} < m$. Asimismo, la probabilidad de que una nueva petición se encuentre con $\mathcal{C}_{ReR} = m$ clientes, será mayor a medida que m sea menor y $\mathcal{C}$ mayor.

Otro factor importante a tener en cuenta, es el número de peticiones de paso a modo ReR que realizarán los clientes, que será dependiente de los diversos parámetros controlados por su memoria. En particular, para una transmisión concreta, el parámetro más crítico es el tamaño máximo de la memoria del cliente, ya que es el que incide directamente en el umbral Max_n que se utiliza y por tanto en el número de periodos ReR necesarios (véase el Capítulo 4).

5.4 Resultados de simulación

El sistema servidor se ha implementado en el simulador ns-2 y se describe en el Anexo A. En este apartado, se presentan los distintos experimentos realizados, para observar las prestaciones del sistema propuesto.

En el primer experimento, se muestra la evolución de la eficiencia media del sistema en función de m y de $\mathcal{C}$. En el segundo, se ilustra la evolución de la cantidad de señalización extra necesaria en función de los mismos parámetros.

El sistema que se simula es un sistema cliente servidor (descrito en el Anexo A), donde se da servicio a $\mathcal{C}$ clientes de forma simultánea. Los clientes son todos homogéneos, es decir, leen los datos a la misma tasa (250 *kbits/s*) y el tamaño de su memoria es de 1 *Mbyte*. Además, inician las sesiones de forma desincronizada entre ellos.

El tráfico de *background* es de tasa constante (*Constant Bit Rate*, CBR), como el utilizado en algunas simulaciones del Capítulo 4, utilizando todo el ancho de banda disponible en los enlaces hacía los clientes, menos aproximadamente unos 200 *kbits/s*, para obtener una carga apreciada por el cliente de aproximadamente un 25%.

El servidor dispone de un ancho de banda de acceso de 5 *Mbits/s*, de los cuales 4 *Mbits/s* se utilizan para reservas de recursos. En esta situación el número máximo de clientes a los que se puede dar servicio es de 16, pero esto supone no poder reducir el coste, debido a que en ese caso se reserva una tasa igual a la de lectura del cliente. Por otra parte, la señalización extra también afecta al número máximo de clientes real, ya que también consume ancho de banda reservable, y como se verá en algunas simulaciones es elevada. Por todo esto, se supone un número máximo de clientes de 14.

5.4.1 Evolución de la eficiencia media del sistema en función de m y $\mathcal{C}$

En la Figura 5.5, se muestra la evolución de la eficiencia media del sistema en función de m y de $\mathcal{C}$. Como se observa, la eficiencia disminuye a medida que aumentan m y $\mathcal{C}$. Esto depende de forma directa de la cantidad de ancho de banda que se reserva en modo ReR. A medida que este ancho de banda es menor, también es menor la reducción del coste que se obtiene.

La gráfica está dividida en tres zonas destacables para m creciente:

1. $m < \mathcal{C}$: en esta zona, a medida que m crece, disminuye la eficiencia debido a que α_{Res} es cada vez menor. $\mathcal{C}$ no afecta a la eficiencia en este área ya que al ser mayor que m, α_{Res} no varía. Únicamente se aprecia un ligero decrecimiento con $\mathcal{C}$ alta ya que el servidor está muy cargado y el tráfico que circula en BE es elevado.
2. $m = \mathcal{C}$: se observa un punto de inflexión a partir del cual α_{Res} se mantiene constante para $m > \mathcal{C}$.
3. $m > \mathcal{C}$: la eficiencia se mantiene aproximadamente constante, disminuyendo a medida que $\mathcal{C}$ aumenta. Esto es debido a que la tasa α_{Res} no varía para todos estos casos, ya que depende de $\mathcal{C}$ (véase la ecuación (5.4)).

Resumiendo, para conseguir una eficiencia alta, el servidor deberá utilizar una $m < \mathcal{C}$ que sea lo más pequeña posible, ya que a medida que m se aproxima a $\mathcal{C}$, la eficiencia disminuye hacia 0. El caso más favorable es para $m = 1$ (reserva de recursos para un cliente), aunque como veremos a continuación es necesario analizar la cantidad de señalización extra necesaria para determinar el valor óptimo a utilizar.

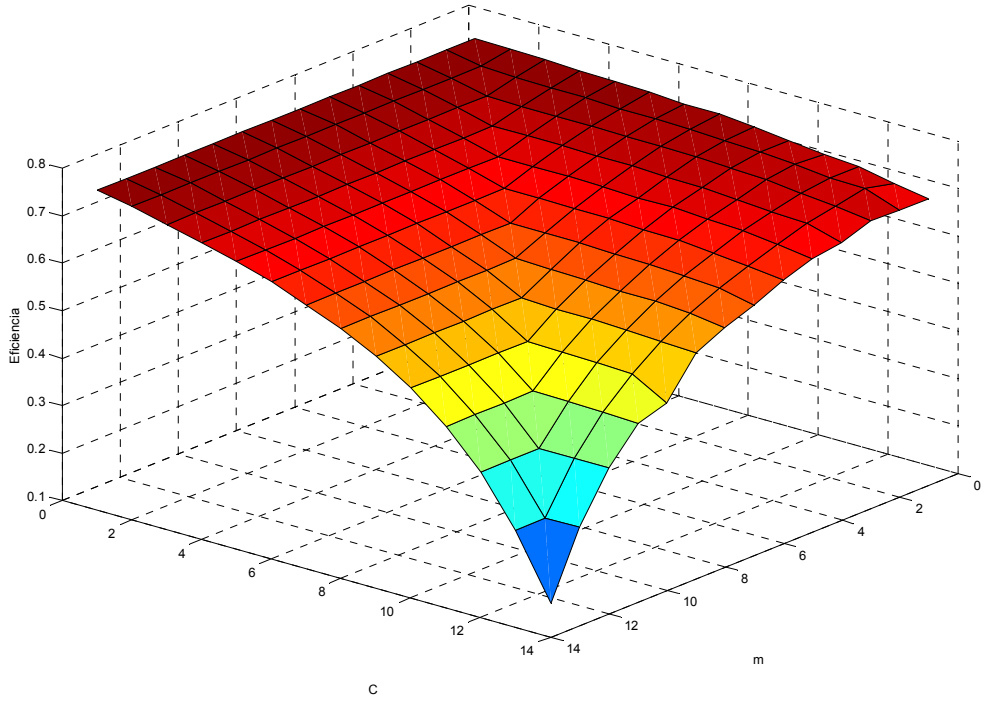

Figura 5.5: Eficiencia media en función de m y $\mathcal{C}$.

5.4.2 Evolución del número extra de mensajes RSVP en función de *m* y $\mathcal{C}$

En la Figura 5.6, se puede observar la evolución de los mensajes de señalización extra debidos al inicio de sesión. Las cantidades indican el total de mensajes utilizados en una simulación donde los $\mathcal{C}$ clientes iniciaban 2 sesiones sucesivas. Similarmente, la Figura 5.7 muestra la dinámica de los mensajes relacionados con el final de las sesiones. Estos dos casos presentan un comportamiento idéntico, debido a que se utiliza el mismo número de mensajes extra tanto al inicio como al final.

Examinando la Figura 5.6 y la Figura 5.7 en las tres zonas destacables se advierte lo siguiente:

1. $m<\mathcal{C}$: en esta zona, a medida que m crece, el número de mensajes extra aumenta, ya que es posible dar servicio a más clientes y por tanto, la señalización necesaria es mayor, ya que depende de $\mathcal{C}$ mientras $m>\mathcal{C}$ (5.7) (5.8). En función de $\mathcal{C}$ la señalización disminuye, debido a que al iniciarse dos sesiones, las segundas se encuentran con el servidor dando servicio a otros clientes. Esto implica que para $\mathcal{C}$ suficientemente

grande, el servidor se encuentre muy probablemente en estado $m<\mathcal{C}$ y por tanto, no sea necesaria señalización extra. A medida que m está más cercana a $\mathcal{C}$ esta probabilidad disminuye.

2. $m=\mathcal{C}$: se observa un punto de inflexión a partir del cual la señalización extra se mantiene constante para $m>\mathcal{C}$.

3. $m>\mathcal{C}$: la señalización extra se mantiene constante, disminuyendo a medida que $\mathcal{C}$ disminuye. Esto es debido a que en estos casos nunca se alcanza a m, por lo que la señalización sólo depende de $\mathcal{C}$.

En este caso, y de forma similar al de la eficiencia, la zona óptima de trabajo es para $m<\mathcal{C}$, con m lo más pequeña posible.

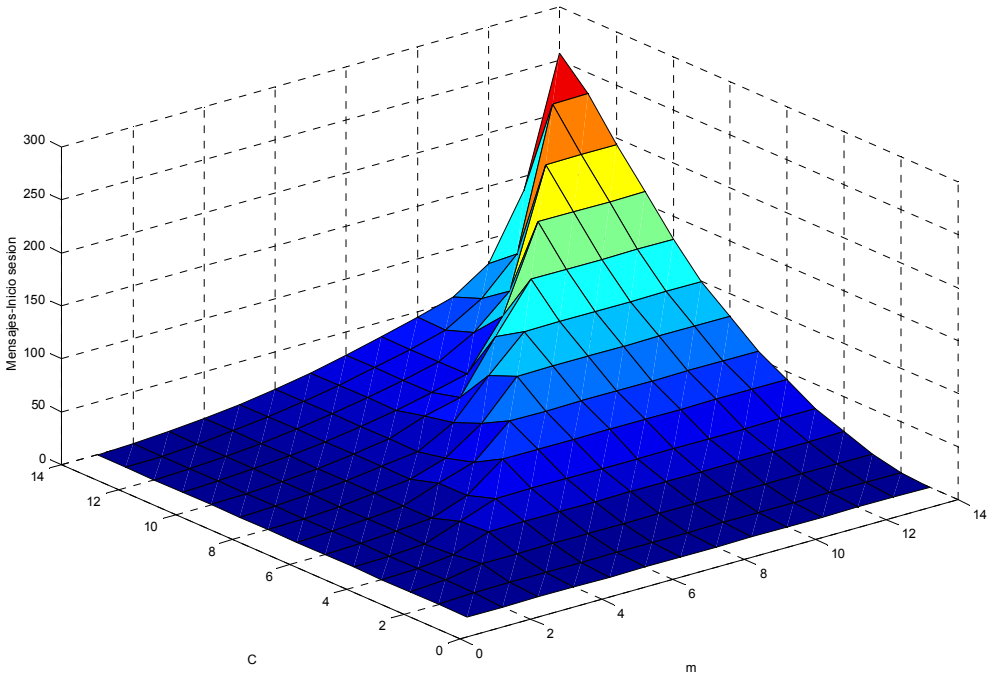

Figura 5.6: Mensajes de señalización extra (inicio sesión) en función de m y $\mathcal{C}$.

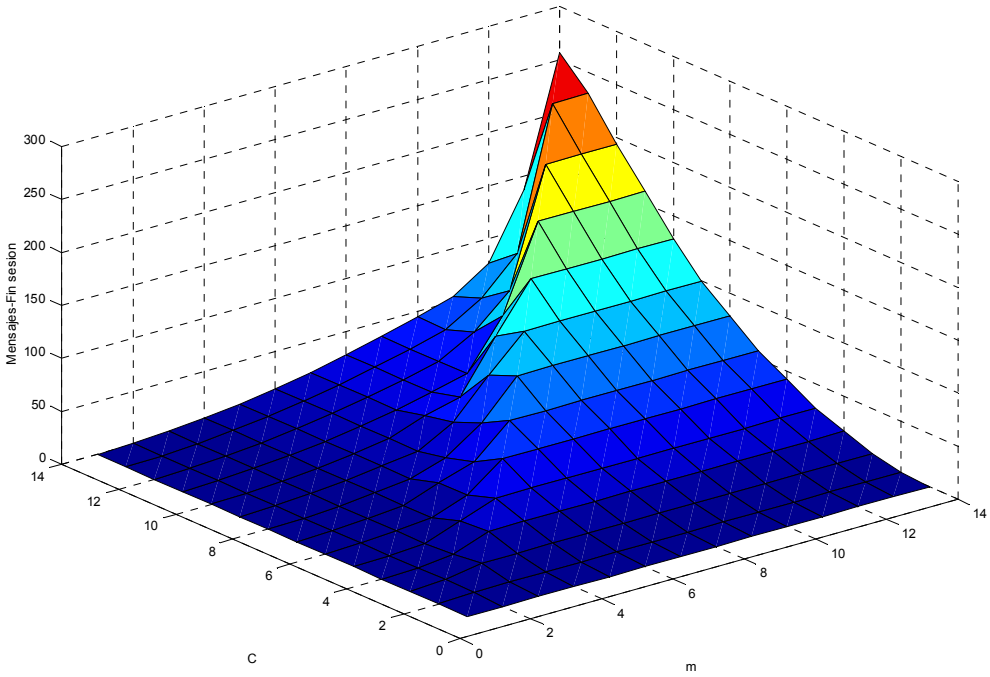

Figura 5.7: Mensajes de señalización extra (fin sesión) en función de m y $\mathcal{C}$.

La Figura 5.8 muestra los mensajes de señalización extra debidos a la necesidad de actualizar la tasa que reservan algunos clientes, para evitar el rechazo de una nueva reserva.

Examinando la Figura 5.8 en las tres zonas anteriores se revela lo siguiente:

1. $m<\mathcal{C}$: en esta zona, a medida que m crece, el número de mensajes extra necesarios aumenta, ya que es posible dar servicio en modo ReR a más clientes y por tanto, la señalización necesaria será mayor, ya que depende de $\mathcal{C}_{ReR}$ y que será mayor con m (5.9). En función de $\mathcal{C}$ la señalización disminuye, debido a que al iniciarse dos sesiones, las segundas se encuentran con el servidor prestando servicio a otros clientes. Esto implica que para $\mathcal{C}$ suficientemente grande, el servidor se encuentre muy probablemente en estado $m<\mathcal{C}$ y con todas las tasas actualizadas. A medida que m está más cercana a $\mathcal{C}$ esta probabilidad disminuye.

2. $m=\mathcal{C}$: se observa un punto de inflexión a partir del cual la señalización extra se mantiene constante para $m>\mathcal{C}$.

3. $m > \mathcal{C}$: la señalización extra se mantiene constante, disminuyendo a medida que $\mathcal{C}$ disminuye. Ello es debido a que en estos casos nunca se alcanza a m, por lo que la señalización sólo depende de la probabilidad de encontrar un determinado $\mathcal{C}_{ReR}$ que es mayor con $\mathcal{C}$.

De nuevo, estos mensajes, requieren que la zona óptima de trabajo sea $m < \mathcal{C}$, con m lo más pequeña posible.

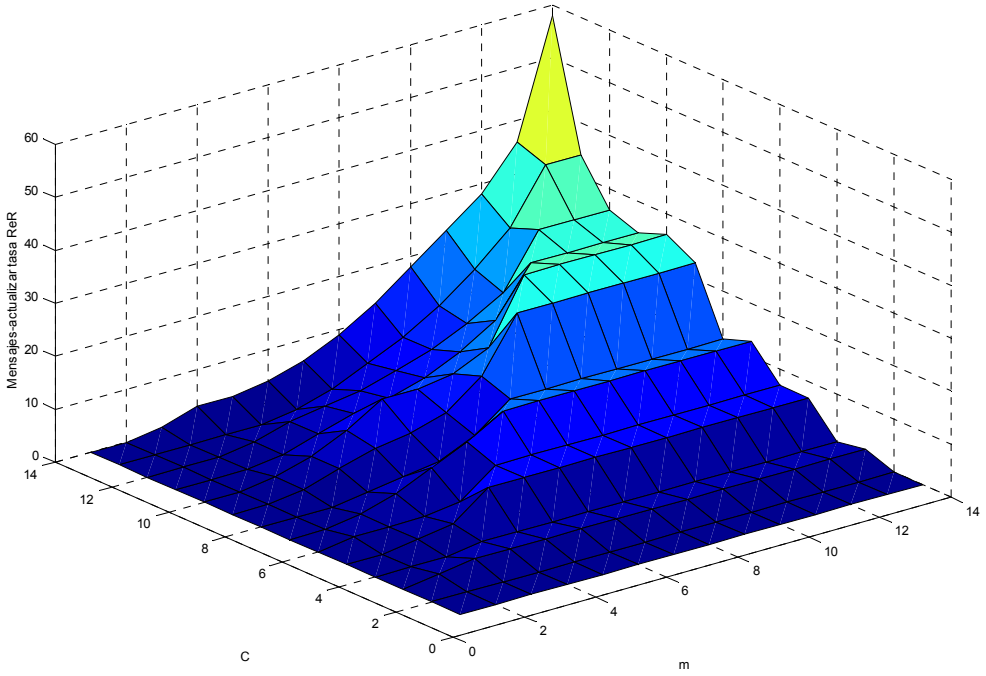

Figura 5.8: Mensajes de señalización extra (actualizar tasa de las reservas) en función de los parámetros m y $\mathcal{C}$.

La evolución de los mensajes de señalización extra debidos a la necesidad de liberar alguna reserva por ser $m = \mathcal{C}_{ReR}$ y la necesidad de dar servicio a una nueva petición se muestra en la Figura 5.9.

Se observan nuevamente, tres zonas destacables:

1. $m < \mathcal{C}$: en esta zona, a medida que m crece, el número de mensajes extra necesarios disminuye, ya que la probabilidad de que una nueva petición se encuentre con $m = \mathcal{C}_{ReR}$ se reduce rápidamente.

2. $m = \mathcal{C}$: se observa un punto de inflexión a partir del cual la señalización extra es nula para $m > \mathcal{C}$.

3. $m > \mathcal{C}$: la señalización extra es nula, ya que m siempre es mayor que $\mathcal{C}_{ReR}$ (5.10).

Este caso a diferencia de los demás requiere que el servidor trabaje en la zona $m > \mathcal{C}$, o lo más cerca posible de $m = \mathcal{C}$.

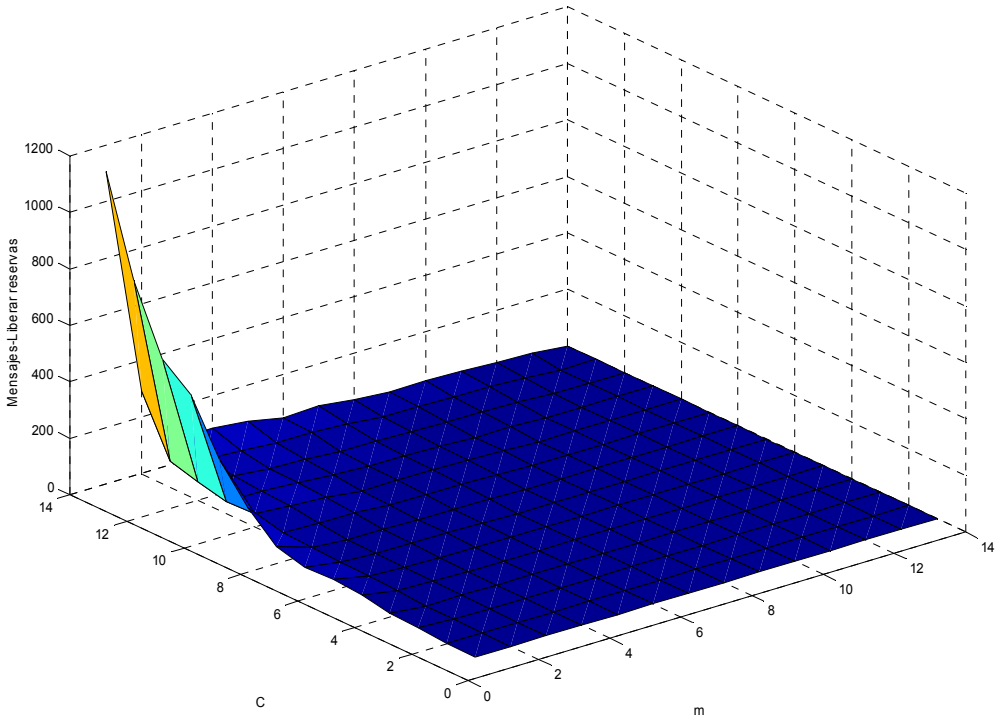

Figura 5.9: Mensajes de señalización extra (liberar reserva) en función de m y $\mathcal{C}$.

Finalmente, la Figura 5.10 muestra la señalización extra total necesaria, suma de las anteriores. Como se puede deducir de la visualización de las gráficas anteriores, la zona óptima de trabajo está en $m < \mathcal{C}$, pero dado que la señalización debida la necesidad de liberar reservas si $m = \mathcal{C}_{ReR}$ es mayor para m menor, se observa una zona óptima donde la señalización es mínima, que es en la que debería de operar el servidor. Esto implica que el valor de m empleado dependerá del número de clientes a los que se de servicio en cada momento, para reducir así la señalización trabajando en la zona óptima. Por otro lado, cabe recordar que m deberá ser lo más pequeña posible para maximizar la eficiencia media del sistema.

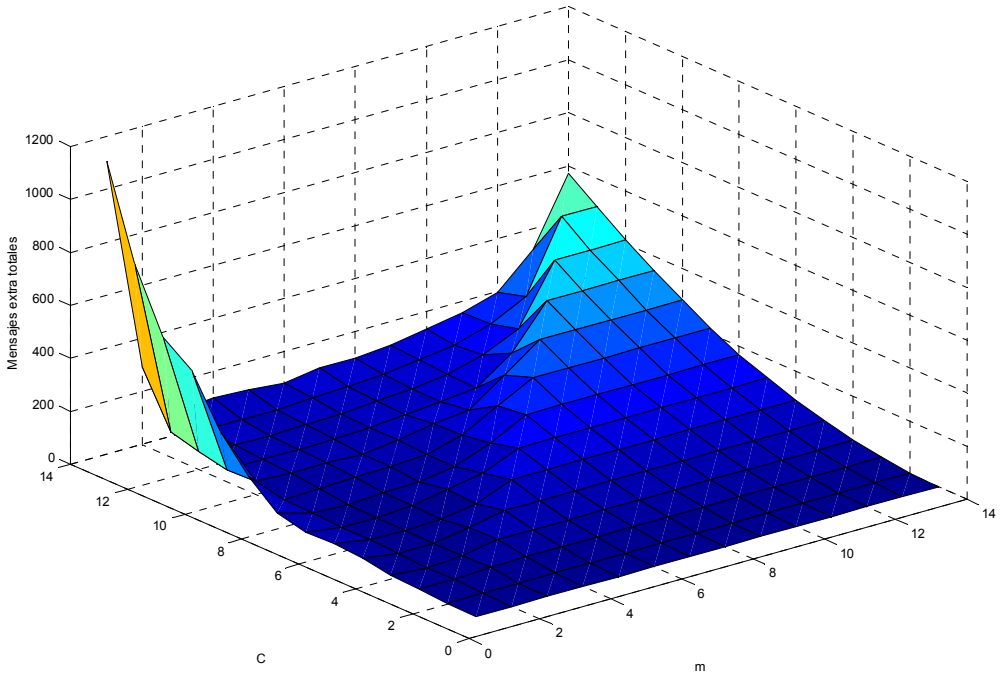

Figura 5.10: Mensajes de señalización extra en función de m y C.

5.5 Conclusiones

En este capítulo, se ha analizado cómo debe realizarse el servicio simultáneo de varios flujos semi-elásticos. El principal problema que se plantea es la gestión del ancho de banda de acceso a la red que posee el servidor. Este ancho de banda se puede agotar fácilmente si no se gestionan debidamente las reservas de recursos.

Se ha propuesto un método genérico de servicio de flujos semi-elásticos, basado en permitir que m clientes reserven recursos simultáneamente, repartiéndose el ancho de banda de acceso de igual forma. Para gestionar este ancho de banda es necesario utilizar señalización extra respecto al servicio a un solo cliente. Esta señalización extra se produce, si es necesaria, al inicio y final de una sesión, para actualizar reservas realizadas por clientes anteriores y para liberar reservas menos prioritarias (las que llevan más tiempo) que la última petición de reserva recibida. Tal y como se ha visto, para realizar correctamente esta gestión se requiere que el protocolo de señalización de reserva de recursos permita al servidor, además de al cliente, también poder cambiar la reserva.

Como casos concretos se ha analizado el servicio con reserva de recursos por cliente (con m igual al número máximo de clientes a los que se puede dar servicio) y el servicio con reserva de recursos para un cliente (con m igual a 1).

El sistema se ha simulado para analizar como evoluciona la eficiencia media del sistema, así como la señalización extra necesaria. De ello se deduce que la eficiencia es mayor a medida que m es menor, dado que el ancho de banda que reserva cada cliente también es mayor. En cuanto a la señalización extra se observa una zona donde la señalización es mínima con m menor que el número de clientes $\mathcal{C}$ pero aumentando con $\mathcal{C}$. Lo que permite concluir que el servidor deberá variar el valor de m en función del número de clientes para trabajar en la zona óptima, utilizando siempre el valor más pequeño posible para adicionalmente maximizar la eficiencia media.

Capítulo 6
Conclusiones y líneas futuras

6.1 Conclusiones

La gran popularidad de Internet ha hecho que las redes IP se conviertan en la vía más utilizada de transporte de información. Esto, a su vez, ha favorecido el rápido crecimiento de la red en cuanto a número de usuarios, número de enlaces y ancho de banda, dispositivos que la forman, velocidad de conmutación, etc. Por otra parte, este mayor uso de la red, junto a las mejoras tecnológicas que aparecen constantemente, hace posible el planteamiento de nuevos servicios y aplicaciones, que requieren mayores niveles de calidad de servicio (*Quality of Service*, QoS) que los ofrecidos actualmente. Debido a esto, uno de los campos de investigación más extensos y activos en este ámbito, es el de provisión de QoS en redes IP, en el que en los últimos años se han propuesto diversos protocolos y mecanismos que intentan solventar al menos en parte este problema.

Hoy por hoy, la provisión de QoS extremo a extremo en una red IP no es aún una realidad en Internet, ya que los dispositivos comerciales que los proveedores de red aportan, todavía no implementan estos mecanismos de forma amplia. De cualquier forma, es posible proporcionar QoS extremo a extremo tanto en entorno LAN (*Local Area Network*, Red de Área Local) como en entorno WAN (*Wide Area Network*, Red de Área Extendida). Así, en entorno LAN existen versiones de la más popular de las redes locales (Ethernet), que presentan mecanismos para ofrecer calidad de servicio diferenciada entre dos nodos locales. Además en este entorno, los

hosts pueden utilizar el protocolo RSVP-E2E (*resource ReSerVation Protocol End-to-End*), característico de la arquitectura de Servicios Integrados (*IntServ*), para señalizar las necesidades de QoS extremo a extremo. Esto es posible, ya que el número de conexiones presentes en este entorno es relativamente pequeño y fácil de gestionar. Por otra parte, también se ha definido la proyección o el "mapeo" necesario entre RSVP-E2E y los mecanismos de provisión de QoS de Ethernet mediante SBM (*Subnet Bandwidth Manager*). Finalmente, en el acceso a la red WAN, un *router* se encargará de gestionar el tráfico y negociar la QoS necesaria con la red WAN. Este es un punto importante, ya que es necesario mapear las necesidades de QoS extremo a extremo, requeridas en el entorno LAN, con las necesarias en el entorno WAN, teniendo en cuenta que ambos entornos utilizan tecnologías de provisión de QoS diferentes.

En entorno WAN, el tráfico proveniente del entorno LAN se agrega en grupos con requerimientos similares. Esto se realiza para evitar gestionar un número de flujos de datos, que pueda llegar a ser muy grande. Así, se utiliza una arquitectura de Servicios Diferenciados (*DiffServ*), junto a tecnología de conmutación MPLS (*MultiProtocol Label Switching*), para permitir que el tráfico fluya de forma rápida y más o menos de forma óptima, teniendo en cuenta el estado de la red.

Una red en la que los usuarios puedan obtener QoS extremo a extremo, presenta la ineludible necesidad de aplicar una tarifa superior a este nuevo servicio, diferenciado del clásico *best-effort*. Ello debe obligar a los demandantes de este servicio que reservará recursos de la red, a no exceder sus necesidades reales de QoS. Dichas necesidades son dependientes del tipo de aplicación generadora del flujo de información que circulará por la red. Estos flujos se pueden clasificar en flujos elásticos, inelásticos y semi-elásticos, según sean más o menos estrictos sus requerimientos de QoS. Los flujos elásticos no necesitan reservar recursos de la red, ya que la información no tiene un instante concreto de consumo. Por el contrario, los flujos inelásticos requieren reservar recursos durante toda la transmisión de datos, ya que la información se genera y se consume en tiempo real. Finalmente, los flujos semi-elásticos se caracterizan por precisar una mayor o menor reserva de recursos, dependiendo del estado de la red. Ello se debe a que en este caso, la información se consume en tiempo real, pero está previamente almacenada en un servidor, por lo que se puede controlar la velocidad con que se envían los datos. Por tanto, para estos flujos será necesario gestionar correctamente las reservas de recursos para evitar un

uso excesivo de este servicio, que encarezca la transmisión y afecte el grado de servicio que ofrezca la red.

Los flujos semi-elásticos necesitan reservar recursos cuando el cliente, que consume los datos, detecta que la cantidad de datos almacenados localmente es demasiado pequeña y existe el peligro de que se agoten. Sin embargo, si el nivel de datos es suficiente, es posible realizar la transmisión en modo *best-effort*. Evidentemente, en este modo de transmisión los datos llegarán al cliente con una tasa dependiente del estado de la red. Así, si dicha tasa es pequeña, se deberán reservar más recursos que si es mayor. Para controlar el modo de transmisión, el cliente determina dos umbrales (uno mínimo y uno máximo) que le permiten detectar si es necesario reservar recursos o no. De este modo, si el nivel de ocupación de su memoria llega al nivel máximo, se podrá utilizar el modo *best-effort*, mientras que si la ocupación alcanza el nivel mínimo, será necesario reservar recursos para garantizar el llenado de la memoria. El umbral mínimo de esta memoria garantiza, por tanto, la disponibilidad de datos durante toda la transmisión, mientras que el umbral máximo se puede fijar a un valor concreto que permita minimizar los recursos que se reservan de la red. Estos valores, tratan de conseguir que la memoria del cliente se encuentre en el nivel mínimo al finalizar la transmisión, la cual cosa implica que los recursos reservados han sido mínimos.

Para poder calcular el umbral máximo óptimo es necesario conocer la tasa de llegada de paquetes en modo *best-effort*, ya que ésta es la que determina la cantidad de recursos a reservar. Para conseguir una primera estimación, sin necesitar de otros procesos para realizarla, se utiliza un nivel máximo inicial que obliga al sistema a pasar a modo de transmisión *best-effort*, en donde se realizará la estimación de la tasa, que se utilizará para calcular el umbral máximo óptimo. Por otra parte, la variabilidad en esta tasa provoca que sea difícil obtener una apropiada minimización del coste, ya que es posible que una estimación determinada no corresponda con la real en el instante de finalización de la transmisión. Para evitar este problema, es posible disminuir el umbral máximo hacia el final de la transmisión, para asegurar que al acabar, el nivel de datos estará cercano al nivel mínimo. Esto se debe realizar de forma que no se incremente excesivamente el número de renegociaciones de modo de transmisión, lo que podría provocar un aumento excesivo del coste de señalización.

También es importante gestionar correctamente el servicio simultáneo de flujos semi-elásticos. Este servicio debe tener en cuenta que varios clientes pueden requerir una reserva de recursos de forma simultánea, lo que puede agotar fácilmente el ancho de banda de acceso a la red, disponible para reservar recursos. Por otra parte, cuanto mayor sea el ancho de banda reservado, mayor podrá ser la reducción del coste obtenida. De esta forma, este servicio deberá satisfacer el que un determinado número de flujos de datos se pueda servir en modo de reserva de recursos, obligando al resto de flujos a servirse en modo *best-effort*. Esto asegura que la tasa que se reserva no sea demasiado pequeña, y además reduce considerablemente la señalización necesaria para gestionar este servicio.

6.2 Líneas futuras

Como líneas futuras de actuación para ampliar la contribución realizada con este trabajo, se plantean los aspectos propuestos a continuación.

Claramente, el sistema cliente optimizador del coste presenta carencias en redes IP donde la QoS no está totalmente implementada, en redes diseñadas de forma inadecuada, con escasez de recursos, o en redes con mala gestión del tráfico que carga la red. En todos estos casos, es posible que se reclame el uso de reserva de recursos, y que la red no la proporcione, provocando un mal funcionamiento del sistema. Por tanto, una de las principales vías de actuación es el estudio de estas situaciones y la propuesta de mecanismos que permitan reducir dicho inconveniente. En esta contribución se ha realizado una primera aproximación a una solución basada en el diseño del nivel mínimo de la memoria, pero se pueden plantear otras opciones.

Otra de las líneas futuras de actuación es la investigación de cómo afectaría al sistema de minimización del coste, el uso de una conexión permanente que utilizara el servicio *best-effort* durante toda la transmisión, haciendo uso de una segunda conexión para enviar datos en modo de transmisión con reserva de recursos cuando fuera necesario. Esta forma de transmisión, reduciría todavía más el coste, ya que se podrían aprovechar los periodos en los que se reserva recursos para continuar enviando en modo *best-effort*.

Es también interesante el estudio de la interactuación del sistema optimizador con las aplicaciones que lo utilizan. Estas aplicaciones pueden requerir que los datos

se envíen o se consuman de una determinada forma. Por ejemplo es posible que los datos viajen comprimidos de forma que el consumo de información de la aplicación sea constante, pero no el consumo de datos de la memoria del proceso optimizador del coste. También podría ocurrir que la aplicación requiera que los bloques de información que se extraen de dicha memoria sean demasiado grandes como para que el control que se realiza de su ocupación se vea afectado negativamente. Esto supondría el proponer un sistema que permitiera que los datos se consumieran de la memoria en bloques más pequeños, que acabarían formando los más grandes, necesarios por la aplicación.

Por otro lado, también se puede plantear el estudio de la interrelación con los protocolos y mecanismos de provisión de QoS concretos. Como se ha visto en este trabajo, para servir a varios clientes simultáneamente utilizando el protocolo RSVP, es necesario añadir un mensaje nuevo que permita modificar la reserva de recursos desde el extremo emisor. Por tanto, sería interesante analizar otras propuestas similares a RSVP que permiten esta habilidad, así como el estudio más en profundidad de las implicaciones que tendría añadir dicho mensaje en la versión actual de RSVP.

Finalmente, se propone como vía futura de estudio, el diseño de un servidor que automáticamente determine la cantidad de clientes a los que se debe permitir el servicio en modo de reserva de recursos, para que la señalización extra necesaria para gestionar el servicio sea mínima. Así como el estudio de un caso genérico con clientes heterogéneos.

Anexo A
Implementación del sistema en ns-2

A.1 Introducción

En este anexo se tratará todo lo referente a las modificaciones, implementaciones y simulaciones realizadas en el simulador *Network Simulator* 2 (ns-2) [NS2] para obtener características y conclusiones sobre el comportamiento del sistema cliente servidor de flujos semi-elásticos en Internet con calidad de servicio extremo a extremo.

Se describirá cómo se ha desarrollado, desde el inicio, el sistema cliente servidor en el ns-2, describiendo las principales funciones, la forma de acceder desde OTcl (lenguaje utilizado en el simulador junto a C++ para desarrollar escenarios de simulación) y algún *script* de ejemplo. También se detallará el entorno propuesto de simulación, así como los distintos parámetros que lo caracterizan. Los principales objetivos planteados mediante la realización de estas simulaciones son los de evaluar el comportamiento del sistema ante diferentes niveles de carga de la red apreciada por el cliente, con diferentes tipos de tráfico de relleno (*background*) y también probar su comportamiento en función de alguno de los parámetros controlables por la aplicación (longitud del paquete y tamaño de la memoria del cliente).

A partir de los resultados obtenidos mediante las simulaciones, se confeccionan las diversas gráficas, utilizadas en los capítulos 4 y 5, que permiten analizar

visualmente la respuesta del sistema y poder deducir las conclusiones oportunas en cada caso.

A.2 Network Simulator 2 (ns-2)

El *Network Simulator* 2 (ns-2) es un simulador ampliamente aceptado y utilizado en el análisis de redes. Ns-2 ha sido desarrollado como parte del proyecto VINT (*Virtual InterNetwork Testbed*) [VIN96], proyecto en el que han colaborado la Universidad de California del Sur, el Laboratorio Nacional Lawrence Berkeley, y la Universidad de Berkeley de California. El principal objetivo del proyecto VINT es el de proporcionar una estructura de simulación adecuada para el análisis y el desarrollo de los protocolos de Internet.

Ns-2 es un simulador de eventos discretos que permite la simulación de entornos diversos, tales como: varios protocolos de red (transporte, *multicast*, *routing*), topologías simples o complejas (incluyendo la generación de dichas topologías), agentes (que se definen como puntos extremos donde los paquetes se crean o se consumen a nivel de red), varias fuentes de generación de tráfico, simulación de aplicaciones (FTP, *Telnet*, *Web*), diversas políticas de gestión de colas, modelado de errores, redes de área local, redes inalámbricas, etc.

El código del simulador ns-2 está escrito en C++ y en *Object Tcl* (OTcl). Hay una correspondencia biunívoca entre una clase en C++ (jerarquía compilada de clases en ns-2) y una clase en OTcl (jerarquía interpretada). Esta arquitectura de programación permite una gran personalización de la simulación de las rutinas a nivel de paquete *software* (implementadas en C++), y una configuración y un control flexible de la simulación usando un lenguaje sencillo de interpretar, como el OTcl [BAJ99].

A.2.1 Instalación de ns-2

El *software* necesario para la instalación de ns-2 en el sistema operativo Linux (en el caso que nos ocupa es concretamente *SUSE 7.1*) puede conseguirse de forma gratuita en la siguiente página web: *http://www.isi.edu/nsnam*. En esta misma página, se encuentran enlaces a diferentes manuales, tutoriales, preguntas frecuentes (*Frequent Answer Questions*, FAQ's), acceso a foros, etc. El simulador consta de los paquetes básicos mostrados en la Tabla A.1.

Se puede instalar el producto de dos formas, paquete por paquete o todo a la vez. Se recomienda para aquellos usuarios que no sean grandes expertos en ns-2 que utilicen esta segunda opción. Para ello, se deberá bajar un fichero comprimido con todos los paquetes, con "todo en uno", que ocupa unos 50 *Mbytes* de espacio sin descomprimir. El nombre del fichero para la versión utilizada es el siguiente: *ns-allinone-2.1b8a.tar.gz*.

Paquete	Versión Instalada para este Proyecto
Tcl	8.3.2
Tk	8.3.2
Otcl	1.0a5
TclCL	1.0b9
Ns	2.1b8a
Nam	1.0a8 / 1.0a11
Xgraph	12.1
SGB	1.0
CWEB	3.4g
Zlib	1.1.3

Tabla A.1: Paquetes instalados con ns-2.

Una vez descomprimido el paquete, se obtienen todos los paquetes desplegados en el directorio *ns-allinone-2.1b8a* en la estructura de ficheros adecuada para su correcto funcionamiento. Para compilar los ficheros fuente se deberá ejecutar el *script* de instalación en el directorio del ns.

También se recomienda ejecutar el *script* de validación para ver si funcionan correctamente los paquetes en el sistema operativo.

Se recomienda configurar adecuadamente los *path's* del sistema para poder ejecutar Ns, Nam y Xgraph desde cualquier directorio. Lo más sencillo será editar el fichero *.bashrc* añadiendo los *path's* y las librerías necesarias.

Si es la primera vez que se usa ns-2, resulta muy útil (y recomendable) estudiar el tutorial que se encuentra en la siguiente dirección: *http://www.isi.edu/nsnam/ns/tutorial/index.html*.

A.2.2 *Scripts* de entrada para simular

Toda la información de configuración y control de una simulación en ns-2 está especificada en el *script* OTcl de entrada. Los objetos de simulación (nodos, enlaces, fuentes de tráfico, etc.) son creados a través de instancias en el *script*, e inmediatamente se reflejan en la jerarquía compilada de C++. El *script* de entrada define la topología, construye los agentes (fuentes y destinos), especifica los ficheros de trazas y los tiempos de comienzo de los eventos iniciales en la simulación. Estos eventos iniciales, probablemente creen nuevos eventos a lo largo de la simulación, estos harán lo propio, y así sucesivamente. El simulador siempre ejecuta los eventos en el orden especificado en la lista de eventos, y en esta lista los eventos estarán siempre ordenados por tiempo.

A.2.3 Simulaciones conducidas por trazas

En ns-2 se permite que la simulación dependa de un fichero de trazas a través de la clase Otcl denominada *Application/Traffic/Trace*, que es una subclase de las *Application/Traffic*. Los objetos de esta clase *../Trace* se relacionan a un fichero de trazas particular, que deberá ser una lista de parejas de 32 *bits* indicando lo siguiente: <*tiempo entre paquetes, tamaño de paquete*>. El primer valor indica el tiempo en μs hasta la llegada del siguiente paquete y el segundo indica el tamaño del paquete en *bytes*. Para evitar la sincronización del tráfico generado por las fuentes, cada fuente empezará a emitir en un instante aleatorio y además, si se alcanza el final de la traza antes del fin de la simulación, la fuente continúa emitiendo, leyendo de nuevo desde el comienzo del fichero que contiene la traza.

A.2.4 Trazas de salida de las simulaciones

La forma de conseguir trazas en ns-2 es a través del uso de objetos *trace* o *monitor*. Los objetos *trace* recolectan los datos para cada llegada de paquete, salida o descarte. Los objetos *monitor* recolectan los datos en un nivel agregado y se implementan como contadores de parámetros específicos (número total de paquetes o de *bytes* que llegan, salen o se descartan), siendo especialmente útiles cuando se necesita saber información básica de la dinámica de la simulación.

El formato de la traza de salida en ns-2 tiene un formato fijo; como ejemplo de una traza de salida podemos ver la Figura A.1 mostrada a continuación.

```
1     2       3   4   5   6       7   8   9      10    11   12
↓     ↓       ↓   ↓   ↓   ↓       ↓   ↓   ↓      ↓     ↓    ↓
r 17.922645 201 202 tcp 552 ------- 0 10.47 191.17 28 49925
+ 17.922645 202 203 tcp 552 ------- 0 10.47 191.17 28 49925
d 17.922645 202 203 tcp 552 ------- 0 10.47 191.17 28 49925
r 17.922648 203 199 udp 200 ------- 0 204.0 199.60 3143 49809
r 17.922744 203 202 ack 40 ------- 0 191.12 10.42 394 49934
+ 17.922744 202 201 ack 40 ------- 0 191.12 10.42 394 49934
- 17.922744 202 201 ack 40 ------- 0 191.12 10.42 394 49934
- 17.922913 202 203 ack 40 ------- 0 133.90 180.10 72 49904
r 17.922968 203 199 udp 200 ------- 0 204.0 199.60 3144 49813
```

Figura A.1: Ejemplo de traza de salida del ns-2.

Cada una de las columnas anteriores indica lo siguiente:

1) Tipo de evento. Puede ser *received* (r), *enqued* (+), *dequed* (-) o *dropped* (d).

2) Instante de tiempo del correspondiente evento.

3) Uno de los dos nodos en la topología entre los que ocurre la traza.

4) El otro nodo implicado en la traza.

5) Descripción del paquete particular (TCP *data*, TCP *ack*, UDP, CBR, etc.).

6) Tamaño del paquete en *bytes*.

7) Puede contener diversos *flags*, pero no se suele utilizar.

8) Identificador del flujo IP definido en IPv6.

9) Dirección origen utilizada internamente por el simulador.

10) Dirección destino utilizada internamente por el simulador.

11) Número de secuencia del paquete.

12) Identificador único de paquete.

A.2.5 Nam

El *Network Animator*, más conocido como Nam, viene incluido en la implementación de ns-2 como una aplicación externa que permite visualizar de forma gráfica lo que sucede durante la simulación. Su funcionamiento es sencillo, de

modo que a partir de las definiciones del entorno establecidas en el fichero Tcl, crea un entorno gráfico que representa los nodos y enlaces del sistema.

El Nam permite, además, ver qué es lo que pasa a lo largo del tiempo, pudiendo personalizar la velocidad de representación, retroceder, avanzar o manejar el tiempo a gusto propio. Esto lo consigue a partir del fichero de traza definido anteriormente, que especifica los eventos en cada instante de tiempo, de modo que el Nam se encarga de interpretar dicho fichero en formato de salida de ns-2 y convertirlo a un entorno visual más agradable para el usuario. Un ejemplo de una representación visualizada con el Nam, podemos verlo a continuación en la Figura A.2.

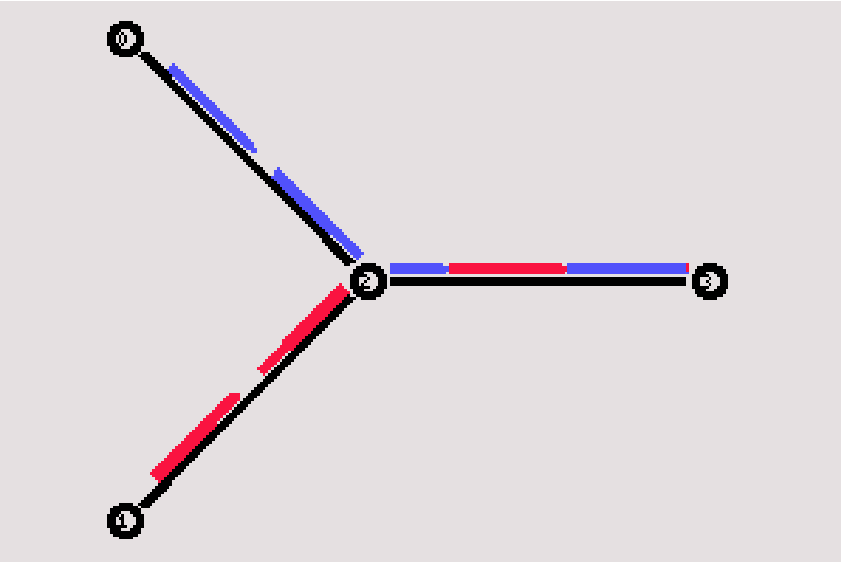

Figura A.2: Ejemplo de animación visualizada con el Nam.

La simulación puede controlarse a través de un panel visual y de botones específicos, los cuales quedan representados en la Figura A.3, donde podemos apreciar las múltiples opciones que nos facilita este Nam a la hora de visualizar cómo se desarrolla la animación.

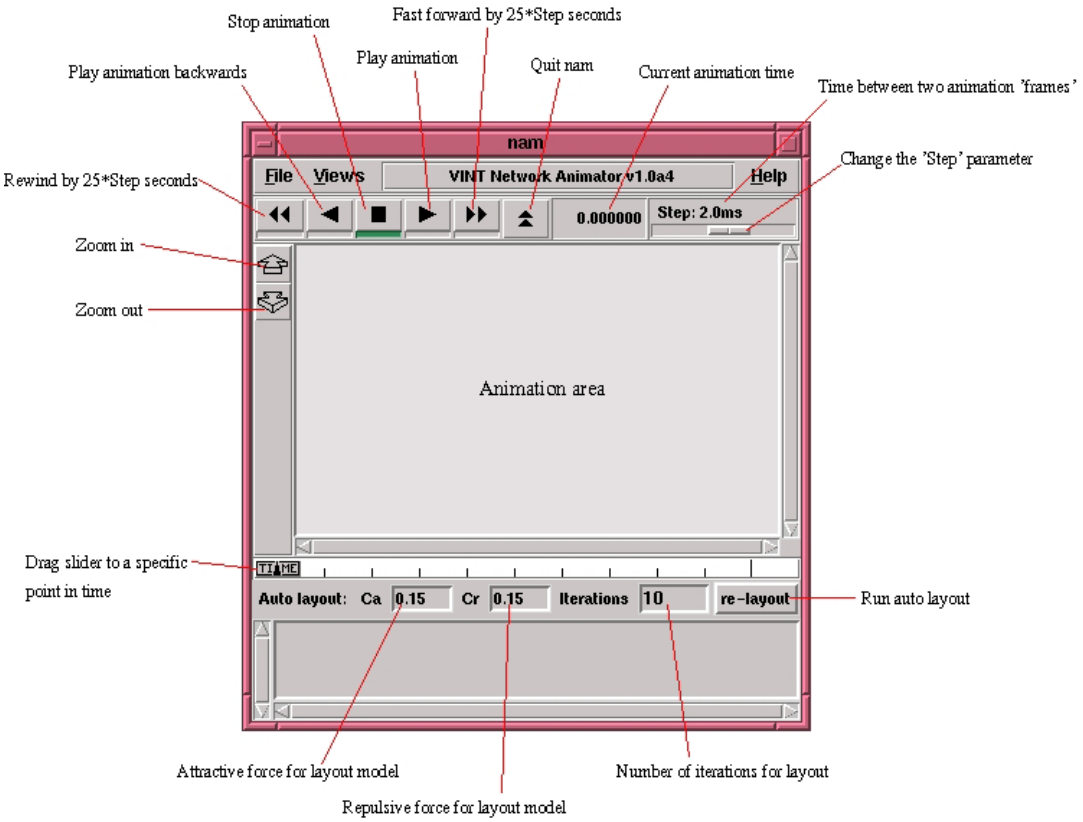

Figura A.3: Panel de Control del Nam.

A.2.6 Xgraph

También es posible obtener trazas personalizadas en ficheros externos, de modo que se puedan monitorizar distintas variables del programa y su evolución en determinados momentos de la simulación. Xgraph es una aplicación incluida en la instalación del *Network Simulator* que permite dibujar gráficas en 2 dimensiones a partir de los datos de ficheros organizados de una forma determinada. El formato que admite este programa es el de ficheros con dos columnas de datos: la primera indica la coordenada X (abscisas) y la segunda indica su correspondiente valor en el eje Y (ordenadas).

Un uso común de esta utilidad suele ser el generar un evento que cada cierto tiempo vaya apuntando en un fichero el valor del tiempo actual de simulación y también el valor de la variable a monitorizar, de modo que al final lo que obtenemos es la evolución temporal de dicha variable y la gráfica correspondiente. Podemos ver un ejemplo de dicha representación en la Figura A.4.

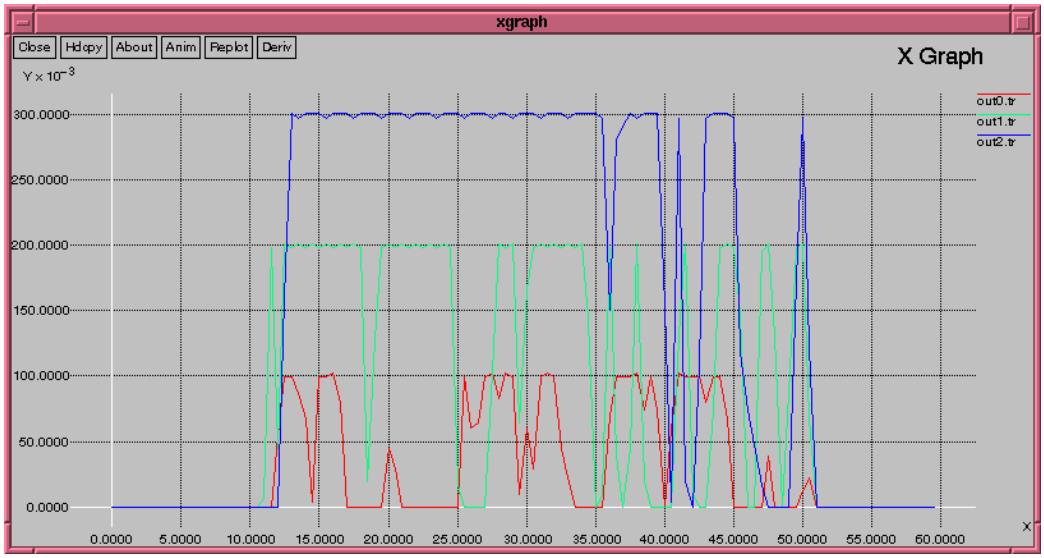

Figura A.4: Ejemplo de Visualización de resultados con Xgraph.

A.2.7 RSVP/NS

El protocolo de reservas de recursos RSVP (*resource ReSerVation Protocol*) forma parte de la arquitectura de Servicios Integrados (*IntServ*), y permite a las aplicaciones llevar a cabo reservas de recursos.

Para utilizar RSVP/NS, básicamente lo que debemos saber es cómo crear/configurar enlaces y agentes RSVP.

Configuración de un enlace RSVP

Para crear un enlace RSVP entre los nodos *n1* y *n2* se utiliza el siguiente comando:

```
ns duplex-rsvp-link <n1> <n2> <bw> <delay> <reservable> <rsvp> <queue>
   <adc> <est>
```

Donde cada argumento significa lo siguiente:

- ns: es una instancia del simulador.
- `bw`: el ancho de banda para este enlace.
- `delay`: el retardo del enlace.
- `reservable`: la porción del ancho de banda del enlace que se puede usar para RSVP.
- `rsvp`: el ancho de banda (en *bits* por segundo) que queda reservado para mensajes RSVP.

- `queue`: el tamaño de la cola para la clase *best-effort* (en *bytes*).
- `adc`: el algoritmo de control de admisión.
- `est`: el estimador usado para las medidas en las que se basan los algoritmos de control de admisión.

Creación y configuración de agentes RSVP

Un agente RSVP se añade a un nodo con el comando:

```
n1 add-rsvp-agent

set rsvpagent [$n1 add-rsvp-agent]
```

Todas las opciones se pueden fijar para cada agente RSVP o globalmente mediante el siguiente comando:

```
Agent/RSVP set <option> <value>
```

El resto de opciones y parámetros disponibles se encuentran detallados en [GRE02], así como diversa información sobre cómo implementar rsvp/ns.

A.3 Implementación del Cliente-Servidor en ns

La implementación del sistema cliente-servidor de flujos semi-elásticos ha sido desarrollada específicamente para este trabajo, no habiendo nada relacionado con este tipo de flujos en las distribuciones estándar de ns-2. Para entender cómo se ha creado este sistema cliente-servidor en C++, resulta básico el crear un árbol de clases en el que queden reflejadas de forma visual las relaciones entre las diferentes clases. También será importante saber qué funciones y parámetros tiene cada fichero, así como las características que éstos implementan.

A.3.1 Árbol de Clases

Existen un total de 5 clases, que forman el sistema cliente-servidor de flujos semi-elásticos, relacionadas entre sí según se muestra en la Figura A.5 y son las siguientes:

- *Cliente*
- *ClienteQoS* → Hereda de *Cliente*
- *Servidor*
- *ServidorQoS* → Hereda de *Servidor*

- *ServidorQoSRated* → Hereda de *ServidorQoS*

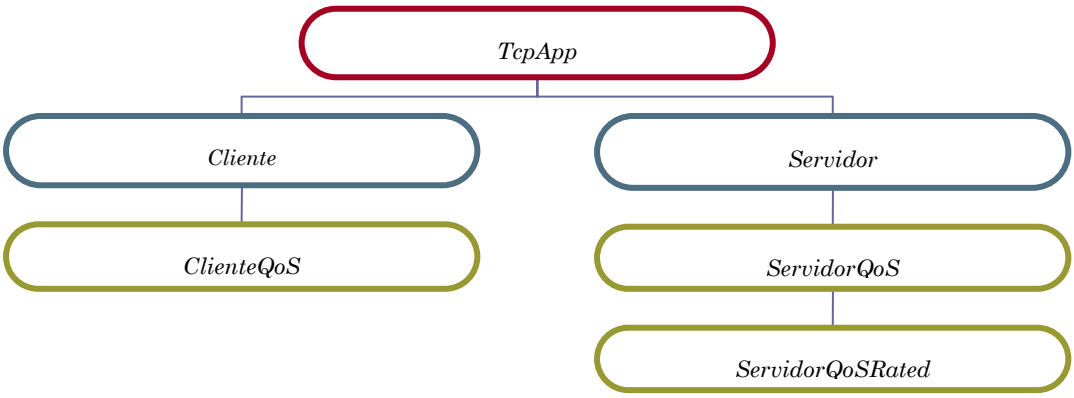

Figura A.5: Árbol de Clases del Sistema Cliente-Servidor.

Como se observa, tanto el cliente como el servidor heredan de *TcpApp*, que es la clase genérica utilizada para crear aplicaciones que utilizan el protocolo TCP.

Los dos primeros objetos desarrollados son un *Cliente* y un *Servidor* genéricos. El *Cliente* pide una cierta cantidad de información al *Servidor*. Cuando el servidor recibe la petición, empieza la transmisión de un fichero del tamaño requerido, usando el protocolo TCP. El fichero se divide en paquetes de longitud constante que posteriormente se entregan al cliente. Esta división en paquetes es útil, ya que el TCP del extremo cliente espera hasta que llega el paquete entero para pasarlo a la aplicación, y el cliente requiere poder controlar su ocupación. Por tanto, la ocupación de la memoria del cliente variará de forma más o menos brusca dependiendo del tamaño de este paquete, afectando al comportamiento del método de minimización propuesto. El objeto *Cliente*, simplemente espera la llegada de información y la almacena en su memoria, para su posterior consumo.

El escenario de simulación contempla el estudio del sistema cliente servidor de flujos semi-elásticos en una red con calidad de servicio QoS extremo a extremo. Por esta razón, seleccionamos RSVP (*resource ReSerVation Protocol*) para reservar recursos de red. En [GRE02], hay una implementación de RSVP para ns-2. Obviamente, en una Internet real, RSVP se utilizará junto con otros mecanismos de provisión de QoS, pero el objetivo es obtener una primera aproximación al problema usando módulos de ns-2 existentes, para centrarse en el desarrollo del sistema cliente-servidor.

De la clase *Servidor* hereda la clase *ServidorQoS*, que es un servidor que utiliza RSVP para reservar recursos de la red. Por otro lado, la Clase *ClienteQoS* hereda de *Cliente* y es un nuevo cliente que usa el método expuesto en el Capítulo 4 para minimizar el uso de recursos de red reservados.

Inicialmente, *ServidorQoS* espera una petición de *ClienteQoS*. Cuando la recibe, *ServidorQoS* inicia una sesión RSVP y envía los paquetes de datos al TCP para que sean transmitidos. Estos paquetes se envían al TCP con tasa controlada para no incumplir el contrato de QoS establecido con la red y no saturar la memoria del cliente, en caso de que la red esté poco utilizada. Esto se consigue mediante la clase *ServidorQoSRated* que hereda de *ServidorQoS*.

Por su parte, *ClienteQoS* pide información al servidor y espera respuesta. Cuando empieza a recibirla, el cliente inicia la lectura de datos inmediatamente, ya que la reserva de recursos inicial (véase el Capítulo 4) asegura el llenado de la memoria. Por tanto, el cliente inicia la gestión de la ocupación de su memoria cuando llega el primer paquete. Esto se consigue controlando que la ocupación de la memoria no supere el umbral máximo. Si esto sucede, *ClienteQoS* cambia el modo de envío de datos del servidor a *best-effort*, indicando al RSVP el envío de un mensaje para eliminar la reserva de recursos, pero manteniendo la sesión RSVP (este mensaje llega al servidor rápidamente, ya que se reserva ancho de banda extra para mensajes RSVP). Además, *ClienteQoS* controla que el nivel de ocupación no sea inferior al umbral mínimo. De forma similar, si esto ocurre, el cliente cambia a modo de reserva de recursos indicándole a RSVP la reserva de recursos adecuada para la sesión activa.

Además de estas 5 clases, se utiliza otra, *Servidor_BWM*, que se encarga de la gestión del ancho de banda de acceso a la red de *ServidorQoS* y *ServidorQoSRated*. Esta clase permite el servicio a múltiples clientes según lo descrito en el Capítulo 5. Recibe llamadas desde el proceso que gestiona el protocolo RSVP[i] cuando se reciben peticiones de reserva de recursos o cuando se finalizan sesiones, para gestionar de forma adecuada el ancho de banda de acceso.

[i] Este mecanismo no viene con la distribución de rsvp/ns y ha debido de ser añadido para realizar las simulaciones.

A.3.2 Descripción de los ficheros

Se detallan uno a uno los principales ficheros fuente desarrollados en C++ (*.cc), explicando de forma estructurada los siguientes contenidos: descripción, forma de llamarlos desde un *script* Tcl, parámetros, ficheros cabecera (*includes*), funciones.

Los ficheros creados son los 8 siguientes: cliente, clienteqosvarbe, rsvpapp, lector, servidor, servidorqos, servidorqosrated y bwm.

- **Cliente.cc:**

Aplicación que emula el comportamiento de un cliente que efectúa una petición de un fichero al servidor indicándole el tamaño del fichero deseado.

La relación biunívoca entre las clases en C++ y Tcl queda reflejada en la siguiente línea de código del cliente:

```
ClienteClass():TclClass("Application/TcpApp/Cliente")
```

Los principales parámetros accesibles desde el *script* Tcl son los siguientes:

```
client_rate_          // Tasa de lectura del Cliente

level_                // Nivel de ocupación del buffer

size_                 // Tamaño del fichero que se demanda
```

Los *Includes* necesarios son:

```
Cliente.h

Lector.h
```

Las principales funciones del Cliente son las detalladas a continuación:

```
Void Cliente::process_data (int size, AppData* data)
```

Si la cola de salida está vacía, se activa el lector para que lea el siguiente paquete al acabar de procesar el paquete actual.

```
Void Cliente::timeout (int tno)
```

Si se agota el *timeout* en la lectura de un paquete de la cola, lo borra y se lee el siguiente.

```
Int Cliente::command (int argc, const char*const* argv)
```

Si es una petición, el cliente envía un *request* con el tamaño del fichero demandado (*size_*).

- **Clienteqosvarbe.cc:**

Es el cliente que ofrece Calidad de Servicio a través de la monitorización del nivel de datos en su memoria y al realizar las reservas de recursos con RSVP en momentos determinados.

Además, también se encarga de ir reduciendo el valor de *Max* hacia el final de la transmisión, de modo que se consigue reducir el error en la estimación de la tasa de *best-effort* y por tanto, también se reduce el error respecto a la minimización del coste de la conexión a costa de unas cuantas renegociaciones extras.

La relación biunívoca entre las clases en C++ y Tcl queda reflejada en la siguiente línea de código del cliente:

```
ClienteQoSClass():TclClass("Application/TcpApp/ClienteQoS")
```

Los principales parámetros accesibles desde el *script* Tcl son los siguientes:

```
max_level_          // Máximo inicial (Max_ini)

min_level_          // Nivel mínimo del buffer del Cliente

TQoS_               // Tiempo Reservando Recursos

rate_be_            // Tasa Best Effort

buffer_max_         // Tamaño máximo memoria del Cliente

data_rec_RES_       // Cantidad de datos recibidos con QoS

delta_rate_d        // Diferencia máxima entre las estimaciones de α_BE
   durante la transmisión
```

Los *includes* necesarios son:

```
Math.h

ClienteQoS.h

Lector.h

Scheduler.h
```

Las principales funciones del Cliente con calidad de servicio son las detalladas a continuación:

```
Void ClienteQoS::process_data (int size, AppData* data)
```

Se llama a la función del Cliente sin QoS: Cliente::process_data (int size, AppData* data).

Se controla el nivel máximo en la cola, de modo que si se iguala o supera el máximo se cambia el modo de transmisión a modo *best-effort*.

Se van tomando estadísticas en las variables sobre el tiempo en que se está en Calidad de servicio (TQoS) y se va estimando la tasa de *best-effort*.

```
Void ClienteQoS::calculate_max_level()
```

Se calcula el nivel del *max_level* a partir de las estimaciones actuales de α_{BE} (*rate_o_*), sin permitir que sólo se realice una estimación de la misma.

```
Void ClienteQoS::timeout (int tno)
```

Se llama a la función del Cliente sin QoS: Cliente::timeout (int tno).

Se controla el nivel mínimo, de modo que si el nivel de ocupación es menor que el mínimo, entonces pasaremos a transmitir reservando recursos con RSVP (modo ReR).

```
Int ClienteQoS::command (int argc, const char*const* argv)
```

Si se quiere adjuntar un agente RSVP, se le pasa como segundo argumento *attach-rsvpagent* y esta función se encarga de hacerlo si los parámetros son los correctos.

- **Rsvpapp.h:**

Sirve para definir la estructura usada por las aplicaciones que usen RSVP durante la transmisión.

Los *Includes* necesarios son:

```
Rsvp.h

Tcp-full.h
```

Se define la estructura requerida:

```
Struct RSVPApp{
        Int sid_;
        Int bucket_;
        Double rate_;
        Nsaddr_t dest_;
        RSVPAgent *rsvpagent_;
        FullTcpAgent *apptcp_;
```

```
        Double ts_;
};
```

- **Lector.cc:**

  Permite leer un paquete de la cola tanto al Cliente como al Servidor.

  Los *Includes* necesarios son:

  ```
  Cliente.h
  Servidor.h
  Lector.h
  ```

  Las principales funciones del Lector son las detalladas a continuación:

  ```
  Lector::Lector (Cliente *a)
  ```

  Para que lea el Cliente.

  ```
  Lector::Lector (Servidor *a)
  ```

  Para que lea el Servidor.

  ```
  Lector::expire (Event *e)
  ```

  Llama al *timeout* del Cliente o del Servidor, según el caso.

- **Servidor.cc:**

  Aplicación que emula el comportamiento de un servidor de ficheros, al cual se le indica el tamaño del fichero demandado y éste lo va enviando fraccionado en paquetes.

  La relación biunívoca entre las clases en C++ y Tcl queda reflejada en la siguiente línea de código del servidor:

  ```
  ServidorClass():TclClass("Application/TcpApp/Servidor")
  ```

  Los principales parámetros accesibles desde el *script* Tcl son los siguientes:

  ```
  long_paq_      // Tamaño del paquete de información que envía el servidor
  ```

  Los *Includes* necesarios son:

  ```
  Servidor.h
  ```

  Las principales funciones del Servidor son las detalladas a continuación:

  ```
  Int Servidor::command (int argc, const char*const* argv)
  ```

Si el servidor recibe un *request*, mira el tamaño del fichero que se pide (*size_*) y comienza a generar paquetes de longitud *long_paq_* hasta igualar ese tamaño.

- **ServidorQoS.cc:**

Es el Servidor anterior, pero añadiendo Calidad de Servicio a la conexión, es decir, que éste permite transmitir en modo RSVP o en modo *best-effort*.

La relación biunívoca entre las clases en C++ y Tcl queda reflejada en la siguiente línea de código del servidor:

```
ServidorQoSClass():TclClass("Application/TcpApp/ServidorQoS")
```

Los principales parámetros accesibles desde el *script* Tcl son los siguientes:

```
rate_   // Tasa del Servidor

bucket_ // Tamaño del token bucket

ttl_          // Time To Live
```

Los *Includes* necesarios son:

```
ServidorQoS.h

Tclcl.h

Rsvp.h
```

Las principales funciones del Servidor con Calidad de Servicio son las detalladas a continuación:

```
Int ServidorQoS::command (int argc, const char*const* argv)
```

Si se recibe un request, genera paquetes igual que en el caso anterior hasta igualar el tamaño del fichero pedido.

Si lo que se recibe es una petición de *attach-rsvpagent*, entonces adjunta este agente según la tabla *<fid> <rsvpapp>*, pasada también por línea de comandos.

- **ServidorQoSRated.cc:**

Se diferencia del anterior en que además de ofrecer QoS, también se encarga de controlar la tasa del Servidor para no desbordar al Cliente.

La relación biunívoca entre las clases en C++ y Tcl queda reflejada en la siguiente línea de código del servidor:

```
ServidorQoSRatedClass():TclClass("Application/TcpApp/ServidorQoSRated")
```

Los principales parámetros accesibles desde el *script* Tcl son los que hereda de ServidorQoS (que a su vez hereda también del Servidor), o sea, los mismos que en el fichero anterior:

```
rate_          // Tasa del Servidor

bucket_        // Tamaño del token bucket

ttl_           // Time To Live
```

Los *Includes* necesarios son:

```
ServidorQoSRated.h

Tclcl.h

Rsvp.h

Tcp-full.h
```

Las principales funciones del Servidor con Calidad de Servicio y control de tasa son las detalladas a continuación:

```
Int ServidorQoSRated::command (int argc, const char*const* argv)
```

Si recibe petición, reserva ancho de banda con RSVP y empieza a generar paquetes hasta igualar el tamaño del fichero pedido.

Se va recalculando el siguiente instante de envío en función de la tasa a la que se pueda enviar (hay control de tasa) y según si estamos con Calidad de Servicio o bien en modo *best-effort*.

```
Int ServidorQoSRated::timeout (int tno)
```

Envía la información y calcula el siguiente instante de envío en función de si ha llegado un mensaje de reserva o si se ha pasado de modo *best-effort* a ReR o viceversa.

```
Int ServidorQoSRated::handle (Event* event)
```

Comprueba que no ha habido cambio de tasa, y si ha habido, entonces cambia el instante de envío.

Si el instante actual es mayor que el instante en que se permite el envío, lo envía directamente y calcula el siguiente instante de envío. Si el instante actual es menor que el instante en que se permite el envío, entonces lo envía en ese instante.

- **BWM.cc:**

En este fichero se define el objeto Servidor_BWM que permite la gestión de ancho de banda de acceso a la red del servidor, en el caso de prestar servicio a múltiples clientes. Este objeto, es único para un servidor concreto, mientras que por cada cliente se crea un objeto ServidorQoSRated, para ofrecer servicio individualizado. El ancho de banda que se reserva para cada conexión, así como el modo de servicio que se utiliza para cada una, está controlado por este objeto.

La relación biunívoca entre las clases en C++ y Tcl queda reflejada en la siguiente línea de código:

```
Servidor_BWMClass():TclClass("Servidor_BWM")
```

Los principales parámetros accesibles desde el *script* Tcl son los que se enumeran a continuación:

```
max_rate_            // Tasa máxima disponible para reservar
bucket_              // Tamaño de bucket a utilizar en las reservas
num_mess_inicio_
num_mess_fin_
num_mess_tasa_
num_mess_quitares_
num_clients_         // Número máximo de clientes
max_resv_clients     // Número máximo de clientes en modo ReR
```

Los *includes* necesarios son:

```
bwm.h
rsvp.h
ServidorQoS.h
```

Las principales funciones del gestor de ancho de banda de acceso del servidor son las detalladas a continuación:

```
Int Servidor_BWM::command (int argc, const char*const* argv)
```

Permite adjuntar los distintos subservidores al objeto Servidor_BWM. Estos subservidores son los encargados de realizar la transmisión del flujo semi-elástico.

```
Int Servidor_BWM::management (int subserver, double rate, long size)
```

Realiza la gestión del ancho de banda de acceso del servidor en función de los diversos parámetros del sistema. Cuando se recibe una petición de reserva de recursos, el proceso RSVP realiza una llamada a esta función antes de continuar a realizar las acciones pertinentes. Indica el subservidor que hace la petición.

```
Int Servidor_BWM::blocking_management (ServidorQoS *server, int block)
```

Gestiona el bloqueo del servidor debido al inicio de sesiones, para evitar que se pase a modo BE a un servidor con una ocupación de su memoria menor que el mínimo necesario.

```
Int Servidor_BWM::ini_sesion (int subserver)
```

Gestiona el inicio de sesión, ajustando debidamente las tasas a reservar por cada cliente.

```
Int Servidor_BWM::fin_sesion (int subserver)
```

Gestiona el final de la sesión, ajustando la tasa a reservar por cada cliente.

```
Int Servidor_BWM::rate_resv (int subserver)
```

Devuelve la tasa que debe reservar un determinado servidor.

A.3.3 Ejemplos de utilización

A modo de ejemplo, se muestran diversos fragmentos ilustrando cómo se usan en la práctica estos ficheros en un *script* Tcl para realizar funciones de creación de aplicaciones o para acceder a las distintas variables de las mismas.

1) Creación de aplicaciones en el Servidor y especificación de sus parámetros.

```
# Crear aplicaciones en el SERVIDOR
#
for {set i 0} {$i < $num_cli} {incr i} {
   set app($i) [new Application/TcpApp/Servidor/ServidorQoSRated $tcp($i)]
   $app($i) attach-rsvpagent $rsvp0 10000
   $app($i) set rate_ 1.8Mb
   $app($i) set bucket_ 100000
   $app($i) set ttl_ 32
   $app($i) set long_paq_ $longitud
```

```
   $bwm attach-subserver $app($i)
}
```

2) Creación de aplicaciones en el Cliente y especificación de sus parámetros.

```
# Crear aplicaciones en CLIENTES
#
for {set i $num_cli} {$i < (2*$num_cli)} {incr i} {
        set app($i) [new Application/TcpApp/Cliente/ClienteQoS $tcp($i)]
        $app($i) attach-rsvpagent $rsvp1 10000
        $app($i) set max_level_ $max
        $app($i) set min_level_ $minimo
        $app($i) set client_rate_ $tasa_lect
        $app($i) set size_ $tam_fich
        $app($i) set buffer_max_ [lindex $argv 0]
}
```

3) Conexión de las aplicaciones del Cliente al Servidor y petición (*request*) por parte del Cliente.

```
# Conectar aplicaciones de Clientes al servidor
#
for {set i 0} {$i < $num_cli} {incr i} {
   $app($i) connect $app([expr ($i+$num_cli)])
}
set req_time 1.0
# Las aplicaciones hacen un request
for {set i 0} {$i < $num_cli} {incr i} {
   $ns at $req_time "$app([expr ($i+$num_cli)]) send_req"
   set req_time [expr ($req_time+0.01)]
}
```

4) Acceso a las variables de las aplicaciones.

```
# Crear Bandwidth Manager
set bwm [new Servidor_BWM]
```

```
$bwm set rate_ [expr ($link_rate*0.8)]

$bwm num_client $num_cli

$bwm max_resv_clients $m
```

4) Acceso a las variables de las aplicaciones.

```
proc finish {} {

        global ns nf f0 app tipo_fuente fuente0

   .......

        #Para acceder a las variables de las aplicaciones

        set t [$app(1) set TQoS_]

        set r [$app(1) set rate_o_]

        set c [$app(1) set client_rate_]

        set f [$app(1) set size_]

        set d [$app(1) set delta_rate_d]

        set v [$app(1) set data_rec_RES_]

        set g [$app(0) set rate_]

   .......

}
```

A.4 Descripción del Entorno

Como escenario de simulación se ha elegido una topología simple, de modo que pueda evaluarse el comportamiento del sistema de forma adecuada en función de otros parámetros del entorno. Estos entornos de simulación se desarrollan en OTcl utilizando los objetos creados específicamente para formar el sistema cliente-servidor de flujos semi-elásticos.

A.4.1 Escenario de Simulación

El escenario utilizado en las simulaciones con un solo cliente (Figura A.6) consta de un total de tres enlaces y cuatro nodos.

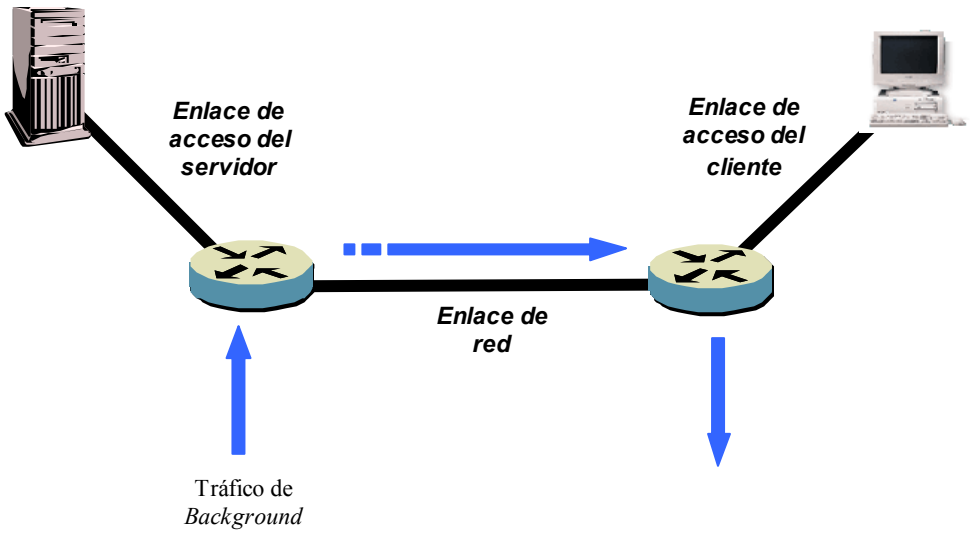

Figura A.6: Topología de red para escenarios con un solo cliente.

Representa un entorno habitual de aplicación cliente-servidor en Internet, aunque en este caso la aplicación garantizará Calidad de Servicio con un coste mínimo de la conexión, gestionando de forma eficiente los recursos disponibles. Así, un ejemplo de esta aplicación podría ser aquella en que el cliente quiere obtener del servidor una secuencia de vídeo e irla visualizando en tiempo real a través de Internet. Para ello, el cliente consta de una memoria determinada para ir almacenando la información que llega, y de un mecanismo de gestión de dicha memoria que permite ir liberando los recursos y minimizar el coste, a la vez que garantiza que la aplicación no se quede sin datos para visualizar la película.

El modelo simplificado de Internet tomado, es el que representa que una conexión TCP que atraviesa la red puede ser modelada como el enlace más restrictivo, es decir, como aquel enlace de red donde se produce el cuello de botella (*bottleneck*) de la conexión a lo largo del trayecto. Este modelo está ampliamente aceptado, y es correcto siempre que la capacidad de los *buffers* y el ancho de banda disponible en el resto de nodos atravesados por la conexión sean suficientemente grandes en comparación con los del nodo elegido como cuello de botella. Sin embargo, si esto no fuera así, deberíamos tener en cuenta otros nodos de la conexión, como se comenta en la referencia [BAR99], si bien este no es el caso que nos ocupa.

El protocolo de transporte utilizado es el más habitual en Internet, el TCP (*Tranport Control Protocol*); en cuanto a las versiones disponibles de este protocolo, podemos elegir entre varias (*Tahoe, Reno, NewReno, Sack TCP*) [FAL96], si bien la elegida es la versión implementada en ns-2 denominada *Tahoe*. Esta versión se

caracteriza por reiniciar siempre mediante *Slow Start*, tanto si detecta una pérdida como si recibe tres ACK's duplicados, lo cual supone una política un tanto conservadora[ii]. Se ha optado por usar la versión *Tahoe* del TCP ya que la versión *NewReno* daba ciertos problemas (que no fueron analizados por el grupo de investigación) con los retardos de los enlaces y provocaba resultados incoherentes de las simulaciones. En los *routers* se emplearán colas FIFO (*First In First Out*) con una política de gestión de colas *DropTail*.

Para caracterizar el tráfico existente en Internet, se inyecta tráfico de *background* entre los dos *routers*, de modo que se deterioran las prestaciones en ese enlace en función de la cantidad de tráfico de *background* que inyecte la fuente creada y en función de las características de la misma.

Para las simulaciones con varios clientes, se utiliza una topología distinta (Figura A.7). En este caso el tráfico de *background* se hace pasar por los enlaces de acceso de los clientes (homogéneos en cuanto a requerimiento de datos) para controlar la carga de red de forma individual e independiente del modo de servicio de cada conexión en el servidor, ya que en el enlace de red habría interferencia entre los distintos flujos de datos en modo *best-effort*. Cabe recordar, que la simulación propuesta en el Capítulo 5, es una primera aproximación al problema de servir a varios clientes, y queda pendiente el estudiar casos más reales (menos homogéneos).

[ii] La tasa de llegada de paquetes a la memoria del cliente es algo menor con esta versión de TCP, por lo que se estudia el peor caso.

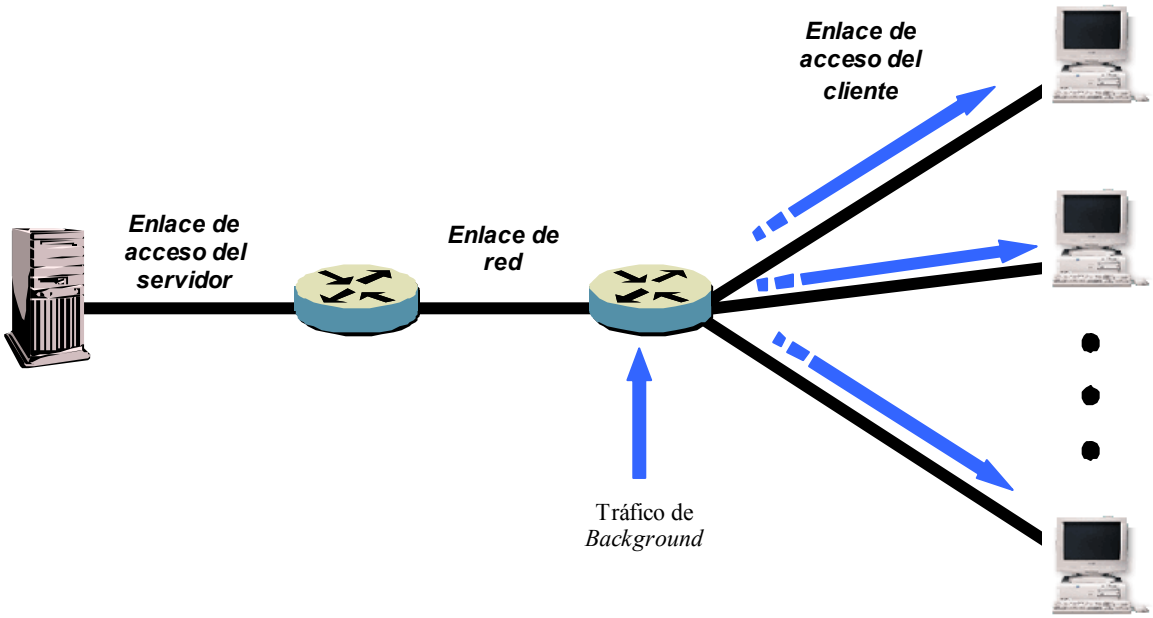

Figura A.7: Topología de red para escenarios con múltiples clientes.

Parámetros de la topología de red para escenarios con un cliente

Los principales parámetros del entorno se comentan a continuación:

- **Datos generales:**
  - Nodos: 4 (n0, n1, n2 y n3)
  - Enlaces: 3
  - *Routers*: 2 (n2 y n3)
  - Clientes: 1 (n1)
  - Servidores: 1 (n0)
  - Protocolo de Transporte: TCP (*Tahoe*)
- **Enlaces:**
  - Ancho de Banda: se elige la capacidad necesaria para dimensionar de forma adecuada el acceso del Cliente o del Servidor y el transporte por la red según SDH/PDH (Jerarquía Digital Síncrona/Plesiócrona). La jerarquía PDH establece diversos niveles según la capacidad deseada [SDH02], obteniendo las siguientes capacidades para los diversos enlaces del entorno:
    - Acceso del Cliente a Internet: E1 → 2 *Mbit/s*
    - Acceso del Servidor a Internet: E3 → 34 *Mbit/s*

  - Modelo Internet (*Bottleneck Node*): E2 → 8 *Mbit/s*
- Retardo: 0.5 ms en los enlaces de acceso y 1 ms en el enlace troncal

- **Parámetros RSVP:**
  - Porcentaje de Ancho de banda Reservable: 90 %.
  - Ancho de Banda Reservado para mensajes RSVP: 10 % de capacidad del enlace.
  - Tamaño cola para clase *best-effort*: 50000 *bytes*
  - Algoritmo Control de Admisión: Param
  - Estimador: Null

- **Servidor: QoSRated**
  - Longitud del paquete: 1500 *bytes* (si bien variará en determinadas simulaciones)
  - Tasa del Servidor: 1.8 *Mbit/s*
  - Bucket: 100000 *bytes* (utilizado en la mayoría de simulaciones en ns-2)
  - TTL: 32
  - Blocking: 1 (para que controle la tasa)

- **Cliente: QoSVarbe**
  - File Size: 300 *Mbytes* (tamaño del fichero pedido)
  - Tasa de lectura del Cliente: 500 *Kbit/s*
  - Buffer-Max: 200 *Mbytes* (tamaño máximo de la memoria del cliente)
  - Mínimo: 1.5 x Longitud del paquete (*Min_level* en función de tamaño de paquete)
  - Max_Ini: suponiendo que el método es válido para cargas > 0.05 %, obtendremos el valor de Max_{ini} a partir de:

$$Max_{ini} = \left[FS - (1-\rho)(FS - Min)\right]\frac{\alpha_{ReR} - \alpha_r}{\alpha_{ReR} - \alpha_r(1-\rho)} \qquad \text{(A.1)}$$

- **Tráfico de *background*:**

Se utilizan diversas fuentes de tráfico de fondo, cada una de ellas con sus parámetros específicos y configurables en función de la situación que deseemos simular. En el próximo apartado se comentará de forma más extensa cada tipo de fuente y sus parámetros más relevantes, si bien a grandes rasgos podemos enumerar los siguientes:

- CBR:
    - Tamaño de Paquete: 500 *bytes*.
    - Tasa: entre 7.8 *Mbit/s* y 8.8 *Mbit/s*, en función de la carga que se desee.
    - Random: 1 (para que no salgan todos los paquetes equiespaciados se utiliza una distribución de probabilidad uniforme para la distancia entre paquetes).
- PARETO:
    - Tamaño de Paquete: 500 *bytes*
    - BurstTime: 500 ms
    - IdleTime: 500 ms
    - Tasa: entre 8.7 *Mbit/s* y 15.5 *Mbit/s* para obtener cargas de 0 a 1.
    - Shape: 200
- WEB_TRAFFIC:
    - InterPage: función de Pareto-II (avg_ 5, shape_ 2)
    - PageSize: función de Pareto-II (avg_ 0.167, shape_ 1.5)
    - ObjSize: función de Pareto-II (avg_ 2, shape_ 1.2)
    - NumPage: 300 páginas por sesión.
    - Num_traficos: entre 40 y 500 fuentes para conseguir cargas bajas o elevadas.
- VIDEO (Mpeg-4):
    - Nombre del fichero: jurassic.nsformat
    - Num_traficos: entre 6 y 16 fuentes para conseguir cargas entre 0 y 1.

Parámetros de la topología de red para escenarios con varios clientes

Los principales parámetros del entorno son idénticos a la anterior topología con las siguientes diferencias:

El ancho de banda es igual en todos los enlaces (5 Mb/s) para que en el caso de dar servicio a un solo cliente se utilice todo el ancho de banda reservable en el enlace de acceso del servidor. En un caso general esto no será así, ya que el ancho de banda de la reserva vendrá limitado por el ancho de banda del cliente (normalmente más restrictivo que el del servidor).

Se utiliza como tráfico de *background* sólo CBR, para estudiar un caso más cercano al caso teórico, y centrar el estudio en la eficiencia conseguida (cercana a la máxima) y en la señalización extra necesaria.

Anexo B

Implementación del sistema para servicio a un cliente sobre sistema operativo Linux

B.1 Introducción

Para evaluar la complejidad práctica del sistema cliente-servidor de flujos semi-elásticos, propuesto y analizado en el Capítulo 4, se plantea su implementación real sobre sistema operativo Linux, utilizando como protocolo de señalización RSVP (*resource ReSerVation Protocol*). Para ello, se ha diseñado una aplicación en la que tanto el cliente (constituido por 3 procesos) como el servidor (4 procesos) hacen uso de la interfaz RAPI (*RSVP Application Programming Interface*) para interactuar con el Demonio o *Daemon* RSVP y poder negociar con la red la reserva de recursos. En esta aplicación se crean diferentes *sockets*: Aplicación cliente/Aplicación servidor (transmisión de datos), *Daemon* cliente/*Daemon* servidor (transmisión de señalización), Aplicación cliente – *Daemon* cliente (el cliente informa al *Daemon*, a través de la RAPI, de las necesidades de reserva) y Aplicación servidor/*Daemon* servidor (a través del cuál el servidor recibe la solicitud del cliente).

Toda la programación se ha realizado en código C y, posteriormente, se ha implementado en una plataforma LINUX (SUSE 7.1). La implementación de la versión 1 del protocolo RSVP para plataforma LINUX es la distribuida por el ISI (*Information Sciences Institute*), institución líder en el desarrollo del protocolo RSVP.

B.2 Diseño del sistema Cliente-Servidor

En este apartado se presenta el diseño de un sistema cliente-servidor, en un entorno Internet con calidad de servicio extremo a extremo que proporcione un adecuado nivel de servicio para la transmisión de flujos semi-elásticos con el menor coste económico posible, en base a dotar a dicho sistema de la capacidad de negociar reservas de recursos cuando sea necesario. Para ello se ha diseñado un sistema cliente-servidor en el que se utilizan los mecanismos *Daemon* RSVP y API. En el diseño de la aplicación se han tenido cuenta las siguientes consideraciones:

- Dado que el objetivo final es la transmisión de datos de un servidor a un cliente, es necesario diseñar tanto la aplicación del cliente como la del servidor.
- Las dos aplicaciones, además de realizar su correspondiente tarea en la transmisión de datos, deben ser capaces de negociar con la red la reserva de recursos en función de las necesidades de cada momento. Para ello, se utilizan los mecanismos *Daemon* RSVP y API.
- Para la minimización del coste se ha implementado el método de gestión de memoria por parte del cliente descrito en el Capítulo 4.

Antes de entrar en detalle en el desarrollo de la aplicación diseñada en este trabajo, se explicará la implementación del *Daemon* RSVP y la API con la cual la aplicación podrá comunicarse para solicitar o anular la reserva de recursos según convenga en cada momento.

B.2.1 Distribución RSVP ISI

La implementación de RSVP utilizada ha sido la proporcionada por el ISI (*Information Sciences Institute*). Ésta es una distribución muy extendida dentro del ámbito RSVP y, en muchos casos, es considerada como sistema de referencia. La página Web de esta Institución dedicada a RSVP es *http://www.isi.esu/rsvp* y las diferentes versiones de implementación se pueden encontrar en la dirección *ftp://ftp.isi.edu/rsvp*.

En este trabajo, se ha utilizado la versión rel4.2a4-1, preparada para trabajar bajo LINUX.

Las características más importantes de esta versión son:

- Realización en C.
- Interoperabilidad con la implementación KOM y con la de Cisco.
- Funcionamiento bajo LINUX, FreeBSD, SunOS, Solaris e IRIX.
- Posibilidad de funcionar como *router* o como *host*.
- Posibilidad de utilizar el *Daemon* UNIX a través de una interfaz de programación (API) o a través de línea de comandos. Con la segunda opción, el usuario puede decidir qué mensajes enviar, los parámetros a utilizar, etc.
- Soporte unicast y multicast.
- Soporte de encapsulado sobre UDP.
- Soporte de IPv6.
- Soporte de extensiones IPSEC a través de los algoritmos MD5 (*Message-Digest Algorithm*) y HMAC (*Keyed-Hashing for Message Authentification*).
- Soporte de *multihoming* (equipos con varias interfaces de red).

B.2.2 RAPI (RSVP Application Programming Interface)

En esta sección se describe la versión 5 de la RAPI, una interfaz de programación de RSVP basada en una librería cliente que se comunica con la aplicación y cuyas llamadas se describen en los siguientes apartados.

En la Figura B.1 se muestra un breve esquema del modelo de implementación de la RAPI, término con el que nos referimos a la interfaz entre la aplicación y la librería cliente RSVP. Las funciones del módulo librería cliente RSVP se comunican con el proceso local RSVP a través de *sockets*.

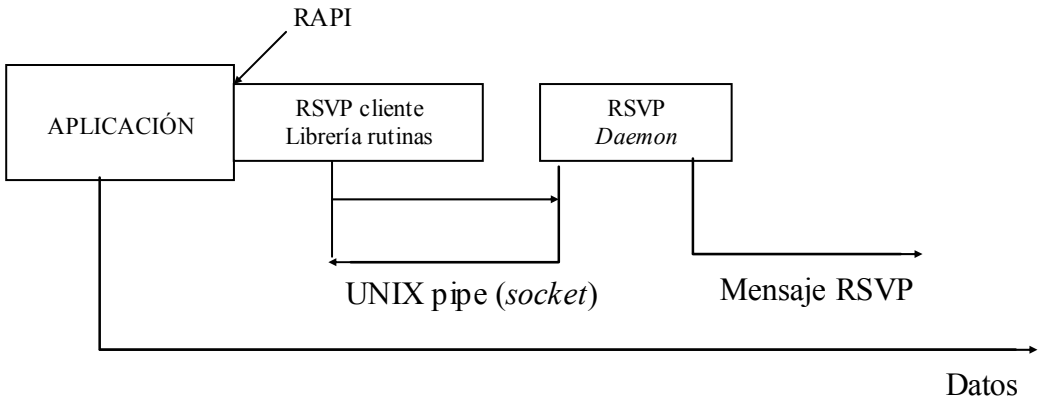

Figura B.1: Esquema de una aplicación con RSVP.

Modelo de reserva

El protocolo RSVP realiza la señalización necesaria para hacer una reserva de recursos sobre un flujo de datos y de forma unidireccional hacia una dirección destino *unicast* o *multicast*. Aunque RSVP distingue los emisores de los receptores, una aplicación puede actuar tanto de emisor como de receptor simultáneamente.

RSVP asigna QoS a un flujo específico de datos multipunto-multipunto, en lo que se conoce como sesión, y se define por un protocolo de transporte particular (una dirección IP destino y un puerto destino).

Una fuente de datos o emisor se define por su dirección IP y puerto. Una sesión puede tener varios emisores (*S1*, *S2*,..., *Sn*) y, si el destino es una dirección *multicast*, también varios receptores (*R1*, *R2*,..., *Rn*). Con RSVP, las peticiones de QoS son realizadas por los receptores.

Resumen de la API

Utilizando la interfaz RAPI, una aplicación hace la llamada *rapi_session* () para definir una "sesión API" que permitirá enviar un único flujo de datos "simplex" y/o recibir un flujo de datos. La llamada *rapi_sender*() se utiliza para registrar los datos del emisor y la llamada *rapi_reserve*(), para solicitar una reserva de QoS sobre los datos que recibe del receptor. Ambas llamadas se pueden repetir con diferentes parámetros, lo que permitirá poder modificar dinámicamente el estado de la sesión en cualquier momento o bien pueden ser tratadas mediante llamadas nulas para borrar el estado correspondiente. La llamada *rapi_release*() cierra la sesión, eliminando todos los recursos reservados. En la Figura B.2 se representa un diagrama de estados con las llamadas correspondientes. En esta figura las llamadas *rapi_sender*(0) y *rapi_reserve*(0) representan llamadas nulas.

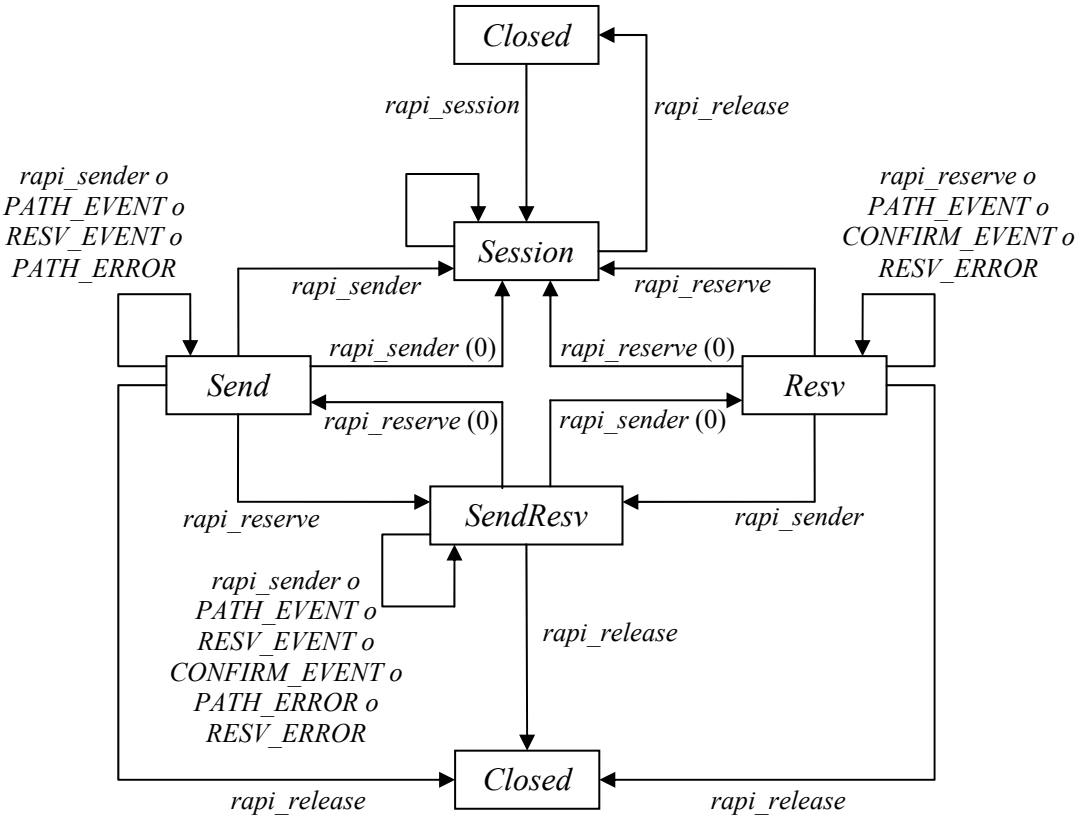

Figura B.2: Diagrama de estados de la RAPI.

Cabe destacar que una sesión API, definida por una única llamada *rapi_session*, únicamente permite definir un emisor aunque una misma sesión RSVP puede establecer varias sesiones API de forma simultánea. Por ejemplo, supongamos una aplicación que envía múltiples flujos de datos UDP, definidos por el puerto origen. En este caso, se deberán realizar las llamadas *rapi_session*() y *rapi_sender*() para cada uno de los flujos.

La llamada *rapi_session* permite a la aplicación especificar una rutina asíncrona (llamada *upcall*) que será la que reciba y trate los eventos relacionados con el cambio de estado y los errores RSVP. Hay cinco posibles eventos que hacen que se ejecute la rutina *upcall*:

- **RAPI_PATH_EVENT**

Este evento indica que el *path state* de un emisor RSVP ha cambiado o bien que ha llegado uno nuevo al nodo local. El parámetro que se le pasa a la llamada *upcall* es una lista de todos los emisores que tienen esta sesión. Este evento lo activa la llamada *rapi_session*().

- **RAPI_RESV_EVENT**

Este evento indica que el estado de una reserva ha cambiado o bien que ha llegado una nueva reserva al nodo local, informando (pero no asegurando) que las reservas han estado establecidas o borradas a lo largo del todo el camino hasta uno o más receptores. Este evento lo activa la llamada *rapi_sender*().

- **RAPI_PATH_ERROR**

Este evento indica que se ha producido un error asíncrono en la información del emisor especificada en una llamada *rapi_sender*() que es quien activa este evento.

- **RAPI_RESV_ERROR**

Este evento indica que se ha producido un error asíncrono en la reserva y es activado por la llamada *rapi_reserve*().

- **RAPI_RESV_CONFIRM**

Este evento indica que se ha realizado una reserva hasta, como mínimo, un punto intermedio y un emisor siendo probable, aunque no necesario, que se haya realizado en todo el camino. Este evento lo activa la llamada *rapi_reserve*().

La rutina *upcall* es invocada por la aplicación de forma indirecta y síncrona, mediante el siguiente mecanismo:

La aplicación ejecuta la llamada de la librería RAPI *rapi_getfd*() para conocer el *file descriptor* del *socket* LINUX que utiliza la API.

La aplicación detecta eventos de lectura en este *file descriptor*, utilizando directamente una llamada *select*.

Cuando hay un evento de lectura en el *file descriptor*, la aplicación llama a la rutina *rapi_dispatch*(). Y esto hace que la API ejecute la rutina *upcall* apropiada.

Un error síncrono en una rutina de la librería RAPI devuelve el código de error apropiado. Los errores RSVP asíncronos son entregados a la aplicación a través de la rutina *upcall*.

La primera llamada *rapi_session*() abre un dominio *socket* de LINUX hacia el proceso *Daemon* RSVP asíncrono y pasa la información de la sesión del *socket*. Si la aplicación (o el *Daemon*) aborta sin haber cerrado correctamente el *socket*, en el otro extremo debería producirse la notificación correspondiente. Concretamente, si el

proceso de usuario acaba sin cerrar explícitamente la sesión RAPI, el *Daemon* RSVP borrará los correspondientes estados de reserva que puedan quedar en los *routers*.

Crear una sesión

La llamada *rapi_session*() crea una sesión API. Esta llamada devuelve un identificador de sesión diferente de cero que se utiliza en las siguientes llamadas que se producen en dicha sesión API. Si la llamada falla síncronamente, devuelve cero (*NULL_SID*) y guarda el código de error en una variable entera que apunta al parámetro *errnop*.

Tras efectuar la llamada *rapi_session*(), la aplicación puede recibir *upcalls* de tipo *RAPI_PATH_EVENT*. El formato de la llamada es el siguiente:

```
unsigned int rapi_session (
   struct sockaddr          *Dest,
   int                      Protid,
   int                      flags,
   int                      (*Event_rtn)(),
   void                     *Event_arg,
   int                      *errnop
)
```

Los parámetros son los siguientes:

- Dest: es un puntero a una estructura *sockaddr* donde está la dirección destino IP (v4 ó v6) y un número de puerto al que serán enviados los datos.
- Protid: es el identificador del protocolo IP para la sesión. Por defecto, toma el valor 17 (UDP). Este parámetro junto con el anterior definen una sesión RSVP.
- `flags`: parámetro que puede tomar el valor de *RAPI_GPI_SESSION* (0x40) o bien el valor de *RAPI_USE_INTSERV* (0x10). En el primer caso, indica que la sesión API sea definida en el formato GPI. En el segundo caso, son los formatos *IntServ* los que se utilizan en los *upcalls*.
- `Event_rtn`: es un puntero a una rutina *upcall*.

- `Event_arg`: es un parámetro opcional, un puntero a un argumento que se puede pasar en alguna invocación de la rutina *upcall*.
- `errnop`: la dirección de un entero en el cual se almacena un código de error RAPI.

Una aplicación puede tener de forma simultánea múltiples sesiones API para la misma o para diferentes sesiones RSVP. Tal y como se ha mencionado anteriormente, aunque sólo un emisor puede estar asociado a una sesión API, una aplicación puede tener múltiples emisores para una sesión RSVP creando una sesión API para cada uno de ellos.

Registro como emisor

Una aplicación puede hacer la llamada *rapi_sender* si tiene la intención de enviar un flujo de datos en el cual los receptores puedan hacer reservas. Esta llamada define, redefine, o borra los parámetros de flujo correspondientes. Una llamada *rapi_sender* puede ser ejecutada más de una vez por la misma sesión API pero la más reciente es la que la que tiene preferencia.

Si se produce un error síncrono, la llamada *rapi_sender* devuelve un código RAPI de error y en cualquier otro caso devuelve cero. Si la llamada ha tenido éxito, la aplicación puede recibir *upcalls* de tipo *RAPI_RESV_EVENT* o *RAPI_PATH_ERROR*.

El formato de la llamada es el siguiente:

```
int rapi_sender(
   int                     Sid,
   int                     flags,
   struct sockaddr         *Lhost,
   rapi_filter_t           *SenderTemplate,
   rapi_tspec_t            *SenderTspec,
   rapi_adspec_t           *SenderAdspec,
   int                     TTL ;
)
```

Los parámetros se exponen a continuación:

- `Sid`: identificador de sesión que devuelve la llamada *rapi_session*.

- `flags`: no hay *flags* definidos para esta llamada
- `Lhost`: es opcional y apunta a una estructura *sockaddr* donde se especifica la dirección IP y el puerto a través del cual el emisor emite los datos, o puede ser *NULL*. Si la dirección IP del emisor es *INADDR_ANY*, la API utilizará la dirección IP (v4 ó v6) por defecto del *host* local.
- `SenderTemplate`: parámetro opcional que es un puntero a una estructura RAPI *filter spec*, que especifica el formato de los paquetes de datos que serán enviados, pudiendo ser *NULL*.
- `SenderTspec`: puntero a un *Tspec* que define el tránsito que el emisor generará.
- `SenderAdspec`: parámetro opcional que es un puntero a una estructura RAPI *Adspec*, pudiendo ser *NULL*.
- `SenderPolicy`: parámetro opcional que es un puntero a una estructura *sender policy*, o también puede ser *NULL*.
- `TTL`: parámetro que especifica el valor TTL (*Time-to Live*) IP cuando se envían datos. En esta llamada, el TTL permite a RSVP enviar sus mensajes de control con el mismo alcance TTL que los paquetes de datos.

Petición de reserva de recursos

La función *rapi_reserve* es llamada para crear, modificar o borrar la reserva. La llamada puede ser repetida con diferentes parámetros, permitiendo a la aplicación modificar o borrar la reserva. Dependiendo de los parámetros, cada llamada puede o no pasar el control de admisión, el cual puede fallar asíncronamente.

Si hay un error síncrono en la llamada, *rapi_reserve* devuelve un código RAPI de error, devolviendo cero en cualquier otro caso. Una vez se ha ejecutado esta llamada, es posible recibir *upcalls* del tipo *RAPI_RESV_ERROR* o *RAPI_RESV_CONFIRM*.

Si el control de admisión falla (por ejemplo, se rechaza una petición de QoS), se informará asíncronamente con un evento del tipo *RAPI_RESV_ERROR*.

El formato de la llamada es el siguiente:

```
int rapi_reserve (
   int                      Sid,
   int                      flags,
   struct sockaddr          *Rhost,
   int                      Styleld,
   rapi_stylex_t            *Style_Ext,
   rapi_policy_t            *Rcvr_Policy,
   rapi_filter_t            *FilterSpec_list,
   int                      FlowspecNo,
   rapi_flowspec_t          *FlowSpec_list
)
```

Los parámetros de la llamada son los siguientes:

- `Sid`: identificador de la sesión que devuelve la llamada *rapi_session*.
- `flags`: si se le da el valor *RAPI_REQ_CONFIRM,* se puede pedir la confirmación de la reserva recibiendo eventos de confirmación (*RAPI_RESV_CONFIRM*).
- `Rhost`: parámetro opcional que puede ser utilizado para definir la dirección de la interfaz a través de la cual se recibirán los datos.
- `Styleld`: especifica el identificador del estilo de reserva.
- `Style_Ext`: parámetro opcional que es un puntero a una extensión *style_dependent* del parámetro *list*.
- `Rcvr_Policy`: parámetro opcional que es un puntero a una estructura de datos *policy*, o puede ser NULL.
- `FilterSpec_list, FilterSpecNo`: el primer parámetro es un puntero a un área que contiene un vector secuencial de objetos RAPI *filter spec*. El número de objetos en este vector se especifica en *FilterSpecNo*. Si este parámetro es cero, el valor de *FilterSpec_list* es ignorado.
- `Flowspec_list, FlowspecNo`: el primer parámetro es un puntero a un área que contiene un vector secuencial de objetos RAPI *flowspec*. El

número de objetos en este vector se especifica en *FlowspecNo*. Si este parámetro es cero, el valor de *Flowspec_list* es ignorado.

Si el valor del parámetro *FlowspecNo* es cero, la llamada *rapi_reserve* borra la reserva que está en curso en una determinada sesión y los parámetros *FilterSpec_list* y *Flowspec_list* son ignorados. En el caso de que no sea así, los parámetros dependen del estilo de la reserva pudiendo ser:

- `Wildcard Filter (WF)`: se utiliza *Styleld* = *RAPI_RSTYLE_WILDCARD*. El parámetro *Flowspec_list* puede estar vacío (para borrar la reserva) o puede apuntar a un único *filter spec* que contenga el apropiado *wildcard(s).*
- `Fixed Filter (FF)`: se utiliza *Styleld* = *RAPI_RSTYLE_FIXED*. Los parámetros *FilterSpecNo* y *FlowspecNo* han de ser iguales.
- `Shared Eplicit (SE)`: se utiliza *Styleld* = *RAPI_RSTYLE_SE*. El parámetro *Flowspec_list* debe apuntar a un único *flowspec*. El parámetro *FilterSpec_list* ha de apuntar a una lista de una cierta longitud.

Otras llamadas RAPI

Para liberar una sesión API, se utiliza la llamada *rapi_release*, pasando como parámetro el identificador de la sesión correspondiente. Esta llamada puede hacerse implícitamente si la aplicación acaba sin tratar sus sesiones RSVP. Si el identificador de la sesión no es válido, la llamada devuelve un código de RAPI de error, o cero en cualquier otro caso.

Otra llamada interesante es la *rapi_getfd*, que se utiliza para obtener el *file descriptor* asociado a un *socket* con el que se establece la comunicación con el *Daemon* RSVP. Se puede utilizar después de la sesión *rapi_session* y antes de la llamada *rapi_release*. Esta llamada tiene como parámetro un identificador de sesión. Si este identificador es ilegal o no está definido, la llamada devuelve –1; en el caso de que la llamada tenga éxito, devuelve el *file descriptor*.

Cuando se produce un evento de lectura en el *file descriptor* que devuelve *rapi_getfd*, la aplicación puede llamar a la rutina *rapi_dispatch*. Esta llamada puede ser ejecutada en cualquier momento, pero no tendrá ningún efecto a no ser que haya algún evento esperando en el *socket*. Puede provocar más de un *upcall*. No se le

pasan parámetros y devuelve un código RAPI de error, en el caso de que éste se produzca, o un valor cero, en el caso de que tenga éxito.

B.2.3 Diseño y arquitectura del *software*

El desarrollo de la presente Tesis, se ha basado en el diseño e implementación de una aplicación cliente-servidor que permitiera al cliente gestionar de forma dinámica la reserva de recursos, mediante el conocimiento de la cantidad de información almacenada en su memoria, con el fin de minimizar el coste de la transmisión de un flujo semi-elástico manteniendo la QoS deseada.

Como se mencionó anteriormente, en el sistema diseñado tanto el cliente como el servidor hacen uso de la RAPI para interactuar con el *Daemon* RSVP y poder negociar con la red la reserva de recursos más óptima, asumiendo que el tipo de datos a transmitir son del tipo semi-elástico. Toda la programación se realizó en código C y, posteriormente, se implementó en una plataforma LINUX (SUSE 7.1).

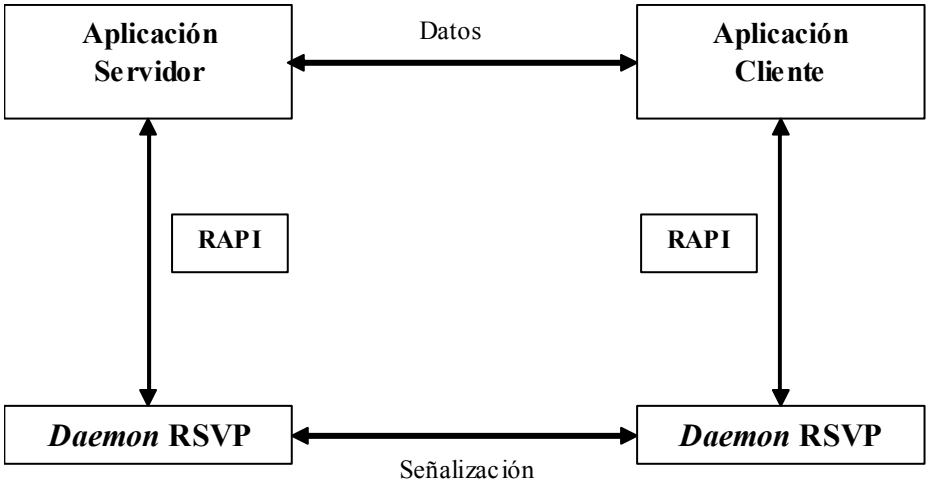

Figura B.3: Arquitectura cliente-servidor.

Tal y como se esquematiza en la Figura B.3, en la aplicación se crean diferentes *sockets*:

- Aplicación cliente ↔ Aplicación servidor, a través del cuál se transmitirán los datos.
- *Daemon* cliente ↔ *Daemon* servidor, a través del cuál se transmitirá la información de señalización.

- Aplicación cliente ↔ *Daemon* cliente, a través del cuál el cliente informa al *Daemon* (a través de la RAPI) de las necesidades de reserva de recursos en cada momento.
- Aplicación servidor ↔ *Daemon* servidor, a través del cuál el servidor recibe la solicitud del cliente.

Diseño del Servidor

En líneas generales, en nuestra aplicación, cuando el cliente hace una petición al servidor (*S_inicial*), éste genera dos procesos: uno destinado a transmitir la información de los datos (*S_transmisor*) y otro destinado a analizar el estado de la red para poder determinar la velocidad de transmisión adecuada (*S_analizador*).

Una vez hecha la petición, el *S_transmisor* empieza la transmisión en el modo reserva de recursos y, de forma simultánea, el *S_analizador* se pone en alerta para recibir la información del *Daemon* RSVP sobre el estado de la red. En función de dicha información, y a través del fichero "parámetros servidor", informa al *S_transmisor* a qué tasa debe transmitir. Esto se ilustra de forma esquemática en la Figura B.4.

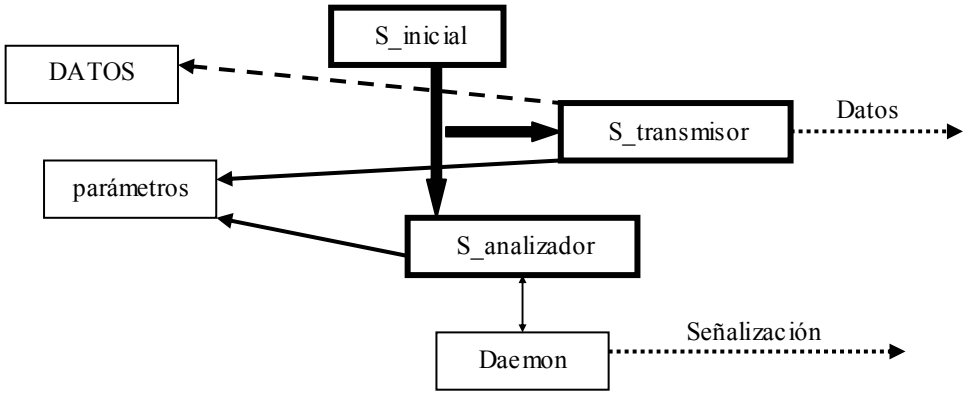

Figura B.4: Arquitectura del Servidor.

A continuación se describe con más detalle cada uno de los procesos que intervienen en el Servidor.

- **Proceso *S_inicial***

Inicialmente, es el único proceso que se mantiene activo (en la máquina servidor) permaneciendo a la escucha del canal en espera de que llegue la petición del cliente. Una vez que ésta llega, tal y como se ha comentado anteriormente, genera dos procesos y queda en espera de que éstos acaben su función.

El pseudo-código de este proceso es el siguiente:

```
Inicio

   mientras   no llegue la petición
      del cliente

         esperar

   fin_mientras

   crear S_transmisor

   crear S_analizador

   Si soy S_transmisor

         código S_transmisor

   fin_si

   Si soy S_analizador

         código S_analizador

   fin_si

   esperar a que acabe S_transmisor

   esperar a que acabe S_analizador

fin
```

Figura B.5: Pseudo-código S_inicial.

- **Proceso *S_transmisor***

Su función consiste en transmitir la información almacenada localmente a una determinada velocidad binaria que dependerá de las instrucciones generadas por el proceso *S_analizador* y que están almacenadas en el fichero "parámetros servidor".

Para el control de esta tasa, hace uso de un *SIGALRM* que le hace atender, a intervalos regulares (1 segundo), una subrutina que cambia el valor de la variable (control de tasa) de "1" a "0" por lo que este proceso (*S_transmisor*) queda desbloqueado, es decir, una vez que haya transcurrido el tiempo restante del segundo se podrá continuar transmitiendo.

A continuación se especifica el pseudo-código de este proceso.

```
Inicio
   mientras no se acabe el fichero
         leer del fichero
         escribir en el socket
         incrementrar contador
         Si contador = velocidad_transmisión [ej:1024]
               mientras control_velocidad=1
                        esperar
               fin_mientras
               control_velocidad=1
         fin_si
   fin_mientras
Fin
```

Figura B.6: Pseudo-código S_transmisor.

- **Proceso *S_analizador***

Es el encargado de atender al *Daemon* RSVP para informar del modo de transmisión: *best-effort* o con reserva de recursos. En base a dicha información, modificará la variable *velocidad_transmisión* (si procede, es decir, si el cliente solicita un cambio de *best-effort* a reserva por haber alcanzado su memoria el umbral mínimo o viceversa, cuando el umbral alcanzado sea el máximo) que se halla en el fichero "parámetros servidor". Fichero que el *S_transmisor* consultará para determinar en cada momento a la velocidad que se debe transmitir.

El pseudo-código de este proceso es el siguiente:

```
Inicio
   mientras (1)
      Si hay cambio de en el modo de transmisión
        escribir en el fichero "parámetros servidor" nueva_velocidad
      fin_Si
   fin_mientras
Fin
```

Figura B.7: Pseudo-código S_analizador.

Diseño del Cliente

En la aplicación cliente operan dos procesos: *C_socket* y *C_fifo*.

El *C_socket* es el encargado de leer la información del *socket* por el que se está transmitiendo un determinado fichero y de enviarlo a una FIFO, fichero en el que el *C_fifo* hallará la información que debe entregar a la aplicación.

La velocidad de lectura es un parámetro fijo, establecido de forma previa, en base a las necesidades de la aplicación utilizada. En la siguiente figura se esquematiza la estructura de esta aplicación.

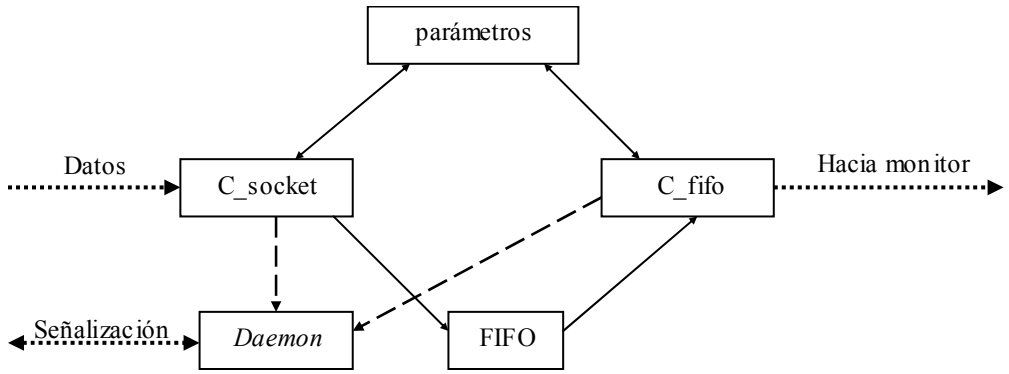

Figura B.8: Arquitectura del Cliente.

En la aplicación cliente, a diferencia de lo que ocurría en la aplicación servidor, los dos procesos interactúan con el fichero "parámetros cliente" tanto en modo escritura como en modo lectura. Por ello, es necesario implementar un mecanismo de exclusión mutua. En nuestro caso, hemos utilizado un "semáforo".

A continuación se describen con más detalle las variables más importantes que intervienen en las funciones que utilizan los dos procesos:

- *final_fifo*, indica el estado de lectura del proceso *C_fifo* (1 = ha terminado de leer la información de la FIFO, 0 = está en proceso de lectura).
- *final_socket*, indica el estado de lectura del proceso *C_socket* (1 = ha terminado de leer la información del *socket*, 0 = está en proceso de lectura).
- *reserva*, indica el modo de transmisión (1 = reserva de recursos, 0 = best-*effort*).
- *cont*, indica el tamaño de memoria utilizado por la aplicación.
- *max,* indica el máximo umbral de memoria que puede utilizar la aplicación
- t_0, t_1 y t_2, que son los tiempos utilizados para el cálculo del máximo de memoria (ver fig. 12, pag. 46).
- *tam*_t_0, *tam*_t_1 y *tam*_t_2, que son los tamaños de memoria correspondientes a t_0, t_1 y t_2.
- *calcular,* indica la necesidad de volver a realizar el cálculo de *max* (1 = necesidad de calcular, 0 = no necesidad).
- *est_lec,* utilizada para estimar la velocidad de lectura.
- *est_be,* utilizada para estimar la velocidad de transmisión en modo *best-effort*.
- *est_re,* utilizada para estimar la velocidad de transmisión en modo reserva de recursos.

▪ ***C_socket***

Este proceso continuamente está leyendo la información del *socket*. Por cada *byte* que lee, cuando el semáforo se lo permita, realiza las siguientes operaciones: escribe el *byte* en la FIFO, incrementa la variable *cont* y comprueba si esta variable ha alcanzado el umbral máximo. Si se ha alcanzado dicho umbral y la variable *reserva* tiene el valor 1 (modo reserva de recursos), llama a la función *rapi_release* para informar al *Daemon* de que debe liberar la reserva y pasar al modo *best-effort* y toma los valores de t_1 y *tam*_t_1.

Una vez ha finalizado la lectura, cambia el valor de la variable *final_socket* para comunicárselo al proceso *C_fifo*.

El pseudo-código es el siguiente:

```
Inicio
   Realiza la peticion al servidor
   Crear C_fifo
   rapi_session (abre una sesión RSVP)
   rapi_reserve (solicita la reserva)
   mientras hay información en el socket
      Leer un byte
      Escribir en la FIFO
      Entrar en el semáforo
      Incrementar cont
      Salir del semáforo
      Si (cont > max) and (reserve = 1)
         rapi_release (quitar reserva)
         tomar t1 y tam_t1
      fin_Si
   fin_mientras
   final_socket=1
   esperar a que acabe C_fifo
Fin
```

Figura B.9: Pseudo-código C_socket.

- ***C_fifo***

Lo primero que realiza este proceso es tomar el valor t_0 e iniciar la lectura de la información de la FIFO, en el caso de que el valor de *cont* le indique que hay información pendiente de leer. En el caso de que no hubiera información pendiente, mira la variable *final_socket* para comprobar que la transmisión del fichero ha finalizado y en el caso de que sea así, mediante la modificación del valor de la variable *final_fifo*, acaba el proceso.

Por cada *byte* que lee, cuando el semáforo se lo permita, realiza las siguientes operaciones: envía la información a la salida estándar (monitor), decrementa la variable *cont* y comprueba si esta variable ha alcanzado el umbral mínimo. Si se ha alcanzado dicho umbral, si la variable *reserva* tiene el valor 0 (modo *best-effort*) y además la variable *calcular* tiene el valor 1 (hay que calcular *max*), deberá indicar al *Daemon*, mediante la función *rapi_reserve*, la necesidad de reservar recursos. A continuación, llama a las siguientes funciones:

- *estimar_re* (), estima el valor de la velocidad de transmisión en modo reserva del siguiente modo: calcula la tasa α_i que es igual a la diferencia en valor absoluto de t_0-t_1 dividido por la diferencia en valor absoluto del *tam*_t_0 – *tam*_t_1 y a este valor de α_i le suma la velocidad de lectura.
- *estimar_be* (), estima el valor de la velocidad de transmisión en modo *best-effort* del siguiente modo: calcula la tasa α_o que es igual a la diferencia en valor absoluto de t_2-t_1 dividido por la diferencia en valor absoluto del *tam*_t_2 – *tam*_t_1 y a este valor de α_o le resta la velocidad de lectura.
- *calcular_máximo* (*est_re*, *est_be*), calcula el valor del umbral máximo de memoria mediante el procedimiento descrito en el Capítulo 4.

Acaba asignándole a t_0 el valor que correspondía, en ese momento, a t_2 y a *tam*_t_0 el valor que correspondía a *tam*_t_2, por si fuera necesario volver a calcular el umbral de memoria (*max*).

En relación al control de la velocidad de lectura, utiliza el mismo algoritmo explicado para la aplicación servidor.

El pseudo-código es el siguiente:

```
Inicio
   Tomar valor de t0
   mientras final_fifo = 0
      Si cont > 0
         Leer un byte
         Enviar byte a pantalla
         Entrar en el semáforo
         Decrementar cont
         Salir del semáforo
         Si (cont < min) and (reserve = 0) and (calcular = 1)
                rapi_reserve (solicitar reserva)
                tomar t2 y tam_t2
                estimar_be
                estimar_re
                calcular_máximo
                t0 = t2; tam_t0 = tam_t2
         fin_Si
      fin_Si
      Si cont < 0
         Si final_socket = 1
                final_fifo = 1
         fin_Si
      fin_Si
   fin_mientras
Fin
```

Figura B.10: Pseudo-código C_fifo.

En la implementación de este *software* han acontecido dos problemas principales: la sincronización de los múltiples procesos implicados en la aplicación y la necesidad de realizar unas pequeñas modificaciones en el código RSVP, dada su gran dimensión, para su acoplamiento a la aplicación.

B.3 Implementación del sistema Cliente-Servidor

La aplicación ha sido implementada en tres PCs. En uno de ellos, se instaló la aplicación cliente; en otro, la aplicación servidor; y a un tercero, situado entre ambos, se le asignó la función de *router*.

B.3.1 Topología

En la Figura B.11, se representa la topología de los PCs en los que se implementó la aplicación cliente-servidor.

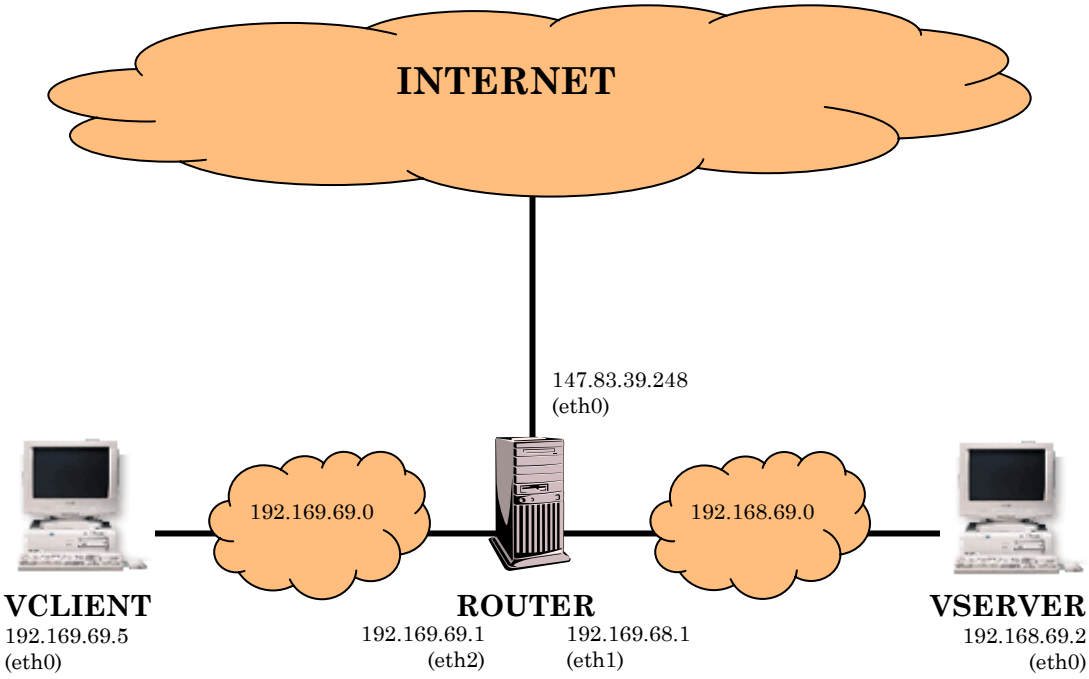

Figura B.11: Topología de la red.

B.3.2 Políticas de colas en RSVP

El *Daemon* es el encargado de gestionar el tráfico. Para ello, se generó un fichero *script* de política de colas que actúa de la siguiente forma:

Inicialmente, crea una clase de servicio con un ancho de banda de 10 Mb/s para asociarla al máximo de ancho de banda de la tarjeta *ethernet* utilizada. A continuación, con el fin de no ocupar todo el ancho de banda existente en la tarjeta de

red y provocar errores[i], crea otra subclase de servicio de 5 Mb/s que asocia a 2 nuevas clases: una de 4 Mb/s dedicada al tráfico realizado en modo reserva de recursos y otra de 1 Mb/s para el tráfico en modo *best-effort*.

El tráfico entrante pasa por los clasificadores RSVP IPv4 e IPv6 que examinarán los diferentes paquetes con el fin de determinar qué nivel de servicio requieren y, en consecuencia, dirigirlos al modo de transmisión adecuado. En el ancho de banda dedicado a la transmisión en modo de reserva de recursos, se crearán tantas subclases como sesiones haya abiertas. Adicionalmente, también se crea una subclase por la que, si procede, se enviará aquel tráfico que excede el ancho de banda contratado en cada una de las diferentes sesiones. En el caso de que las sesiones con reserva de recursos no ocupen los 4 Mb/s asignados a este tipo de transmisión, el modo *best-effort* aprovechará el ancho de banda no utilizado, mientras que sea posible, para mejorar la calidad de servicio. En la siguiente figura se esquematiza el funcionamiento de esta política de gestión de colas.

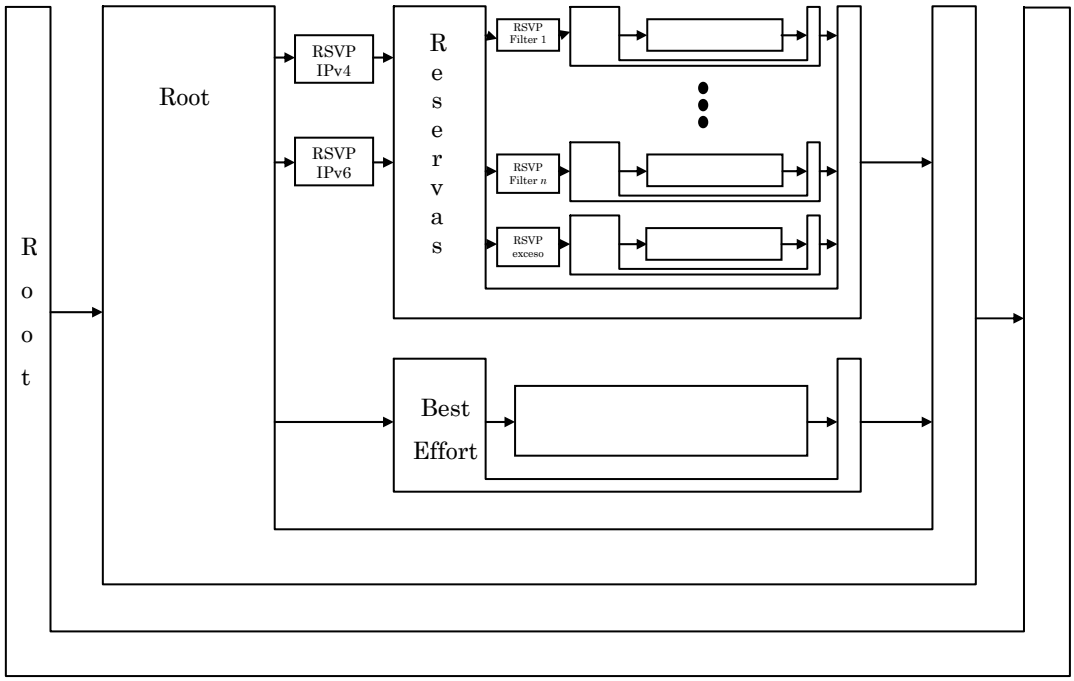

Figura B.12: Estructura de la política de colas.

[i] Si se utiliza todo el ancho de banda, se aprecien pérdidas aún cuando no debieran haberlas por estar reservados los recursos. Esto se debe a la implementación del sistema de gestión actual en el sistema operativo Linux.

A continuación se presenta el fichero que utiliza el *Daemon* para llevar a cabo la política de colas.

```
#! /bin/sh[ii]
TC=/usr/sbin/tc
IP=/usr/sbin/ip
DEVICE=eth1
BANDWIDTH="bandwidth 10Mbit"
# Poner CBQ en $DEVICE. Tendrá handle 1:.
#   $BANDWIDTH es el ancho de banda real de $DEVICE (10Mbit).
#   avpkt es el tamaño medio de paquete.
#   mpu es el tamaño mínimo de paquete.
$TC qdisc add dev $DEVICE root handle 1:  cbq \
$BANDWIDTH avpkt 1000 mpu 64
# Crear una clase root con classid 1:1.
#   BANDWIDTH es el mismo que el de la CBQ.
#   rate == todo el ancho de banda
#   allot es el tamaño de las cabeceras MTU + MAC
#   maxburst mide el bustiness de la clase permitido
#   est 1sec 8sec significa que el kernel evaluará la tasa media
#   en esta clase con periodo 1s y tiempo constane de 8s.
#   Esta tasa puede consultarse con "tc -s class ls dev $DEVICE"
$TC class add dev $DEVICE parent 1:0 classid :1 est 1sec 8sec cbq \
$BANDWIDTH rate 5Mbit allot 1514 maxburst 50 avpkt 1000
# Bulk.
#    Los nuevos parámetros son:
#    weight, debe ser proporcional a
#    "rate". No es necesario, weight=1 funcionará igual.
#    defmap y split indican que el tráfico best effort, no clasificado
#    se desviará a esta clase.
```

[ii] Indica el intérprete de comandos que utiliza el *script*.

```
$TC class add dev $DEVICE parent 1:1 classid :2 est 1sec 8sec cbq \

$BANDWIDTH rate 1Mbit allot 1514 weight 500Kbit \

prio 6 maxburst 50 avpkt 1000 split 1:0 defmap ffff

# OPCIONAL.

# Poner "sfq" qdisc en la clase, quantum es MTU, perturb

# indica el periodo de perturbación de la función de hash en segundos.

#

$TC qdisc add dev $DEVICE parent 1:2 sfq quantum 1514b perturb 15

# Clase de tiempo real for RSVP

$TC class add dev $DEVICE parent 1:1 classid 1:7FFE cbq \

rate 4Mbit $BANDWIDTH allot 1514b avpkt 1000 \

maxburst 20

# Tráfico en tiempo real reclasificado

#

#     Nuevo elemento: split no es 1:0, sino 1:7FFE. Significa,

#     que solo los paquetes en tiempo real, que han violado los filtros de
   policía

#     o han exedido los buffers de reshaping se desviarán hacia él.

$TC class add dev $DEVICE parent 1:7FFE classid 1:7FFF  est 4sec 32sec cbq
   \

rate 500Kbit $BANDWIDTH allot 1514b avpkt 1000 weight 10Kbit \

prio 6 maxburst 10 split 1:7FFE defmap ffff
```

B.3.3 Resultados

A continuación se presentan las gráficas con los resultados obtenidos cuando se aplicaron los siguientes parámetros[iii]:

- velocidad de transmisión en modo de reserva = 10240 *bytes/s* (81920 *bits/s*)
- velocidad de transmisión en modo *best-effort* = 1280 *bytes/s* (1024 *bits/s*)

[iii] Los parámetros son conservadores, ya que el sistema de gestión de colas para proporcionar la reserva no acababa de responder bien en las máquinas Linux.

- velocidad de lectura = 5120 *bytes/s* (40960 *bits/s*)
- tamaños del fichero a transmitir = 1160898 *bytes* (9287184 *bits*)

Y para tres valores distintos de *n* (véase el Capítulo 4): número de veces en que se utiliza el modo de reserva de recursos menos 1.

Resultados obtenidos para n=2

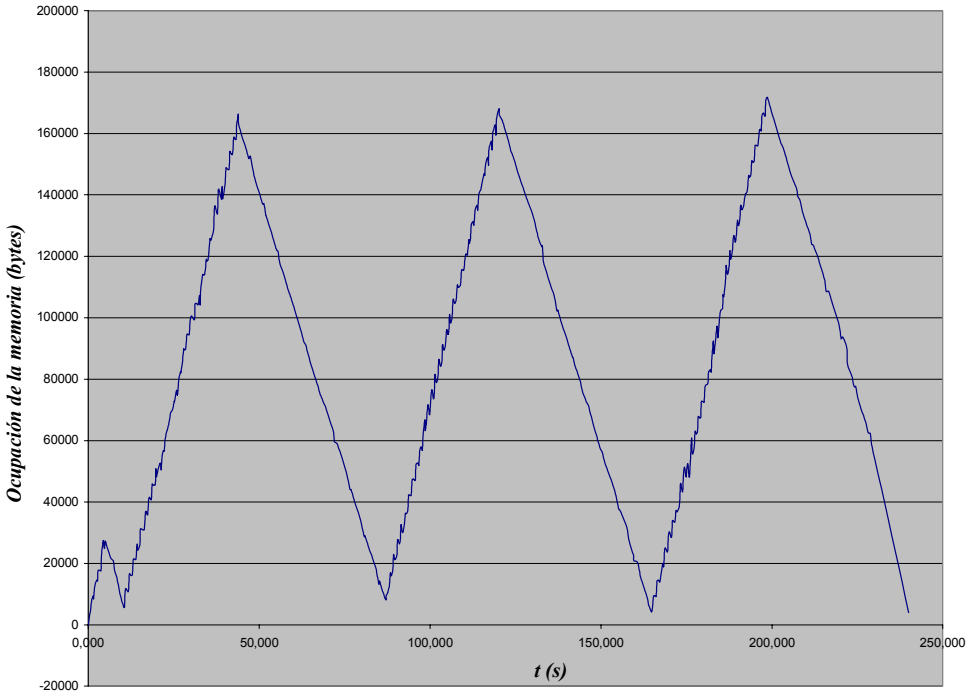

Figura B.13: Dinámica de los datos almacenados en la memoria del cliente para n=2.

Resultados obtenidos para n=3

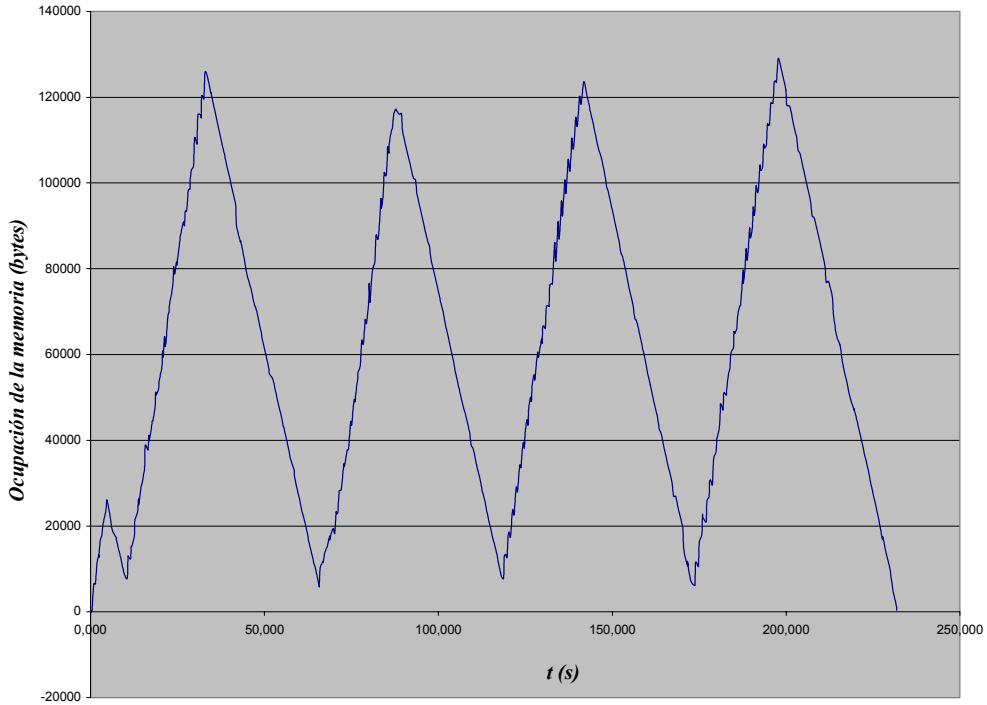

Figura B.14: Dinámica de los datos almacenados en la memoria del cliente para n=3.

Resultados obtenidos para n=4

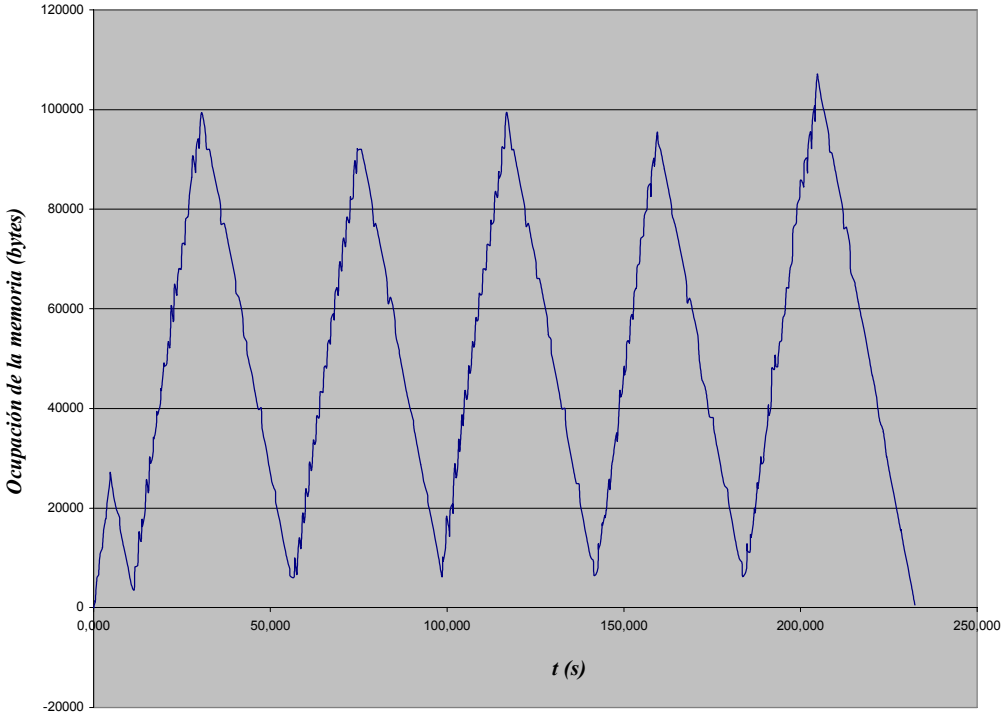

Figura B.15: Dinámica de los datos almacenados en la memoria del cliente para n=4.

Trazas obtenidas con el analizador de protocolos Ethereal[iv].

A continuación se muestran las trazas obtenidas en el escenario de pruebas de la figura 24, para el caso de $n = 0$. Las trazas correspondientes a los mensajes RSVP, se han obtenido desde el servidor (*vserver*). Se marcan los mensajes que solicitan/quitan la reserva de recursos solicitada por el cliente.

No.	Time	Source	Destination	Protocol	Info
1	0.000000	vserver.ssade	vclient.sertel	RSVP	PATH Message
2	**2.952354**	**router.ssade**	**vserver.ssade**	**RSVP**	**RESV Message**
3	**7.758991**	**router.ssade**	**vserver.ssade**	**RSVP**	**RESV TEAR Message**
4	**13.157852**	**router.ssade**	**vserver.ssade**	**RSVP**	**RESV Message**
5	30.018687	vserver.ssade	vclient.sertel	RSVP	PATH Message
6	**41.116728**	**router.ssade**	**vserver.ssade**	**RSVP**	**RESV TEAR Message**
7	70.028682	vserver.ssade	vclient.sertel	RSVP	PATH Message
8	78.387924	vserver.ssade	vclient.sertel	RSVP	PATH TEAR Message

Los valores de cada uno de los campos de los mensajes RSVP capturados de la Ethernet, se detallan a continuación.

```
Frame 1 (178 on wire, 178 captured)
    Arrival Time: Feb 10, 2002 14:16:24.3104
    Time delta from previous packet: 0.000000 seconds
    Frame Number: 1
    Packet Length: 178 bytes
    Capture Length: 178 bytes
Ethernet II
    Destination: 00:a0:24:9e:ea:66 (3Com_9e:ea:66)
    Source: 00:10:5a:af:24:cc (00:10:5a:af:24:cc)
    Type: IP (0x0800)
Internet Protocol
    Version: 4
    Header length: 24 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 164
    Identification: 0xe211
    Flags: 0x00
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64
    Protocol: RSVP (0x2e)
    Header checksum: 0xf7bc (correct)
    Source: vserver.ssade (192.168.69.2)
    Destination: vclient.sertel (192.169.69.5)
    Options: (4 bytes)
        Unknown (0x94) (4 bytes)
Resource ReserVation Protocol (RSVP)
    RSVP Header
        RSVP Version: 1
        Flags: 00
        Message Type: PATH Message (1)
        Message Checksum
        Sending TTL: 62
        Message length: 140
    SESSION: 1
        Length: 12
        Class number: 1 - SESSION object
```

[iv] Ethereal es un analizador de protocolos (sniffer) ampliamente utilizado en entorno Linux que permite obtener información acerca de las tramas Ethernet que circulan por esta red.

```
        C-type: 1 - IPv4
        Destination address: vclient.sertel (192.169.69.5)
        Protocol: TCP (6)
        Flags: 155
        Port number: 22000
    HOP: 3
        Length: 12
        Class number: 3 - HOP object
        C-type: 1 - IPv4
        Neighbor address: 192.168.69.2
        Logical interface: 0
    TIME VALUES: 5
        Length: 8
        Class number: 5 - TIME VALUES object
        C-type: 1
        Refresh interval: 30000 ms (30 seconds)
    SENDER TEMPLATE: 11
        Length: 12
        Class number: 11 - SENDER TEMPLATE object
        C-type: 1 - IPv4
        Sender IPv4 address: vserver.ssade (192.168.69.2)
        Sender port number: 20000
    SENDER TSPEC: 12
        Length: 36
        Class number: 12 - SENDER TSPEC object
        Message format version: 0
        Data length: 7 words, not including header
        Service header: 1 - Traffic specification
        Length of service 1 data: 6 words, not including header
        Parameter 127 - Token bucket TSpec
        Parameter 127 flags: 0
        Parameter 127 data length: 5 words, not including header
        Token bucket rate: 100
        Token bucket size: 10000
        Peak data rate: 100
        Minimum policed unit: 100
        Maximum policed unit: 65535
    ADSPEC: 13
        Length: 52
        Class number: 13 - ADSPEC object
        Message format version: 0
        Data length: 11 words, not including header
        Default General Parameters
            Service header 1 - Default General Parameters
            Break bit not set
            Data length: 8 words, not including header
            IS Hop Count - 1 (type 4, length 1)
            Path b/w estimate - 1250000 (type 6, length 1)
            Minimum path latency - 0 (type 8, length 1)
            Composed MTU - 1500 (type 10, length 1)
        Guaranteed
            Service header 2 - Guaranteed
            Break bit set
            Data length: 0 words, not including header
        Controlled Load
            Service header 5 - Controlled Load
            Break bit set
            Data length: 0 words, not including header

Frame 2 (1514 on wire, 1514 captured)
    Arrival Time: Feb 10, 2002 14:16:27.2627
    Time delta from previous packet: 2.952354 seconds
    Frame Number: 2
    Packet Length: 1514 bytes
    Capture Length: 1514 bytes
Ethernet II
    Destination: 00:10:5a:af:24:cc (00:10:5a:af:24:cc)
    Source: 00:a0:24:9e:ea:66 (3Com_9e:ea:66)
    Type: IP (0x0800)
Internet Protocol
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
```

```
        .... ...0 = ECN-CE: 0
    Total Length: 128
    Identification: 0x1c00
    Flags: 0x04
        .1.. = Don't fragment: Set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64
    Protocol: RSVP (0x2e)
    Header checksum: 0x12fc (correct)
    Source: router.ssade (192.168.69.1)
    Destination: vserver.ssade (192.168.69.2)
Resource ReserVation Protocol (RSVP)
    RSVP Header
        RSVP Version: 1
        Flags: 00
        Message Type: RESV Message (2)
        Message Checksum
        Sending TTL: 63
        Message length: 108
    SESSION: 1
        Length: 12
        Class number: 1 - SESSION object
        C-type: 1 - IPv4
        Destination address: vclient.sertel (192.169.69.5)
        Protocol: TCP (6)
        Flags: 155
        Port number: 22000
    HOP: 3
        Length: 12
        Class number: 3 - HOP object
        C-type: 1 - IPv4
        Neighbor address: 192.168.69.1
        Logical interface: 0
    TIME VALUES: 5
        Length: 8
        Class number: 5 - TIME VALUES object
        C-type: 1
        Refresh interval: 30000 ms (30 seconds)
    STYLE: 8
        Length: 8
        Class number: 8 - STYLE object
        C-type: 1
        Style: 10 - Fixed Filter
    FLOWSPEC: 9
        Length: 48
        Class number: 9 - FLOWSPEC object
        Message format version: 0
        Data length: 10 words, not including header
        Service header: 2 - Guaranteed
        Length of service 2 data: 9 words, not including header
        Parameter 127 - Token bucket TSpec
        Parameter 127 flags: 0
        Parameter 127 data length: 5 words, not including header
        Token bucket rate: 100
        Token bucket size: 10000
        Peak data rate: 100
        Minimum policed unit: 100
        Maximum policed unit: 65535
        Parameter 130 - Guaranteed-rate RSpec
        Parameter 130 flags: 0
        Parameter 130 data length: 2 words, not including header
        Rate: 10000
        Slack term: 1
    FILTERSPEC: 10
        Length: 12
        Class number: 10 - FILTER SPEC object
        C-type: 1 - IPv4
        Sender IPv4 address: vserver.ssade (192.168.69.2)
        Sender port number: 20000

Frame 3 (1514 on wire, 1514 captured)
    Arrival Time: Feb 10, 2002 14:16:32.0694
    Time delta from previous packet: 4.806637 seconds
    Frame Number: 3
```

```
    Packet Length: 1514 bytes
    Capture Length: 1514 bytes
Ethernet II
    Destination: 00:10:5a:af:24:cc (00:10:5a:af:24:cc)
    Source: 00:a0:24:9e:ea:66 (3Com_9e:ea:66)
    Type: IP (0x0800)
Internet Protocol
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 72
    Identification: 0x1d00
    Flags: 0x04
        .1.. = Don't fragment: Set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64
    Protocol: RSVP (0x2e)
    Header checksum: 0x1234 (correct)
    Source: router.ssade (192.168.69.1)
    Destination: vserver.ssade (192.168.69.2)
Resource ReserVation Protocol (RSVP)
    RSVP Header
        RSVP Version: 1
        Flags: 00
        Message Type: RESV TEAR Message (6)
        Message Checksum
        Sending TTL: 63
        Message length: 52
    SESSION: 1
        Length: 12
        Class number: 1 - SESSION object
        C-type: 1 - IPv4
        Destination address: vclient.sertel (192.169.69.5)
        Protocol: TCP (6)
        Flags: 55
        Port number: 22000
    HOP: 3
        Length: 12
        Class number: 3 - HOP object
        C-type: 1 - IPv4
        Neighbor address: 192.168.69.1
        Logical interface: 0
    STYLE: 8
        Length: 8
        Class number: 8 - STYLE object
        C-type: 1
        Style: 10 - Fixed Filter
    FILTERSPEC: 10
        Length: 12
        Class number: 10 - FILTER SPEC object
        C-type: 1 - IPv4
        Sender IPv4 address: vserver.ssade (192.168.69.2)
        Sender port number: 20000

Frame 4 (1514 on wire, 1514 captured)
    Arrival Time: Feb 10, 2002 14:16:37.4682
    Time delta from previous packet: 5.398861 seconds
    Frame Number: 4
    Packet Length: 1514 bytes
    Capture Length: 1514 bytes
Ethernet II
    Destination: 00:10:5a:af:24:cc (00:10:5a:af:24:cc)
    Source: 00:a0:24:9e:ea:66 (3Com_9e:ea:66)
    Type: IP (0x0800)
Internet Protocol
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
```

```
    Total Length: 128
    Identification: 0x1e00
    Flags: 0x04
        .1.. = Don't fragment: Set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64
    Protocol: RSVP (0x2e)
    Header checksum: 0x10fc (correct)
    Source: router.ssade (192.168.69.1)
    Destination: vserver.ssade (192.168.69.2)
Resource ReserVation Protocol (RSVP)
    RSVP Header
        RSVP Version: 1
        Flags: 00
        Message Type: RESV Message (2)
        Message Checksum
        Sending TTL: 63
        Message length: 108
    SESSION: 1
        Length: 12
        Class number: 1 - SESSION object
        C-type: 1 - IPv4
        Destination address: vclient.sertel (192.169.69.5)
        Protocol: TCP (6)
        Flags: 155
        Port number: 22000
    HOP: 3
        Length: 12
        Class number: 3 - HOP object
        C-type: 1 - IPv4
        Neighbor address: 192.168.69.1
        Logical interface: 0
    TIME VALUES: 5
        Length: 8
        Class number: 5 - TIME VALUES object
        C-type: 1
        Refresh interval: 30000 ms (30 seconds)
    STYLE: 8
        Length: 8
        Class number: 8 - STYLE object
        C-type: 1
        Style: 10 - Fixed Filter
    FLOWSPEC: 9
        Length: 48
        Class number: 9 - FLOWSPEC object
        Message format version: 0
        Data length: 10 words, not including header
        Service header: 2 - Guaranteed
        Length of service 2 data: 9 words, not including header
        Parameter 127 - Token bucket TSpec
        Parameter 127 flags: 0
        Parameter 127 data length: 5 words, not including header
        Token bucket rate: 100
        Token bucket size: 10000
        Peak data rate: 100
        Minimum policed unit: 100
        Maximum policed unit: 65535
        Parameter 130 - Guaranteed-rate RSpec
        Parameter 130 flags: 0
        Parameter 130 data length: 2 words, not including header
        Rate: 10000
        Slack term: 1
    FILTERSPEC: 10
        Length: 12
        Class number: 10 - FILTER SPEC object
        C-type: 1 - IPv4
        Sender IPv4 address: vserver.ssade (192.168.69.2)
        Sender port number: 20000

Frame 5 (178 on wire, 178 captured)
    Arrival Time: Feb 10, 2002 14:16:54.3291
    Time delta from previous packet: 16.860835 seconds
    Frame Number: 5
    Packet Length: 178 bytes
```

```
    Capture Length: 178 bytes
Ethernet II
    Destination: 00:a0:24:9e:ea:66 (3Com_9e:ea:66)
    Source: 00:10:5a:af:24:cc (00:10:5a:af:24:cc)
    Type: IP (0x0800)
Internet Protocol
    Version: 4
    Header length: 24 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 164
    Identification: 0xe55e
    Flags: 0x00
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64
    Protocol: RSVP (0x2e)
    Header checksum: 0xf46f (correct)
    Source: vserver.ssade (192.168.69.2)
    Destination: vclient.sertel (192.169.69.5)
    Options: (4 bytes)
        Unknown (0x94) (4 bytes)
Resource ReserVation Protocol (RSVP)
    RSVP Header
        RSVP Version: 1
        Flags: 00
        Message Type: PATH Message (1)
        Message Checksum
        Sending TTL: 62
        Message length: 140
    SESSION: 1
        Length: 12
        Class number: 1 - SESSION object
        C-type: 1 - IPv4
        Destination address: vclient.sertel (192.169.69.5)
        Protocol: TCP (6)
        Flags: 155
        Port number: 22000
    HOP: 3
        Length: 12
        Class number: 3 - HOP object
        C-type: 1 - IPv4
        Neighbor address: 192.168.69.2
        Logical interface: 0
    TIME VALUES: 5
        Length: 8
        Class number: 5 - TIME VALUES object
        C-type: 1
        Refresh interval: 30000 ms (30 seconds)
    SENDER TEMPLATE: 11
        Length: 12
        Class number: 11 - SENDER TEMPLATE object
        C-type: 1 - IPv4
        Sender IPv4 address: vserver.ssade (192.168.69.2)
        Sender port number: 20000
    SENDER TSPEC: 12
        Length: 36
        Class number: 12 - SENDER TSPEC object
        Message format version: 0
        Data length: 7 words, not including header
        Service header: 1 - Traffic specification
        Length of service 1 data: 6 words, not including header
        Parameter 127 - Token bucket TSpec
        Parameter 127 flags: 0
        Parameter 127 data length: 5 words, not including header
        Token bucket rate: 100
        Token bucket size: 10000
        Peak data rate: 100
        Minimum policed unit: 100
        Maximum policed unit: 65535
    ADSPEC: 13
        Length: 52
```

```
        Class number: 13 - ADSPEC object
        Message format version: 0
        Data length: 11 words, not including header
        Default General Parameters
            Service header 1 - Default General Parameters
            Break bit not set
            Data length: 8 words, not including header
            IS Hop Count - 1 (type 4, length 1)
            Path b/w estimate - 1250000 (type 6, length 1)
            Minimum path latency - 0 (type 8, length 1)
            Composed MTU - 1500 (type 10, length 1)
        Guaranteed
            Service header 2 - Guaranteed
            Break bit set
            Data length: 0 words, not including header
        Controlled Load
            Service header 5 - Controlled Load
            Break bit set
            Data length: 0 words, not including header

Frame 6 (1514 on wire, 1514 captured)
    Arrival Time: Feb 10, 2002 14:17:05.4271
    Time delta from previous packet: 11.098041 seconds
    Frame Number: 6
    Packet Length: 1514 bytes
    Capture Length: 1514 bytes
Ethernet II
    Destination: 00:10:5a:af:24:cc (00:10:5a:af:24:cc)
    Source: 00:a0:24:9e:ea:66 (3Com_9e:ea:66)
    Type: IP (0x0800)
Internet Protocol
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 72
    Identification: 0x1f00
    Flags: 0x04
        .1.. = Don't fragment: Set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64
    Protocol: RSVP (0x2e)
    Header checksum: 0x1034 (correct)
    Source: router.ssade (192.168.69.1)
    Destination: vserver.ssade (192.168.69.2)
Resource ReserVation Protocol (RSVP)
    RSVP Header
        RSVP Version: 1
        Flags: 00
        Message Type: RESV TEAR Message (6)
        Message Checksum
        Sending TTL: 63
        Message length: 52
    SESSION: 1
        Length: 12
        Class number: 1 - SESSION object
        C-type: 1 - IPv4
        Destination address: vclient.sertel (192.169.69.5)
        Protocol: TCP (6)
        Flags: 55
        Port number: 22000
    HOP: 3
        Length: 12
        Class number: 3 - HOP object
        C-type: 1 - IPv4
        Neighbor address: 192.168.69.1
        Logical interface: 0
    STYLE: 8
        Length: 8
        Class number: 8 - STYLE object
        C-type: 1
        Style: 10 - Fixed Filter
```

```
    FILTERSPEC: 10
        Length: 12
        Class number: 10 - FILTER SPEC object
        C-type: 1 - IPv4
        Sender IPv4 address: vserver.ssade (192.168.69.2)
        Sender port number: 20000

Frame 7 (178 on wire, 178 captured)
    Arrival Time: Feb 10, 2002 14:17:34.3391
    Time delta from previous packet: 28.911954 seconds
    Frame Number: 7
    Packet Length: 178 bytes
    Capture Length: 178 bytes
Ethernet II
    Destination: 00:a0:24:9e:ea:66 (3Com_9e:ea:66)
    Source: 00:10:5a:af:24:cc (00:10:5a:af:24:cc)
    Type: IP (0x0800)
Internet Protocol
    Version: 4
    Header length: 24 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 164
    Identification: 0xeb38
    Flags: 0x00
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64
    Protocol: RSVP (0x2e)
    Header checksum: 0xee95 (correct)
    Source: vserver.ssade (192.168.69.2)
    Destination: vclient.sertel (192.169.69.5)
    Options: (4 bytes)
        Unknown (0x94) (4 bytes)
Resource ReserVation Protocol (RSVP)
    RSVP Header
        RSVP Version: 1
        Flags: 00
        Message Type: PATH Message (1)
        Message Checksum
        Sending TTL: 62
        Message length: 140
    SESSION: 1
        Length: 12
        Class number: 1 - SESSION object
        C-type: 1 - IPv4
        Destination address: vclient.sertel (192.169.69.5)
        Protocol: TCP (6)
        Flags: 155
        Port number: 22000
    HOP: 3
        Length: 12
        Class number: 3 - HOP object
        C-type: 1 - IPv4
        Neighbor address: 192.168.69.2
        Logical interface: 0
    TIME VALUES: 5
        Length: 8
        Class number: 5 - TIME VALUES object
        C-type: 1
        Refresh interval: 30000 ms (30 seconds)
    SENDER TEMPLATE: 11
        Length: 12
        Class number: 11 - SENDER TEMPLATE object
        C-type: 1 - IPv4
        Sender IPv4 address: vserver.ssade (192.168.69.2)
        Sender port number: 20000
    SENDER TSPEC: 12
        Length: 36
        Class number: 12 - SENDER TSPEC object
        Message format version: 0
        Data length: 7 words, not including header
```

```
        Service header: 1 - Traffic specification
        Length of service 1 data: 6 words, not including header
        Parameter 127 - Token bucket TSpec
        Parameter 127 flags: 0
        Parameter 127 data length: 5 words, not including header
        Token bucket rate: 100
        Token bucket size: 10000
        Peak data rate: 100
        Minimum policed unit: 100
        Maximum policed unit: 65535
    ADSPEC: 13
        Length: 52
        Class number: 13 - ADSPEC object
        Message format version: 0
        Data length: 11 words, not including header
        Default General Parameters
            Service header 1 - Default General Parameters
            Break bit not set
            Data length: 8 words, not including header
            IS Hop Count - 1 (type 4, length 1)
            Path b/w estimate - 1250000 (type 6, length 1)
            Minimum path latency - 0 (type 8, length 1)
            Composed MTU - 1500 (type 10, length 1)
        Guaranteed
            Service header 2 - Guaranteed
            Break bit set
            Data length: 0 words, not including header
        Controlled Load
            Service header 5 - Controlled Load
            Break bit set
            Data length: 0 words, not including header

Frame 8 (82 on wire, 82 captured)
    Arrival Time: Feb 10, 2002 14:18:10.2425
    Time delta from previous packet: 13.873441 seconds
    Frame Number: 8
    Packet Length: 82 bytes
    Capture Length: 82 bytes
Ethernet II
    Destination: 00:a0:24:9e:ea:66 (3Com_9e:ea:66)
    Source: 00:10:5a:af:24:cc (00:10:5a:af:24:cc)
    Type: IP (0x0800)
Internet Protocol
    Version: 4
    Header length: 24 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 68
    Identification: 0xecfa
    Flags: 0x00
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64
    Protocol: RSVP (0x2e)
    Header checksum: 0xed33 (correct)
    Source: vserver.ssade (192.168.69.2)
    Destination: vclient.sertel (192.169.69.5)
    Options: (4 bytes)
        Unknown (0x94) (4 bytes)
Resource ReserVation Protocol (RSVP)
    RSVP Header
        RSVP Version: 1
        Flags: 00
        Message Type: PATH TEAR Message (5)
        Message Checksum
        Sending TTL: 62
        Message length: 44
    SESSION: 1
        Length: 12
        Class number: 1 - SESSION object
        C-type: 1 - IPv4
        Destination address: vclient.sertel (192.169.69.5)
```

```
    Protocol: TCP (6)
    Flags: 155
    Port number: 22000
HOP: 3
    Length: 12
    Class number: 3 - HOP object
    C-type: 1 - IPv4
    Neighbor address: 192.168.69.2
    Logical interface: 0
SENDER TEMPLATE: 11
    Length: 12
    Class number: 11 - SENDER TEMPLATE object
    C-type: 1 - IPv4
    Sender IPv4 address: vserver.ssade (192.168.69.2)
    Sender port number: 20000
```

Referencias

[AWD99] D. Awduche, L. Berger, D. Gan, T. Li, G. Swallow, V. Srinivasan, "Extensions to RSVP for LSP Tunnels", *RFC 3209*, December 2001.

[BAJ99] S. Bajaj, L. Breslau, D. Estrin, K. Fall, and Rally Floyd, "Improving simulation for network research". *Technical Report 99-702*, University of Southern California, March 1999.

[BAR98] P. Barford, Mark Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation", in *Joint International Conference on Measurement and Modeling of Computer Systems - Performance Evaluation Review*, 1998, pp. 151-160.

[BAR99] C. Barakat, E. Altman, "Analysis of TCP with Several Bottleneck Nodes", Rapport de recherche, *INRIA, n°3620*, February 1999.

[BER00] Y. Bernet et al., "A Framework for Integrated Services over DiffServ Networks", *RFC 2998*, November 2000.

[BER97] L. Berger, and T. O'Malley, "RSVP Extensions for IPSec Data Flows", *RFC 2207*, September 1997.

[BLA98] S. Blake et al., "An Architecture for Differentiated Services", *RFC 2475*, December 1998.

[BRA97] R. Braden, E., L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource ReSerVation Protocol (RSVP) – version 1 functional specification," *RFC 2205*, September 1997.

[BUR01] Burkhard Stiller, Peter Reichl, Simon Leinen, "Pricing and Cost Recovery for Internet Services: Practical Review, Classification, and Application of Relevant Models", *NETNOMICS: Economic Research and Electronic Networking*, vol. 3, num. 2, pp. 149-171, September 2001.

[CHE96] Z. Chen, S. M. Tan, R. H. Campbell, and Y. Li. "Real Time Video and Audio in the World Wide Web". *World Wide Web Journal*, vol. 1, 1996.

[COM95] D. E. Comer, "Internetworking with TCP/IP", Prentice Hall, 1995.

[CRO95] M. E. Crovella, A. Bestavros, “Explaining World Wide Web traffic self-similarity”, Computer Science Dept., Boston Univ., *Tech Rep. TR-95-015*. October 1995.

[FAL96] K. Fall, S. Floyd, “Simulation-based Comparisons of Tahoe, Reno, and SACK TCP”, *Computer Communication Review*, V. 26 N. 3, July 1996, pp. 5-21.

[FIN02] V. Fineberg, “A Practical Architecture for Implementing End-to-End QoS in an IP Network”, *IEEE Communications Magazine*, pp. 122-130, January 2002.

[GIO03] S. Giordano, S. Salsano, S. Van den Berghe, G. Ventre and D. Giannakopoulos, “Advanced QoS Provisioning in IP Networks: The European Premium IP Projects”, *IEEE Communications Magazine*, pp. 30-36, January 2003.

[GRE02] M. Greis, “RSVP/ns: An Implementation of RSVP for the Network Simulator ns-2”, Computer Science Department IV, University of Bonn.

[JAM02] B. Jamoussi et al., “Constraint-Based LSP Setup using LDP”, *RFC 3212*, January 2002.

[KEN98] S. Kent, and R. Atkinson, “IP Encapsulating Security Payload (ESP)”, *RFC 2406*, November 1998.

[MAC95] J. MacKie-Mason, H. Varian, “Pricing the Internet", in B. Kahin, J. Keller(eds), “Public Access to the Internet", Prentice Hall, 1995.

[MPEG] MPEG-4 and H.263 Video Traces for Network Performance Evaluation, *http://www-tkn.ee.tu-berlin.de/research/trace/trace.html*

[NIC98] K. Nichols et al., “Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers”, *RFC 2474*, December 1998.

[NS2] The Network Simulator – ns-2, *http://www.isi.edu/nsnam/ns/*

[PAX97] V. Paxson, “Measurements and Analysis of End-to-End Internet Dynamics,” Ph.D. dissertation, University of California, Berkeley, April 1997.

[POS00] M. Postigo-Boix, J. García-Haro, M. Aguilar-Igartua, "Transmisión Eficiente de Bloques en Tiempo Real sobre Redes IP", *XV Simposium Nacional de la Unión Científica Internacional de Radio*, Zaragoza, Spain, septiembre 2000, pp. 405-406

[POS01a] M. Postigo-Boix, J. García-Haro, M. Aguilar-Igartua, "Cost Minimization Study in the Client-Server Transmission of Semi-Elastic Flows Using Internet", en *Proceedings of the 2001 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, Victoria, B.C., Canada, agosto 2001, pp. 188-191.

[POS01b] M. Postigo-Boix, J. García-Haro, M. Aguilar-Igartua, "Análisis de Minimización de Costes en la Transmisión de Flujos Semi-Elásticos sobre Internet", en las actas de las *III Jornadas de Ingeniería Telemática*, Barcelona, España, septiembre 2001, pp. 37-44.

[POS02a] M. Postigo-Boix, J. García-Haro, M. Aguilar-Igartua, "Cost Minimization Study of Semi-Elastic Flows Using Internet", en *Proceedings of the IEEE International Conference on Communications 2002*, New York, USA, Abril 2002, pp. 2237-41.

[POS02b] M. Postigo-Boix, J. García-Haro, M. Aguilar-Igartua, "A Cost Optimization Methodology for Internet Transmission of Semi-Elastic Traffic Flows", en *Proceedings of the 3rd International Conference on Internet Computing*, Las Vegas, USA, junio 2002, pp. 311-16.

[POS02c] M. Postigo-Boix, J. García-Haro, M. Aguilar-Igartua, "Simulation of a Reliable Client-Server System for Semi-Elastic Flows", en *Proceedings of the 2002 IASTED International Conference on Communication Systems and Networks*, Malaga, España, septiembre 2002, pp. 332-37.

[POS02d] M. Postigo-Boix, J. García-Haro, J. L. Melús-Moreno, "Transmission of Semi-Elastic Flows under High Variability Background Traffic", en *Proceedings of the 2002 IASTED International Conference on Applied Modelling and Simulation*, Cambridge, USA, noviembre 2002, pp. 105-110.

[REI01] P. Reichl, B. Stiller, “Edge Pricing in Space and Time: Theoretical and Practical Aspects of the Cumulus Pricing Scheme”, *Proceedings of the 17th International Teletraffic Conference*, ITC-17, Salvador da Bahia, Brazil, September 2001.

[ROB98] J. Roberts, “Quality of Service Guarantees and Charging in Multi-service Networks”, *IEICE Transactions on Communications*, May 1998.

[ROS01] E. Rosen, A. Viswanathan, R. Callon, “Multiprotocol Label Switching Architecture”, *RFC 3031*, January 2001.

[SCH98] H. Schulzrinne, A. Rao, R. Lanphier, “Real Time Streaming Protocol (RTSP)”, *RFC 2326*, April 1998.

[SDH02] SONET / SDH Technical Summary, http://www.techfest.com/networking/wan/sonet.htm

[SHA00] C. Shahabi, M. H. Alshayeji, “Super-streaming: a New Object Delivery Paradigm for Continuous Media Servers”, *Journal of Multimedia Tools and Applications*, vol.11, issue 1, May 2000, pp. 275-298.

[SON97] D.J. Songhurst, F.P. Kelly, “Charging schemes for multiservice networks", in *Proc. ITC15, (“Teletraffic Contributions for the Information Age")*, Elsevier, 1997.

[STA99] Stardust.com. White Paper – The Need for QoS. July 1999.

[STE00] Stemm, M., Katz, R., and Seshan, S.: “A Network Measurement Architecture for Adaptive Applications”, *Proceedings of IEEE Infocom 2000*, Vol. 1, pp. 185-294, Tel Aviv, Israel, March 2000.

[STI01] B. Stiller, P. Reichl. J. Gerke, P. Flury, “A Generic and Modular Internet Charging System for the Cumulus Pricing Scheme”, *Journal of Network and Systems Management*, Vol. 9, No. 3, September 2001.

[THO99] B. Thomas, N. Feldman, P. Doolan, L. Andersson, A. Fredette, “LDP Specification”, *RFC 3036*, January 2001.

[VIN96] VINT (Virtual InterNetwork Testbed), 1996. *http://www.isi.edu/nsnam/vint*

[WAN00] Wang, x., and Schulzrinne, h.: “An Integrated Resource Negotiation, Pricing, and QoS Adaptation Framework for Multimedia Applications”, *Journal of Selected Areas in Communications*, Vol. 18, Num. 12, pp. 2514-29, December 2000.

[WRED] Technical Specification from Cisco, Distributed Weighted Random Early Detection, *http://www.cisco.com/univercd/cc/td/doc/product/software/ios111/cc111/wred.pdf*

[XIA02] X. Xiao, T. Telkamp, V. Fineberg, C. Chen, and L. M. Ni, “A Practical Approach for Providing QoS in the Internet Backbone”, *IEEE Communications Magazine*, pp. 56-62, December 2002.

[YAV00] R. Yavatkar, D. Hoffman, Y. Bernet, F. Baker, “SBM (Subnet Bandwidth Manager): A Protocol for RSVP-based Admission Control over IEEE 802-style networks”, *RFC 2814*, May 2000.