TESIS DOCTORAL

TÉCNICAS DE OPTIMIZACIÓN DE PARÁMETROS DE RED PARA LA MEJORA DE LA COMUNICACIÓN EN SERVICIOS DE TIEMPO REAL

Autor

José Mª Saldaña Medina

Director

Julián Fernández Navajas

Doctorado en Tecnologías de la Información
y Comunicaciones en Redes Móviles

Departamento de Ingeniería Electrónica y Comunicaciones

Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

2011

RESUMEN

En esta tesis se presenta un conjunto de estudios sobre técnicas de optimización del tráfico de servicios de tiempo real, aplicables en el caso de que un número de flujos compartan la misma ruta.

En estos escenarios la eficiencia se puede mejorar mediante la compresión de cabeceras cuyos campos se repiten o se incrementan de uno en uno para todos los paquetes del flujo. Posteriormente se pueden multiplexar varios paquetes en uno más grande, que se envía extremo a extremo utilizando un túnel. Algunos de estos métodos de optimización, como ocurre para servicios basados en RTP, han sido estandarizados por el IETF. También pueden adaptarse a otros servicios con características similares.

Se han realizado pruebas encaminadas a medir la mejora que se puede obtener mediante estos métodos en servicios de tiempo real. Se han realizado medidas tanto en entornos reales como en otros emulados mediante virtualización o simulados. Se han evaluado los parámetros de calidad objetiva resultantes, como el retardo o las pérdidas, que posteriormente se han traducido en valores de calidad percibida, utilizando estimadores del MOS *(Mean Opinion Score,* puntuación de la opinión media).

El primer servicio estudiado ha sido el de Voz sobre IP *(Voice over IP*, VoIP). Para ello, se ha diseñado un sistema de telefonía IP similar a los utilizados en entornos empresariales, incluyendo un sistema que multiplexa las llamadas simultáneas entre el mismo par de sucursales. El sistema se ha implementado en un entorno de pruebas, utilizando herramientas de *software* libre. Se han realizado también medidas de la mejora que se puede obtener en los parámetros de calidad de servicio cuando se multiplexa un número de flujos nativos. Por último, se han realizado simulaciones para evaluar el efecto en la probabilidad de admisión.

El segundo servicio de tiempo real que se ha probado ha sido el del género FPS *(First Person Shooter,* Tirador en Primera Persona) de juegos *online*. Se ha adaptado el estándar de multiplexión para el tráfico de estos juegos, debido a que, a diferencia del servicio de voz, no usan el protocolo RTP sino que envían paquetes UDP. Se han estudiado, analíticamente y mediante simulaciones, las mejoras que se pueden obtener, mostrando que se consiguen ahorros significativos en ancho de banda y paquetes por segundo, que se pueden traducir en mejoras en los parámetros de calidad.

ABSTRACT

This Doctoral Thesis presents a set of studies about optimization techniques for real-time services, which can be used in the case of a number of flows sharing the same path.

In these scenarios, the efficiency can be improved by header compression, as many header fields are the same for every packet, or increase by one with respect to the previous packet. In addition, many packets can be multiplexed in a bigger one, which is sent to the destination via a tunnel. Some multiplexing methods have been standardized by IETF, as it happens with the ones based on RTP. The methods could be adapted to other services with similar characteristics.

Some measurements have been carried out so as to quantify the improvement which can be achieved by the use of these methods for real-time services. Different test environments have been used: real hardware, virtual emulation and simulation. Some objective quality parameters, like delay and packet loss, have been obtained, and they have been properly combined in order to calculate some subjective quality estimators, in terms of MOS (Mean Opinion Score).

First, VoIP service has been tested using an IP telephony system which has been designed using the scheme of commercial solutions. It includes a multiplexing system, which merges the calls that share the same origin and destination offices. The system has also been implemented in a testbed using free software tools. Some measurements of the obtained improvements in terms of quality parameters have been carried out. Finally, a simulation environment has allowed us to study the improvements of admission probability obtained by multiplexing.

The optimization method has also been tested for another real-time service, i.e. First Person Shooter (FPS) games. The multiplexing standard has been adapted to this service, as these games do not use RTP but only UDP. The savings in terms of bandwidth and packets per second have been calculated analytically, and corroborated by simulations. Finally, some tests have been carried out in order to show the improvements in terms of quality parameters.

AGRADECIMIENTOS

En primer lugar, quiero manifestar mi más sincero agradecimiento a los profesores Julián Fernández Navajas y José Ruiz Mas, por darme la oportunidad de trabajar en este campo de investigación tan interesante. A lo largo de estos años hemos compartido muchos éxitos y algunos fracasos, que siempre han terminado convertidos en éxitos a base de trabajo, paciencia y optimismo.

También quiero dar las gracias a todas las demás personas que forman o han formado parte de este equipo de investigación: José Ignacio Aznar, con el que compartí los primeros pasos en el mundo de la investigación, Eduardo Viruete, que siempre me dio buenos consejos basados en su experiencia, Luis Casadesus, con quien he compartido tantos buenos momentos. También los estudiantes que realizaron sus Proyectos Fin de Carrera con nosotros, empezando por Esmeralda Betrián, siguiendo por Laura Esteban, Adrián Rejas y Jenifer Murillo, que después de acabar su Proyecto participó e hizo posible la realización de muchas de las pruebas que ahora se presentan. También Luis Sequeira ha colaborado en estos últimos meses.

Muchas de las pruebas que se presentan aquí han sido posibles gracias a la instalación de algunas máquinas *(cajalon04* y *cajalon05),* que puso en marcha José Alberto Royo, y luego Agustín Sin y José Antonio Alvarado han sabido mantener y mejorar.

También quiero agradecer su ayuda a todo el Personal de Administración y Servicios del Departamento, de modo especial a Ignacio Mendieta por su disponibilidad para ayudar a solucionar muchos problemas del día a día, y a María Ángeles Serrano por su ayuda con todos los trámites.

No puedo olvidarme de citar aquí a Ignacio Ferrero, que fue quien primero me sugirió hacer los cursos de Doctorado, que luego se convirtieron en Máster. También a José Sanvicente, que siempre me animó a dedicarme a la investigación, a Jesús Plaza, que me dio ánimos en muchos momentos, y a Emiliano Conejo, que nos asesoró con el Inglés en algunos trabajos.

Por último, quiero también manifestar mi agradecimiento a algunos de los revisores anónimos de los congresos y revistas que, aunque a veces han sido duros, en otras ocasiones nos han dado pistas sobre cómo avanzar en la investigación. En concreto, estoy especialmente agradecido al revisor del

Workshop DENVECT del *CCNC 2011* que escribió la frase: *"I suggest the authors, if they want to target an enterntainment conference, to put their contribution in the context of MMOG and/or virtual collaborative environments"*, dándonos así una magnífica idea para iniciar una nueva línea de investigación.

Y por supuesto, no puedo olvidarme de mi familia y de todos los amigos que me han animado y se han interesado por mi trabajo en estos años, sin olvidar a María y Álvaro Castellet.

TABLA DE CONTENIDO

LISTA DE FIGURAS

GLOSARIO

Best-effort: Término utilizado para denominar una red en la que no existen garantías de tiempos de entrega, es decir, los paquetes son entregados con el mínimo retardo posible según el estado y el tráfico de la red.

Bridge: Dispositivo que conecta dos segmentos en una red de conmutación de paquetes.

Buffer: Memoria que se utiliza para almacenar datos pendientes de envío, evitando así que se descarten al no poder transmitirse instantáneamente.

Checksum: Redundancia que se añade a un conjunto de datos, para comprobar que no se han modificado durante la transmisión.

Codec: Abreviatura de codificador-decodificador. Especificación que permite transformar un flujo de datos de un formato a otro.

Downlink: Aunque el término se originó para referirse al enlace radio que comienza en un satélite y acaba en una estación base terrestre, se utiliza en sentido amplio para referirse al enlace de la red de acceso a Internet, en el sentido que va desde la red hasta el usuario.

Ethernet: Tecnología de conexión usada en redes locales, que define el nivel físico y de enlace, y utiliza control de acceso al medio por contienda. Se define en el estándar IEEE 802.3.

FIFO (First Input First Output): Política usada en una cola, por la que los elementos son atendidos según el orden de llegada.

Gateway: Dispositivo que conecta redes con distintos protocolos y arquitecturas. En esta tesis se utilizará para referirse al dispositivo que conecta una red de datos con la Red Telefónica Conmutada (RTC).

Jitter: En redes de conmutación de paquetes, se utiliza este término para referirse a la variación entre el retardo de unos paquetes con respecto a otros.

LAN (Local Area Network): Red de área local, que conecta máquinas en un entorno reducido, como puede ser un domicilio o una oficina.

Look-ahead: Técnica que usan algunos *codec*, que consiste en utilizar información de las siguientes tramas para poder comprimir las anteriores. Provoca un retardo adicional.

Módem (Acrónimo de *mod*ulación-*dem*odulación): Aparato que convierte las señales digitales en analógicas para su transmisión, o a la inversa (DRAE).

Overhead: Información que se transmite en un sistema de comunicaciones, que es necesaria para la comunicación, pero no es originada por el usuario.

Payload: Parte de un paquete de información en la que se encuentran los datos.

RFC (Request For Comments): Memoria publicada por el IETF *(Internet Engineering Task Force)* para definir protocolos, comportamientos o innovaciones referidas al comportamiento de Internet.

Router (enrutador): Dispositivo que conmuta paquetes entre varias redes a las que se encuentra conectado.

Softphone: Aplicación informática que se comporta de modo similar a un teléfono, utilizando la tarjeta de sonido del equipo, y enviando los flujos de voz a través de la red de datos.

Testbed (banco de pruebas): Sistema que se utiliza para realizar medidas, en el que algunos de los elementos son idénticos al sistema original, mientras que el resto son emulados o sustituidos por otros que imitan su comportamiento.

Trunking: Agrupamiento de varias comunicaciones con el mismo origen y destino, con la finalidad de ahorrar *overhead*.

Uplink (ver *Downlink*): Enlace de la red de acceso a Internet, en el sentido desde el usuario a la red.

WiMAX (Worldwide Interoperability for Microwave Access, Interoperabilidad mundial para acceso por microondas): Familia de protocolos para proporcionar acceso a Internet fijo y móvil. Se especifica en la norma IEEE 802.16.

Capítulo 1

INTRODUCCIÓN

Aunque Internet se desarrolló inicialmente como una red que no garantizaba la entrega de la información en tiempo real, con el paso de los años también se está usando para proporcionar servicios que tienen requerimientos temporales estrictos. Esto ha llevado a la búsqueda de métodos para conseguir que la infraestructura de red sea capaz de dar soporte a los nuevos servicios, pero con el problema de que Internet no se puede rediseñar desde el principio, debido a su gran tamaño actual [Han06]. De hecho, en sus comienzos, su pequeño tamaño permitía poner de acuerdo a todos los usuarios para realizar modificaciones en los protocolos. La última gran modificación se hizo el 1 de enero de 1983, cuando el protocolo NCP, que combinaba direccionamiento y transporte, se sustituyó por TCP/IP, separando así ambas tareas. En ese momento Internet sólo estaba compuesta por unos cuatrocientos nodos. Pero en la actualidad, dado su gran tamaño, estos cambios no se pueden realizar súbitamente: los nuevos avances se abren paso poco a poco, y necesitan casi siempre ser compatibles con lo ya implantado.

Los nuevos servicios interactivos despiertan un gran interés, en parte por las posibilidades de negocio que presentan. Pero con frecuencia se ofrecen al usuario sin tener en cuenta los recursos necesarios para garantizar un servicio de calidad. Para dar solución a esta situación, se ha hecho un gran esfuerzo de investigación en definir los parámetros que determinan la *Calidad de Servicio (Quality of Service,* QoS) en cada caso, preocupándose tanto de los requerimientos de los servicios como de la calidad proporcionada por la infraestructura de red. Cuando la calidad de la red no es suficiente para proporcionar el servicio en todos los casos, hay que buscar un compromiso, de modo que las limitaciones de la red afecten lo menos posible a la calidad experimentada por el usuario. Para ello se utiliza el concepto de *Calidad de la Experiencia (Quality of Experience,* QoE), que está más centrado en el usuario que el de QoS.

En primer lugar, se pueden medir en la red diferentes parámetros objetivos que afectan al tráfico, como el retardo, la probabilidad de pérdidas, la variación del retardo o el ancho de banda. Cada uno de estos parámetros puede caracterizarse estadísticamente, obteniendo su media o varianza. Posteriormente, mediante la

realización de encuestas a usuarios sobre servicios concretos, se pueden convertir esos parámetros objetivos en estimaciones de calidad subjetiva.

Una vez realizado un número suficiente de encuestas, se trasladan los resultados a un modelo matemático que nos proporciona una estimación de la percepción del usuario que se puede obtener a partir de los valores objetivos medidos en la red.

Aunque se han presentado muchas soluciones para mejorar la QoS de las infraestructuras, una de las más usadas es el denominado *Control de Admisión (Call Admission Control,* CAC*)*. De modo general, se puede decir que consiste en llevar un control de los recursos de red disponibles y ocupados, para así conocer la QoS que podremos proporcionar, y tomar correctamente la decisión de admitir o no un nuevo requerimiento de servicio. Esto provoca que en algunos casos las peticiones sean rechazadas, pero evita proporcionar un mal servicio a las sesiones en curso y a las que se quieren establecer. El *Control de Admisión* busca adecuar la prestación de los servicios a la provisión de recursos y al dimensionado que se realizó en el momento de su puesta en marcha.

En esta tesis nos centraremos en servicios de tiempo real. Un problema que presentan estos servicios es la poca eficiencia de los paquetes que se envían por la red. Al ser Internet una red de conmutación de paquetes, cada uno debe llevar una cabecera IP y otra TCP o UDP. Los requerimientos temporales hacen que la información se deba enviar con mucha frecuencia y en paquetes muy pequeños, lo que provoca que la información útil *(payload)* suponga un porcentaje pequeño del tamaño del paquete, que en ocasiones puede llegar a ser solamente el 25 o 30% del total.

En los últimos años se está agravando el problema de la escasez de direcciones IPv4, y esto está llevando a la adopción de IPv6, la siguiente versión del protocolo de nivel de red, que tiene la gran ventaja de proporcionar un gran espacio de direcciones. Esta nueva versión se está introduciendo poco a poco [CGKR10], aunque en algunos países se adopta con mayor rapidez. Pero IPv6 ofrece una eficiencia peor, pues su cabecera mínima es de 40 bytes, es decir, el doble que la de su predecesor IPv4.

Así pues, se deben buscar soluciones para mejorar la eficiencia de los protocolos. Una de ellas es la multiplexión, un concepto muy conocido y utilizado, aunque no por ello menos interesante, que puede tener también una gran utilidad en los escenarios donde funcionan estos nuevos servicios. El aumento del número de muestras que se envían en cada paquete podría verse como una solución, pero

también aumenta el retardo, porque deberemos esperar a tener un número de muestras mayor. Pero en el caso en que varios flujos compartan la misma ruta, se pueden utilizar técnicas de optimización, como son el multiplexado de los paquetes de diferentes usuarios, al que se puede añadir la compresión de cabeceras. De esta forma se puede mejorar la eficiencia, añadiendo solamente un pequeño retardo, que corresponde al tiempo de encolado en el multiplexor.

La multiplexión puede lograr ahorros significativos de ancho de banda, consiguiendo que aumente el número de usuarios que pueden usar el servicio simultáneamente, y mejorando por tanto la probabilidad de admisión.

En este trabajo nos centraremos en dos servicios de tiempo real muy significativos en la actualidad: los sistemas de telefonía IP empresariales, por un lado, y los juegos *online* por otro. Ambos servicios envían paquetes pequeños a altas tasas y, por tanto, su eficiencia es susceptible de mejora mediante técnicas de optimización.

Estructura de la tesis

La tesis se ha dividido en tres secciones, de las que incluimos aquí un breve resumen. En la primera sección se abordará el estado de la cuestión. Se explicará la problemática de los servicios de tiempo real que utilizan Internet, y por qué se han elegido la telefonía IP y los juegos *online* como servicios significativos y susceptibles de optimización. Se introducirá el concepto de *Calidad de Servicio,* y los parámetros y métricas existentes para definirla. Posteriormente se tratará específicamente el problema del dimensionado del *buffer*, que en caso de competencia entre servicios, afecta directamente al rendimiento de los de tiempo real. Otro capítulo se dedicará a las diferentes técnicas de optimización del tráfico, haciendo hincapié en la compresión de cabeceras y en la multiplexión de varios paquetes en uno más grande. Finalmente, se abordará el tema de las distintas herramientas que se pueden usar en las pruebas de tráfico: máquinas reales, *testbed* (banco de pruebas) basado en virtualización y simuladores.

La segunda sección se dedicará al servicio de telefonía IP. En primer lugar, se presentará el diseño de un sistema de telefonía distribuido, con una estructura similar a la de algunas soluciones comerciales actuales, pero con la peculiaridad de estar integrado solamente con *software* libre, lo que facilitará su estudio y modificación. El sistema contará con un CAC basado en parámetros, y también se incluirán algunas mejoras, como la compartición de los *gateway* de cada sucursal, para así conseguir ahorro en llamadas internacionales, que se establecen

en dos tramos, uno a través de Internet, y otro por RTC (Red Telefónica Conmutada) hasta el usuario final. Se presentará también la implementación del sistema en una plataforma de pruebas basada en virtualización. Posteriormente se evaluarán algunos esquemas de optimización del tráfico. La sección se cerrará con un capítulo en el que se realizan simulaciones para comprobar la mejora obtenida con la optimización del tráfico.

La última sección adapta las ideas utilizadas para la optimización del tráfico de telefonía IP, al caso de otro servicio de tiempo real, como son los juegos *online*. Dado que en estos juegos no tiene sentido incluir un control de admisión, pues son soluciones comerciales cerradas que ya tienen sus limitaciones en cuanto a número de usuarios, se adaptarán los esquemas de optimización, pensados para tráfico de voz, al tráfico de los juegos. Posteriormente, se evaluará la influencia de estos esquemas de optimización en la calidad experimentada por los usuarios.

Sección A: Estado de la Cuestión

Después del capítulo inicial de introducción, se ha incluido esta sección en la que se trata el estado de la cuestión en los diferentes temas que van a ser objeto de estudio en la presente tesis. En primer lugar, se explicará la problemática del uso de Internet para el transporte de servicios de tiempo real, y se expondrán las razones por las que se han elegido la telefonía IP y los juegos *online* como dos servicios significativos en los que se pueden usar técnicas de optimización del tráfico.

En el siguiente capítulo trataremos sobre la *Calidad de Servicio*, explicando este concepto, así como los parámetros y modos existentes para medirla.

El capítulo cuarto tratará específicamente sobre un problema que estará presente a lo largo de todo el trabajo: el dimensionado del *buffer* del *router*. Es un tema que ha dado lugar a muchos trabajos en los últimos años, y que, aunque se ha estudiado principalmente en la literatura para *router* de la red troncal, afecta también al tráfico de servicios de tiempo real cuando usan *router* de más baja gama.

En el siguiente capítulo se expondrán los sistemas actuales de optimización del tráfico, especialmente los de compresión de cabeceras, por un lado, y multiplexión, por otro.

La sección se cierra con un capítulo dedicado a los diferentes tipos de herramientas que se usan en la actualidad para las pruebas y medidas. Nos centraremos especialmente en el uso de la virtualización para realizar pruebas en sistemas distribuidos.

Capítulo 2

SERVICIOS DE TIEMPO REAL

En este capítulo presentaremos los dos servicios interactivos en tiempo real que van a ser objeto de estudio en este trabajo. En particular, los servicios de tiempo real se caracterizan por requerir una respuesta rápida, de forma que el usuario los perciba como interactivos.

El primer servicio, muy extendido en Internet, es el de la telefonía IP. En una conversación telefónica la interactividad quiere decir que el retardo boca-a-oído es lo suficientemente pequeño como para permitir una conversación fluida entre los hablantes, evitando que a causa del retardo se interrumpan el uno al otro.

El segundo servicio que estudiaremos será el de los juegos *online,* y más en concreto aquellos en los que la interactividad también es importante, como son los denominados FPS *(First Person Shooter,* Tirador en primera persona). Las acciones de los jugadores deben llegar al servidor con rapidez, para que éste pueda calcular el nuevo estado del juego y transmitirlo a todos los participantes en un intervalo pequeño. Si el usuario percibe que sus acciones no tienen efecto inmediato en el resto de jugadores, o que su dispositivo de visualización no muestra una secuencia de vídeo continua, su experiencia empeorará.

En definitiva, se han seleccionado estos dos servicios como ejemplos en los que la interactividad es un requisito muy claro de forma que, si falta, la calidad experimentada por el usuario desciende con rapidez.

Pasamos ahora a explicar más detenidamente cada uno de ellos.

Sistemas de telefonía IP

Con el término "telefonía IP" nos estamos refiriendo a un concepto más específico que "Voz sobre IP" o VoIP. VoIP se suele entender simplemente como el uso de Internet para el transporte de conversaciones de voz entre usuarios. Telefonía IP se refiere más a los sistemas instalados en empresas u otras organizaciones, que añaden a VoIP una gestión centralizada, disponibilidad, seguridad, y un conjunto de servicios, como un plan de numeración, llamadas en espera, buzón de voz, etc. De todas formas, en algunos lugares puede ocurrir que estos conceptos se utilicen indistintamente.

En la actualidad, en entornos empresariales, se está tendiendo a introducir nuevas soluciones de telefonía IP, en parte buscando disminuir costes. En [SSS06] Intel publicó los resultados de un programa piloto en el que un grupo de empleados utilizó telefonía IP basada en SIP *(Session Initiation Protocol)* [RSCP+02] durante unos meses. La conclusión fue que esta tecnología era beneficiosa para la empresa, en términos de costes y también de productividad. En [AHT06] se puede encontrar otro estudio que ilustra las mejoras obtenidas al usar telefonía IP en lugar de telefonía tradicional. Se obtenían ahorros en costes de equipos, aprovisionamiento, facturación, mantenimiento y servicio, y se recomendaba VoIP como la nueva solución de telefonía para las empresas.

A pesar de la demanda y de la rápida implantación que están teniendo estos servicios, a día de hoy no existe una solución completa que proporcione a los protocolos de señalización una gestión y configuración dinámica de QoS. Las principales arquitecturas desarrolladas para garantizar la QoS no están preparadas para soportar la implantación masiva de nuevos servicios multimedia. Así, los mecanismos basados en *Integrated Services* (IntServ) [BCS94] requieren que todos los *router* situados en el camino de red almacenen el estado de señalización de los flujos que los atraviesan, lo que hace que sea una solución difícil de administrar y no escalable. Por otro lado, los mecanismos basados en *Differentiated Services* (DiffServ) [Nic98], [Bla98] utilizan el campo TOS *(Type of Service)* para clasificar el tráfico y establecen unas normas generales de cómo debe comportarse cada nodo ante un tipo de tráfico determinado *(Per-Hop Behaviour*, PHB). Los principales problemas de esta arquitectura son que se requiere un mapeo previo y estático entre las aplicaciones y las clases de servicio y que las RFC *(Request for Comments)* que lo normalizan dejan muchas opciones abiertas a la implementación.

Los usuarios de estos sistemas esperan encontrar una Calidad de Servicio similar a la que proporciona RTC. En VoIP, que es un servicio en tiempo real, el retardo de los paquetes es uno de los parámetros que más afecta a la calidad de las llamadas. Por ello la ITU *(International Telecommunication Union*, Unión Internacional de Telecomunicaciones) recomienda un retardo en un sentido (OWD) máximo de 150 ms en la Recomendación G.114 [ITU96]. En cuanto a otros parámetros de QoS, en la actualidad existen diversos métodos para su mejora, que pueden actuar en el plano de información o el plano de control [CWXL+03].

El Control de Admisión de Llamadas (*Call Admission Control*, CAC) [YA07] es un método que trabaja en el plano de control, y acepta o rechaza llamadas en función de los parámetros de QoS en cada momento, dependiendo del estado de la red. El paradigma de aceptación de una nueva petición consiste en que, al aceptarla, las demás llamadas en curso no se vean afectadas viendo degradada su calidad, aumentando las pérdidas de paquetes y los retardos [WMXZ06].

Diferentes sistemas CAC ya han sido ampliamente utilizados en otras tecnologías de red, como las redes móviles o ATM [MH02]. De hecho, en la actualidad, el CAC constituye uno de los elementos fundamentales en las arquitecturas de QoS de los organismos de estandarización de las redes de nueva generación, como 3GPP (*3rd Generation Partnership Project*), WiMAX Forum (*Worldwide Interoperability for Microwave Access)* y TISPAN *(Telecommunications and Internet converged Services and Protocols for Advanced Networking)*. Sus recomendaciones definen un elemento central para gestionar la aplicación de políticas de QoS y la reserva de recursos.

Los sistemas CAC se pueden clasificar en dos categorías principales [SCV07]: basados en parámetros y basados en medidas. Los primeros requieren unos parámetros para regir su funcionamiento, como por ejemplo el número máximo de llamadas simultáneas. Para obtener estos parámetros requieren la realización de una serie de medidas durante la puesta en marcha del sistema. En el caso de Cisco, se denominan [WMXZ06] *Site-Utilization-Based* CAC (SU-CAC); en el momento de la configuración del sistema, reservan un ancho de banda en el *host* o en la red para llamadas de VoIP. Por otra parte, este fabricante también define los *Link-Utilization-Based* CAC (LU-CAC), que basan sus decisiones en la utilización del ancho de banda individual, permitiendo el multiplexado a nivel de enlace, pero añadiendo mucha complejidad y usando protocolos de reserva de recursos como RSVP (*Resource ReSerVation Protocol*) [Bra97].

Los sistemas CAC basados en medidas se denominan también MBAC *(Measurement Based Admission Control)*. Actualmente son utilizados en algunas soluciones comerciales [Cis01A], pero están limitadas a los equipos del fabricante. En el caso de Cisco, hay dos sistemas MBAC que funcionan para SIP: *AVBO* y *PSTN Fallback* [Cis01B]. Estos sistemas utilizan un elemento en cada sucursal, denominado *Call-Manager,* que interactúa con el *Gatekeeper,* situado en el nodo central. Para que un esquema de MBAC sea funcional, debe cumplir las siguientes condiciones [MH02], [GT03]:

- Asegurar que se alcanza el nivel de QoS deseado (precisión).

- Maximizar la utilización de los recursos.

- Equiparar los costes de implementación y los beneficios.

Continuamente van apareciendo diferentes propuestas sobre sistemas MBAC para tráfico en tiempo real, tanto de voz como de vídeo. Recientemente, en [WKKG06] se definió un sistema CAC que trataba de mantener altos los parámetros de QoS en una red mallada inalámbrica, con un servicio de VoIP. En [CGS05] se presentó un algoritmo predictivo autorregresivo, aplicado a información de vídeo, en el que se basan las decisiones de CAC.

En todo caso, la implementación de un MBAC requiere el uso de herramientas de estimación y monitorización de parámetros de QoS [GWCC+07]. Existen herramientas como *nettimer*, *pathchar*, *clink*, así como las bien conocidas *ping* y *traceroute*, que sirven para caracterizar diversos parámetros de una red: retardo, variación del retardo *(jitter),* ancho de banda máximo, ancho de banda disponible y tasa de pérdidas. Según el tipo de herramienta de medida, tendremos los distintos tipos de MBAC [BJS00], [JENN04]. Estas herramientas de medida pueden clasificarse en dos grandes grupos: extremo a extremo *(end-to-end)* y centralizadas. Las primeras se basan en la obtención de datos desde los extremos de la red, sin preocuparse por su estructura interna. Por el contrario, las segundas utilizan información obtenida dentro de la propia red, como es la estadística de los *router*, para cuantificar los parámetros de QoS. En el caso de no tener control sobre la red, las medidas deben realizarse extremo a extremo. Asimismo, las herramientas pueden dividirse en activas [IK01] y pasivas [CK00]. Las activas se basan en analizar paquetes de prueba introducidos deliberadamente en la red. Las herramientas pasivas, por contra, se basan en la captura de paquetes ya existentes en la red y en su análisis en tiempo real *(online)* o posteriormente *(offline).*

Por último, se puede añadir que una mejora para el CAC es buscar la mejor ruta (en cuanto a parámetros de QoS y costes) para el establecimiento de las conexiones, teniendo en cuenta que pueden existir diversas ubicaciones disponibles desde las que establecer una llamada a RTC.

SIP: Session Initiation Protocol

En el ámbito de la VoIP existen gran variedad de protocolos y configuraciones posibles, tanto para la información multimedia como para la señalización. Aunque para la transmisión multimedia se ha adoptado como estándar el protocolo RTP *(Real-Time Protocol),* en el ámbito de la señalización y el establecimiento de la llamada se dispone de varias opciones como SIP *(Session*

Initiation Protocol) [RSCP+02], H.323, IAX *(Inter-Asterisk eXchange protocol)* o MGCP *(Media Gateway Control Protocol),* o bien soluciones propietarias. Aunque las soluciones de Cisco, muy extendidas en empresas, suelen utilizar H.323 [Ale02], el análisis desarrollado para telefonía IP en este trabajo se ha centrado en SIP porque se trata de un protocolo abierto, sencillo y muy usado en la actualidad en redes IP [Zav08].

SIP ha sido adoptado por el 3GPP como el protocolo de señalización para IMS (*IP Multimedia Subsystem)* [3GPP06]. Otra ventaja de SIP es que no sólo puede utilizarse para administrar sesiones de VoIP, sino que también puede emplearse para otros servicios. Como veremos, existen centralitas *software* que lo utilizan, y permiten que el sistema CAC pueda ser fácilmente integrado.

SIP es un protocolo de nivel de aplicación basado en texto, que incorpora muchos elementos de HTTP y de SMTP (Fig. 2.1). Utiliza el método petición-respuesta de HTTP, y al incorporar muchas de sus funcionalidades, es un protocolo legible. SIP solamente se encarga de la señalización. Utiliza el protocolo SDP *(Session Description Protocol)* para establecer los parámetros de los flujos multimedia que se utilizarán en la sesión, que luego tiene lugar mediante el protocolo RTP.

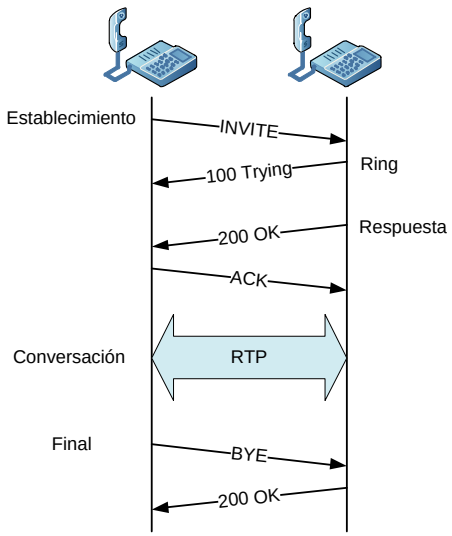


Fig. 2.1. Esquema de una llamada SIP

La recomendación RFC 3261, que define SIP, incluye el concepto de *proxy* SIP, un elemento que puede concentrar o redirigir tráfico, añadiendo escalabilidad, porque permite transferir carga de trabajo del núcleo a los bordes de la red.

Una centralita por software proporciona las funcionalidades de una centralita tradicional, pero, en lugar de ser un dispositivo específico, se encuentra incluida como un proceso dentro de una máquina. Suele resultar una solución más económica que el recurso a una centralita tradicional, resultando más flexible, porque se le pueden añadir funcionalidades mediante la instalación o activación de nuevos módulos. Una de las centralitas *software* más populares es Asterisk, creada en 1999 por Mark Spencer, de la empresa Digium. Estas centralitas admiten una gran variedad de protocolos, tanto de señalización como de tráfico de voz o vídeo.

Para el tráfico de señalización, como es el caso de SIP, se debe usar un sistema centralizado, ya que la PBX actúa como *"Back to back user agent"*, es decir, une dos llamadas: la primera desde el origen hasta la PBX, y otra desde la centralita hasta el destino (Fig. 2.2). Pero por otro lado, el tráfico del servicio de tiempo real se puede configurar de dos maneras: o bien pasando por la PBX (Fig. 2.2 b), o bien

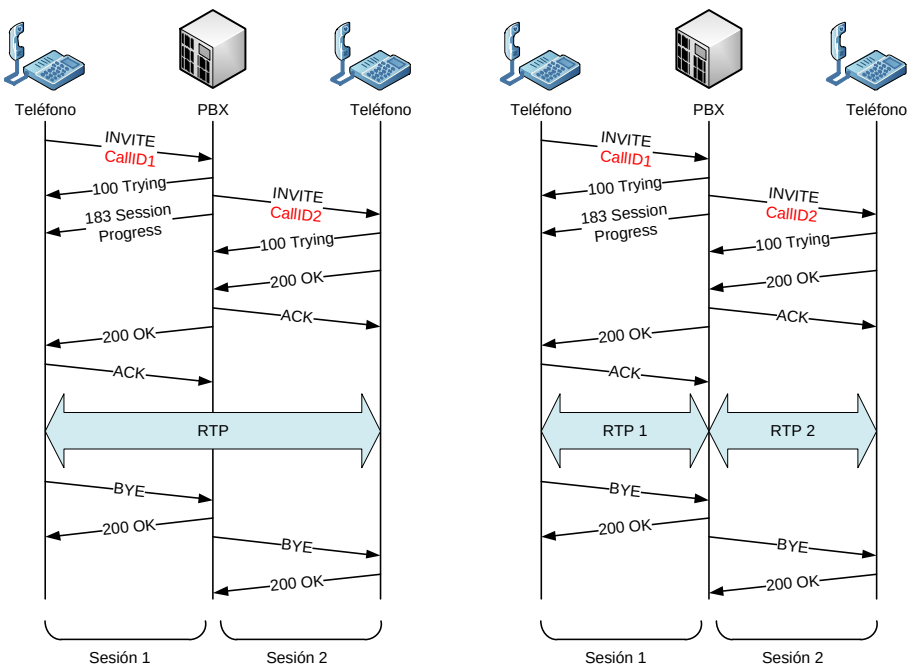


Fig. 2.2. Esquema de una llamada SIP a través de una centralita: a) el tráfico RTP va del origen al destino; b) el tráfico RTP pasa por la centralita

usando una topología en estrella (Fig. 2.2 a), de modo que el tráfico vaya directamente desde una máquina a otra. Esta topología evita los retardos que aparecerían si ese tráfico tuviera que pasar por la PBX. Si se usa una topología centralizada, se pueden aprovechar otras ventajas, como por ejemplo la posibilidad de cambiar el *codec (transcoding),* usando uno diferente en el terminal origen y destino para optimizar los recursos de QoS.

Juegos *online*

Los juegos *online* son un servicio que crece día a día en Internet. Algunos títulos tienen millones de usuarios, y por eso las empresas desarrolladoras se enfrentan a un difícil problema cada vez que lanzan un nuevo juego: necesitan recursos *hardware* y de red para evitar que su infraestructura se sature. Dado que el éxito de un nuevo título no es muy predecible, en ocasiones se puede recurrir al sobredimensionado de los recursos para dar un buen servicio a los usuarios. De hecho, en [CFSS05] se presentó un estudio del comportamiento de los jugadores *online*, y los autores llegaron a la conclusión de que son muy difíciles de satisfacer: si encuentran problemas, suelen abandonar ese servidor, y tienden a variar mucho sus preferencias.

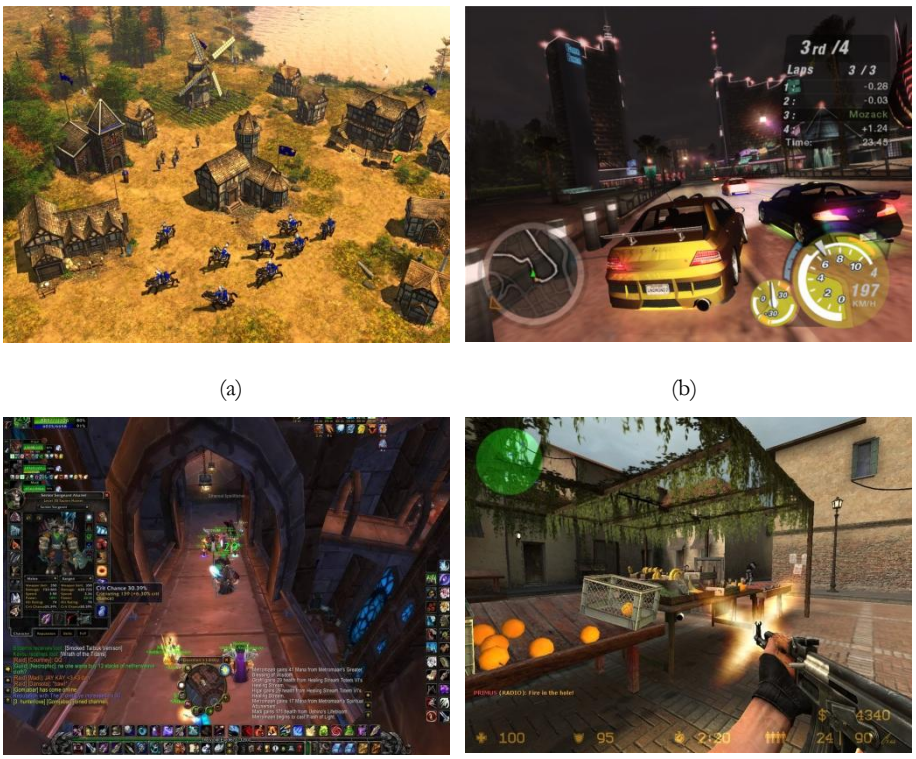
Clasificación

Algunos juegos *online* presentan unos requerimientos de tiempo real muy estrictos, similares en parte a los de VoIP. El comportamiento de los usuarios es muy exigente, y son muy sensibles al retardo [CFSS05]. Este hecho hace que las empresas que proporcionan este servicio se enfrenten a un problema a la hora de dimensionar los recursos a dedicar para el soporte del juego. Entre estos recursos está el ancho de banda, y también el número de paquetes por segundo que los elementos de la red deben ser capaces de gestionar. De hecho, los juegos no usan grandes cantidades de ancho de banda, ya que suelen generar altas tasas de paquetes pequeños.

Algunos de los géneros que permiten jugar en red son los denominados RTS *(Real Time Strategy,* Estrategia en Tiempo Real) (Fig. 2.3 a), en los que existe un escenario virtual en el que el jugador tiene que gestionar recursos como ejércitos, edificios, etc. También existen simuladores de diferentes deportes, y entre ellos destacan los relacionados con el mundo del motor y las carreras (Fig. 2.3.b).

Dos de los géneros más populares de juegos en red son los MMORPG *(Massive Multiplayer Online Role Playing Game,* Juegos Masivos de Rol Multijugador Online) y los FPS *(First Person Shooter,* Tirador en Primera Persona). Los primeros crean un

mundo virtual en el que miles de personas pueden jugar simultáneamente. Cada jugador maneja a un personaje, que puede obtener diferentes habilidades y poderes (Fig. 2.3 c). Este género requiere fiabilidad, pero no una gran interactividad, ya que las luchas no están basadas en disparos, sino en los poderes de cada personaje. Por esta razón, estos juegos utilizan principalmente TCP [CHL05], [SKR07]. La duración de las sesiones suele ser de varias horas [FCFW05]. En [CHL05] se estudió el tráfico de los MMORPG, llegando a la conclusión de que tienen algunas características como la periodicidad y la autosimilitud. Otra conclusión de dicho estudio es que estos juegos presentan unos requerimientos de ancho de banda y tiempo real menores que los FPS.



(a) (b)

(c) (d)

Fig. 2.3: Capturas de pantalla de juegos: a) RTS *(Age of Empires III)*; b) Deportivo *(Need for Speed 2)*; c) MMORPG *(World of Warcraft)*; d) FPS *(Counter Strike)*

Lo habitual en los juegos FPS (Fig. 2.3 d) es que participen en la partida unas decenas de jugadores, que comparten un escenario virtual, donde tienen que eliminar a los enemigos o lograr un objetivo. Cada usuario tiene un arma, que se puede mejorar conforme a los resultados del juego. La duración de una partida

tiende a ser breve, pero lo normal es jugar unas cuantas rondas en la misma sesión. Los requerimientos de tiempo real resultan muy estrictos, ya que los movimientos y disparos son rápidos y frecuentes. Por esta razón, este género de juegos suele usar el protocolo UDP [FCFW05]. Estos juegos suelen estar pensados para funcionar en PC de gama alta, o en consolas, ya que se requieren tarjetas gráficas muy rápidas.

Como veremos, los juegos FPS comerciales utilizan arquitecturas cliente-servidor. Cada vez que se lanza un nuevo título, el proveedor debe preparar una infraestructura para darle soporte, lo que implica la puesta en marcha de servidores con suficiente capacidad de proceso, y de redes con gran ancho de banda. Por eso, el servidor puede ser un cuello de botella que introduzca una limitación en el número simultáneo de jugadores, a no ser que previamente se hayan sobredimensionado los recursos.

Algunos trabajos han mostrado que los jugadores son un tipo de usuario muy difícil de contentar: en el estudio presentado en [CFSS05] se observó que no tienden a ser leales a un servidor, y tienen muy poca paciencia: si un servidor no funciona correctamente, cambian a otro y no vuelven al anterior. Otro problema que debe resolver el proveedor es la "injusticia" que puede aparecer cuando unos jugadores tienen menos retardo que otros. Una posible técnica para mitigar este problema es incrementar artificialmente el retardo a algunos jugadores, de forma que todos los retardos se igualen.

En los juegos FPS las acciones de los jugadores se deben propagar al servidor y al resto de jugadores en muy poco tiempo, por lo que los retardos de red son muy críticos. Estos juegos producen altas tasas de paquetes UDP de pequeño tamaño (algunas decenas de bytes) desde el cliente al servidor, y por eso el *overhead* causado por las cabeceras IP y UDP es elevado. Por el contrario, los paquetes del servidor al cliente son habitualmente más grandes.

Aunque esta técnica también se puede aplicar a otros géneros, en esta tesis nos centraremos en los juegos FPS, a causa de sus grandes requerimientos de interactividad. La calidad subjetiva depende fundamentalmente del retardo y las pérdidas de paquetes [ZA04]. El tiempo de respuesta del sistema *(System Response Time, SRT*), que se define como el tiempo necesario para detectar un evento del usuario, procesarlo en el servidor actualizando el estado del juego, y presentarlo en el dispositivo de salida correspondiente, debe mantenerse por debajo de unos determinados valores.

Tráfico que generan los FPS

En la literatura se pueden encontrar muchos trabajos sobre el tráfico que generan los juegos *online*. En este trabajo estudiaremos el tráfico activo del juego, que se genera una vez comenzada la partida. Este tráfico presenta dos comportamientos diferentes: por un lado, la aplicación cliente se encarga de comunicar las acciones de los jugadores al servidor, usando para ello paquetes pequeños con una frecuencia alta. Por otro lado, el servidor calcula el nuevo estado del juego y se lo envía a todos los jugadores, usando paquetes más grandes, cuyo tamaño depende del número de jugadores. En [BA06] se presentó un método para extrapolar el tráfico del servidor al cliente, obtenido a partir de medidas empíricas. A partir de las trazas del tráfico de partidas de 2 o 3 jugadores, los autores de este estudio obtuvieron las distribuciones para una partida de *N* jugadores, que se pueden usar para posteriores investigaciones.

En [FCFW02] se analizó una traza de 500 millones de paquetes de un servidor de *Counter Strike*, y a partir de ese análisis se concluyó que el juego está diseñado para saturar el cuello de botella que constituye la red de acceso. Por otro lado, en [FCFW05] se analizaron otros juegos en términos de tamaño de paquete y tiempo entre paquetes. En [RHS10] se presentó un resumen de diferentes modelos de tráfico que existen en la literatura para 17 juegos FPS comerciales. Los estudios citados muestran que estos juegos generan altas tasas de paquetes pequeños, obteniendo una eficiencia muy pobre. Este hecho nos da pie a plantearnos el ahorro de ancho de banda mediante la compresión de cabeceras y la multiplexión de paquetes.

En [FCFW02] también se dice que el cuello de botella no es sólo el ancho de banda de la red de acceso, sino el número de paquetes por segundo que el *router* puede gestionar. Los *router* están diseñados frecuentemente para paquetes grandes, y pueden experimentar problemas al gestionar ráfagas con un gran número de paquetes pequeños.

Infraestructura para soporte de juegos online

Desde el punto de vista del servidor, existen dos posibles arquitecturas para dar soporte a este servicio: centralizadas y distribuidas. En las primeras existe un servidor que mantiene el estado del juego y lo distribuye a los jugadores. El problema que presentan es que el servidor constituye un cuello de botella. En las arquitecturas distribuidas [GD98] no se necesita un servidor central, ya que los jugadores se intercambian directamente la información, pero esta arquitectura no suele usarse en juegos comerciales por los siguientes motivos: la mejor gestión de

la sincronización, la facilidad para mantener la consistencia del juego entre los participantes, evitar las trampas y, sobre todo, razones comerciales, ya que de este modo se puede cobrar por el servicio o vender el *software* servidor del juego.

La escalabilidad de la infraestructura para soportar estos juegos ha sido estudiada por Mauve y otros [MFW02], que propusieron el uso de *proxy* para conseguir control de congestión, robustez, reducción de los retardos y evitar las trampas de algunos jugadores. Algunos *proxy* podrían situarse cerca de los jugadores, ahorrando trabajo al servidor central. Así, en la Ref. [BRS02] también se propuso el uso de *booster-box,* que se podrían situar cerca del *router*, para así conocer el estado de la red, y ser capaces de dar soporte de red a las aplicaciones.

Por último, desde el punto de vista del usuario, en [LKC04] se presentó un algoritmo para permitir que el cliente seleccione adaptativamente el mejor servidor para un juego concreto. Esto podría permitir a un grupo de usuarios jugar en el mismo servidor, y así poder usar técnicas de multiplexión para ahorrar ancho de banda y reducir la cantidad de paquetes por segundo. Es la solución que plantearemos en este trabajo.

Capítulo 3

CALIDAD DE SERVICIO

Definición de Calidad de Servicio y Calidad de la Experiencia

Como se ha explicado en el capítulo anterior, Internet fue diseñada como una red *best-effort*, es decir, la red no puede garantizar un retardo acotado en la entrega de los paquetes. Sin embargo, en los últimos años, está siendo ampliamente utilizada para servicios interactivos en tiempo real, como VoIP, videoconferencias, servicios de telemedicina o juegos *online*.

Además los usuarios están acostumbrados a servicios de conmutación de circuitos, como por ejemplo la telefonía tradicional, en los que disponen de un canal exclusivo para el tráfico de su llamada, lo que proporciona una calidad constante garantizada.

Por otro lado, el despliegue de las redes de datos empresariales en las últimas décadas ha llevado a una situación en la que coincidían en los mismos lugares una red de telefonía y una red de datos. Lógicamente, la convergencia de ambas permite reducir costes de instalación y mantenimiento, lo que ha llevado al uso de redes de conmutación de paquetes para proporcionar servicios de telefonía.

Pero esta convergencia implica el desarrollo de mecanismos que aseguren una calidad mínima, o de lo contrario los usuarios, acostumbrados a la calidad de los anteriores servicios, rechazarán este cambio.

En este contexto, se ha acuñado el concepto de Calidad de Servicio, frecuentemente denominada con las siglas QoS, que corresponden a *Quality of Service,* para englobar las diferentes métricas que nos dan una idea del servicio que estamos dando al usuario final. La QoS se entiende como algo objetivo y, por tanto, medible, que no depende de la experiencia subjetiva del usuario.

Por otro lado, en los últimos años ha aparecido otro concepto, más referido a la experiencia que tiene el usuario cuando usa un servicio: la Calidad de la Experiencia, también denominada QoE, del Inglés *Quality of Experience.* Este concepto engloba también las experiencias del usuario del servicio, y es por tanto más subjetivo y difícil de medir.

Veremos en primer lugar los parámetros objetivos que se pueden medir a partir del tráfico de la red y posteriormente estudiaremos cómo esos parámetros se pueden integrar en un solo indicador que nos dará una idea de la calidad.

Parámetros de red que determinan la Calidad de Servicio

En este apartado veremos los principales parámetros, que son medibles en la red, y determinan la calidad que se presta al usuario de un servicio. Aunque existen otros parámetros que también se pueden medir, como el ancho de banda disponible, nos centraremos en el retardo, las pérdidas y el *jitter*, por ser los que vamos a utilizar posteriormente en las medidas a lo largo de esta tesis.

Retardo

Es el tiempo que invierte el paquete en viajar desde el origen hasta el destino. El origen y el destino dependen del servicio: para la voz, por ejemplo, el retardo total es el de boca-a-oído. Para los juegos *online* se puede considerar el tiempo desde que el usuario realiza una acción hasta que su efecto aparece en su dispositivo de visualización.

El retardo tiene una importancia fundamental en el caso de servicios interactivos. Se suelen distinguir dos formas fundamentales de medirlo: el retardo en un sentido *(One Way Delay,* OWD*),* y el retardo de ida y vuelta *(Round Trip Time,* RTT*)* [CR00].

El OWD se puede medir sin ningún problema en el caso de utilizar entornos de laboratorio en los que toda la información sobre los eventos está disponible para el usuario. También se puede medir correctamente cuando los relojes de las máquinas origen y destino están sincronizados.

Pero en el caso de medidas con máquinas reales, muchas veces nos ocurre que se encuentran alejadas, y por tanto no es fácil sincronizarlas. Existen protocolos de sincronización como NTP *(Network Time Protocol)* [MMBK10], pero con frecuencia su precisión no es suficiente para las medidas que se requieren en el caso de servicios interactivos. Se puede recurrir al uso de GPS *(Global Positioning System)* para sincronizarlos, pero resulta una solución cara.

En muchos casos se utiliza como medida el RTT, que es el tiempo que necesita un paquete para hacer el camino de ida y vuelta, y resulta útil para conocer el retardo boca-a-oído, pues en muchos casos [CR00] se estima el OWD como la mitad del RTT. De esta manera, los tiempos de envío y recepción se miden con el reloj de la máquina origen.

El retardo en servicios multimedia tiene diversos componentes, que se explicarán más detenidamente en los siguientes capítulos, cuando se presenten las medidas de cada servicio. En este capítulo introductorio nos limitaremos a enumerarlos, y se explicarán más detenidamente cuando hablemos de la realización de las pruebas: paquetización, tiempo de transmisión *(store and forward)*, tiempo de retención en el multiplexor, transmisión en la red, retardo en el *buffer* del *router*, tiempos de procesado, y retardo introducido por el *buffer* de *dejitter*.

Pérdidas de paquetes

Las pérdidas de paquetes son una de las principales causas del descenso de la calidad en redes IP.

En los primeros años, Internet estaba diseñada para gestionar paquetes grandes, pues los servicios no tenían requerimientos de tiempo real y, por tanto, lo mejor era maximizar el tamaño del paquete, para que la cabecera fuera compartida por más bytes de información, disminuyendo así el *overhead*. Los *router* de Internet tienen un límite en términos de ancho de banda, que hace que el *buffer* se llene, pero también se ven limitados por el número de paquetes por segundo que pueden gestionar [FCFW02], [YA07]. La causa es que muchos *router* estaban pensados inicialmente para manejar paquetes grandes [FCFW05], y pueden experimentar problemas para enviar altas tasas de paquetes pequeños, como los generados por algunas de las aplicaciones de tiempo real. Por ello, en redes cableadas una causa importante de pérdidas es el descarte en los *buffer* de los *router* [WCL07].

En [BSUB98] se estudiaron las pérdidas de paquetes, enviando tráfico real. Se encontró que muchas de las pérdidas se producían a ráfagas. Unas ráfagas eran cortas, de hasta un segundo, y su causa principal era el descarte en los *buffer* de los *router*. Otras ráfagas más largas, de hasta diez segundos, se debían al reinicio o mantenimiento de los *router*.

Los servicios basados en TCP pueden utilizar la retransmisión para recuperar esos paquetes, pero las aplicaciones de tiempo real, que suelen usar UDP, se pueden ver afectadas seriamente, pues su interactividad hace que no tenga sentido esperar al paquete retransmitido.

Existe una relación entre el tamaño de los paquetes y la probabilidad de pérdidas: en redes inalámbricas, según aumenta el tiempo de transmisión, la probabilidad de que algún bit se corrompa aumenta. Esta relación ha sido estudiada por

ejemplo en [KW05]. Por otra parte, en [DFK06] se presentó un estudio analítico para *buffer* de tamaño limitado en redes cableadas.

Algunos estudios [WCL07], [BG98], [BSUB98] han caracterizado el comportamiento de la Internet pública en términos de pérdidas de paquetes. Estas caracterizaciones pueden ser interesantes a la hora de pensar el mejor modo de transportar nuestro servicio a través de la red. Si observamos, por ejemplo, que los paquetes grandes se ven penalizados por nuestro *router* o por la red, quizá no nos interese emplear técnicas que aumenten el tamaño de los paquetes, pues esto aumentaría la probabilidad de descarte.

Este efecto será particularmente interesante en esta tesis, pues encontraremos un compromiso: al multiplexar, por un lado reducimos el ancho de banda requerido por nuestro servicio; pero por otro lado, al aumentar el tamaño del paquete, hay determinadas políticas de *buffer* que hacen que aumenten las pérdidas. Por tanto, habrá que estudiar en cada caso cuál es la opción que proporciona mayor calidad al usuario.

Variación del retardo (jitter)

En general, cuando se habla de *jitter* en el entorno de comunicaciones en redes de ordenadores, nos estamos refiriendo a la variación entre el retardo de unos paquetes con respecto a otros. Las redes de paquetes pueden retardar de manera diferente los distintos paquetes de un mismo flujo, y eso produce problemas.

El RFC 3393 [DC02] define *Instantaneous Packet Delay Variation* (Variación instantánea del retardo del paquete, *IPDV)* para un par de paquetes del mismo flujo como "*the difference between the one-way-delay of the selected packets*", es decir, la diferencia entre el retardo en un sentido (OWD) de dos paquetes (Fig. 3.1). Ese documento evita denominar *jitter* a esta medida, porque esta palabra tiene otros dos significados: la variación de una señal respecto a otra señal "reloj", o la variación de una métrica respecto a una métrica de referencia.

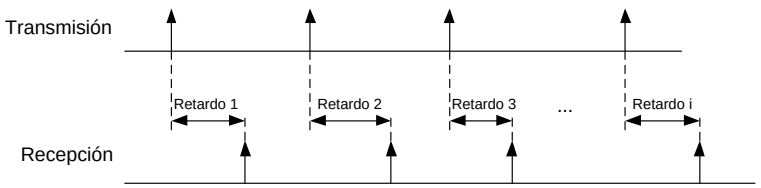


Fig. 3.1. Diferentes retardos para paquetes del mismo flujo

Por tanto, para calcular el IPDV se usaría la fórmula:

$$IPDV = \frac{\sum \left| retardo_i - retardo_{i-1} \right|}{n-1} \tag{3.1}$$

Se puede observar que para calcular su valor es necesario conocer el instante de envío y recepción de cada paquete. Pero puede ocurrir que en recepción no tengamos información sobre el instante en que se ha enviado cada paquete, o que los relojes del emisor y receptor no estén sincronizados. Por eso en ocasiones se utilizan o denominan *jitter* otras magnitudes que dan una idea de la variación del retardo, sin ser exactamente el IPDV. Por ejemplo, en ocasiones se usa como *jitter* sencillamente la varianza o la desviación estándar del retardo.

Otra forma de medir el *jitter* la podemos encontrar en el Apéndice A.8. del RFC 3550 [SCFV03], que estandariza el protocolo RTP, y en su predecesor RFC 1889, que presentan un algoritmo para estimar el *jitter*, ya que su valor se debe incluir en algunos campos usados por el protocolo. El *jitter* se estima para cada nuevo paquete que llega, a partir del valor estimado anteriormente, que se corrige en función de la diferencia entre retardos de llegada. El parámetro se calcula así:

$$s_{jitter} \mathrel{+}= (1./16.) * ((\text{double})d - s_{jitter})$$

Donde s_{jitter} es la nueva estimación del *jitter*. El parámetro *d* representa la diferencia entre los retardos de dos paquetes consecutivos. Lo que significa que la nueva estimación del *jitter* será la suma de su valor actual más 1/16 de la diferencia entre *d* y el valor actual:

$$jitter_{i+1} \approx jitter_i + \frac{1}{16}(d - jitter_i) \tag{3.2}$$

De este modo, el valor estimado del *jitter* se va actualizando dinámicamente y se puede incluir en el campo correspondiente del protocolo, para poder tenerlo en cuenta a la hora de controlar el envío de tráfico.

En el entorno de VoIP, los *codec* están diseñados para enviar las muestras con una frecuencia determinada. Lógicamente, el receptor debe decodificarlas y reproducirlas con la misma frecuencia. Por efecto del *jitter* es posible que, llegado el momento de reproducir una muestra, no hayamos recibido todavía el paquete correspondiente. Por eso, estas aplicaciones utilizan los denominados "*buffer* de *dejitter*" o también "*playout buffer*", que funcionan acumulando varias muestras en

una cola, y reproduciéndolas a la frecuencia correspondiente. El problema es que añaden un retardo debido al tiempo en la cola. También añaden pérdidas, porque los paquetes que llegan demasiado tarde ya no podrán reproducirse y se deberán descartar. Vemos por tanto que hay un compromiso: a mayor tamaño de este *buffer*, menor será la probabilidad de descarte, pero mayor será el retardo añadido. En servicios interactivos como es VoIP no podemos agrandar indefinidamente este *buffer*, porque el retardo es crítico.

En el presente trabajo evitaremos ceñirnos a una implementación concreta del *buffer* de *dejitter*. Por eso, utilizaremos una aproximación para obtener el valor de las pérdidas que ocasiona este *buffer*, que es la presentada en [CR00]:

$$Pérdidas_{dejitter} \approx \Pr (o > bg) \qquad (3.3)$$

Donde o es la diferencia entre el retardo en un sentido de dos paquetes consecutivos, b es la mitad del tamaño del *buffer* y g representa el tiempo entre paquetes. Esta aproximación supone un *buffer* de *dejitter* de tamaño fijo. Si se utilizaran esquemas adaptativos, se podrían conseguir mejores resultados, pero para obtener resultados independientes de la implementación, hemos optado por esta aproximación.

Factor R y MOS

Existen estimadores que integran los valores de los parámetros de red, para reducirlos a un solo número que nos dé una idea de la calidad obtenida. En este apartado veremos algunos de ellos. Lo hemos dividido en dos partes: en primer lugar explicaremos el uso de estos estimadores para calcular la calidad del servicio de voz y posteriormente explicaremos cómo este estimador también se ha adaptado para su uso en juegos *online*.

Uso inicial para voz

La norma G.107 [ITU03] de la ITU presenta el E-Model, una herramienta para la planificación y diseño de redes de conmutación de paquetes que deben dar soporte a aplicaciones de voz. Una herramienta muy utilizada para estimar la calidad es el uso del denominado MOS *(Mean Opinion Score,* que podríamos traducir como "puntuación de la opinión media"). Mediante la realización de encuestas a usuarios en distintas situaciones, se obtienen unos parámetros de calidad percibida, a la que los usuarios asignan una puntuación, que se asocian con los parámetros de red en cada caso.

La herramienta estima los inconvenientes que producen en la calidad de la voz los distintos equipos al modificar los parámetros de la red, y así provee una manera de estimar el MOS para la calidad de la voz. Un resultado del E-Model es el cálculo del Factor R, una medida sencilla de la calidad de la conversación, que varía desde 0 (el peor caso) hasta 100 (la máxima calidad). Normalmente se considera que la calidad es aceptable para valores de R por encima de 70. El Factor R depende de varios parámetros, asociados al canal de voz, como son el eco, el ruido de fondo, las pérdidas, las imprecisiones causadas por el *codec*. El resultado obtenido para el Factor R se puede convertir fácilmente en MOS, que varía desde 1 (mala calidad) hasta 5 (muy buena). La fórmula para obtener el valor del MOS a partir del Factor R es:

$$\begin{aligned} &Si\ R < 0 && MOS = 1 \\ &Si\ R > 100 && MOS = 4{,}5 \\ &Si\ 0 < R < 100 && MOS = 1 + 0{,}035\ R + 7e\text{-}6\ R(R\text{-}60)(100\text{-}R) \end{aligned} \tag{3.4}$$

El Factor R se expresa como la suma de cuatro términos:

$$R = 100 - I_s - I_d - I_{ef} + A \tag{3.5}$$

Donde I_s se asocia a los inconvenientes causados por la conmutación de circuitos, I_d está asociado al retardo boca-a-oído, I_{ef} se debe al equipamiento, es decir, está asociado, entre otros factores, a las pérdidas de calidad del *codec* y a los paquetes perdidos, y A es el factor debido a las expectativas (*Expectation Factor)*, que intenta agrupar las cantidades intangibles más difíciles de cuantificar, como las expectativas del usuario respecto al servicio.

Vemos que los términos I_s, I_d e I_{ef} son aditivos, y también vemos que las contribuciones debidas al retardo se incluyen en I_d, mientras que las causadas por las pérdidas están en I_{ef}, por lo que ambas están separadas. Esto no significa que estén incorreladas, pero sí que sus contribuciones a la pérdida de calidad se pueden estudiar por separado.

Algunos de estos factores pueden simplificarse [CR00], utilizando valores estándar que la propia norma propone. De esta manera, la ecuación 3.5 puede reducirse, asumiendo fijos los valores de I_s y de A. Como en este trabajo vamos a preocuparnos de los problemas debidos a la red, la expresión de R quedaría:

$$R = 94{,}2 - I_d - I_{ef} \tag{3.6}$$

La expresión analítica para I_d dada por la norma G.107 depende de tres retardos diferentes: el OWD absoluto boca-a-oído; el OWD medio desde el receptor hasta el punto donde tiene lugar el acoplamiento de la señal, que puede ser fuente de eco; y el RTT medio. Seguiremos en este trabajo la aproximación de [CR00], que en el caso de VoIP obtiene una aproximación de estos tres retardos con una sola medida, la del OWD. Así se obtiene una expresión analítica simplificada para I_d:

$$I_d = 0{,}024\, d + 0{,}11\, (d - 177{,}3)\, H(d - 177{,}3) \qquad (3.7)$$

Donde d es el OWD en milisegundos, y $H(x)$ es la función escalón:

$$H(x) = 0 \quad \text{para } x < 0$$
$$H(x) = 1 \quad \text{para } x \geq 0 \qquad (3.8)$$

Si representamos la expresión 3.7 (Fig. 3.2), vemos que hay dos zonas prácticamente lineales: hasta el valor de 177,3 ms de retardo, el valor de I_d aumenta siguiendo una pendiente, mientras que a partir de ese valor, la pendiente es mayor. Por tanto, de esta expresión se puede deducir que, mientras el OWD se mantenga por debajo de ese valor, será más fácil obtener una calidad aceptable.

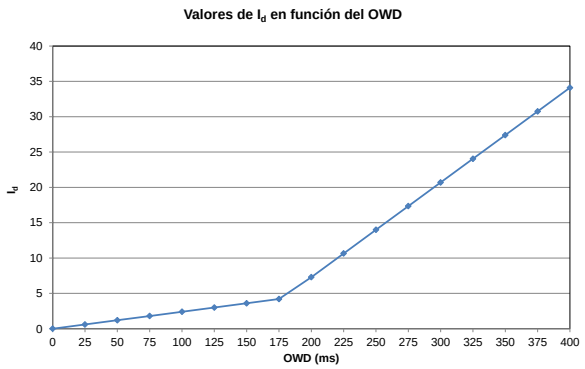


Fig. 3.2. Valores de I_d en función del retardo en un sentido

Si sustituimos el valor de I_d obtenido en 3.7 en la ecuación 3.6, obtenemos la expresión aproximada para el Factor R:

$$R \sim 94{,}2 - 0{,}024d + 0{,}11(d - 177{,}3)\, H(d - 177{,}3) - I_{ef} \qquad (3.9)$$

Así pues, falta por calcular el valor de I_{ef}. Este factor varía según el *codec*, por lo que se requieren estudios empíricos que incluyan pruebas subjetivas, para obtener fórmulas que se correspondan con su comportamiento. En [CR00] se presentan

las fórmulas para algunos de los *codec* más utilizados. En esta tesis se utilizará siempre el *codec* G.729a, para el que se ha obtenido la siguiente fórmula:

$$I_{ef}(G.729a) \sim 11 + 40\ ln(1 + 10\ e) \quad (3.10)$$

Donde *e* es la probabilidad de pérdidas total en tanto por uno.

Por tanto, tenemos ya una fórmula analítica para estimar el Factor R, basada en el OWD *(d)* y las pérdidas en tanto por uno *(e)*:

$$R \sim 94{,}2 - 0{,}024d + 0{,}11(d - 177{,}3)\ H(d - 177{,}3) - 11 + 40\ ln(1 + 10\ e) \quad (3.11)$$

Esta fórmula será utilizada para estimar la calidad, en las pruebas con tráfico de voz que se presentarán a lo largo de la tesis.

Adaptación para juegos online

El problema de obtener estimadores para la calidad subjetiva de los juegos *online* ha sido tratado en muchos trabajos. En [SERZ02] sólo se consideraba el efecto negativo del retardo, en términos de tiempo de respuesta del sistema, también denominado SRT *(System Response Time),* definido como el tiempo que se necesita para detectar un evento de un usuario, procesarlo y representar el nuevo estado del juego en el dispositivo de visualización. Se calculó una sencilla fórmula para el MOS, pero sólo dependía del SRT.

Algunas de las técnicas de adaptación a los problemas de red, que usan los desarrolladores de juegos para mantener una buena calidad percibida se explican en [OH03]. Entre ellos destacamos la predicción de los movimientos del usuario o la distorsión del tiempo entre los diversos jugadores.

Otro estudio [ZA04] llevó a cabo un conjunto de experimentos, usando un emulador para añadir retardos y pérdidas controlados en la red, pidiendo posteriormente a los usuarios que rellenaran unos cuestionarios sobre la calidad percibida. En este estudio se probaron dos juegos: *Halo* y *Quake III*. El trabajo estudió por separado el efecto del retardo y las pérdidas, por lo que no se elaboró una fórmula para el MOS. Algunas de las conclusiones fueron que el retardo tiene una mayor influencia que las pérdidas. Otro resultado interesante estaba relacionado con las pérdidas: mientras que *Halo* dejaba de funcionar a partir del 4% de pérdidas, *Quake III* podía funcionar incluso con un 35%. Esto implica que los distintos juegos implementan métodos diferentes para ocultar los inconvenientes de la red a los jugadores. Este estudio no consideraba el *jitter*

como un inconveniente, pues se decía que su efecto era significativamente menor que el del retardo.

En [DWW05] se realizó un conjunto de encuestas a jugadores, y también se evaluaron en la práctica cuatro juegos: dos FPS, un simulador deportivo y un RTS. Se estudió solamente el efecto del retardo y el *jitter*, dejando las pérdidas para trabajos futuros. Los resultados mostraron los diferentes efectos de los problemas de red en el MOS para cada juego.

El primer modelo de MOS para un juego FPS, adaptado del E-Model para voz explicado en el apartado anterior, fue presentado en [WKVA06]. El juego seleccionado fue *Quake IV*. Se mostró que este juego, al igual que sus predecesores, tiene un algoritmo muy efectivo para ocultar a los jugadores el efecto de las pérdidas de paquetes. Por eso sólo se consideró el retardo y el *jitter*. Se desarrolló una fórmula, basada en un polinomio, para el MOS, usando regresión multidimensional.

En [Ubi05] se presentó una fórmula que utiliza diferentes factores que multiplican al retardo y el *jitter* según cada juego. Este modelo también considera las pérdidas. Pero los valores de estos factores para cada juego no están públicamente disponibles, porque este trabajo se desarrolló dentro de una empresa.

Un análisis similar se llevó a cabo en [RSR08], y se desarrolló una fórmula para el MOS de un juego MMORPG *(World of Warcraft)*. Se probaron diferentes combinaciones de retardo y *jitter* usando un emulador, y se realizaron encuestas para evaluar la experiencia de varios jugadores durante el juego.

De todas formas, dado que estos estimadores sólo se pueden utilizar para títulos concretos, y que todavía no han alcanzado el alto grado de consenso que existe con el uso del E-Model para voz, no se utilizarán en esta tesis, sino que nos limitaremos a estudiar por separado el efecto de los diferentes inconvenientes de la red.

PROBLEMÁTICA DEL DIMENSIONADO DEL *BUFFER*

Este breve capítulo se dedicará a explicar el estado del arte en lo que se refiere al dimensionado del *buffer* del *router*. Se le dedica un capítulo independiente debido a que consideramos que es un tema con una entidad propia.

Se trata de un problema que afecta muy directamente al tráfico de servicios de tiempo real, puesto que en muchos casos los usuarios se conectan a Internet (Fig. 4.1) desde una red de acceso, como por ejemplo ADSL *(Asymmetric Digital Subscriber Line)* o módem de cable, y todo su tráfico atraviesa en primer lugar un *router*. Las características de este dispositivo pueden modificar sustancialmente los parámetros de calidad del servicio proporcionado al usuario.

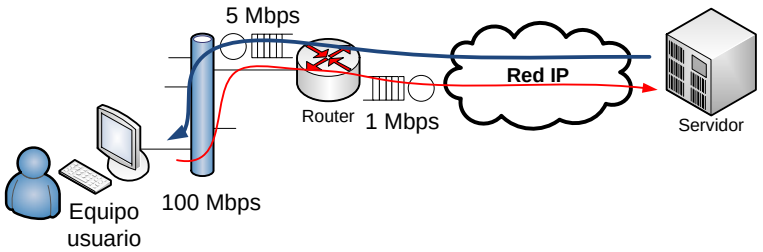


Fig. 4.1. Esquema de una red de acceso

Normalmente la red de acceso supone un cuello de botella para el tráfico del usuario. Puede ocurrir que en su domicilio disponga de una LAN de alta velocidad (por ejemplo, Ethernet a 100 Mbps o incluso a 1 Gbps), mientras que la velocidad de su acceso sea menor de 10 Mbps. Una vez que el tráfico llega a la red troncal, también ahí dispone de un mayor ancho de banda, por lo que el acceso es el cuello de botella que más limita al tráfico.

Por otra parte, muchas de las tecnologías de acceso, especialmente el ADSL, tienen un ancho de banda asimétrico (en la Fig. 4.1 se ha supuesto un ancho de banda de 1 Mbps en el *uplink* y de 5 Mbps en el *downlink)*, pues están diseñados para que el usuario descargue más bytes de la red de los que va a *subir*. Por eso, como vemos en la figura, el *buffer* del *uplink* del *router* tenderá a llenarse si el tráfico ofrecido es mayor que el del acceso. En cambio, el tráfico no tendrá problemas

en el otro sentido, pues pasará de un ancho de banda menor a otro mayor, que es el de la LAN del usuario.

Podemos ver por tanto que existe un compromiso, dado que el tráfico en Internet es a ráfagas: si este *buffer* es grande, al recibir una ráfaga podrá almacenar más paquetes, y la probabilidad de descartarlos será menor. Esto es interesante para servicios en los que prima la fiabilidad sobre la interactividad, como son el correo electrónico, la navegación *web,* la transferencia de ficheros, etc. Pero si se trata de servicios de tiempo real, un *buffer* con un tamaño excesivo provocará retardos elevados, que degradarán la calidad del servicio.

Esta problemática se puede mitigar con técnicas avanzadas de priorización de tráfico, pero en el escenario considerado lo habitual será encontrar un *router* de gama baja, con una cola FIFO fácil de implementar.

En los últimos años, el problema del dimensionado del *buffer* ha dado lugar a muchos estudios. Un buen resumen de la cuestión se puede encontrar en [VST09]. Aunque el problema se ha planteado fundamentalmente para *router* de alta gama, que se usan en la red troncal, tiene también implicaciones en los *router* de gama media y baja, que podemos encontrar en las redes de acceso comerciales.

La regla comúnmente aceptada para establecer el tamaño del *buffer* del *router* ha sido durante años el uso del producto del ancho de banda por el retardo de ida y vuelta [VS94]:

$$B = C \times RTT \tag{4.1}$$

Donde B es el tamaño del *buffer,* C es la capacidad del enlace y RTT es el retardo de ida y vuelta. El objetivo fundamental de esta regla era maximizar la utilización del enlace, intentando que alcanzase el 100%. Dimensionando así el *buffer* garantizamos que cuando se llene, y se empiecen por tanto a descartar paquetes, y TCP reaccione bajando la tasa de transmisión, tendremos suficientes paquetes almacenados como para mantener ocupado el canal mientras TCP va aumentando de nuevo la tasa de transmisión. Esta regla se obtuvo experimentalmente [VS94] usando 8 flujos TCP en un enlace de 40 Mbps. En ese trabajo no se estudiaba lo que ocurre cuando hay un gran número de flujos TCP, y cada uno tiene un RTT diferente. Con los tamaños que se usan actualmente, en el caso de tener por ejemplo una capacidad de 40 Gbps, y un RTT de 250 ms, se obtendría un tamaño del *buffer* de 1,25 Gigabytes, que es un tamaño excesivo.

Pero esta regla fue cuestionada en 2004 por el llamado *"Stanford model"* [AKM04], que reduce el tamaño del *buffer*, dividiéndolo por la raíz cuadrada del número de flujos TCP presentes. Esto se debe a que la ausencia de sincronización entre los flujos permite realizar una aproximación, basada en el teorema central del límite, para calcular la ocupación del *buffer*. De esta manera, el tamaño sería:

$$B = C \times RTT / \sqrt{N} \tag{4.2}$$

Donde N es el número de flujos TCP. Se está asumiendo que ese número es lo suficientemente grande como para considerarlos no sincronizados e independientes unos de otros. También asume que solamente hay flujos TCP de larga duración. Usando esta aproximación, un *router* que gestione 10.000 flujos solamente necesitará 12,5 Megabytes de tamaño.

Este tamaño de *buffer* se ha denominado en la literatura *small buffer*. Este trabajo provocó que posteriormente aparecieran muchos otros sobre el asunto, tratando de encontrar el tamaño óptimo en cada caso. Muchos de ellos se centran en estudiar el comportamiento del *buffer* en presencia de un número de flujos TCP en *router* pertenecientes a la red troncal.

Posteriormente, en [EGGMR05] se propuso la utilización de *buffer* todavía más pequeños, denominados *tiny buffer*, considerando que un tamaño de entre 20 y 50 paquetes (que equivale a algunas decenas de kilobytes) es suficiente como para alcanzar una utilización del enlace de entre el 80 y el 90%.

Teniendo en cuenta el crecimiento de los servicios de tiempo real en Internet en los últimos años, otros trabajos [VS08], [VSR09] han considerado el tráfico combinado TCP y UDP en *buffer* pequeños, descubriendo una región anómala, en la que las pérdidas de paquetes de UDP crecen con el tamaño del *buffer*.

En [DD 06] se presentó una comparativa entre diferentes políticas. Una de ellas era el *buffer* limitado en tiempo, que controla el máximo tiempo que un paquete puede pasar encolado. De hecho, es similar al *buffer* limitado en tamaño, ya que la relación entre tamaño y tiempo viene dada por el ancho de banda del enlace. En este trabajo estudiaremos y utilizaremos esta política, ya que, como veremos, es muy adecuada para mantener los retardos por debajo de una cota superior, cosa interesante en servicios de tiempo real.

Al usar ciertas políticas, como el *buffer* limitado en tiempo, el tamaño del paquete tendrá influencia en la probabilidad de descarte, puesto que los paquetes más

grandes tendrán una mayor probabilidad de no tener sitio en la cola. Y dado que las técnicas de optimización del tráfico, como la multiplexión, incrementan el tamaño del paquete, puede ocurrir que las pérdidas aumenten. Por otro lado, estas técnicas ahorran ancho de banda, por lo que, con la misma cantidad de tráfico de fondo, habrá menos tráfico total, y por tanto menos probabilidad de pérdidas. Por tanto, tenemos una situación de equilibrio, con varios factores que influyen en las pérdidas, así que habrá que estudiar cómo afecta cada uno para valorar el interés de las técnicas de optimización.

Existen *router* que miden el tamaño de sus *buffer* en términos de bytes, mientras que otros lo dimensionan según el número de paquetes que puede almacenar. Por ejemplo, en [SPGW08] se comparaban los *router* de dos fabricantes, y esta diferencia aparecía. Esto también tiene implicaciones para nuestro trabajo, en el que la multiplexión modifica el tamaño del paquete: el hecho de que un *router* pueda almacenar un número de paquetes, y no una cantidad de bytes, hace que el tamaño del paquete no tenga influencia, por lo que resultará más interesante utilizar paquetes grandes, puesto que ocupan un hueco en la cola, exactamente igual que los paquetes pequeños. Esta implementación producirá también el efecto de que el retardo de encolado variará en función del tamaño de los paquetes que haya almacenados, lo que tiene implicaciones en servicios de tiempo real.

TÉCNICAS DE OPTIMIZACIÓN DEL TRÁFICO

Algoritmos de compresión

En una red de conmutación de paquetes mediante *datagramas*, como son las redes IP, cada unidad de información debe llevar una cabecera en la que se especifican su origen, destino y algunos otros campos. Esto provoca que en un flujo de paquetes con el mismo origen y destino, esa misma información aparezca en cada paquete. También hay otros campos, como por ejemplo los números de secuencia, que aumentan su valor de uno en uno. Esto ha llevado a buscar métodos para reducir el *overhead* mediante la supresión de algunos campos de estas cabeceras.

Se han desarrollado diversos algoritmos que comprimen la cabecera, pero con el inconveniente de que sólo pueden funcionar nodo a nodo, o bien mediante un túnel, ya que un paquete sin una cabecera completa no puede circular por una red IP.

Concretamente, los flujos de tiempo real que utilizan RTP, requieren como mínimo 40 bytes de cabecera (20 de IPv4, 8 de UDP y 12 de RTP). Si el *payload* es pequeño (por ejemplo, en el caso de voz puede ser de 20 bytes), vemos que sólo un tercio de los bytes enviados llevan información (Fig 5.1).

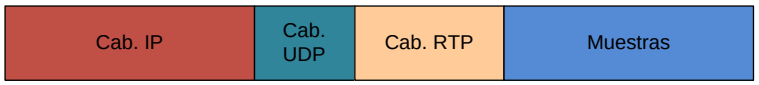


Fig. 5.1. Esquema de un paquete de voz RTP con 20 bytes de *payload*

El IETF (*Internet Engineering Task Force*) ha desarrollado a lo largo de los años diferentes estándares para aumentar la eficiencia, aprovechando esta redundancia de los campos de las cabeceras IP, UDP, TCP y RTP.

En estos protocolos se define un *contexto*, que se transmite inicialmente con las primeras cabeceras. Los diferentes campos de las cabeceras se dividen en *estáticos, aleatorios, delta* e *inferidos*. Los *estáticos* se envían sólo en las primeras cabeceras. Los *aleatorios* no se comprimen. Los clasificados como *delta,* que son los que se van incrementando de uno en uno, se codifican con menos bytes que en el campo

original. Finalmente, los *inferidos* se deducen de los campos de los niveles inferiores: por ejemplo, la longitud del paquete se puede inferir del campo correspondiente del nivel 2.

En primer lugar, VJHC [Jac98] presentó un método para comprimir las cabeceras IP/TCP. Poco después se presentó IPHC (IP *Header Compression)* [DNP99], capaz de comprimir también las cabeceras UDP e IPv6. En ese mismo momento se publicó CRTP (*Compressed* RTP) [CJ99] para comprimir las cabeceras IP/UDP/RTP. Varios años después se mejoró su comportamiento en enlaces con retardos grandes, pérdidas de paquetes y llegadas desordenadas, y se denominó ECRTP (*Enhanced Compressed* RTP) [KCGT+03]. Las mejoras intentan reducir los problemas asociados a la desincronización del *contexto*, modificando el modo de sincronización: los campos se actualizan de manera repetida, y se incluyen actualizaciones tanto de los campos totalmente comprimidos como los que se comprimen de manera diferencial.

Uno de los principales problemas de estos protocolos es el grave impacto que tiene la desincronización del contexto. Si se pierde algún paquete de los que llevan la información de las cabeceras completas, esto producirá que en el origen y en el destino no coincidan los valores del *contexto*, con lo que los paquetes no se podrán reconstruir correctamente. Otro problema, ya citado anteriormente, es que estos protocolos de compresión sólo pueden actuar nodo a nodo.

ROHCv2 *(RObust Header Compression Version 2)* [PS08] es un estándar más reciente, que puede comprimir las cabeceras IP, TCP, UDP y RTP. Reduce el impacto de la desincronización del contexto mediante un sistema de realimentación que funciona entre el descompresor y el compresor. Utiliza diferentes niveles de compresión, que se corresponden con los modos de operación: *inicialización*, *primer orden* y *segundo orden.* En el último modo, la cabecera se puede comprimir a un solo byte [CLMR+06]. De todas formas, el uso de estas técnicas avanzadas hace más difícil su implementación [EC04], añadiendo más retardo de procesado que en el caso de los protocolos de compresión explicados anteriormente.

Algunos de los protocolos de compresión son adecuados para el tráfico de VoIP, que utiliza RTP. Otros se pueden utilizar para comprimir solamente las cabeceras IP y UDP. Por tanto, en servicios de tiempo real que no usan RTP, se deberán usar estos últimos. Un ejemplo de este tipo de servicio son los juegos *online,* que utilizan UDP pero no RTP.

Hubo una propuesta interesante [MHKE01] para usar un protocolo similar a RTP en el tráfico juegos *online*, lo que daría la posibilidad de reutilizar servicios genéricos, evitando así la necesidad de implementarlos para cada juego. Pero hoy en día los juegos comerciales utilizan principalmente paquetes IP/UDP.

Métodos de multiplexión

Puede ocurrir que diferentes flujos RTP se quieran agrupar o modificar de diferentes maneras, por tener un origen o destino comunes. Si se busca reducir el *overhead* mediante el aumento de muestras que comparten una misma cabecera, la primera idea podría ser juntar un número de paquetes de un mismo usuario en otro paquete más grande, con una sola cabecera (Fig.5.2). Esto tiene una contrapartida, y es que cada paquete añadido aumentaría el retardo en el emisor, como puede observarse. Un caso particular sería el aumento del número de muestras por paquete.

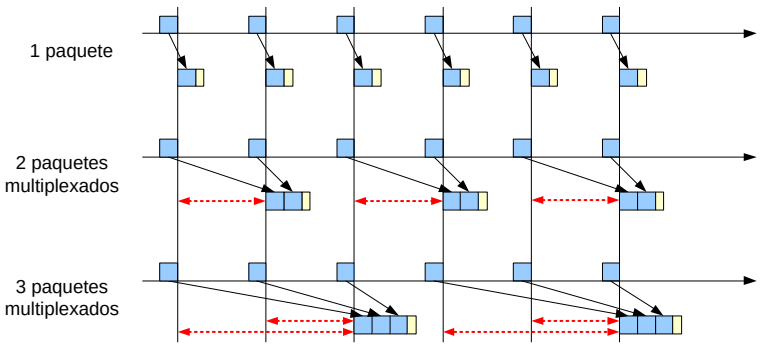


Fig. 5.2. Unión de paquetes del mismo flujo

El estándar que define RTP [SCFV03] incluye los conceptos de *translators* (traductores) y *mixers* (mezcladores). Un traductor (Fig. 5.3) puede cambiar la codificación de un flujo. Un mezclador (Fig. 5.4) es una entidad que recibe flujos de distintas fuentes, puede cambiar algunos datos y los reenvía como un flujo combinado. Por ejemplo, en una multiconferencia se pueden unir varios flujos de voz en uno solo, que se transmite al receptor. Por tanto, estas dos entidades envían un flujo RTP, que puede ser la combinación de varios. Estos dispositivos son capaces de mejorar la eficiencia, pues pueden reducir el número de paquetes enviados y, por tanto, el de cabeceras.

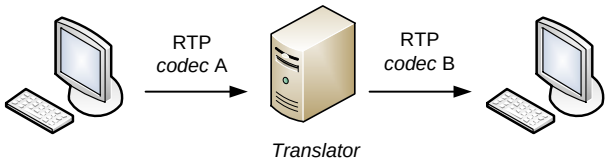


Fig. 5.3. Funcionamiento de un *translator*

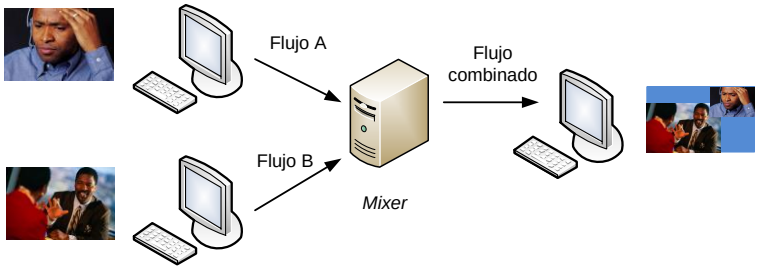


Fig. 5.4. Funcionamiento de un *mixer*

Existen situaciones en las que varios flujos de un servicio en tiempo real comparten el mismo camino (Fig. 5.5): por ejemplo, un conjunto de máquinas de la misma oficina pueden usar una PBX situada en el centro de datos de su empresa; o también diferentes usuarios de dos sucursales de la misma empresa pueden establecer simultáneamente varias conversaciones entre ellos.

En estos escenarios, se puede pensar en las ventajas del uso de un multiplexor, que recibe datos de varias fuentes y combina todos los flujos en uno solo, pero de forma que se puedan volver a separar en el destino. En telefonía, este esquema se suele denominar *trunking*. El mecanismo consiste en tomar un paquete de cada flujo, y unirlos en un paquete multiplexado. Se añade un retardo a causa de la espera hasta tener un paquete de cada flujo, pero es un retardo acotado, que será como máximo el tiempo entre paquetes, como se ve en la Fig. 5.6. Esta solución envía el mismo número de muestras, con la misma frecuencia que en el tráfico nativo. Se debe tener en cuenta que también aparecerán otros retardos, como el de procesado, que habrá que estudiar para valorar su influencia.

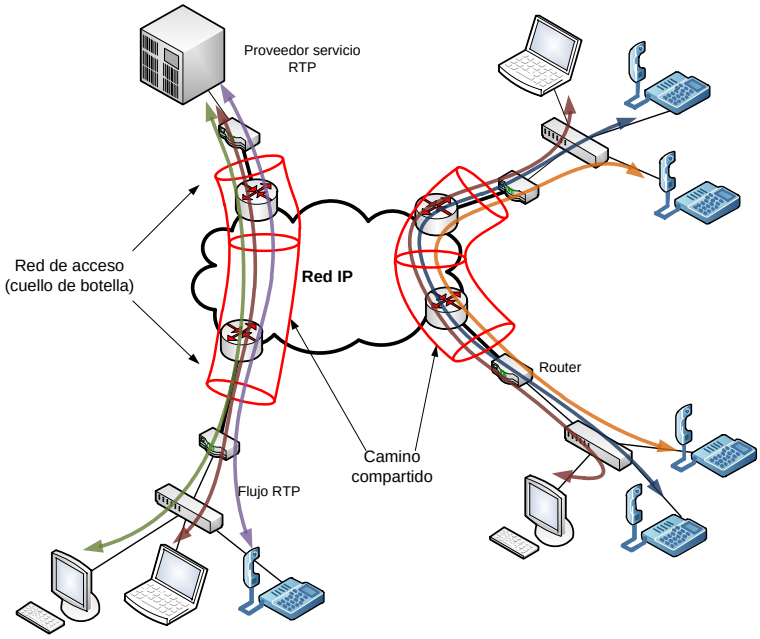


Fig. 5.5. Escenarios donde varios flujos de tiempo real comparten un camino

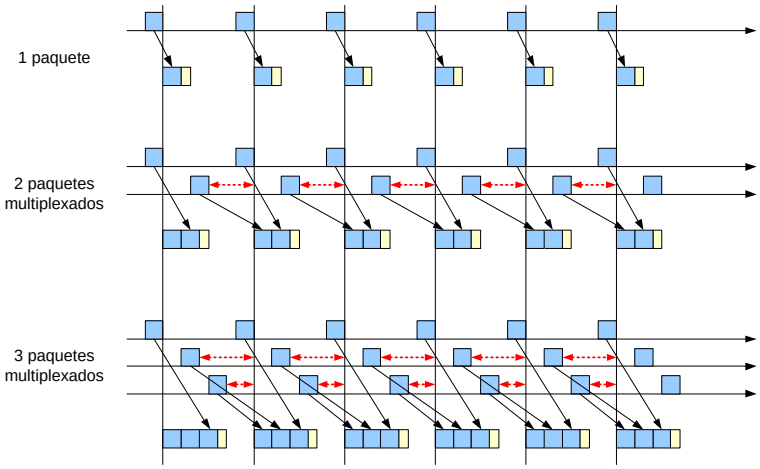


Fig. 5.6. Unión de paquetes de diferentes flujos

Un multiplexor-demultiplexor tiene que ser transparente para los dos extremos de la comunicación (Fig. 5.7). El paquete enviado desde el origen debe ser exactamente igual que el recibido en el otro extremo. Por tanto, el demultiplexor necesita información para reconstruir el paquete original y entregarlo a su destinatario.

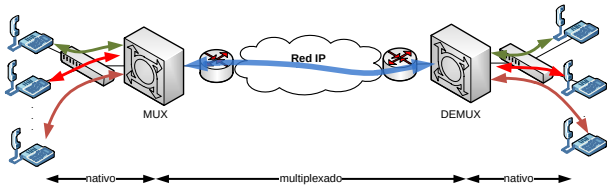


Fig. 5.7. Esquema genérico de un sistema de multiplexión

El IETF, después de valorar diferentes propuestas para la multiplexión de flujos RTP, definió en 2005 TCRTP [TKW05], con la categoría de *"best current practice"*, lo que significa que el estándar no define ningún nuevo protocolo, sino que combina varios ya existentes y recomienda su uso siguiendo un esquema determinado.

Explicaremos ahora dicho estándar, que será ampliamente utilizado en esta tesis. Su pila de protocolos se puede ver en la Fig. 5.8. En primer lugar, se usa ECRTP para comprimir las cabeceras IP, UDP y RTP. Posteriormente se usa PPPMux *(PPPMultiplexing)* [PAF01] para incluir varios paquetes en uno que finalmente se envía mediante PPP *(Point-to-point Protocol)* [Sim94] y un túnel L2TP *(Layer 2 Tunneling Protocol)* [TVR+99]. El uso de tunelado permite utilizar ECRTP extremo a extremo, evitando así la necesidad de aplicarlo en cada *router* del camino.

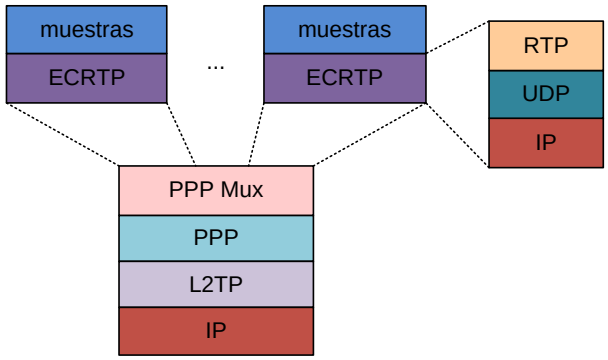


Fig. 5.8. Pila de protocolos de TCRTP

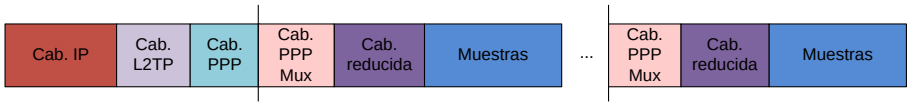


Fig. 5.9. Esquema de un paquete TCRTP

Además del estándar TCRTP, en la literatura existen otras propuestas. Una de ellas es la presentada por Sze *et al.* [SLLY02]. Consiste en incluir un número de paquetes RTP en un solo paquete UDP. Las cabeceras RTP se comprimen, por lo que se necesitan unas tablas en las que almacenar la información para poder descomprimirlas en el receptor. En la Fig. 5.10 se muestra el esquema de un paquete multiplexado según esta propuesta. En [HTT99] se propuso una solución similar, pero sin incluir la compresión de las cabeceras RTP.

Fig. 5.10. Esquema de un paquete multiplexado según el método de Sze.

Existen en la literatura otras propuestas de multiplexión, como la presentada en [TA02], que propone un sistema que adapta la tasa de envío según la congestión. Por otro lado, GeRM [Per03] propuso la idea de incluir varios *payload* RTP, cada uno con su cabecera comprimida, dentro de un único paquete RTP (Fig. 5.11). Finalmente, en [SS98] se propuso ensamblar muestras de audio de distintos usuarios en un solo *payload* RTP, utilizando una mini-cabecera de 2 bytes para identificar a los usuarios.

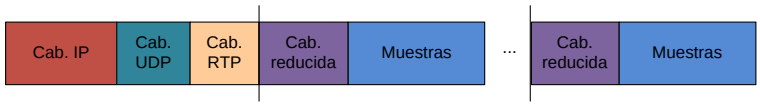


Fig. 5.11. Esquema de un paquete multiplexado según el método GeRM.

Capítulo 6

HERRAMIENTAS A UTILIZAR EN LAS PRUEBAS

> La parte de este capítulo referida al *testbed* basado en virtualización ha sido publicada en el artículo **"Hybrid Testbed for Network Scenarios,"** en **SIMUTools 2010**, the Third International Conference on Simulation Tools and Techniques. Torremolinos, Malaga (Spain). Mar. 2010, cuyos autores son José Mª Saldaña, Eduardo Viruete, Julián Fernández-Navajas, José Ruiz-Mas, José I. Aznar.

Aunque en sus comienzos Internet se entendía como una infraestructura de investigación, hoy en día se ha convertido en un entorno operativo y es más difícil de usar como plataforma de pruebas [Liu07]. Muchas universidades y departamentos de I+D que trabajan en redes, protocolos y aplicaciones distribuidas, necesitan herramientas y entornos donde verificar su funcionamiento en condiciones reales. En ocasiones, las pruebas son muy difíciles de realizar debido al gran número de máquinas y elementos que se requieren.

Tipos de herramientas

Hay varias soluciones posibles para este problema. Una pasa por el uso de herramientas de simulación, como Opnet, ns-2, ns-3, etc., que permiten realizar medidas controladas y repetibles a bajo coste. Están compuestos por módulos de *software* que representan diferentes elementos de red, protocolos, etc. Su característica fundamental es que se definen por eventos discretos, y existe una variable tiempo, que va avanzando, pero no se corresponde con el tiempo real [Liu07].

Tienen dos inconvenientes: el primero es la carga computacional, especialmente cuando el escenario de red incluye un gran número de máquinas; el segundo se debe al hecho de que los simuladores simplifican el sistema a medir, alejándolo de la realidad, y usan implementaciones específicas de los protocolos, lo que no permite la utilización en las pruebas de cualquier aplicación o sistema operativo reales.

Otra opción es utilizar *hardware* real, que incluye la pila de protocolos completa del sistema operativo. Esta puede ser una buena solución en muchos casos, pero tiene como inconvenientes el coste de los equipos y la dificultad de gestionarlos todos a un tiempo.

Por eso, también se han desarrollado muchos *testbed* y emuladores [Gök07] para facilitar las pruebas y reducir el coste. Cuando se utiliza emulación, el sistema a medir se representa con algunas de sus partes modificadas y otras tratadas exactamente igual que en el caso real [Gök07]. Algunas partes de la prueba tienen un mayor nivel de abstracción que otras; algunas son simuladas y otras reales.

Como la emulación combina elementos reales con otros modificados o simplificados, se debe ejecutar en tiempo real. Esto hace que las pruebas no sean totalmente repetibles, ya que puede haber pequeñas diferencias entre unas realizaciones y otras.

En los últimos años se han desplegado grandes infraestructuras donde probar protocolos y aplicaciones como GENI [Tur06] o FIRE [GKFM+ 07]. Estos entornos suelen ser compartidos por grupos de investigación de diferentes países. Algunos integran emuladores para imitar el comportamiento de diferentes elementos del sistema, como los movimientos de los nodos, los cambios en el canal radio, etc. Una de estas grandes plataformas de pruebas es *PlanetLab* [BBCC+04], que permite desarrollar y evaluar nuevos protocolos y servicios. Cada nodo contiene un monitor de máquinas virtuales que aísla entre sí los servicios y aplicaciones.

Algunos emuladores y *testbed* son híbridos, combinando las ventajas de la simulación, la emulación y las pruebas con equipos reales. Entre las plataformas híbridas existentes en la literatura podemos destacar en primer lugar EMWin [ZN02], que emula la capa MAC, mientras que la pila de protocolos y las aplicaciones son reales. NCTUns [Wan06] emula nodos que generan tráfico, que luego es capturado por el simulador para introducir el comportamiento de un escenario inalámbrico. Netbed/ Emulab [WLSR+02] y W-NINE [CGPD07] usan conformadores de tráfico después de una etapa de simulación. También hay emuladores que usan virtualización, como vBET [JX03].

En las pruebas presentadas en esta tesis, se utilizarán las herramientas que se consideren más adecuadas en cada caso. De este modo, cuando sea necesaria la utilización en las pruebas de herramientas *software* reales, se deberá recurrir a máquinas reales o a *testbed*. Pero cuando sea necesario realizar pruebas con un mayor número de máquinas, la simulación también puede resultar útil para modelar determinados comportamientos. También se pueden usar resultados obtenidos en un *testbed* como parámetros de una simulación posterior, utilizando por tanto una aproximación híbrida.

Solución utilizada para el *testbed:* Virtualización

En la actualidad, se habla de virtualización con significados muy distintos. En el presente trabajo, el término "virtualización" se refiere a la creación de una versión virtual de una máquina, que se ejecuta sobre el *hardware* real de otra. Existe una capa de abstracción que permite a múltiples máquinas virtuales con sistemas operativos distintos ejecutarse en la misma máquina física. Cada máquina virtual tiene su propio *hardware* virtual.

La virtualización ha sido empleada frecuentemente en el ámbito de la investigación. Por ejemplo, se ha empleado para construir grandes infraestructuras como la ya citada *PlanetLab* [BBCC+ 04], aunque también se ha empleado en plataformas más pequeñas [BSPL+09], [WLSR+02]. También se pueden integrar con algunos simuladores de red, como ns-3, que permite integrar máquinas virtuales dentro de las simulaciones, pudiendo ejecutarse sobre sus dispositivos y canales.

El *software* de virtualización emula un *hardware* suficiente para poder ejecutar de forma aislada a un sistema operativo cliente. Lógicamente, ese sistema operativo debe estar diseñado para la misma CPU. Muchas máquinas virtuales pueden funcionar al mismo tiempo en una máquina física.

Para las pruebas que requieran el uso de *software* real, se utilizará una plataforma de pruebas basada en virtualización. De esta forma, un conjunto de máquinas virtuales se podrá incluir dentro de una sola máquina física, evitando de este modo la necesidad de utilizar un número elevado de equipos. Se creará un entorno en el que cada máquina disponga de varios interfaces de red (Fig. 6.1), uno de los cuales se utilizará exclusivamente para control, es decir, a través de él se enviarán a la máquina los comandos para ejecutar las aplicaciones correspondientes. El resto de interfaces serán usados para crear la red correspondiente al escenario que se está emulando. De este modo, el tráfico de control no interferirá con el de las pruebas, evitando así perder realismo.

Por tanto, es necesaria una plataforma de virtualización que permita ejecutar un buen número de máquinas virtuales sin penalizar el rendimiento ni requerir mucha capacidad computacional. Aunque no existe un consenso total, algunos autores [Jon11] consideran cuatro tipos fundamentales de virtualización. Pasaremos ahora a verlos.

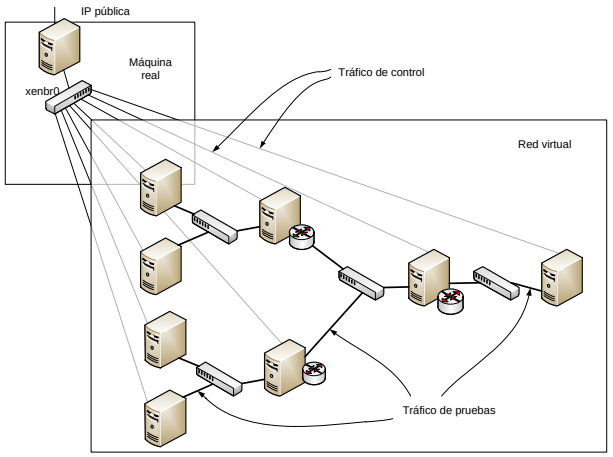


Fig. 6.1. Máquina real y red de máquinas virtuales

Tipos de virtualización

Emulación del Hardware

Este método de virtualización crea una máquina virtual en el sistema operativo de la máquina física, que emula el *hardware* deseado (Fig. 6.2). Tiende a ser muy lento, porque cada instrucción debe ser simulada en el *hardware* real. Tiene la ventaja de que se puede ejecutar un sistema operativo sin modificar en cualquier procesador. Algunos ejemplos son Bochs y QEMU.

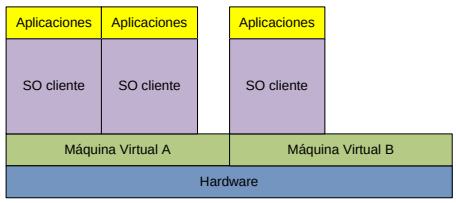


Fig. 6.2. Emulación del *Hardware*

Virtualización completa

Se denomina también nativa. Utiliza una máquina virtual que media entre los sistemas operativos cliente y el *hardware* nativo (Fig. 6.3). No es necesario modificar el sistema operativo cliente, lo que constituye su principal ventaja. Un requisito es que el sistema operativo del cliente debe estar preparado para el *hardware* físico.

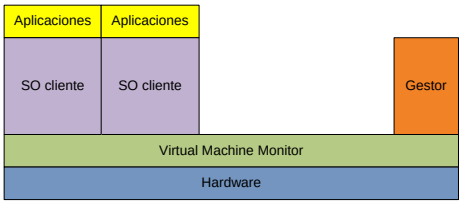


Fig. 6.3. Virtualización completa

Algunas instrucciones protegidas deben ser tratadas de un modo especial por el *hypervisor* o *VMM (Virtual Machine Monitor*, Monitor de máquinas virtuales), porque el *hardware* no es manejado directamente por el sistema operativo, sino por el *hypervisor*, que debe estar constantemente al tanto de estas instrucciones, para tratarlas de una manera especial y permitir su correcta ejecución.

Es un método más rápido que la emulación del *hardware*, pero pierde algo de eficiencia a causa del *hypervisor*. Un ejemplo de este tipo de virtualización es *VMware*.

Paravirtualización

Es muy semejante al caso anterior, pero integra en el sistema operativo cliente las instrucciones necesarias para evitar que ninguna instrucción deba ser tratada de modo especial (Fig. 6.4).

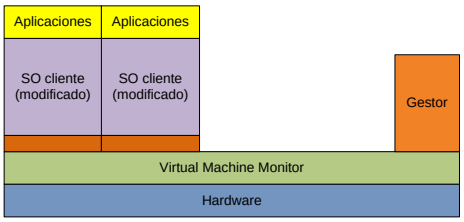


Fig. 6.4. Paravirtualización

Requiere que los sistemas operativos cliente estén compilados para funcionar en ese entorno de virtualización, lo que constituye su principal desventaja, pero esto les permite conseguir una velocidad similar a la que se daría en un sistema no virtualizado. Por supuesto, varios sistemas operativos diferentes pueden ejecutarse a la vez. Ejemplos de este tipo de virtualización son Xen y UML.

Virtualización a nivel de Sistema Operativo

Soporta un solo sistema operativo, y aísla unos grupos de procesos de otros, dándoles acceso solamente a un subconjunto de los recursos. El sistema

operativo oculta los grupos de procesos entre sí (Fig. 6.5). Su gran ventaja es que las prestaciones son las mismas que se obtendrían de forma nativa. Dos ejemplos son Linux-VServer y OpenVZ.

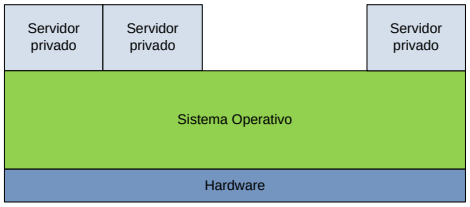


Fig. 6.5. Virtualización a nivel de Sistema Operativo

Solución elegida

Se ha optado por la paravirtualización, y en particular por Xen. La razón principal ha sido el interés en que las máquinas virtuales sean rápidas y consuman sólo los recursos necesarios. En [QNC06] se presentó una comparativa entre diferentes tecnologías de virtualización, mostrando que Xen obtenía buenos resultados en rendimiento, linealidad y aislamiento entre las máquinas.

El buen rendimiento de esta solución nos permitirá disponer de un número aceptable de máquinas virtuales dentro de una sola máquina física. Nos interesa que, para no falsear las pruebas, funcionen a la misma velocidad que en modo nativo. El problema es la no conveniencia de utilizar entornos gráficos.

Como hemos visto, un requisito de esta solución es que las máquinas deben estar preparadas para funcionar sobre ese *hardware* virtual: algo comparable a compilar el *kernel* para una nueva arquitectura. Eso no supone ningún problema en el caso de máquinas Linux.

Modo de emular redes en Xen

En esta sección explicaremos el método usado para crear redes con máquinas virtuales Xen, así como las herramientas que se han usado para emular las condiciones de red de los distintos escenarios.

Tarjetas de red y *bridge* virtuales

Como ya hemos explicado anteriormente, se ha separado la red de control de la de medidas. Esto se ha logrado gracias a que Xen permite crear varias tarjetas de red en cada máquina. La primera máquina que inicia Xen se llama *dom0*, y es la que arranca todas las máquinas virtuales. Por cada interfaz de cada máquina virtual existe otro en *dom0* (denominado *vifA.B*) que está conectado a él (Fig. 6.6).

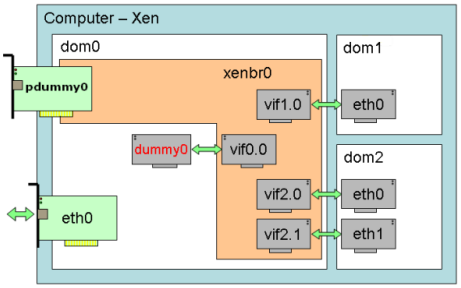


Fig. 6.6. Esquema de los interfaces de red en Xen

El comando Linux utilizado para unir interfaces es *brctl*, que permite crear *bridge* que unen las diferentes tarjetas de red. Para crear la red de control, el interfaz *eth0* de cada máquina virtual se ha unido a un *bridge* denominado *xenbr0*, que es accesible desde el dominio principal. Para crear las diferentes topologías de red, basta con crear otros *bridge* y unir a ellos los interfaces correspondientes.

Si en caso particular se desea crear redes independientes, basta con unir sus máquinas virtuales a través de *bridge* que no tengan ninguna máquina en común. Otro ejemplo de uso sería la creación de un nodo que actúe como *router*, conectando sus interfaces a *bridge* diferentes.

Traffic Control

El entorno incorpora también emulación del enlace, gracias a la herramienta Linux *traffic control (tc)* [1], que permite establecer para un interfaz de red diferentes políticas de transmisión, con sus parámetros correspondientes, como el número de paquetes o bytes que caben en el *buffer*, el límite de tiempo máximo de estancia en el *buffer*, la tasa de envío, el tamaño máximo de ráfaga, etc.

Esta herramienta dispone de diferentes tipos de colas, que además pueden anidarse unas con otras. También ofrece la posibilidad de incorporar prioridades por dirección IP o puerto.

Algunas de las colas más sencillas son la *bfifo*, para la que se puede establecer un límite de tamaño en bytes, y la *pfifo*, que permite establecer el tamaño en número de paquetes. Esta última se ha usado en algunas de las pruebas, para emular el comportamiento de un *router* que puede almacenar un número máximo de paquetes.

[1] La herramienta *tc* tiene en cuenta las cabeceras de nivel *Eth* para calcular el ancho de banda, por lo que las cantidades de tráfico deberán ser corregidas adecuadamente.

Otra cola incluida dentro de las opciones de *tc* es la *tbf* (*Token Bucket FIFO),* cuyo esquema puede verse en la Fig. 6.7. Existe un almacén de *token* o "testigos", que el sistema operativo va rellenando con una cierta periodicidad, según el parámetro *rate*. Este almacén también tiene un límite en bytes dado por el parámetro *burst* o "ráfaga". Cada vez que se envía un paquete, se retira del almacén un número de *token* equivalente al tamaño del paquete, consiguiendo así limitar el ancho de banda que sale de la cola. También existe un tamaño de la cola de paquetes, que se puede medir en bytes *(limit)* o en tiempo máximo de encolado *(latency)*.

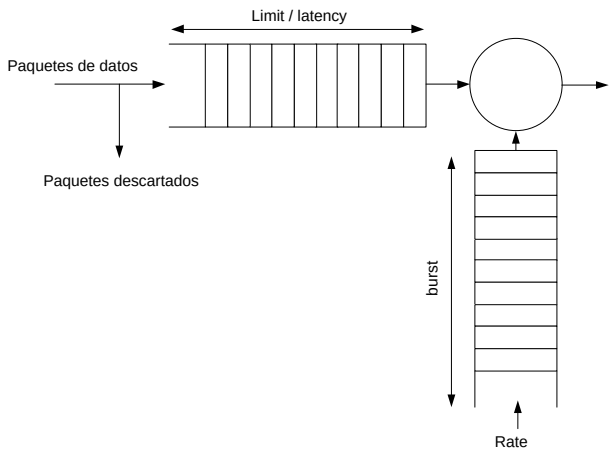


Fig. 6.7. Esquema de funcionamiento de la política *token bucket FIFO*

Esta cola es la que utilizaremos principalmente para implementar el *buffer* limitado en tiempo, usando para ello el parámetro *latency*.

Netem

También se utiliza emulación de red mediante *netem* [Hem05], que permite añadir retardos con diferentes distribuciones estadísticas, así como pérdidas de paquetes, paquetes corruptos, paquetes que llegan fuera de orden, etc.

Generadores de tráfico

Los generadores de tráfico que se han usado son JTG [Man11] y D-ITG [BDP07], que permiten enviar ráfagas con diferentes estadísticas. Ambos se componen de un módulo que envía paquetes, y otro que los captura, generando un fichero de resultados. También ofrecen unas aplicaciones que permiten realizar un cierto tratamiento estadístico de los resultados en términos de retardo, pérdidas de paquetes, *jitter*, etc.

En algunos casos se ha optado por prescindir de la aplicación que calcula los resultados, y se ha optado por programar un *script* en el lenguaje *awk* [awk11], para poder así realizar los cálculos de una manera más adecuada a nuestras necesidades. Un ejemplo de esto es el uso de *awk* para calcular el efecto del *buffer* de *dejitter* en aplicaciones de voz, que se explicará más adelante.

Sincronización

Un problema que se plantea al medir retardos es el de la sincronización entre las máquinas origen y destino. Ya hemos visto en los capítulos iniciales el problema de medir el OWD, o el RTT. Para medir el primero se requiere sincronización, por lo que, si no es posible conseguirlo con suficiente precisión, se debe recurrir a la medida del RTT.

En el caso de Xen, debemos tener en cuenta que todas las máquinas se encuentran dentro del mismo *hardware* físico, y esto nos debería ayudar a sincronizarlas. Cuando las máquinas virtuales arrancan en Xen, toman el valor del reloj de la máquina principal. Pero se ha comprobado que según pasa el tiempo, las máquinas se van desincronizando entre sí.

Se ha recurrido por tanto a la opción de Xen denominada *independent_wallclock*, que permite a cada máquina mantener su propio reloj. Una de las máquinas virtuales se ha usado como servidor NTP, y las otras se han sincronizado con ella. Este método ha dado resultados aceptables, al lograr, tras varias iteraciones de la sincronización, unas diferencias entre los relojes menores al milisegundo. Esto es posible porque las máquinas virtuales no están conectadas a través de dispositivos de red reales, sino mediante *bridge* virtuales, que funcionan a velocidad de procesador. Se ha considerado suficiente esta precisión para las medidas realizadas. Por eso, en muchas de las medidas utilizaremos el OWD como medida del retardo.

Máquinas Utilizadas en el testbed

La máquina utilizada en primer lugar tiene el Sistema Operativo CentOS 5, una distribución Linux derivada de Red Hat Enterprise Linux 5. La versión del núcleo es la 2.6.18-8.1.15. Dispone de un procesador Core 2 Duo a 2.40 Ghz, 2Mb de Cache nivel 2, de 4Gb de RAM, y un disco duro SATA2 de 320 Gb. Las máquinas virtuales tienen instalado también el Sistema Operativo CentOS 5. Los ficheros imagen ocupan 4 Gb por máquina virtual. La versión de Xen instalada es la 3.03-25.0.4.

En algunas pruebas se ha usado otra con características más avanzadas: el sistema operativo es CentOS 5.5 x86_64, con un *kernel* 2.6.18-194.32.1.el5xen. La versión de Xen es: *Hypervisor* 3.1 y Dominio 0: 3.0-x86_64. La memoria es de 8 Gb, el procesador es un Intel i5 750 y el disco duro es de 800 Gb.

Herramienta de simulación: Matlab

Matlab es un entorno *software* ampliamente utilizado en el ámbito de la investigación, ya que permite realizar con eficiencia operaciones matemáticas. También permite representar funciones, implementar algoritmos y crear interfaces de usuario. Además resulta adecuado para realizar simulaciones.

En algunas de las pruebas de esta tesis se ha utilizado para generar las variables aleatorias que se corresponden con los momentos de llegada de diferentes eventos, como por ejemplo llamadas telefónicas, siguiendo la distribución estadística que se desee.

También se ha utilizado para implementar algoritmos que calculan tamaños de paquetes multiplexados, así como los instantes en que se generan, a partir de trazas originales de tráfico.

Uso de esquemas de pruebas híbridos

En algunas de las pruebas se utilizará un sistema híbrido, es decir, integrando simulación y emulación (Fig. 6.8). Como se ha visto al principio del presente capítulo, este tipo de integración se puede encontrar en otros trabajos de investigación. Se realizan en primer lugar unas pruebas con tráfico real en el *testbed*, y se analizan para obtener los parámetros de QoS del servicio, en un conjunto de posibles situaciones: número de flujos, anchos de banda, etc. Posteriormente esos parámetros son utilizados por el simulador cada vez que se dan esas situaciones en tiempo de simulación, obteniendo así resultados más realistas.

Un ejemplo de este modo de trabajo se verá en el cálculo de parámetros de calidad en un escenario de telefonía. Si queremos probar el sistema con un número muy grande de usuarios, las limitaciones en cuanto a número de máquinas físicas disponibles harán más conveniente el uso de simulación. Pero en este caso podemos en primer lugar realizar una batería de pruebas en el *testbed*, donde se obtienen los parámetros en todo conjunto de situaciones posibles, es decir, con diferentes valores para el número de llamadas establecidas, tráfico de fondo, etc. Estos parámetros se almacenan en tablas de Matlab, para utilizarlos

posteriormente en las simulaciones. De este modo se pueden conseguir resultados realistas sin necesidad de implementar un escenario completo con máquinas reales.

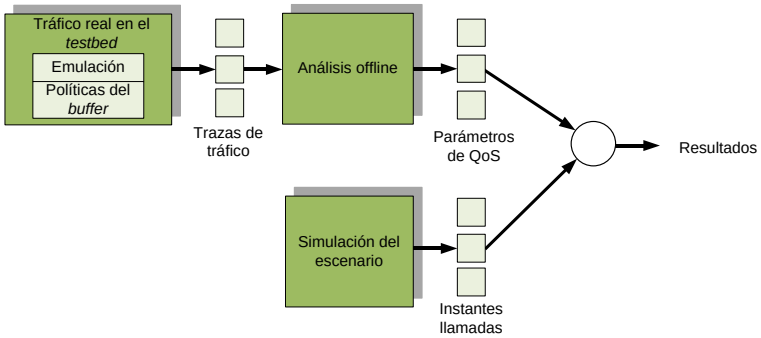


Fig. 6.8. Esquema de las pruebas híbridas.

Sección B: Optimización del tráfico de un sistema de telefonía IP

En esta sección se presenta el diseño de un sistema de telefonía IP distribuido, con una estructura similar a la de soluciones comerciales existentes, pero diferenciándose de ellas en la característica básica de que sólo se utiliza *software* libre, de modo que la solución sea modificable y fácil de estudiar. Por otro lado, se ha dotado al sistema CAC de un cierto grado de inteligencia, para poder conseguir ahorros a la hora de establecer llamadas entre diferentes sucursales, mediante la compartición del *gateway* presente en cada sucursal.

El sistema se implementa posteriormente en una plataforma de pruebas *(testbed)* basada en virtualización. De esta manera, se puede validar en un entorno controlado y con un número suficiente de máquinas virtuales.

Posteriormente se evalúan sistemas de optimización del tráfico, principalmente compresión y multiplexión, con la idea de estudiar su comportamiento para ver si es posible y conveniente incluirlos en el sistema de telefonía.

Para terminar la sección se realizan algunas pruebas incluyendo los esquemas de optimización, y comparando los resultados así conseguidos, con los obtenidos anteriormente.

DISEÑO DE UN SISTEMA DE TELEFONÍA IP CON CONTROL DE ADMISIÓN

Partes del contenido de este capítulo han sido publicadas en **"Distributed IP Telephony System with Call Admission Control,"** Proc. Conference on ENTERprise Information Systems **CENTERIS 2010**. Viana do Castelo, Portugal. Oct. 2010, cuyos autores son José Mª Saldaña, Jenifer Murillo, Julián Fernández-Navajas, José Ruiz-Mas, Eduardo Viruete y José I. Aznar.

Otras partes se han publicado en **"Implementación de un CAC basado en medidas de QoS para sistemas de telefonía IP,"** Actas de las VIII Jornadas de Ingeniería Telemática (JITEL 2009), cuyos autores son José Mª Saldaña, Jenifer Murillo, Julián Fernández-Navajas, José Ruiz-Mas y Eduardo Viruete. También en los Proyectos Fin de Carrera *"***Implementación de control de admisión en sistemas ToIP,"** realizado por Laura Esteban Boillos y *"***Análisis de un sistema CAC para telefonía IP,"** realizado por Jenifer Murillo Royo, dirigidos por el autor de esta tesis, se definieron y estudiaron las principales características del sistema de telefonía IP.

Como se ha visto en el capítulo 2, en los últimos años muchas empresas están sustituyendo sus antiguos sistemas de telefonía basados en RTC (Red Telefónica Conmutada) por otros nuevos que usan IP, de manera que las llamadas y las videoconferencias se pueden establecer a través de redes de datos. Uno de los objetivos es la reducción de costes, que se puede lograr aprovechando los accesos a Internet existentes para transmitir vídeo, voz y datos.

En ese mismo capítulo se explicaron las razones que nos han llevado a elegir el protocolo SIP para la señalización en el sistema. La principal es que se trata de un estándar abierto, que se encuentra muy extendido.

La VoIP es un servicio en tiempo real, que en muchos casos utiliza una red que fue diseñada para servicios *best-effort*. Sin embargo, los usuarios desean una calidad similar a la que acostumbran a tener utilizando los sistemas de telefonía RTC tradicionales. Este hecho ha llevado a los investigadores a buscar soluciones para añadir calidad a las redes de telefonía IP. Una manera de evitar el problema consiste en sobredimensionar los recursos, pero el crecimiento continuo del tráfico y la aparición de nuevos servicios han llevado a la búsqueda de soluciones que aporten más inteligencia a los sistemas. Una de las más utilizadas para telefonía IP, como hemos visto, es el CAC, que acepta o rechaza las nuevas

llamadas para evitar la degradación del servicio. Puede estar basado en parámetros que se establecen en el momento de la instalación, o en medidas de QoS que se van realizando en paralelo con el funcionamiento normal del sistema, y modifican el número de líneas disponibles. Nuestro sistema podrá funcionar en ambos modos: basado en parámetros o en medidas.

Esquema y elementos del sistema de telefonía IP

Muchas de estas empresas tienen sus recursos descentralizados, de manera que cada oficina o sucursal (utilizaremos indistintamente estos dos términos) es independiente del resto y se encarga de gestionar tanto su conexión a Internet como sus líneas telefónicas. Pero si se realizase una gestión centralizada de los recursos globales de la empresa (Fig. 7.1), se podrían ahorrar costes realizando parte de las llamadas a través de Internet, manteniendo, e incluso aumentando, la QoS mediante el control de la probabilidad de admisión de las llamadas y de sus parámetros de calidad.

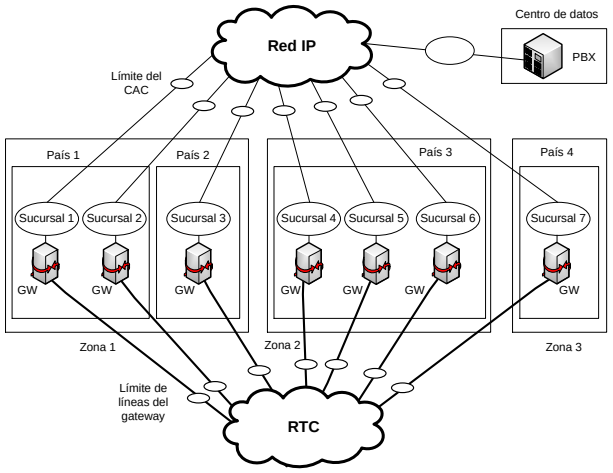
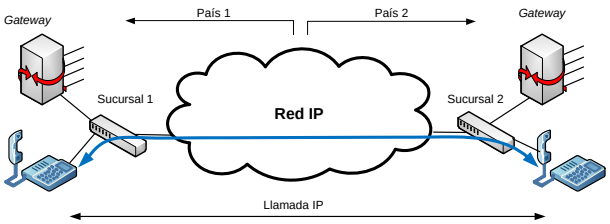


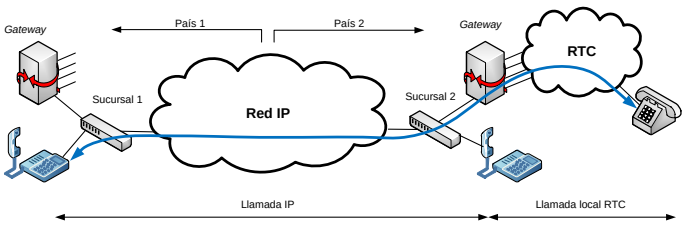
Fig. 7.1. Esquema del sistema de telefonía IP distribuido

Como solución centralizada, se puede usar una centralita o PBX *software* para unir los sistemas de telefonía de cada sucursal, creando así un sistema que cuenta con varias ventajas, como el poder compartir las líneas que conectan a RTC las diferentes sucursales. Las llamadas entre sucursales podrían establecerse utilizando simplemente una aplicación de VoIP (Fig. 7.2.a), pero las empresas suelen optar por instalar sistemas de telefonía IP para gestionar sus comunicaciones de voz. Y para las empresas con presencia en varios países, una de las ventajas de usar un sistema de este tipo es que se puede hacer una mejor

gestión de las conferencias internacionales con destino a terminales tradicionales. Estas llamadas podrían llevarse a cabo en dos tramos, uno a través de Internet hasta el país destino, y otro por RTC hasta el usuario destino, con tarifa de llamada local (Fig. 7.2.b), logrando así el consiguiente ahorro económico.



(a)



(b)

Fig. 7.2. a) Esquema usando una aplicación de VoIP y b) esquema de telefonía IP propuesto

Por tanto, el objetivo que nos planteamos es el diseño de un sistema de telefonía IP con CAC, que permita estos servicios. El esquema de partida será similar al de algunas soluciones propietarias, por ejemplo las de Cisco, que fueron estudiadas en [WMXZ06]. Este sistema se deberá corresponder con el de una empresa con sucursales en diferentes países (Fig. 7.3), que podrán estar agrupadas en zonas geográficas. Las zonas son agrupaciones de países entre los que existen tarifas algo más reducidas.

Como vimos al hablar del CAC en el capítulo 2, existen soluciones comerciales que integran un elemento en cada sede, que se comunica con el nodo central. Usaremos este mismo esquema para el diseño de nuestro sistema, incluyendo un agente local que estará a cargo de la señalización. Supondremos que no vamos a actuar sobre la PBX, y para reducir los costes de operación, el plan de numeración del sistema de telefonía IP se mantendrá solamente en la PBX, y no distribuido entre las sucursales. Además, se utilizará Internet como medio de transmisión del tráfico telefónico entre las sucursales, prescindiendo de otro tipo

de enlaces. El *gateway,* que conecta la red IP con RTC, ofrece un número determinado de líneas en cada sucursal. Sus líneas están controladas por el agente local.

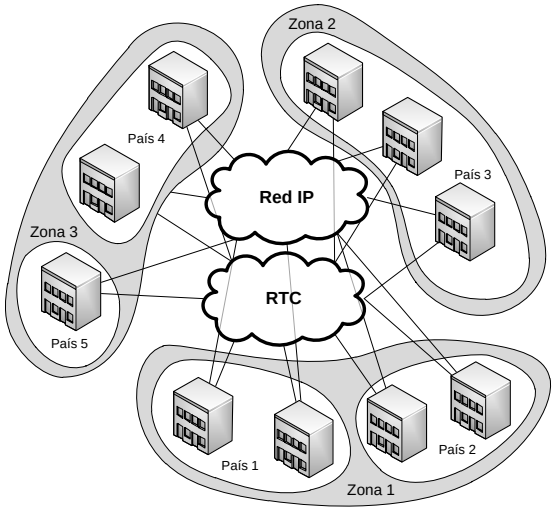


Fig. 7.3. Esquema de zonas y países del sistema de telefonía IP

Vemos, por tanto, que, además de la PBX central, en cada sucursal se dispondrá de los siguientes elementos (Fig. 7.4): teléfonos IP, *softphone* (aplicaciones informáticas que se utilizan como teléfono), *gateway* y un agente local.

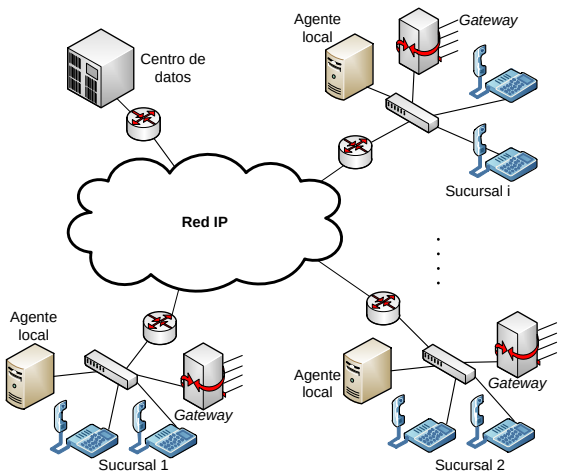


Fig. 7.4. Elementos del sistema de telefonía IP

Tipos de llamadas

El sistema permite el establecimiento de diferentes tipos de llamadas. Las clasificaremos en seis (Fig. 7.5):

1. Llamada entre terminales de la misma sucursal.
2. Llamada interna entre oficinas diferentes.
3. Llamada a RTC de un país que no tiene sucursal. Será gestionada por el *gateway* local.
4. Llamada a RTC del país en el que se encuentra la sucursal. También será gestionada por el *gateway* local.
5. Llamada a RTC gestionada por el *gateway* de una sucursal remota.
6. Llamada procedente de un usuario de RTC externa. Llegará al *gateway*, con destino a un terminal de la sucursal.

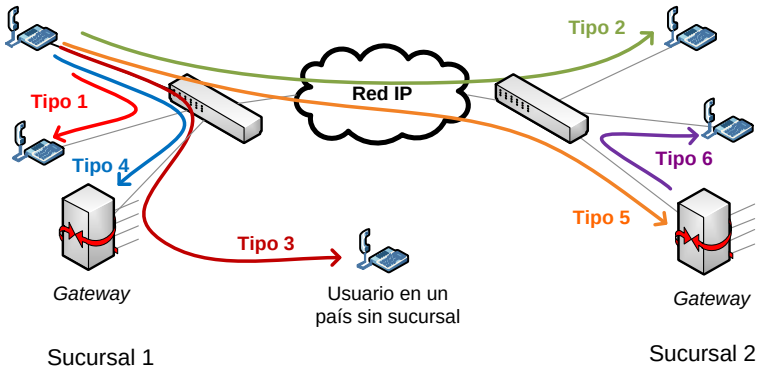


Fig. 7.5. Tipos de llamadas

Funcionamiento del agente local

El agente local juega un papel importante para lograr que las llamadas sean cursadas por la mejor ruta en cuanto a QoS y costes. En la Fig. 7.6 puede verse un esquema con sus elementos.

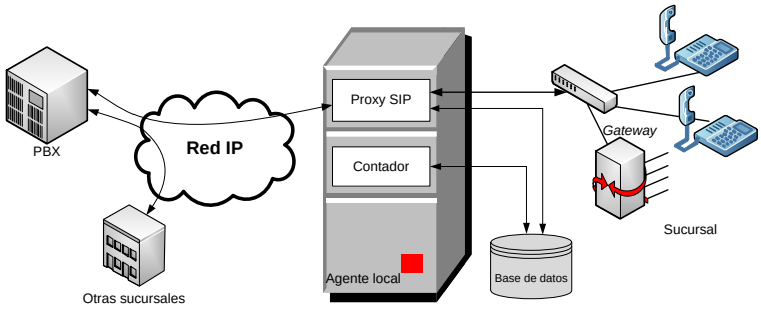


Fig. 7.6. Esquema del agente local

El agente se encargará de la elección de la mejor ruta para realizar las llamadas telefónicas. Basándose en la calidad esperada, las tarifas y el número de líneas libres y ocupadas del *gateway*, rellenará unas tablas en las que se basarán las decisiones de CAC. Explicaremos ahora más en detalle algunos de los elementos que lo componen:

- Un *proxy* SIP procesa la señalización de las llamadas e implementa el mecanismo de CAC, basándose en información de la tabla de decisiones que se encuentra en la base de datos. Esta tabla, cuya estructura se detalla en el siguiente apartado, se actualiza en función del estado del sistema, e indica si se puede aceptar o no una llamada. En general, un *proxy* SIP no requiere una gran capacidad de procesado. En [WSKW07] se realizaron algunas medidas de su comportamiento, comprobando que un único *proxy* puede administrar 2.000 mensajes SIP por segundo.

- El módulo *contador* se encarga de controlar las líneas libres y ocupadas en el *gateway*. También realiza un conteo del número de llamadas telefónicas en curso en la oficina en cada momento. Teniendo en cuenta esta información, las tarifas telefónicas y el número de líneas disponibles en el *gateway*, el agente rellena la tabla en que se basan las decisiones del sistema CAC.

- Adicionalmente, se podría incluir un módulo encargado de realizar medidas periódicas. No obstante, no se ha incluido dicho módulo en el esquema porque también se podrían encargar estas medidas a otra entidad, que las almacenase en la base de datos correspondiente, donde el agente pudiese consultarlas.

El esquema de los protocolos puede verse en la Fig. 7.7. Observamos cómo el tráfico de señalización SIP va desde el teléfono hasta el *proxy* de la sucursal origen, y posteriormente a la centralita. Después, a través del otro *proxy*, llega al teléfono destino. La centralita actúa por tanto como *"back to back user agent"*, es decir, une dos llamadas diferentes, una con cada uno de los teléfonos que intervienen. Para el tráfico RTP se ha utilizado un esquema en estrella, en el que el tráfico circula entre cada par de teléfonos directamente, sin pasar por la PBX.

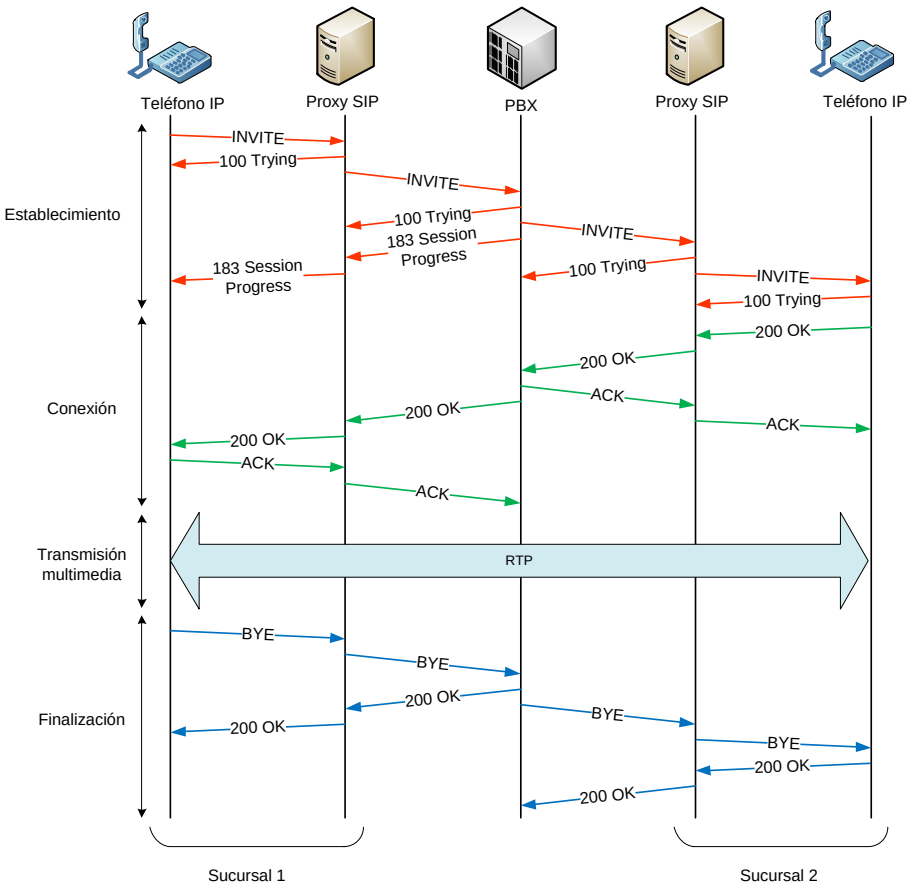


Fig. 7.7. Esquema de una llamada en el sistema

El agente local se ha situado en el escenario de manera que todos los mensajes de señalización entre los terminales y la PBX pasen a través de él. De esta forma podrá introducir señalización para implementar las decisiones de CAC, y llevar cuenta de las llamadas establecidas en el *gateway* en cada momento. Esta situación tiene la ventaja de que, mientras no haya que rechazar llamadas, el agente se limitará a retransmitir la señalización.

Las llamadas entre terminales de la misma sucursal (tipo 1) deberán ser gestionadas por el agente local, y por tanto no necesitarán acudir a la PBX para establecerse. En el caso de las llamadas entre sucursales, que no salen a RTC (tipo 2), las decisiones de CAC serán sólo de aceptación o no de la llamada, en función de la QoS, puesto que no tiene sentido redirigirlas a otra sucursal diferente a la del teléfono destino. Si el usuario solicita una llamada a RTC (tipo 3, 4 y 5), el sistema podrá elegir el *gateway* a través del que se establecerá la llamada local. Por último, las llamadas tipo 6, que llegan de usuarios de RTC, ocupan una línea del *gateway*.

Para tomar las decisiones de CAC, cada agente local utiliza una "tabla de decisiones" (Tabla 7.1), en la que se especifica cómo actuar en el caso de recibir una petición de llamada (mensaje *INVITE* del protocolo SIP) desde una determinada sucursal. Las llamadas internas son las que se dan entre diferentes sucursales de la empresa. Las llamadas al *gateway* son las que van destinadas a usuarios de RTC. El CAC no afecta a las llamadas entre usuarios de la misma sucursal, pues las redes locales suelen tener un ancho de banda elevado que garantiza la calidad en condiciones normales.

Origen	**Llamada interna**	**Llamada al *gateway* (RTC)**
1	Aceptar / rechazar	Aceptar / rechazar / redirigir a i
2	Aceptar / rechazar	Aceptar / rechazar / redirigir a i
...	...	...
N	Aceptar / rechazar	Aceptar / rechazar / redirigir a i

Tabla 7.1. Tabla de decisiones

Esta tabla se construirá a partir de otras que veremos posteriormente, y que reflejan las líneas disponibles en cada *gateway*, la tarificación de cada sucursal y, en el caso de usar el modo basado en medidas, de las estimaciones de QoS:

Tabla decisiones = f (líneas disponibles, tarifas, medidas QoS)

Cuando el agente recibe un mensaje *INVITE* de la PBX (Fig. 7.8), con destino a un usuario de su sucursal, lo acepta o rechaza en función de la entrada de la tabla que corresponda a la sucursal origen. La llamada puede ser rechazada por no estar disponible el usuario destino, o bien, en el modo basado en medidas, por falta de QoS entre las dos sucursales.

Si el agente local recibe un *INVITE* de una llamada con destino al *Gateway,* y la tabla de decisiones indica *rechazar*, se enviará un mensaje SIP "*480 Temporarily Unavailable*" (Fig. 7.9), a la PBX. Ésta, según su plan de numeración, podrá acudir

a otra sucursal que pueda realizar la llamada, a ser posible en la misma zona para así conseguir una tarifa económica.

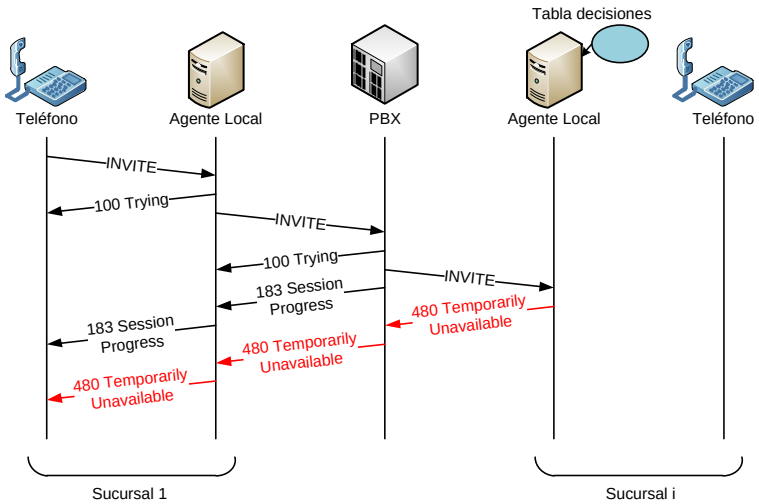


Fig. 7.8. Llamada entre sucursales rechazada

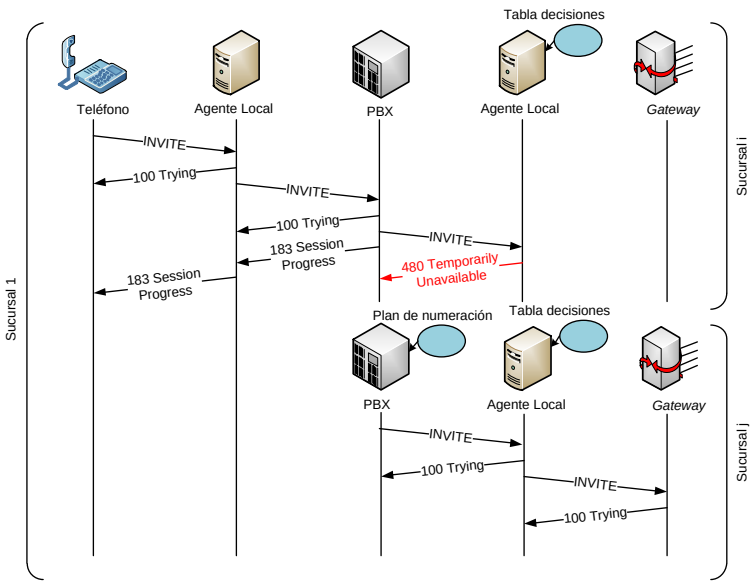


Fig. 7.9. La PBX intenta la llamada por el *gateway* de otra sucursal

Si la tabla de decisiones indica *redirigir*, el agente puede actuar también como *redirect server,* reencaminando la llamada a la sucursal donde haya líneas libres para establecer la llamada (Fig. 7.10). El *redirect server* envía mensajes del tipo *3xx*,

informando sobre la ruta alternativa, y entonces la PBX intenta cursar la llamada mediante el *gateway* de otra sucursal, sin necesidad de acudir a su plan de numeración. El agente que actuó como *redirect server* ya no vuelve a intervenir en la señalización de esa llamada.

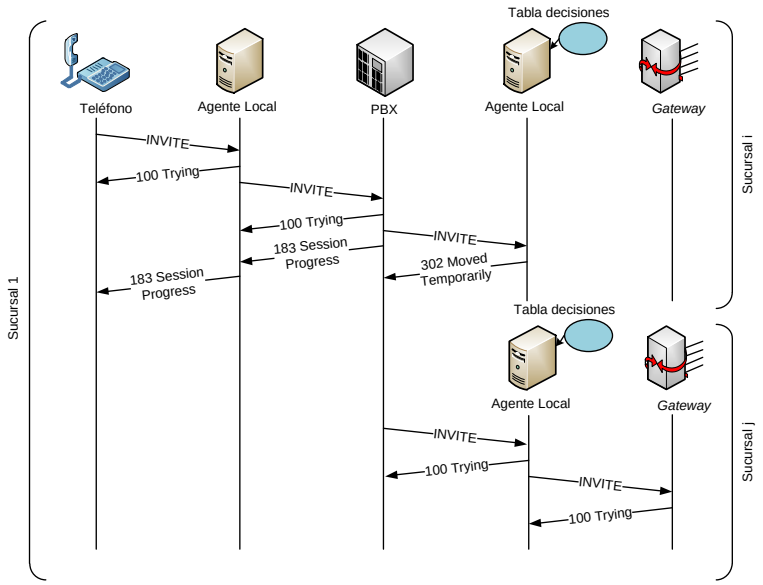


Fig. 7.10. Llamada redirigida de una sucursal a otra

Mejora esperada de la probabilidad de admisión

Para ilustrar el ahorro en costes y el incremento del grado de servicio de las llamadas a RTC al utilizar redirecciones, incluimos aquí un ejemplo utilizando un escenario simplificado. Los teléfonos de cada sucursal generan tráfico hacia RTC (AP_i) y hacia otras sucursales a través de la red IP (AI_{ij}). El tráfico de desbordamiento (AO_i) representa las llamadas que no pueden ser cursadas por el *gateway* local. N_i representa el número de líneas del *gateway*, y M_i el máximo número de llamadas que pueden ser aceptadas por el sistema CAC (Fig. 7.11).

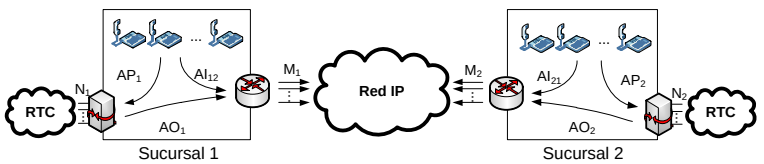


Fig. 7.11. Líneas compartidas del *gateway*

En este ejemplo simplificado supondremos que solamente hay dos sucursales. Sea $\gamma(A,N)$ la probabilidad de bloqueo del sistema con tráfico A y N servidores, P_{bRTC} y P_{bIP} las probabilidades de bloqueo de los *gateway* y del sistema CAC respectivamente. En caso de no compartir los *gateway,* la probabilidad de bloqueo de la sucursal 1 será:

$$P_{bRTC} = \gamma(AP_1, N_1) \tag{7.1}$$

$$P_{bIP} = \gamma(AI_{12} + AI_{21}, M_1) \tag{7.2}$$

Pero si los dos *gateway* se comparten entre las dos sucursales, el número de circuitos que se pueden utilizar para la conexión a RTC crece a $N_1 + N_2$. Así que las nuevas probabilidades de bloqueo serían:

$$P_{bRTC} = \gamma(AP_1 + AP_2, N_1 + N_2) \tag{7.3}$$

$$P_{bIP} = \gamma(AI_{12} + AI_{21} + AO_1 + AO_2\ , M_1) \tag{7.4}$$

Podemos hacer una simplificación llamada *Erlang Fixed Point*, que se usa para redes grandes con poca probabilidad de bloqueo. Asume que todos los tráficos tienen distribución de Poisson. Por tanto, la fórmula Erlang B podría utilizarse como la función γ para calcular las probabilidades de bloqueo. Utilizando esta simplificación, la probabilidad de bloqueo de los *gateway* mejora, ya que (7.3) será menor que (7.1), si suponemos que las dos sucursales tiene cantidades de tráfico y número de líneas similares.

Por otra parte, (7.4) será mayor que (7.2), ya que el número de llamadas que utilizan la red IP crecerá. Como hemos definido un número máximo de llamadas en nuestro sistema, éste rechazará llamadas con mayor probabilidad. Lógicamente, la simplificación asumida podría ser muy severa, de manera que se necesitarían más cálculos y simulaciones para obtener una función γ válida para todos los casos. Hemos utilizado esta simplificación para ilustrar las ventajas de compartir recursos entre las diferentes sucursales.

Dejando a un lado el ejemplo, se puede concluir que nuestro sistema establece un compromiso, que podemos modificar, y en el que están implicados, por un lado, la probabilidad de bloqueo en los *gateway* y, por otro, el tráfico de la red IP. Si se comparten los *gateway,* tendrán menos bloqueo pero al precio de aumentar el tráfico en la red IP, lo que, debido al cuello de botella que supone el acceso, puede perjudicar a la probabilidad de admisión del CAC. Si no se desea incrementar la probabilidad de bloqueo, existen dos soluciones: reducir el tráfico

interferente o incrementar el ancho de banda, que será probablemente más económico que incrementar el número de líneas del *gateway*.

Tablas de control

Como hemos visto, la tabla de decisiones de cada agente se construye a partir de otras tablas en las que se almacena la información de las líneas totales y disponibles en cada *gateway,* la tarificación y también, en el caso de usar el modo basado en medidas, la QoS entre las distintas sucursales.

1) *Tabla de líneas* de cada *gateway* (Tabla 7.2): Permite conocer las líneas RTC disponibles en cada sucursal. Tiene dos columnas: una con el número total de líneas de cada *gateway*, y otra con el número de líneas ocupadas. En el caso de usar el modo basado en medidas, el número total de líneas disponibles puede aumentar o disminuir según sea la QoS medida.

	Total Líneas	**Líneas Ocup.**
Gateway 1	TL 1	LO 1
Gateway 2	TL 2	LO 2
...	...	...
Gateway i	TL i	LO i

Tabla 7.2. Tabla de líneas

2) *Tabla de tarifas* (Tabla 7.3): En cada campo se incluye el tipo de tarifa con varios niveles, representados por un número entero, correspondiente a una llamada desde el *gateway* de la red *i* hasta el país *j*. Las llamadas locales tienen un nivel 1, y el nivel va subiendo con el precio.

	País 1	**País 2**	**...**	**País j**
Gateway 1	Tar 1→1	Tar 1→ 2	...	Tar 1→ j
Gateway 2	Tar 2→ 1	Tar 2→2	...	Tar 2→ j
...	...	...	...	...
Gateway i	Tar i→ 1	Tar i→ 2	...	Tar i→ j

Tabla 7.3. Tabla de tarifas

3) *Tabla de QoS* (Tabla 7.4): En cada sucursal existirá una tabla con las estimaciones de cada parámetro de QoS. La tabla no tiene por qué ser simétrica, puesto que existen redes de acceso a Internet que por definición son asimétricas. Los elementos de la diagonal no se definen, ya que representarían medidas dentro de una misma sucursal. Esta tabla sólo se utiliza en el modo basado en medidas.

	Agente 1	**Agente 2**	**...**	**Agente i**
Agente 1	-	par 1→ 2	...	par 1→ i
Agente 2	par 2→1	-	...	par 2→ i
...	...	...	...	...
Agente i	par i→ 1	par i→ 2	...	-

Tabla 7.4. Tabla de QoS

Según las medidas que se utilicen, podrían existir varias tablas con diferentes parámetros de QoS, como ancho de banda, OWD, *jitter,* tasa de pérdidas, etc. Estas medidas permitirán modificar el número de líneas disponibles, en la *tabla de líneas*.

Como ya hemos visto, la tabla de decisiones se construye a partir de las tablas de líneas, de tarifas y de QoS. La *tabla de decisiones* puede construirse a partir de la información de una sola sucursal, o bien teniendo en cuenta las medidas entre todas las sucursales.

En el caso de que cada agente conociera sólo los parámetros de su sucursal en relación con las demás, la *tabla de QoS* sería solamente un vector columna. La *tabla de líneas* quedaría reducida en ese caso a un contador del número de líneas libres en el *gateway* de la propia sucursal. Como cada tabla tendría una frecuencia de actualización diferente, la velocidad de las conexiones podría o no permitir a las diferentes sucursales compartir su información con el resto. La *tabla de tarifas* sí se podría tener completa en cada agente, puesto que su periodo de actualización será muy largo, del orden de días o semanas.

En el caso de que los agentes pudieran disponer de la información de todas las sucursales, la función que decide cuál es la mejor sucursal en cada momento para establecer una llamada, según las tarifas, la QoS y las líneas libres tendría mayor complejidad. Si el tiempo de procesado de las tablas creciese mucho, podrían dejar de tener validez por haber pasado mucho tiempo desde que se hicieron las medidas.

Los *proxy* SIP que hay en los agentes pueden actuar como *redirect server*, redirigiendo las llamadas (Fig. 7.10) a la sucursal por la que sea mejor establecer la llamada. Para poder hacer esto, es necesario disponer de información del resto de sucursales. De lo contrario, podría haber problemas como, por ejemplo, bucles en la señalización.

Conclusiones

En este capítulo se ha definido el diseño de un sistema de telefonía que responde a un esquema distribuido, similar al que ofrecen algunas soluciones comerciales. Está basado en el protocolo de señalización SIP. En cada una de las localizaciones de la empresa, además de los terminales de telefonía IP y del *gateway* que conecta con RTC, se incluye un agente local, que está al tanto de la señalización.

Este agente también se encarga de implementar un CAC, mediante el uso de un *proxy* SIP que tiene incluido, y limita el número de llamadas para garantizar una calidad mínima en el tráfico de voz. También se aprovecha el sistema CAC para redirigir llamadas de una sucursal a otra, cuando el *gateway* está totalmente ocupado, buscando siempre la opción con tarifa más económica.

El CAC basa sus decisiones en una tabla, que se gestiona desde una base de datos, y se actualiza en función del número de llamadas en curso, tarifas de cada zona y, en su caso, medidas de QoS, permitiendo así a los agentes locales realizar las funciones de CAC.

Capítulo 8

IMPLEMENTACIÓN DEL SISTEMA DE TELEFONÍA IP Y REALIZACIÓN DE PRUEBAS

Partes del contenido de este capítulo se han publicado en **"QoS Measurement-Based CAC for an IP Telephony system," Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering** 22, Springer 2009, cuyos autores son José Mª Saldaña, José I. Aznar, Eduardo Viruete, Julián Fernández-Navajas y José Ruiz-Mas. Otras partes se publicaron en **"QoS and Admission Probability Study for a SIP-based Central Managed IP Telephony System,"** Proc. 5th International Conference on New Technologies, Mobility and Security, **NTMS 2011**, Paris. Feb. 2011, cuyos autores son José Mª Saldaña, Jenifer Murillo, Julián Fernández-Navajas, José Ruiz-Mas, Eduardo Viruete y José I. Aznar.

En el capítulo anterior se ha definido la arquitectura de un sistema de telefonía IP distribuido. En el que comenzamos ahora se va a presentar su implementación en una plataforma de pruebas. Se ha buscado un entorno que se adapte bien al sistema de telefonía IP, y logre emularlo con realismo, permitiendo pruebas y medidas con flexibilidad.

Se ha recurrido por tanto a una plataforma de pruebas basada en virtualización Xen, presentada en el capítulo 6, pues al incluir máquinas virtuales, permite instalar aplicaciones de VoIP reales. La ventaja de utilizar máquinas virtuales es que las aplicaciones se pueden instalar sin apenas modificaciones, y que las implementaciones de los protocolos son reales, ya que las máquinas virtuales tienen instalados los mismos sistemas operativos que se usan en *hardware* real.

Para las pruebas que requieran un escenario más extenso se utilizará también simulación, que permite simplificar muchos de los aspectos del sistema, centrándose en los que se quiere estudiar.

En primer lugar describiremos la implementación realizada, y después presentaremos algunas pruebas que se han realizado, principalmente medidas de tiempos de procesado, tiempos de establecimiento y probabilidad de admisión.

Aplicaciones seleccionadas

Los elementos que componen el escenario propuesto (PBX, *softphone*, *proxy* SIP, *gateway* VoIP) deben tener poca carga computacional, dado que se van a utilizar

en un entorno de máquinas virtuales. Se usarán soluciones de *software* libre, que están pensadas para funcionar en sistemas reales. Por tanto, la solución que se va a implementar sobre máquinas virtuales sería fácilmente trasladable a un entorno real.

Centralita (PBX)

Para la PBX utilizaremos la versión 1.6. de Asterisk [Ast11], de Digium, que se está utilizando en muchos entornos por su flexibilidad, actualizaciones y por su distribución bajo licencia GNU-GPL. Además de SIP, soporta otros protocolos como H.323, IAX y MGCP.

Se ha utilizado un plan de numeración que permite redirigir la llamada a otra sucursal en el caso de que no sea aceptada por el *gateway* seleccionado como primera opción.

En el caso de plantearnos un escenario en el que pudiésemos actuar sobre la PBX, se debe tener en cuenta que Asterisk dispone de la denominada *Asterisk Realtime Architecture (ARA)* [MSM05], que proporciona un método para almacenar los ficheros de configuración en una base de datos MySQL o PostgreSQL. Mientras el uso *estático* del plan de numeración requiere hacer un *reload* después de cada cambio, la utilización del tiempo real *dinámico* hace que Asterisk acceda a la base de datos según lo necesita, actualizándose así sus ficheros de configuración en tiempo real. El plan de numeración cambiaría dinámicamente en función de los parámetros de QoS medidos por los agentes, las líneas ocupadas y la tarificación.

Proxy SIP

Necesitamos también un *proxy* SIP que tenga opciones de *redirect server*, y que se pueda programar para implementar de este modo las decisiones de CAC. Debe ser capaz de consultar información externa en una base de datos. La solución elegida ha sido OpenSIPS [Ope11], un *proxy* SIP realizado con *software* libre, continuación del proyecto OpenSER. Dispone de funciones de *registrar server, location server, proxy server* y *redirect server*. También presenta poca carga computacional, y permite añadir o quitar funcionalidades mediante el uso de módulos. A nivel de transporte, soporta UDP, TCP, TLS y SCTP. A nivel de red se puede usar con IPv4 e IPv6. Para su configuración se debe manipular el archivo *opensips.cfg*, en el que se especifica cómo actuar en función de cada mensaje que se reciba. Se hace mediante programación en un lenguaje propio, de

alto nivel. OpenSIPS también permite la posibilidad de acceder a una base de datos MySQL.

La autenticación, autorización y gestión de cuentas (AAA) se pueden llevar a cabo mediante una base de datos (MySQL, PostgreSQL), ficheros de texto, protocolos RADIUS o DIAMETER. La versión de OpenSIPS utilizada es la 1.4.

Softphone (Teléfono por software) y Gateway

Utilizamos el *softphone* PJSUA [PJS11], que funciona por interfaz de comandos. Forma parte del proyecto PJSIP, que ofrece bajo licencia GPL el *software* de una pila SIP completa. Tiene poca carga computacional, ocupando menos de 150 kB para lograr una implementación completa del protocolo SIP.

Dispone de soporte para llamadas múltiples, llamadas en espera, mensajería, protocolos UDP, TCP, TLS y SRTP. Soporta diversos *codec*, aunque en nuestro caso, por motivos prácticos, no lo vamos a utilizar para generar el flujo multimedia, que obtendremos mediante un generador de tráfico. También dispone de opciones para atravesar los NAT, por medio de ICE o STUN. Se ha usado la versión 1.0.

Por último, necesitamos una manera para emular los *gateway* de un sistema de telefonía IP. Hay que tener en cuenta que en nuestra plataforma no disponemos de conexiones reales con RTC. La solución adoptada ha sido el uso de PJSUA, ya que dispone de la posibilidad de establecer varias llamadas simultáneas, limitando ese número al de líneas del *gateway* a emular. De esta manera, cuando se encuentren ocupadas todas las líneas, se considerará que ese *gateway* está totalmente ocupado, y se rechazarán las llamadas.

El *codec* a utilizar en transmisión será el G.729a con dos muestras por paquete. Esto supone un tiempo de paquetización de 25 ms, que incluye 10 ms por cada muestra, más otros 5 ms de *look-ahead*. Cada flujo envía un paquete cada 20 ms, y el paquete es de 20 bytes de *payload*, más 40 bytes que corresponden a las cabeceras IP, UDP y RTP. Por tanto, el ancho de banda a nivel IP es de 24 kbps (60 bytes / 20 ms). A nivel *Ethernet* el ancho de banda será de 29,6 kbps, pues se añaden otros 14 bytes a cada paquete.

Pruebas realizadas en la plataforma

Una vez descritas las aplicaciones que hemos utilizado para implementar el sistema, pasaremos a presentar algunas pruebas realizadas para validarlo. Los

resultados deben mantenerse dentro de lo aceptable para ofrecer una buena calidad al usuario.

Como se ve en la Fig. 8.1, los mensajes SIP pasan siempre por los agentes locales, y también por la PBX cuando se debe establecer una comunicación entre sucursales distintas. Pero para el tráfico RTP se ha optado por el envío directo desde un terminal a otro, sin pasar por la PBX. Así pues, como se aprecia en la figura, en las pruebas que hagamos con el tráfico de señalización deberemos tener en cuenta que pasa por la PBX, mientras que en las del tráfico de voz RTP, sólo se deberá contar el retardo desde una sucursal a otra.

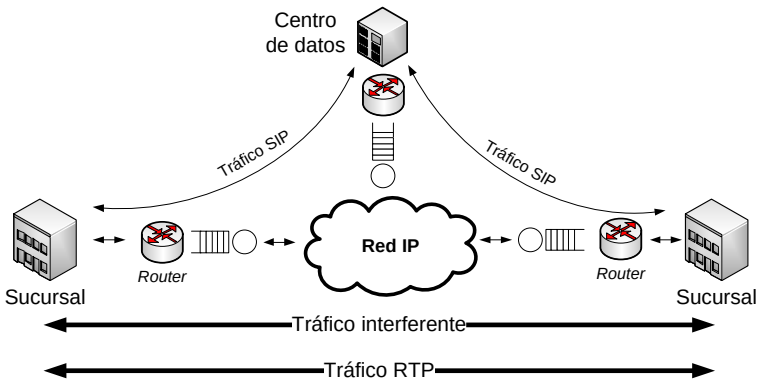


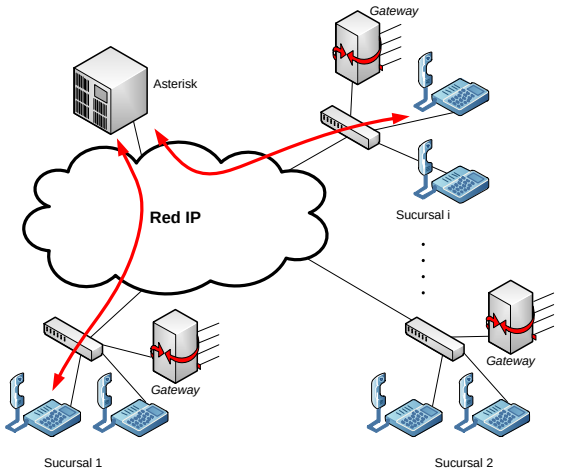
Fig. 8.1. Esquema de las pruebas

Flujo del protocolo

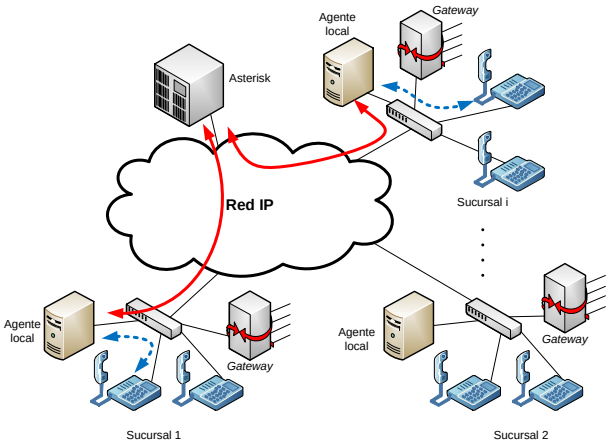
En primer lugar, se han incluido en el entorno de pruebas todos los elementos del sistema, construyendo así un modelo con tres sucursales en el entorno virtual, donde se ha podido comprobar que los protocolos funcionan según el diseño previsto en el capítulo anterior.

En este apartado se identifican los diferentes retardos que sufren los mensajes, para ver cuáles dependen de la red, cuáles están en función de la capacidad de proceso de las máquinas, y cuáles se añaden al incluir el sistema CAC. Medir estos tiempos directamente en la plataforma de pruebas no tendría sentido, pues por un lado los elementos de red que conectan las máquinas son virtuales, y no tienen las limitaciones de ancho de banda que aparecerían si se usaran elementos reales. Por otro lado, los tiempos de procesado dependerán de la plataforma concreta en que se implemente el sistema en cada caso.

Se han comparado dos casos diferentes: si no se utiliza CAC, es decir, si los *softphone* se comunican directamente con la *PBX* (Fig. 8.2. a) y si se utiliza CAC, en cuyo caso los mensajes de señalización deben atravesar también los dos *proxy*, tanto de la sucursal origen como de la destino (Fig. 8.2. b). Se ha supuesto que el sistema CAC no utiliza un método reactivo de estimación de parámetros de QoS. Esto añadiría retardos, por tener que realizar medidas cada vez que apareciese una petición de establecimiento de llamada.



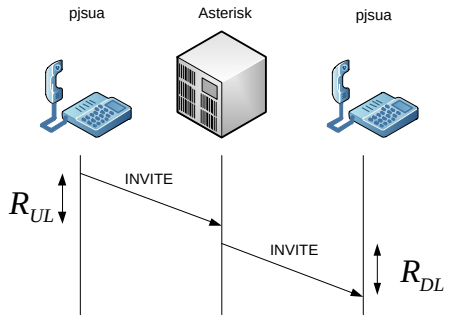
(a)



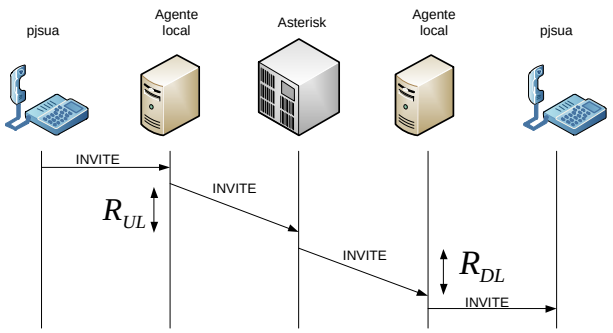
(b)

Fig. 8.2. Esquema del sistema a) sin CAC; b) con CAC

Definimos T_{Psc}, como el tiempo total de procesado cuando no se usa CAC, y T_{Pac}, como el tiempo de procesado usando CAC. Además, denominaremos R_{UL} al retardo de red en el *uplink*, y R_{DL} al retardo de red en el *downlink*. Al introducir el sistema CAC, estamos añadiendo los agentes locales, es decir, dos nuevas máquinas en el camino a seguir por los paquetes de señalización. Esto provocará que en cada mensaje se añada dos veces el tiempo de retardo en una red local (Fig. 8.3), que consideraremos despreciable en comparación con los tiempos de propagación en Internet.



(a)



(b)

Fig. 8.3. Establecimiento de llamada a) sin CAC; b) con CAC

Una medida importante es el retardo entre el envío del primer *INVITE* y el comienzo del tono de llamada en el destino, que consideraremos como el tiempo de establecimiento. En un sistema sin CAC este retardo será:

$$R_{\text{sin}CAC} = R_{UL} + R_{DL} + T_{psc} \tag{8.1}$$

Si se considera despreciable el tiempo de propagación en la red local, el retardo con el sistema CAC será:

$$R_{conCAC} = R_{UL} + R_{DL} + T_{pcc} \tag{8.2}$$

Restando (8.1) y (8.2), obtenemos el retardo introducido por el sistema CAC:

$$R_{conCAC} - R_{\sin CAC} = T_{pcc} - T_{psc} \tag{8.3}$$

Cuando una llamada es redirigida de una sucursal a otra, según la decisión del sistema CAC, el agente local actúa como *redirect server,* comunicando a la PBX la sucursal por la que puede establecer la llamada.

El hecho de redirigir una llamada supone un retardo total de procesado que denominaremos T_{Pre}. Además de este retardo adicional, debemos considerar el retardo de transmisión de la señalización de la llamada a través de cada uno de los enlaces de las sucursales involucradas en la redirección hasta la PBX. Este retardo depende del escenario considerado y del número de redirecciones que sufra la llamada. Si definimos R_{ULi} como el retardo en el *uplink* de la sucursal *i*, y R_{DLi} como su retardo en el *downlink,* podemos concluir que el retardo con N redirecciones sería el siguiente:

$$R_{redir} = T_{pcc} + N \cdot T_{p\mathrm{Re}} + R_{UL_1} + R_{DL_{N+2}} + \sum_{i=2}^{N+1}(R_{DL_i} + R_{UL_i}) \tag{8.4}$$

En función de los retardos de cada sucursal y del número de redirecciones, R_{redir} puede llegar a resultar inadmisible. Es necesaria, por tanto, su medida en cada entorno particular, con sus retardos de red y de procesado, para asegurar el buen funcionamiento del sistema CAC.

Parámetros de calidad

Se han realizado algunas medidas para encontrar los parámetros que determinan la calidad subjetiva, que principalmente son el retardo, las pérdidas y el *jitter*. También se han utilizado los resultados obtenidos para calcular el Factor R según el E-Model de la ITU [ITU03].

En estas medidas se ha estudiado solamente la parte del sistema en que se realiza la transmisión del flujo RTP, es decir, se ha enviado tráfico desde una máquina que emula la transmisión por parte de un *softphone*, atravesando los *router* de la sucursal origen y destino, hasta la máquina que emula el *softphone* destino (Fig.

8.4). Como se puede ver, una máquina genera, usando la aplicación D-ITG [BDP07], tráfico de voz y tráfico de fondo, que comparten el mismo *router*. Posteriormente, mediante la herramienta *Traffic Control (tc),* explicada en capítulos anteriores, el *router* es capaz de emular diferentes comportamientos del *buffer*. En este capítulo usaremos dos políticas diferentes: en primer lugar, un *buffer* de alta capacidad, que se emulará con una política *tbf* limitada en tiempo a 800 ms. En segundo lugar, un *buffer* limitado a 60 ms, que se emulará con esa misma política. En ambos casos el ancho de banda será de 1 Mbps.

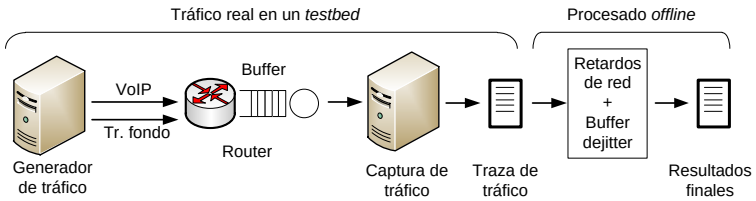


Fig. 8.4. Esquema de las medidas

El tráfico de fondo tiene la siguiente distribución de tamaños: el 50% de los paquetes son de 40 bytes a nivel IP, el 10% de 576 bytes, y el 40% de 1.500 bytes [Nas05]. Este tráfico se ha utilizado para saturar el acceso de cada sucursal. Cada prueba dura 400 segundos, en los que el tráfico RTP y el de fondo comparten el enlace. Los 20 segundos iniciales y finales se descartan para el cálculo de los resultados.

Una vez capturado el tráfico en el receptor, en la fase de procesado *offline,* se añade a cada paquete un tiempo de retardo en la red. Se ha usado el modelo propuesto en [KPLK+09], que está basado en los resultados de algunos proyectos de medidas globales [Pin10]. Añade en primer lugar un retardo fijo causado por la distancia geográfica entre los nodos, y después otro variable con distribución log-normal. Se han usado los siguientes valores: 20 ms para el retardo fijo, y también 20 ms, con varianza 5, para el retardo variable. Por tanto, el retardo medio de red es de 40 ms, que es un valor habitual en Internet [ATT11] para un escenario intrarregional.

También se añaden *offline* los efectos del *buffer* de *dejitter*, que se traducen en más retardo, según el número de muestras que almacena, y en pérdidas de paquetes, que corresponden a los que llegan demasiado tarde como para poder ser reproducidos. Se ha usado la aproximación propuesta en [CR00] y explicada en el capítulo 3, para evitar ceñirnos a una implementación concreta de este *buffer*.

La Fig 8.5 muestra los resultados del Factor R obtenidos cuando se usa el *buffer* de alta capacidad. Se representa el estimador de calidad en función de la cantidad total de tráfico de fondo. Vemos que el comportamiento tiene forma de escalón: cuando se alcanza el límite del ancho de banda, la calidad de la conversación cae con rapidez.

En cambio, en la Fig. 8.6, que muestra los resultados para el *buffer* limitado en tiempo, vemos que las gráficas presentan una pendiente más suave, es decir, el Factor R desciende más despacio según aumenta el tráfico de fondo. Vemos que se pueden obtener valores aceptables de R (por encima de 70) para mayores cantidades de tráfico de fondo que en el caso anterior.

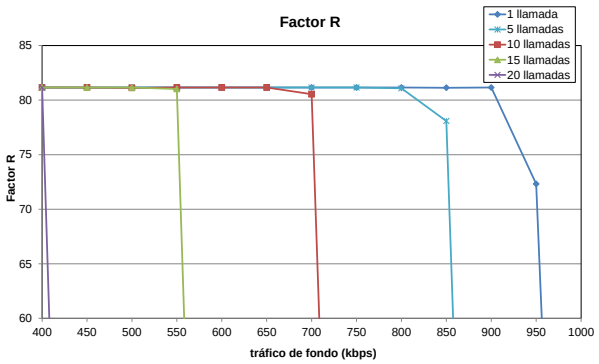


Fig. 8.5. Factor R para el *buffer* de alta capacidad

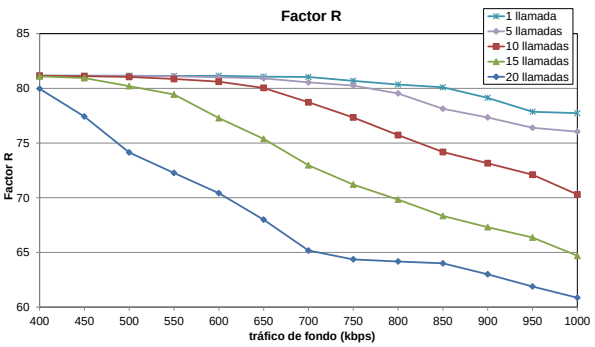


Fig. 8.6. Factor R para el *buffer* limitado en tiempo

Por un lado, el *buffer* limitado en tiempo es capaz de mantener acotado el máximo retardo de los paquetes de voz, pero por otro incrementa las pérdidas de paquetes para el tráfico de fondo. La Fig. 8.7 muestra las pérdidas de paquetes para un

valor de tráfico de fondo de 800 kbps, usando el *buffer* limitado en tiempo. Puede verse que según el tráfico de fondo se acerca al límite (1 Mbps), los paquetes comienzan a ser descartados, y que los paquetes grandes lo hacen en un porcentaje mayor. Esto comienza a notarse más cuando el número de flujos se sitúa por encima de 8, puesto que en ese caso ya tenemos más de 200 kbps de tráfico de voz, y empezamos a superar el límite de ancho de banda. La Fig. 8.8 muestra que en esta misma situación los paquetes pequeños mantienen su ancho de banda, mientras que los grandes son descartados en un alto porcentaje, ya que tienen una mayor probabilidad de no tener sitio en el *buffer*.

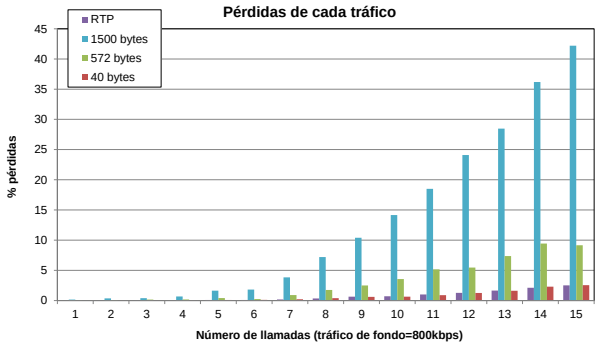


Fig. 8.7. Pérdidas de cada tráfico en función del número de llamadas para el *buffer* limitado en tiempo

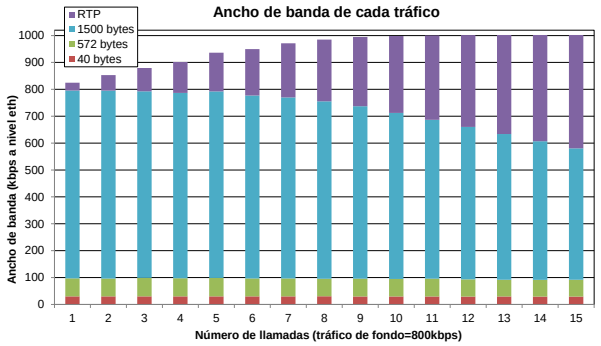


Fig. 8.8. Ancho de banda de cada tráfico en función del número de llamadas para el *buffer* limitado en tiempo

Este resultado es interesante, pues muestra que, para este tipo de *buffer,* el tráfico RTP está protegido a causa de su pequeño tamaño. Por tanto, limitar el máximo número de llamadas simultáneas es importante para evitar la degradación del

resto de tráfico de la sucursal. Esto implica que el CAC no sólo sirve para proteger el tráfico de voz, sino también para evitar que el resto de tráfico experimente una degradación.

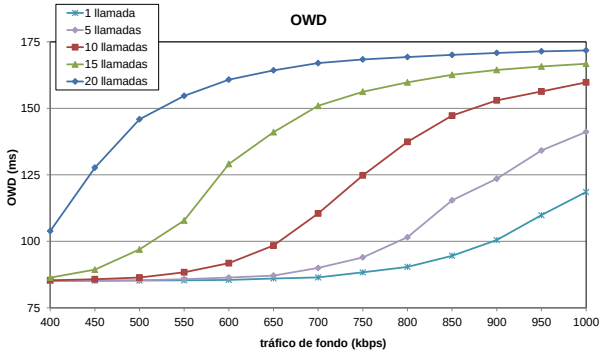
Ya hemos visto que el *buffer* limitado en tiempo presenta características muy interesantes para mantener la calidad del tráfico de voz. Veamos ahora algunos otros resultados para este *buffer*. La Fig. 8.9 muestra el OWD, las pérdidas y el *jitter*, medido como IPDV *(Instantaneous Packet Delay Variation)*. Estos valores serían la cota superior para los parámetros representados, en el caso de establecer el límite del CAC al número de llamadas representado. Las gráficas se representan en función del tráfico de fondo, por lo que cada una de ellas alcanza la congestión en un momento diferente, cuando el tráfico total ofrecido supera el límite de 1 Mbps.

En las gráficas se puede observar que, antes de llegar a la congestión, el OWD tiene más influencia en el Factor R que las pérdidas. Pero una vez alcanzada la congestión, los dos parámetros pasan a influir de modo similar. Pero el hecho de que el *buffer* tenga un límite temporal hace que el OWD se mantenga por debajo de los 175 ms, mientras que las pérdidas crecen indefinidamente.

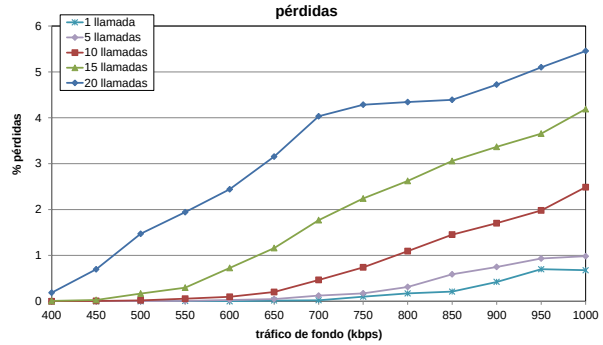
En la Fig. 8.9.c) vemos que el IPDV presenta un máximo en el momento de la saturación, pero que después de él su valor decrece. La causa es que la cola está habitualmente llena, por lo que todos los paquetes experimentan aproximadamente el mismo retardo, siendo su variación muy pequeña.

Pruebas realizadas mediante simulación

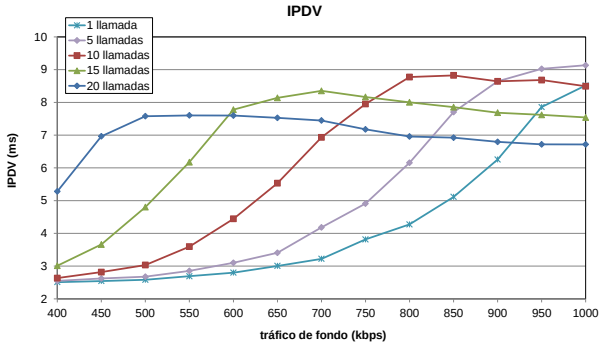
Acabamos de explicar la obtención en el *testbed* de los parámetros de QoS para cada número de llamadas. Ahora, utilizando un sistema híbrido para las pruebas, combinaremos la emulación ya realizada con la simulación. De esta forma, los resultados pueden extenderse a un escenario con un número de máquinas mayor. Todas las pruebas de este apartado se han realizado usando el CAC en el modo basado en parámetros.



(a)



(b)



(c)

Fig. 8.9. Parámetros de calidad para el *buffer* limitado en tiempo: a) OWD; b) pérdidas; c) IPDV

En primer lugar, explicaremos el sistema empleado para las pruebas. Cada prueba se divide en las siguientes etapas, como se ve en la Fig. 8.10:

- Emulación, vista en el apartado anterior, en la que se usan tres máquinas. En primer lugar, el generador de tráfico envía paquetes RTP y tráfico de fondo. Después, estos tráficos atraviesan el *router*, que emula diferentes políticas de *buffer*. Finalmente, una máquina recibe y almacena el tráfico.

- Procesado *offline*, en el que se añaden algunos retardos al tráfico capturado en la fase anterior.

- Simulación del escenario global y resultados finales. Se usa Matlab para simular el escenario y se utilizan los resultados anteriores para obtener la probabilidad de admisión y el Factor R de las llamadas.

- Los resultados finales se obtienen integrando los procedentes de la simulación con los del *testbed*.

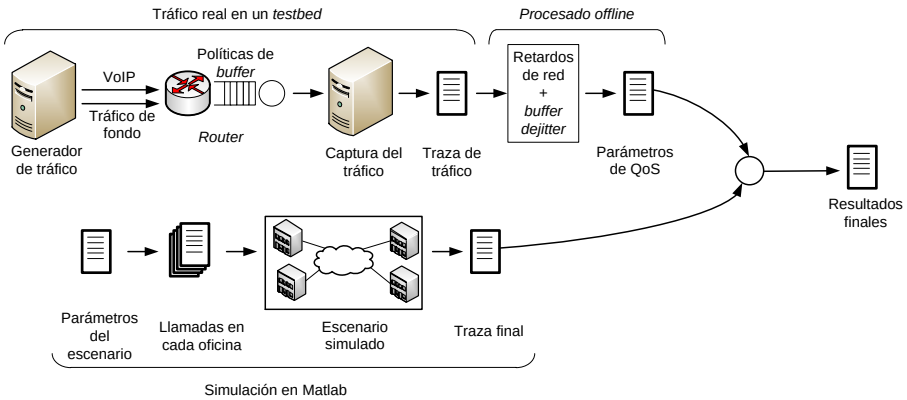


Fig. 8.10. Diagrama del sistema de medidas

Se ha utilizado Matlab para generar realizaciones con diferentes parámetros: número de oficinas, usuarios, líneas de los *gateway*, países, zonas geográficas, retardos de establecimiento, etc. Cada realización obtiene la siguiente información de los diferentes tramos de cada llamada: instante de comienzo, duración y extensiones implicadas. Las llamadas se generan según una distribución de Poisson, para la que se define una tasa de llamadas por hora λ para cada usuario. Su duración se ha modelado con una distribución normal de media 180 seg. y varianza 30. Finalmente, se utilizan diferentes probabilidades para generar el destinatario de la llamada.

Tiempos de establecimiento

En este apartado se utilizará el escenario simulado para medir retardos de establecimiento. La Fig. 8.11 ilustra los retardos que se deben tener en cuenta. Mediremos el tiempo desde que el terminal origen de la llamada envía el *INVITE* hasta que llega a su destino. Los retardos de procesado obtenidos anteriormente se incluirán dentro del retardo total.

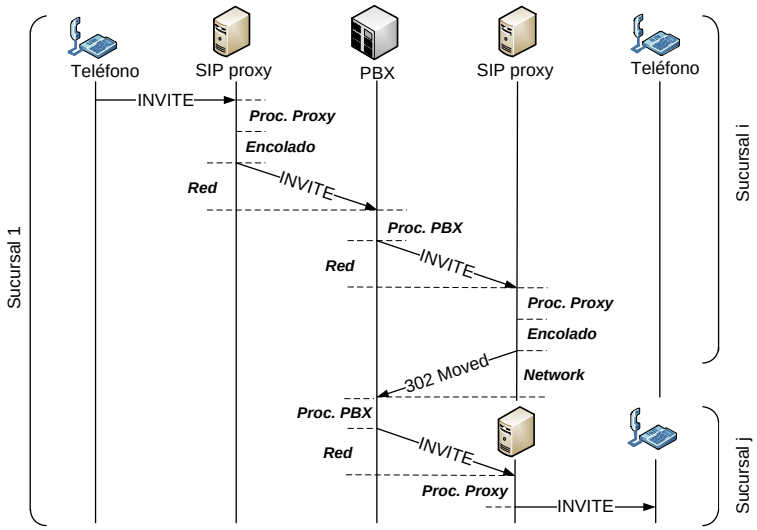


Fig. 8.11. Componentes del retardo de establecimiento de una llamada interna (es decir, entre dos sucursales de la empresa)

En la simulación se tendrán en cuenta los siguientes retardos. Cada uno se cuenta un número de veces, según los valores de la Tabla 8.1:

- Retardo de red: se ha calculado usando el mismo método que para los paquetes RTP. El retardo de las redes locales se considerará despreciable para las redes origen y destino, dado que suelen ser de alta velocidad en el escenario que se está considerando.

- Retardo de procesado: Se utilizará un valor de 5 ms, tanto para el *proxy* como para la *PBX*.

- Retardo de encolado en el *router* origen: Se ha estimado que tendrá el mismo valor que el retardo experimentado por los paquetes RTP, puesto que ambos tipos de tráfico comparten el mismo *buffer*. Por tanto, los valores usados son los obtenidos con el *testbed*. No se ha considerado

este retardo en el *router* destino, ya que los paquetes pasan de una red lenta a otra rápida. Por último, no se ha considerado el retardo en la PBX, porque se ha supuesto en el diseño del sistema que el centro de datos tiene una conexión rápida a Internet.

	Llamada sucursal	**Llamada interna**	**Cada redirección**
Retardo de red	0	2	2
Retardo procesado *proxy*	1	2	1
Retardo procesado PBX	0	1	1
Retardo encolado	0	1	1

Tabla 8.1. Componentes del retardo de establecimiento

El entorno de simulación se ha usado para realizar diferentes pruebas modificando el número de sucursales en el escenario. Cada una incluye 25 usuarios. Cada *gateway* tiene 6 líneas. Las llamadas siguen una distribución de Poisson con $\lambda = 4$ llamadas por hora y usuario. El límite del CAC son 6 llamadas. La Fig. 8.12 muestra el retardo de establecimiento obtenido con diferentes números de sucursales y diferentes retardos de red (RTT), desde 25 hasta 125 ms, que son valores típicos que pueden darse en Internet [ATT11].

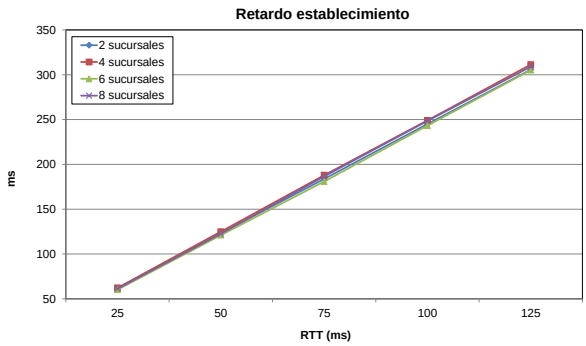


Fig. 8.12. Retardo de establecimiento en función de RTT

En la figura puede observarse que el retardo depende principalmente del RTT, que aumenta linealmente con él, y que es prácticamente independiente del número de sucursales.

Probabilidad de admisión

En primer lugar, para mostrar los beneficios que se obtienen al compartir los *gateway* entre las distintas sucursales, se han realizado unas simulaciones en las que cada realización se puede ejecutar utilizando dos algoritmos. El primero, denominado *aislado,* se usa como referencia. No implementa el sistema CAC, y cada sucursal es independiente del resto, por lo que el sistema no comparte los *gateway* y no se redirigen llamadas. Por tanto, todas las llamadas que van a RTC no pueden utilizar Internet. El segundo algoritmo, denominado *compartido*, simula un sistema centralizado, que comparte todas las líneas de sus *gateway* entre las sucursales e implementa un sistema CAC, permitiendo al agente local redirigir llamadas a otras oficinas, buscando siempre la tarifa más barata. Con este algoritmo se trata de usar Internet para el máximo número de llamadas posible.

El modo *compartido* se usa para intercambiar la probabilidad de bloqueo en los *gateway* por probabilidad de bloqueo en el acceso a Internet, ya que normalmente es más barato contratar ancho de banda que aumentar el número de líneas. Por tanto, es de esperar que la probabilidad de admisión aumente a causa de las redirecciones, y que se consiga un ahorro de costes en llamadas internacionales. Sin embargo, incrementar la probabilidad de admisión también tiene una contrapartida, y es que se introduce más tráfico en el acceso a Internet, por lo que la QoS puede verse afectada.

En estas pruebas se ha calculado la probabilidad de admisión en los modos *aislado* y *compartido*. En el modo *compartido*, el sistema CAC estará basado en parámetros, es decir, simplemente contará las llamadas, manteniéndolas por debajo de un máximo.

Para lograr estos resultados se han requerido varias realizaciones en las que todas las llamadas tenían como destino RTC, es decir, las líneas gestionadas por los *gateway* de las sucursales. Los parámetros que hemos variado han sido la tasa de generación de llamadas λ y el número de sucursales en cada país.

La Fig. 8.13 muestra cómo al usar el modo *compartido*, a mayor número de oficinas se consigue una mayor probabilidad de admisión. Se han incluido 25 usuarios por sucursal, 6 líneas en los *gateway* y un límite del CAC de 6 llamadas. Esto se corresponde con la idea de la fórmula de Erlang de que se consigue más probabilidad de admisión cuantos más recursos se comparten. Naturalmente, si λ aumenta, la probabilidad de admisión se reduce.

Hemos realizado otras pruebas variando el límite del CAC. En la Fig. 8.14 ($\lambda = 3$ llamadas por hora, 25 usuarios por sucursal, 6 líneas por *gateway)* se puede ver que el modo *compartido* da mejores resultados para la probabilidad de admisión que el modo *aislado*, que confirma la idea de que compartir las líneas es beneficioso. Por otro lado, vemos que según aumenta el límite del CAC, en el modo *compartido* aumenta la probabilidad de admisión, mientras que en el modo *aislado* no varía. Concretamente, en el caso de tener dos sucursales podemos ver que la probabilidad de admisión crece hasta que el límite del CAC llega a 6, pero después el valor permanece constante, porque la limitación pasa a deberse a las líneas de los *gateway*. Además, cuando el límite del CAC aumenta en el modo *compartido*, hay más llamadas que utilizan Internet, debido al hecho de que las líneas son compartidas.

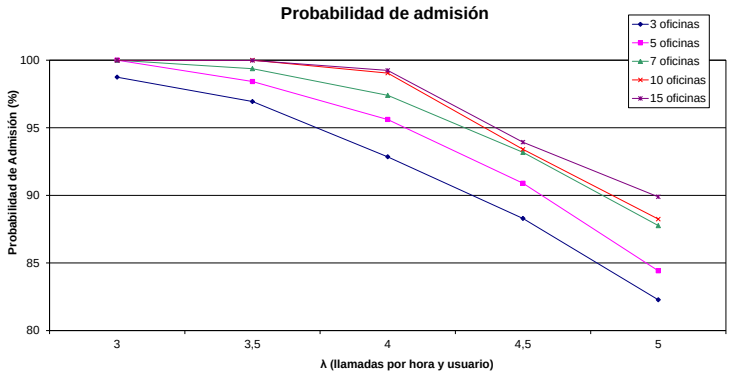


Fig. 8.13. Probabilidad de admisión en modo *compartido*

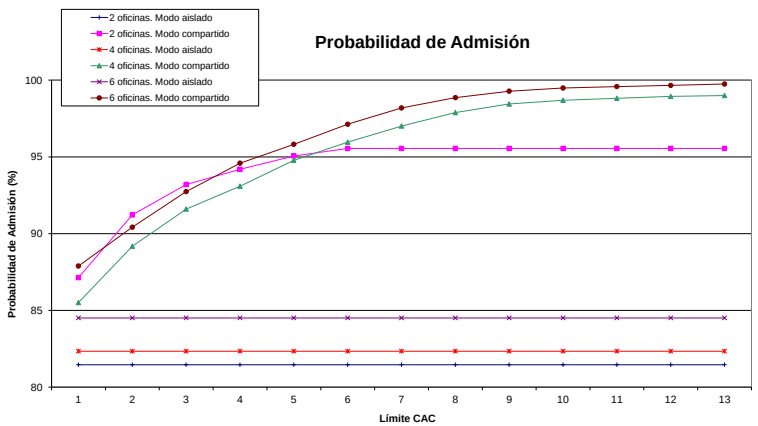


Fig. 8.14. Probabilidad de admisión en modos *aislado* y *compartido*

Método de configuración del sistema CAC

Se ha podido comprobar, con lo visto hasta ahora, que existe un compromiso entre la QoS de las llamadas y la probabilidad de admisión del sistema. Por tanto, los pasos a dar para configurar el sistema CAC serían los siguientes (Fig. 8.15): en primer lugar, se deberían medir los tráficos interferente y telefónico de la sucursal en la Hora Cargada; posteriormente, se decide el valor mínimo del Factor R o MOS que consideraremos aceptable. Después, utilizando los resultados, especialmente los de la Fig. 8.5 u 8.6, según sea nuestro *buffer*, se obtiene el máximo número de llamadas simultáneas que podemos permitir en el sistema. Finalmente, usando un método tradicional, como por ejemplo las tablas de Erlang, o bien las simulaciones presentadas en las Fig. 8.13 u 8.14, se calcula la probabilidad de admisión del sistema. Si este valor resulta inaceptable, la solución pasará por reducir el tráfico de fondo o incrementar el ancho de banda.

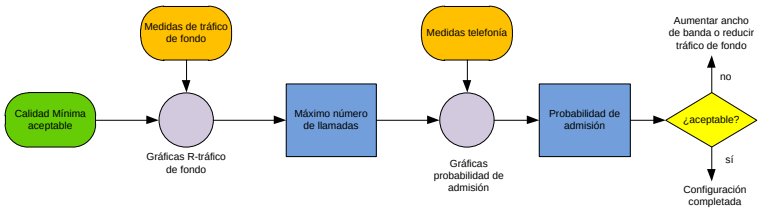


Fig. 8.15. Diagrama de flujo para la configuración del sistema CAC

Conclusiones

En este capítulo hemos comenzado implementando el sistema de telefonía en una plataforma de pruebas. Se han buscado aplicaciones de *software* libre para poder ponerlo en marcha en máquinas virtuales, y así comprobar su funcionamiento, realizando medidas de tiempos de procesado y parámetros de calidad en la conversación. Posteriormente, se ha trasladado el sistema a un entorno de simulación para poder medir tiempos de establecimiento y probabilidad de admisión en función del número de sucursales, límites del CAC, etc.

Las medidas de parámetros de calidad se han traducido a resultados en términos del Factor R, y se ha mostrado cuándo el sistema puede funcionar adecuadamente, sin añadir retardos que resulten inaceptables para los usuarios. También se ha visto que los tiempos de establecimiento se mantienen dentro de límites razonables, y que dependen principalmente del retardo en la red.

Para unos niveles mínimos de R, se ha visto cómo el sistema permite intercambiar probabilidad de bloqueo entre los *gateway* y la red IP, controlada por el CAC: si aumentamos el límite de llamadas permitidas, aumentará el tráfico en la red IP, pero también aumentará la posibilidad de compartir las líneas externas que controlan los *gateway*, con lo cual podremos ahorrar más.

Por último, en el entorno de simulación se ha mostrado cómo la probabilidad de admisión mejora cuando se comparten los *gateway* de las diferentes sucursales, mediante la gestión centralizada de todo el sistema.

Capítulo 9

EVALUACIÓN DE ESQUEMAS DE OPTIMIZACIÓN DE TRÁFICO PARA SU INCLUSIÓN EN EL SISTEMA DE TELEFONÍA IP

Partes de este capítulo se publicaron en el artículo **"Evaluation of Multiplexing and Buffer Policies Influence on VoIP Conversation Quality,"** Proc. **CCNC 2011** - 3rd IEEE International Workshop on Digital Entertainment, Networked Virtual Environments, and Creative Technology, pp 1147-1151, Las Vegas. Ene. 2011, cuyos autores son José Mª Saldaña, Jenifer Murillo, Julián Fernández-Navajas, José Ruiz-Mas, Eduardo Viruete y José I. Aznar.

Otras partes se publicaron en **"Influence of the Distribution of TCRTP Multiplexed Flows on VoIP Conversation Quality,"** Proc. **CCNC 2011**. The 8th Annual IEEE Consumer Communications and Networking Conference - Work in Progress papers, pp 711-712, Las Vegas. Ene. 2011, con los mismos autores.

En los capítulos anteriores se ha diseñado e implementado un sistema de telefonía IP pensado para una empresa con sucursales en diferentes países. Si el número de sucursales y usuarios es elevado, puede ocurrir que haya llamadas que se cursen simultáneamente entre el mismo par de sucursales. Esto nos lleva a plantearnos la posibilidad de recurrir a técnicas de optimización del tráfico, como puede ser la multiplexión de varios flujos utilizando un túnel. En el capítulo 5 vimos los escenarios en que se puede aplicar, así como algunas de las propuestas de multiplexión existentes.

En el capítulo que comenzamos ahora vamos a estudiar analíticamente, y también con medidas en la plataforma de pruebas, una de las propuestas de multiplexión, con la idea de evaluar su utilidad para incluirla en el sistema de telefonía IP.

Se ha seleccionado la propuesta TCRTP [TKW05]. La principal razón es que se trata de un estándar del IETF (RFC 4170) y que, por tanto, es una solución públicamente disponible y conocida. De todas formas, en algunas de las pruebas se ha evaluado también la solución propuesta en [SLLY02], que denominaremos "Sze", para así poder comparar la solución elegida con otra de las encontradas en la literatura.

Realizaremos en primer lugar un análisis teórico de la multiplexión. Pasaremos después a explicar la metodología de las pruebas, cuyos resultados se presentarán posteriormente. En el último apartado se discutirán los resultados.

Análisis teórico de las técnicas de multiplexión

En este apartado se presenta un estudio analítico de la influencia de la multiplexión en tres parámetros: tamaño de los paquetes, eficiencia en términos de relación de ancho de banda, y paquetes por segundo generados. El último subapartado se dedicará a resumir los compromisos derivados de la multiplexión.

Tamaño de los paquetes

Aunque ya se ha visto en capítulos anteriores, por claridad reproducimos aquí en la Fig. 9.1 el esquema de un paquete multiplexado, para poder explicar mejor sus diferentes partes. De acuerdo con la figura, podemos definir los siguientes parámetros en los que se basará el análisis:

- **CH:** Cabecera Común *(Common Header)*: Es el tamaño de la cabecera compartida por todo el paquete multiplexado.

- **MH:** Cabecera de Multiplexión *(Multiplexing Header)*: Corresponde a los bytes del protocolo PPPMux que se incluyen antes de la cabecera reducida de cada paquete RTP.

- **RH:** Cabecera Reducida *(Reduced Header)*: Es el tamaño de la cabecera reducida que precede a las muestras de cada flujo RTP. Los protocolos de compresión producen cabeceras comprimidas de diferentes tamaños, por lo que se calculará su valor esperado $E[RH]$, teniendo en cuenta la probabilidad de obtener una cabecera de cada tamaño.

- **S:** Tamaño de las muestras *(Samples)*: Tamaño en bytes de las muestras que lleva cada paquete RTP.

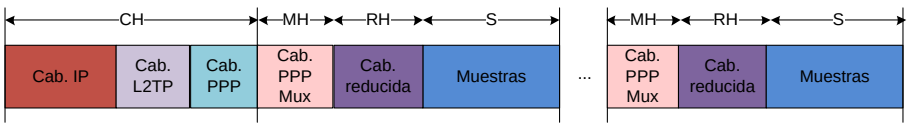


Fig. 9.1. Esquema de un paquete multiplexado

Denominaremos **NH** *(Normal Header,* Cabecera típica), al tamaño de las cabeceras IP/UDP/RTP de un paquete nativo RTP. Por tanto, su tamaño será:

$$PS_{nativo} = NH + S \tag{9.1}$$

Si definimos k como el número de flujos multiplexados, entonces el valor esperado del tamaño de un paquete multiplexado será:

$$E[PS_{k\ flujos}] = CH + k\,(MH + E[RH] + S) \tag{9.2}$$

Necesitamos, por tanto, calcular $E[RH]$. TCRTP define tres tipos de cabeceras, cada una con un tamaño diferente: COMPRESSED_RTP, que es la opción que más comprime; COMPRESSED_UDP, que no comprime la cabecera RTP, sino sólo las cabeceras IP/UDP, y FULL_HEADER, que no comprime. Según un estudio en el que se presentó una comparativa de CRTP y ECRTP para aplicaciones de VoIP sobre enlaces vía satélite [DKP07], más del 99,9% de las cabeceras son comprimidas. Asumiremos por tanto la simplificación de considerar solamente dos valores posibles para el tamaño de cabecera. Definimos RH_1 como el tamaño de la cabecera COMPRESSED_RTP, y RH_2, como el tamaño de la cabecera COMPRESSED_UDP.

Si definimos p como la probabilidad de tener una cabecera de tipo COMPRESSED_RTP, entonces la probabilidad de tener i cabeceras COMPRESSED_RTP en un paquete multiplexado de k flujos, tendrá una distribución binomial, así que podemos expresarla como:

$$\Pr(i) = \binom{k}{i} p^i\,(1-p)^{k-i} \tag{9.3}$$

Por tanto, podremos calcular el tamaño medio de la cabecera reducida como:

$$E[RH] = \frac{1}{k}\sum_{i=0}^{k} \Pr(i)\,[\,RH_1\,i\; + RH_2\,(k - i)\,] \tag{9.4}$$

Si operamos con (9.3) y (9.4) obtendremos una expresión para $E[RH]$ (9.5) que muestra que es independiente de k. Podríamos haber esperado este resultado, ya que las técnicas de compresión son independientes de la multiplexión:

$$E[RH] = p\,RH_1 + (1-p)\,RH_2 \tag{9.5}$$

Por tanto, en el caso de tener más de dos posibles valores para el tamaño de la cabecera comprimida, se puede usar una distribución multinomial. Si definimos p_j como la probabilidad de tener un tamaño de cabecera reducida RH_j, y M como el

número de posibles valores del tamaño de la cabecera, entonces la expresión del valor esperado queda así:

$$E[RH] = \sum_{j=0}^{M} p_j RH_j \tag{9.6}$$

Relación de anchos de banda

Para conocer el ahorro de ancho de banda que se obtiene al multiplexar, calcularemos la relación de anchos de banda entre el tráfico TCRTP y el RTP nativo, que denominaremos *BWR (BandWidth Relationship)*. El ahorro de ancho de banda vendrá dado por la relación:

$$Ahorro = 1\text{-}BWR \tag{9.7}$$

Dado que el periodo en que se envían los paquetes es el mismo en ambos casos, el parámetro *BWR* se podrá también calcular como la relación entre el tamaño de un paquete TCRTP que lleva información de k flujos (9.2) y el tamaño de k paquetes RTP nativos, que es el producto de (9.1) por k. Por tanto, el valor de *BWR* será:

$$BWR = \frac{CH + k(MH + E[RH] + S)}{k(NH + S)} \tag{9.8}$$

Si separamos (9.8) en dos partes, una que da cuenta de la multiplexión (BWR_M) y otra referida a la compresión de cabeceras (BWR_{RH}), obtenemos:

$$BWR_M = \frac{CH}{k(NH + S)} \tag{9.9}$$

$$BWR_{RH} = \frac{MH + E[RH] + S}{NH + S} \tag{9.10}$$

Vemos que BWR_M se hace menor según aumenta el número de flujos k, por lo que dependerá principalmente de la multiplexión. Por otro lado, como BWR_{RH} no depende de k, solo se verá influenciado por la eficacia del algoritmo de compresión de cabeceras, expresado en el término $E[RH]$, y el tamaño de las muestras S, que depende principalmente del *codec* utilizado.

Para hacernos una mejor idea de los resultados numéricos que se pueden obtener, se presentan ahora algunas gráficas de *BWR* en función de k, S y la probabilidad

de tener una cabecera reducida *p*. Los valores usados son los de TCRTP para VoIP con IPv4:

- *CH:* Corresponde a las cabeceras IP/L2TP/PPP, que ocupan 25 bytes.
- *MH*: 2 bytes, correspondientes a PPPMux.
- RH_1: 4 bytes.
- RH_2: 12 bytes.
- *NH*: 40 bytes: Cabeceras IP/UDP/RTP.
- *S*: Utilizaremos los valores de 10, 20 o 30 bytes, que se corresponden al *codec* G.729a con una, dos o tres muestras por paquete respectivamente. El *codec* genera una muestra cada 10 ms.

En la Fig. 9.2 hemos representado el valor de *BWR* para $S = 20$ bytes, con *k* variando desde 1 hasta 20 flujos, y *p* variando desde 0,7 hasta 1. Teniendo en cuenta los resultados de [DKP07] el valor de 0,7 es muy pesimista. Por claridad, se ha representado en la Fig. 9.3 el valor de *BWR* para diferentes números de muestras por paquete, pero manteniendo fijo el valor de $p = 0{,}95$. Y en la Fig. 9.4 se representa *BWR* para $k = 10$ y distintos valores de *p*.

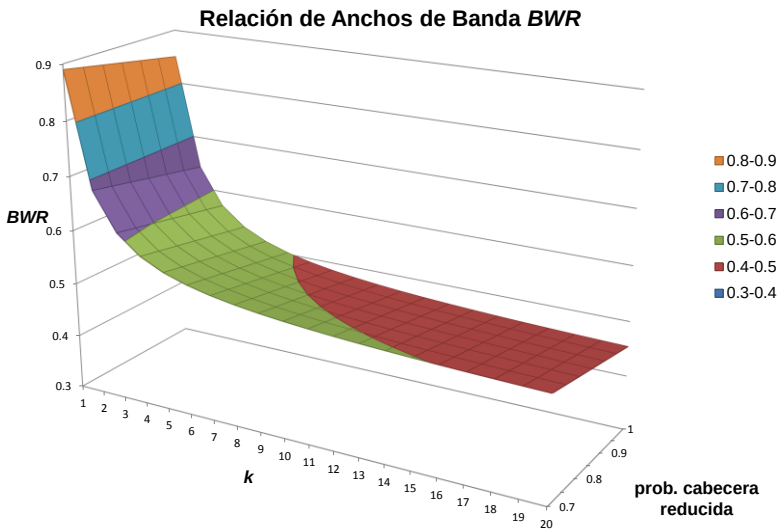


Fig. 9.2. Relación de anchos de banda *BWR* para $S = 20$ bytes

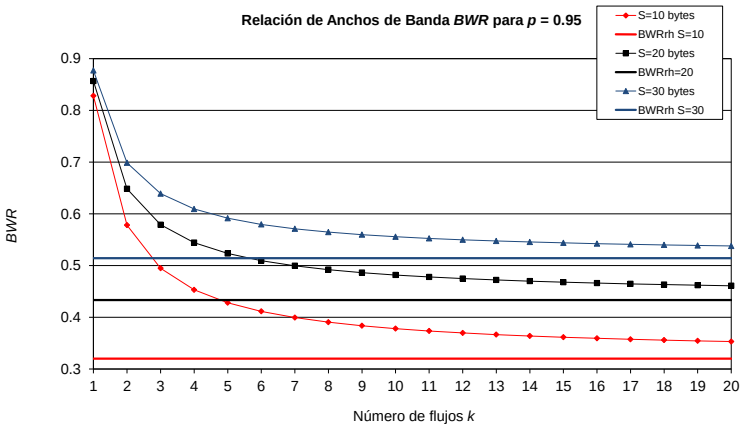


Fig. 9.3. Relación de anchos de banda *BWR* para $p = 0{,}95$

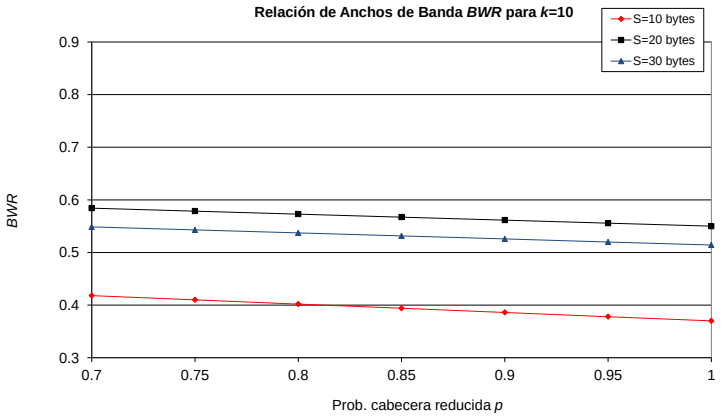


Fig. 9.4. Relación de anchos de banda *BWR* para $k = 10$ flujos

Podemos extraer una primera conclusión a la vista de estas gráficas: la multiplexión con TCRTP siempre ahorra ancho de banda respecto a RTP nativo, pues el valor de *BWR* está siempre por debajo de la unidad, incluso para $k = 1$ flujo. La razón es que el algoritmo de compresión actúa sobre los 40 bytes de cabecera de cada paquete RTP, aunque luego se añadan 25 bytes de cabecera común. Vemos que los efectos se compensan, porque el algoritmo de compresión reduce en más de 15 bytes la cabecera original.

Podemos extraer otra consecuencia de la Fig. 9.2: la influencia de k es mayor que la de p, es decir, incrementar el número de flujos multiplexados es más

beneficioso para la eficiencia que el tener una probabilidad de cabecera comprimida muy grande.

Por otro lado, vemos que, aunque el valor de *BWR* se puede reducir significativamente al aumentar *k*, las gráficas de la Fig. 9.3 muestran una asíntota determinada por el valor de BWR_{RH}. Los valores mínimos de BWR_{RH} son 0,32 para $S = 10$ bytes, 0,43 para $S = 20$ bytes, y 0,51 para $S = 30$ bytes. Estos son, por tanto, los límites del ahorro de ancho de banda. En conclusión, el incremento del número de flujos *k* tiene un efecto en el valor de *BWR*, pero la existencia de una asíntota hace que su efecto disminuya para valores grandes de *k*. Por tanto, si tenemos un gran número de flujos que multiplexar, quizá la mejor solución no sea agruparlos en un solo túnel puesto que el ahorro de ancho de banda crecerá muy poco, pero el tamaño de los paquetes sí puede crecer excesivamente.

La Fig. 9.4 muestra la influencia del algoritmo de compresión. Si no hay muchas transmisiones con la cabecera completa, lo que significa que *p* está cerca de la unidad, entonces se puede obtener un valor menor para $E[RH]$. Pero en la figura vemos que este parámetro no afecta demasiado al valor de *BWR*, ya que solamente le aporta una pequeña mejora.

A modo de ejemplo, en la Tabla 9.1 hemos incluido los anchos de banda a nivel IP que se obtienen para diferentes valores de *k* cuando se usa RTP o TCRTP.

Número de flujos *k*	**5**	**10**	**15**	**20**
RTP	120	240	360	480
TCRTP	62	115	168	221

Tabla 9.1 Ancho de banda utilizado por RTP y TCRTP a nivel IP en kbps

Paquetes por Segundo

Finalmente, estudiaremos la disminución en términos de paquetes por segundo, que en este caso es muy simple: al multiplexar, el número de paquetes generado queda dividido por un factor *k*. Este hecho puede resultar también ventajoso, puesto que hay estudios que han mostrado que el límite de paquetes por segundo que el *router* puede gestionar es un cuello de botella a considerar, además del límite de ancho de banda. Por ejemplo, en [YA07] se mostró este hecho a partir de la realización de medidas con *router* comerciales, y se recomendó considerar el *throughput* (rendimiento) en términos de bits por segundo y paquetes por segundo. Este hecho también ha sido destacado en [FCFW02] y [FCFW05].

Esta ganancia no afectará en nuestro caso a RTCP, que es un protocolo que trabaja en paralelo con RTP, encargándose de la información de control; como se dice en el estándar que define ambos protocolos [SCFV03], el tráfico de RTCP no debe exceder el 5% del total del tráfico RTP. Esta es la razón por la que no vamos a considerar la multiplexión de los paquetes RTCP, o sea, que este protocolo continuaría funcionando normalmente entre los extremos de la comunicación. Por tanto, el número de paquetes de RTCP se mantendrá constante.

En la Fig. 9.5 se representa el número de paquetes por segundo que se generarían al enviar 40 flujos RTP, si éstos se agrupan en diferentes números de túneles TCRTP. El parámetro *l* representa el número de túneles, y *k* sigue refiriéndose al número de flujos de cada túnel, cumpliendo siempre *l* x *k* = 40. Se ha añadido un 5% de paquetes, correspondiendo a RTCP, que es siempre el mismo, puesto que no se multiplexan.

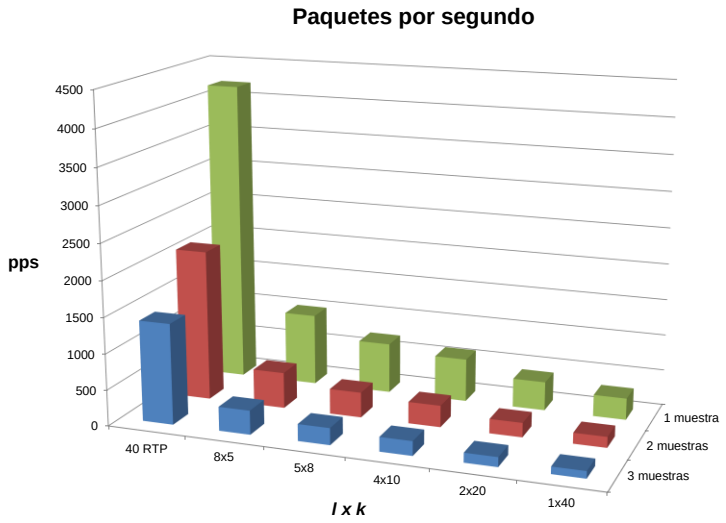


Fig. 9.5. Paquetes por segundo generados por RTP y RTCP en función del número de muestras por paquete y de la distribución de los túneles

La figura muestra que, lógicamente, el número mínimo de paquetes por segundo se logra cuando se usa un solo túnel y tres muestras por paquete. Se puede observar también la gran tasa de paquetes por segundo generada por RTP cuando se usa en su forma nativa, si se compara con la multiplexión. Vemos por tanto otra gran ventaja de multiplexar: la gran reducción en la cantidad de paquetes por segundo.

Al estudiar si nuestro *router* está trabajando en buenas condiciones, habría que calcular la cantidad total de paquetes por segundo, incluyendo los paquetes de voz y los de fondo (por ejemplo, la distribución de tráfico de fondo que usaremos en las pruebas [Nas05] genera 290 paquetes por segundo, para 1.600 kbps), y comprobar que no está por encima del límite del *router*.

Compromisos que aparecen con la multiplexión

Recapitulando lo visto en los apartados anteriores, vemos el triple compromiso de la multiplexión: el ancho de banda, el tamaño medio de paquete y el número de paquetes por segundo son los tres parámetros que podemos modificar al multiplexar el tráfico, cambiar el número de muestras por paquete o la distribución de los flujos.

En las Fig. 9.6 y 9.7 se presentan las tres variables en un diagrama de tres ejes: paquetes por segundo, ancho de banda y tamaño de paquete. El *router* tiene un límite de paquetes por segundo, y la conexión presenta un límite de ancho de banda, por lo que estos ejes tienen un límite fijo, que se verá reducido si consideramos además el efecto del tráfico de fondo. En las figuras estos límites se representan con el triángulo de líneas discontinuas. El eje del tamaño del paquete tiene solamente el límite del MTU *(Maximum Transfer Unit,* Unidad Máxima de Transferencia), que en muchas de las redes actuales es de 1.500 bytes. Pero, dependiendo del comportamiento del *router* y de la red con respecto al tamaño del paquete, nos puede interesar más situarnos en valores de tamaños pequeños o cercanos al MTU. El triángulo sólido representa en las figuras los parámetros para el esquema seleccionado. En la Fig. 9.6 se ha dibujado la distribución para 40 flujos RTP nativos, y en la 9.7 la distribución usando dos túneles TCRTP de 20 flujos cada uno.

La decisión sobre qué esquema utilizar nos permite adaptar nuestro tráfico de VoIP a las limitaciones de nuestro *router* y de nuestra red de acceso, evitando los valores que estén cerca de los límites. Si estamos planificando una red, podemos también considerar la multiplexión como una opción para ahorrar, eligiendo un *router* más económico, capaz de gestionar menos paquetes por segundo, o contratar menos ancho de banda en nuestra conexión.

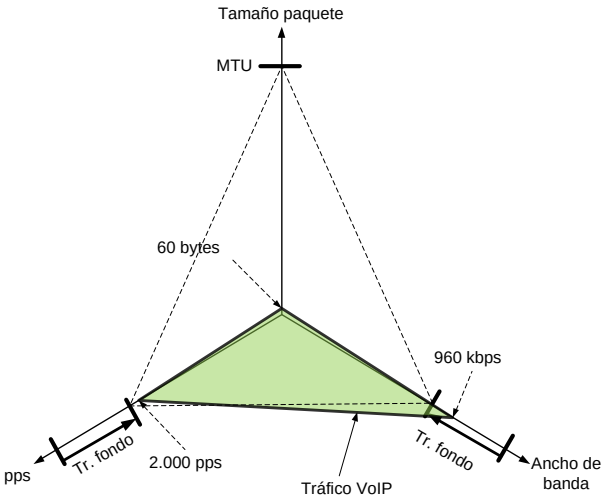


Fig. 9.6. Compromiso entre el ancho de banda, los paquetes por segundo y el tamaño medio de los paquetes para k = 40 flujos nativos

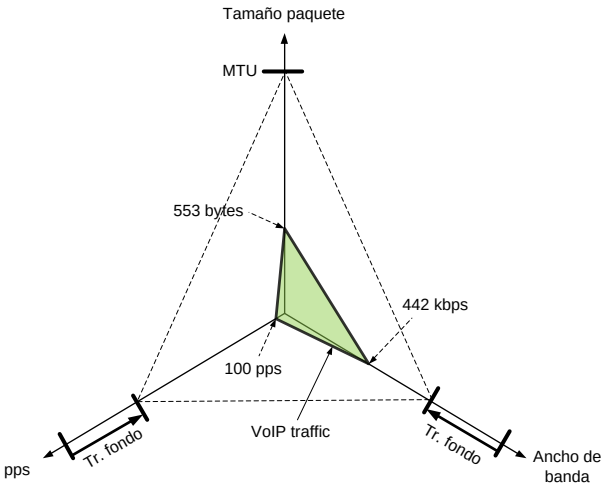


Fig. 9.7. Compromiso entre el ancho de banda, los paquetes por segundo y el tamaño medio de los paquetes para k = 40 flujos multiplexados en dos túneles de 20 flujos cada uno

Metodología de las pruebas

Terminado el estudio teórico, pasamos a presentar algunas medidas y pruebas que se han llevado a cabo utilizando el *testbed* presentado en el capítulo 6. Se han realizado del mismo modo que las medidas de parámetros de QoS presentadas en el capítulo anterior, utilizando tres máquinas: una genera el tráfico, otra actúa como *router* y otra lo recibe (Fig. 8.4). Hemos usado nuevamente la herramienta

Linux *Traffic Control (tc)*, ya explicada anteriormente, para obtener el comportamiento deseado del *buffer*. En este caso, se ha usado la opción *pfifo* de *tc* para emular el *buffer* con un número fijo de paquetes, además de la ya conocida *tbf* para los *buffer* de alta capacidad, limitado en tiempo y también para el que tiene un ancho de banda dedicado para el tráfico de voz.

Se comparará el tráfico multiplexado con el tráfico nativo RTP, pero no con CRTP o ECRTP, puesto que estos protocolos funcionan salto a salto, por lo que no resultan adecuados para nuestro escenario, en el que se utiliza Internet. El *codec* utilizado será siempre G.729a. Generalmente se usarán dos muestras por paquete, aunque en algunas comparativas se mostrarán también los resultados con una o tres muestras.

Generación del tráfico

El tráfico de fondo se envía usando el generador D-ITG [BDP07]. Se ha usado la ya conocida distribución de tamaños de paquete para el tráfico de fondo: el 50% de los paquetes son de 40 bytes, el 10% son de 576 y el restante 40% son de 1.500 bytes [Nas05]. Se ha empleado el mismo generador de tráfico D-ITG para enviar el tráfico de voz, puesto que permite diferentes estadísticas tanto para el tamaño del paquete como para el tiempo entre paquetes.

El tráfico RTP multiplexado debe ser caracterizado con un modelo estadístico adecuado para obtener una distribución realista. En el ya citado estudio [DKP07], los resultados mostraron que, para ECRTP, el 97,3% de los paquetes tenían una cabecera de tipo COMPRESSED_RTP, mientras que el 2,6% usaban una COMPRESSED_UDP. El porcentaje de cabeceras completas era muy pequeño (0,0033%), por lo que en estas medidas lo consideraremos despreciable. Así que usaremos una distribución binomial para calcular el tamaño de los paquetes multiplexados, como un caso particular de la distribución multinomial.

Así pues, se ha modelado el comportamiento de TCRTP en términos de tamaño de paquete añadiendo el número de bytes extra correspondiente a cada cabecera COMPRESSED_UDP, de acuerdo con una distribución binomial usando como parámetro el número de paquetes multiplexados k y la probabilidad de tener una cabecera comprimida. Estos bytes extra corresponden a los campos de la cabecera RTP que contienen una marca de tiempo y a un identificador que se deben actualizar periódicamente.

Para cada prueba se han enviado 400 segundos de tráfico. Posteriormente, los primeros y últimos 20 segundos se han descartado, para tener en cuenta

solamente el comportamiento estacionario. No se ha utilizado supresión de silencios.

Retardos del sistema

Un sistema de multiplexión-demultiplexión debe ser transparente para los extremos de la comunicación: el paquete enviado y el recibido deben ser exactamente iguales. Por eso, el demultiplexor necesita información (el *contexto)* para reconstruir el paquete original y entregarlo a su destinatario. Teniendo esto en cuenta, presentamos ahora un resumen de los diferentes retardos que deben considerarse en nuestro sistema, que se ilustran en la Fig. 9.8. Observamos que algunos de los retardos ya estaban en el sistema tal como se ha estudiado anteriormente, pero con la multiplexión han aparecido otros nuevos, principalmente los de retención y procesado

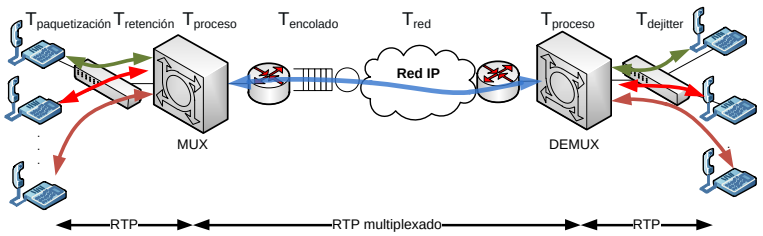


Fig. 9.8. Retardos del sistema de multiplexión

- Retardo de paquetización ($T_{paquetización}$): Depende fundamentalmente del *codec* utilizado. Hemos usado el *codec* G.729a, igual que en el capítulo anterior, para poder comparar en las mismas condiciones. Recordaremos que su retardo de paquetización es de 10 ms por muestra, más 5 ms que corresponden al tiempo de *look-ahead*. Por ejemplo, si se usan dos muestras por paquete, el retardo de paquetización será de 25 ms.

- Tiempo de retención en el multiplexor ($T_{retención}$): El multiplexor debe esperar a tener un paquete de cada flujo RTP para poder construir el paquete multiplexado. En este trabajo se ha asumido que las fuentes RTP están conectadas a través de una red local de alta velocidad, por lo que el tiempo de retención se puede considerar equivalente al tiempo entre paquetes, como cota superior. Naturalmente, si los flujos estuvieran sincronizados, este retardo podría disminuir significativamente, pero esta no es una situación habitual.

- Tiempo de proceso en el multiplexor y demultiplexor ($T_{proceso}$): En el trabajo presentado en [SLLY02] se construyó en Linux un prototipo del esquema de multiplexión propuesto. Se observó que los tiempos de procesado causados por el manipulado de las cabeceras y los tiempos de transmisión, estaban por debajo de 1 ms. Como los paquetes multiplexados son más grandes que los nativos, el tiempo de *store and forward* se incrementará ligeramente. En las medidas presentadas se ha añadido un retardo de 5 ms para tener en cuenta todos estos efectos, así como los tiempos de propagación en las redes locales implicadas.

- Tiempo de encolado en el *buffer* del *router* que envía el tráfico ($T_{encolado}$): El paso de la red local de alta velocidad al acceso a Internet supone un cuello de botella que se debe tener en cuenta. Este retardo dependerá fundamentalmente de la implementación del *buffer*.

- Retardo de red (T_{red}): Los tiempos de llegada de cada paquete se capturan una vez atravesado el *router*, y se añade a cada uno un tiempo de retardo diferente, siguiendo una distribución estadística. Se ha usado nuevamente el modelo propuesto en [KPLK+09], con una parte fija que depende de la distancia geográfica entre los dos nodos implicados, más otro retardo que corresponde a la parte variable del retardo. Se ha considerado en principio un escenario intrarregional, y se han usado valores de retardo extraídos de [ATT11]: 20 ms de retardo mínimo, más el retardo log-normal, con media 20 ms y varianza 5. De todas maneras, en algunas comparativas se emplearán también retardos mayores. Finalmente, se ha considerado que la red no pierde paquetes.

- Retardo de encolado en el *router* destino: Se considera despreciable, porque pasamos de un acceso a Internet a una red local que supondremos de alta velocidad.

- *Buffer* de *dejitter* de la aplicación destino ($T_{dejitter}$): Añade un nuevo retardo e incrementa las pérdidas de paquetes, ya que todos los paquetes que no llegan a tiempo para ser reproducidos equivalen a paquetes perdidos. Para evitar ceñirnos a una implementación concreta, las pérdidas causadas por este *buffer* se han calculado utilizando la aproximación sugerida en [CR00] y presentada en el apartado dedicado al *jitter* en el capítulo 3. El tamaño del *buffer* se ha calculado en cada caso para maximizar el Factor R.

Resultados

Presentaremos aquí los resultados de algunas pruebas realizadas, principalmente en términos del Factor R, que estima la calidad subjetiva de la conversación. También se presentarán algunos resultados de retardos y pérdidas de paquetes. Para analizar mejor la influencia de cada parámetro, se estudiará y comparará el efecto de distintos factores, como el ahorro que supone la multiplexión, el número de muestras por paquete, y la distribución de flujos en diferentes números de túneles. En las comparativas se tendrá también en cuenta la implementación del *buffer* del *router*. Algunos efectos se estudiarán solamente para las implementaciones en las que tengan efectos significativos.

Buffer con ancho de banda dedicado

El primer objetivo de la multiplexión RTP que analizaremos es el ahorro de ancho de banda mediante la compresión de cabeceras. Se han hecho algunas medidas para ilustrar este efecto. En primer lugar se ha estudiado lo que ocurre si se reserva un ancho de banda para el tráfico de voz. En este caso, es de esperar que el sistema se comporte bien mientras el ancho de banda de voz sea menor que el límite establecido.

Se ha utilizado un *buffer* con 200 kbps de ancho de banda. Para implementarlo se ha empleado la opción *tbf* de *tc,* obteniendo un *buffer* limitado en tiempo de 800 ms. Al no haber tráfico de fondo, el tamaño del *buffer* tiene poca influencia.

La Fig. 9.9 muestra el Factor R en función del número de flujos *k*. Puede observarse que, por ejemplo, si se usa RTP con dos muestras por paquete, sólo se pueden soportar 6 flujos simultáneos (7 flujos x 29 kbps a nivel *Ethernet* estaría ya por encima del límite de 200 kbps), mientras que utilizando TCRTP con el mismo número de muestras por paquete, se puede llegar hasta 17 flujos. El *overhead* del esquema de multiplexión es compartido por todos los flujos, pero en el caso de RTP nativo el ancho de banda simplemente se incrementa en un factor *k,* que es el número de flujos.

En la Fig. 9.10 se comparan las soluciones de RTP nativo, TCRTP y Sze, con dos muestras por paquete. Como hemos visto en el capítulo 5, la solución de Sze no utiliza un túnel, sino que directamente incluye varios paquetes RTP con cabecera comprimida dentro del *payload* del paquete UDP. Al ahorrarse el *overhead* correspondiente al túnel, consume menos ancho de banda y obtiene unos resultados algo mejores que TCRTP. De hecho, mantiene el Factor R por encima de 70 incluso con 20 flujos.

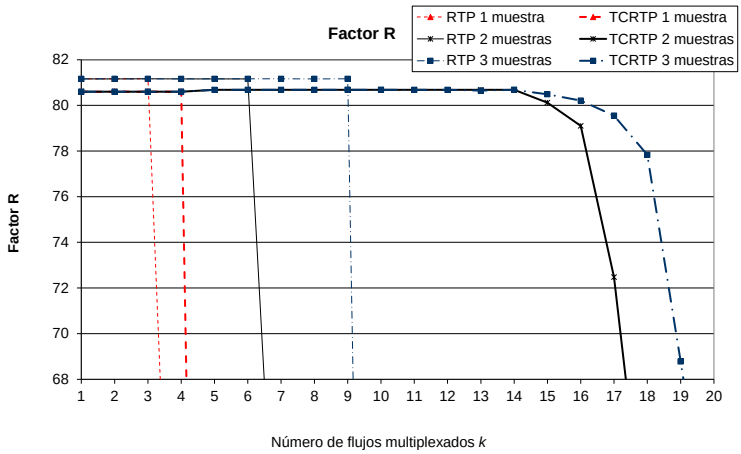


Fig. 9.9. Comparativa utilizando 200 kpbs de ancho de banda dedicado

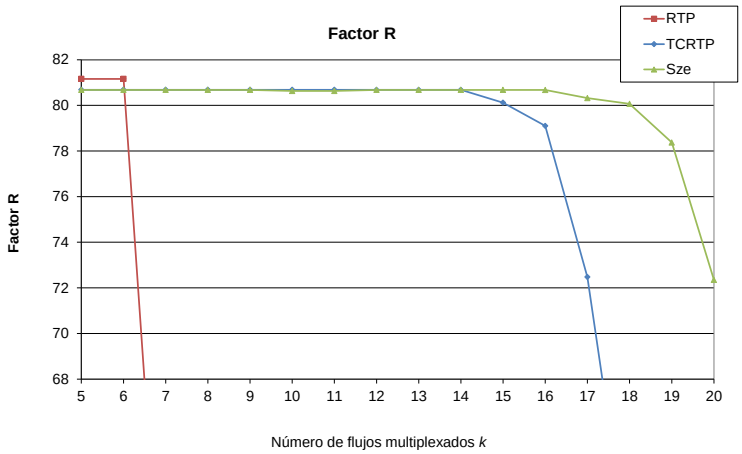


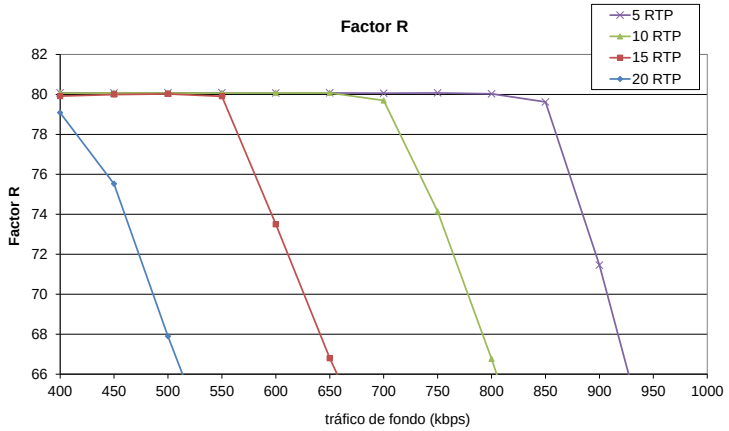
Fig. 9.10. Comparativa entre RTP nativo, TCRTP y Sze utilizando 200 kbps de ancho de banda dedicado

Buffer con un número limitado de paquetes

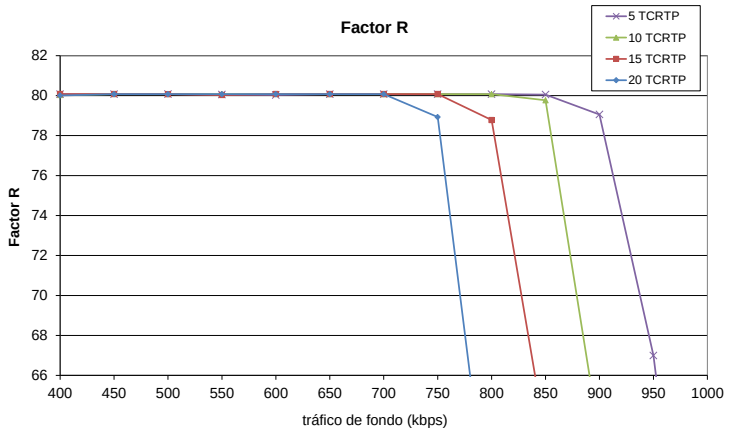
Otra posible implementación que puede encontrarse en los *router* de acceso es un *buffer* cuya capacidad está medida en paquetes y no en bytes, es decir, que puede almacenar un número máximo de paquetes. En este estudio se considerará también este tipo de *buffer*. Se ha elegido un número pequeño de paquetes para poder observar con más claridad el efecto que tiene este tipo de implementación del *buffer*.

Esta política se ha implementado utilizando la opción *pfifo* de la herramienta *tc* de Linux, que permite establecer este parámetro. En nuestro caso se ha utilizado un límite de 50 paquetes. Esta cola se ha anidado con otra que permite limitar el ancho de banda a 1 Mbps.

La Fig. 9.11 presenta el Factor R cuando se usa tráfico RTP nativo (a) y TCRTP (b), en función del tráfico de fondo, para 5, 10, 15 y 20 flujos VoIP simultáneos.



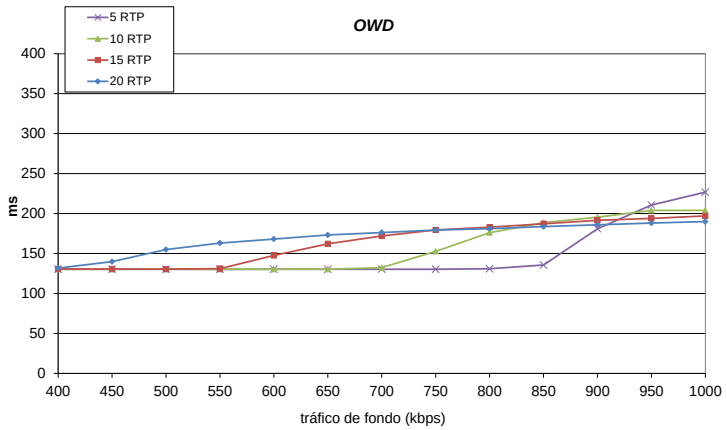
(a)



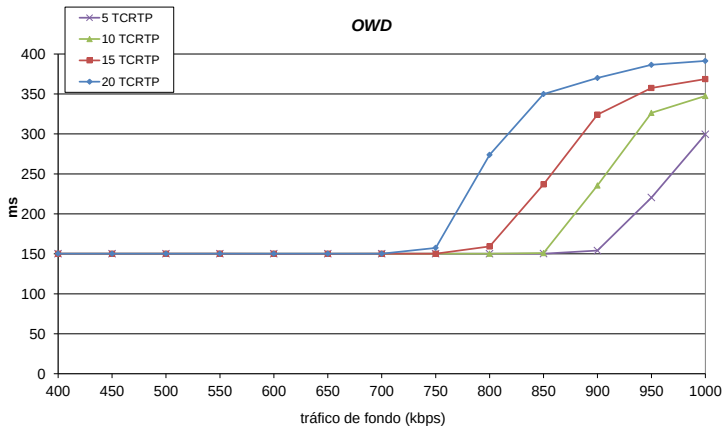
(b)

Fig. 9.11. Factor R en función del tráfico de fondo para a) RTP y b) TCRTP para el *buffer* con un número limitado de paquetes

Se puede observar que TCRTP produce un ligero empeoramiento de la calidad para valores pequeños del tráfico de fondo, a causa del tiempo de retención, pero es capaz de dar un servicio aceptable en presencia de mayores valores de tráfico de fondo. Por ejemplo, cuando se envían 20 flujos, si se usa tráfico RTP nativo, sólo se pueden tolerar 450 kbps de tráfico de fondo, cifra que sube hasta 750 kbps en el caso de TCRTP.



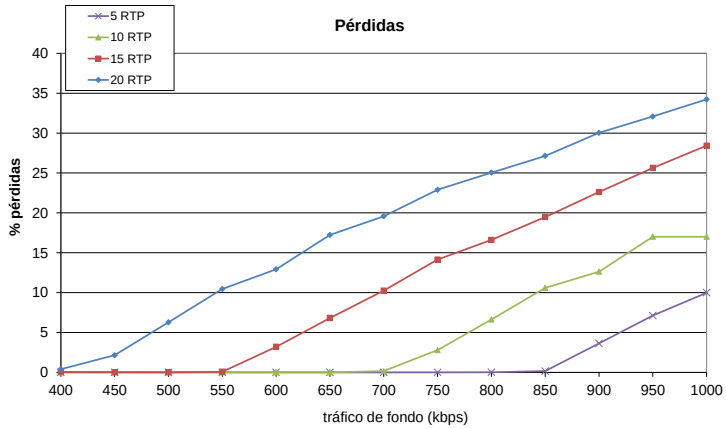
(a)



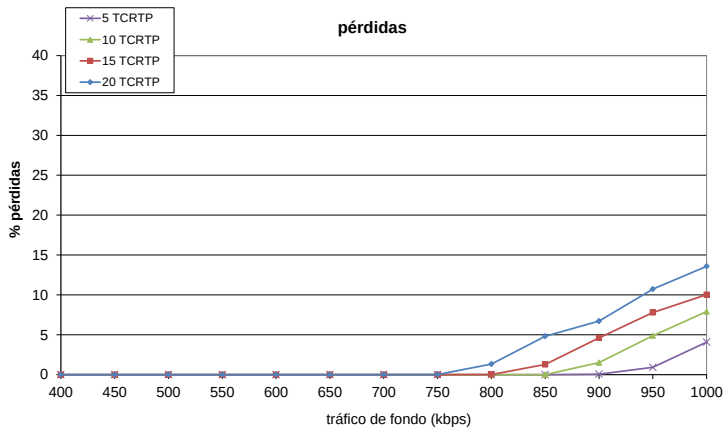
(b)

Fig. 9.12. Retardo en un sentido en función del tráfico de fondo para a) RTP y b) TCRTP

Para hacernos una idea más clara de la causa de este comportamiento, podemos observar las gráficas del OWD (Fig. 9.12) y pérdidas (Fig. 9.13). También resulta interesante calcular el tamaño medio de paquete del tráfico que entra al *buffer* en cada caso, teniendo en cuenta tanto el tráfico de voz como el de fondo. La Tabla 9.2 muestra estos valores. Como los paquetes de fondo son de 40, 576 y 1.500 bytes, el uso de RTP nativo reducirá el tamaño medio, pero al usar TCRTP el tamaño aumentará. Las diferencias pueden ser significativas: por ejemplo, para 20 flujos RTP nativos, el tamaño medio es de 101 bytes, mientras que al usar TCRTP se obtienen 626 bytes, para un tráfico de fondo de 400 kbps.



(a)



(b)

Fig. 9.13. Porcentaje de pérdidas de paquetes en función del tráfico de fondo para a) RTP y b) TCRTP

	400 kbps	700 kbps	1000 kbps
5 RTP	198,84	267,92	319,58
5 TCRTP	465,54	530,73	565,27
10 RTP	138,21	185,00	224,32
10 TCRTP	519,04	567,79	593,61
15 RTP	114,44	149,36	180,21
15 TCRTP	572,55	604,84	621,95
20 RTP	101,75	129,54	154,77
20 TCRTP	626,02	641,87	650,27

Tabla. 9.2. Tamaño medio de paquete (en bytes) a nivel IP incluyendo el tráfico de VoIP y de fondo

Si observamos la Fig. 9.12, podemos ver que el retardo crece, aunque no significativamente, ya que el *buffer* sólo puede almacenar 50 paquetes, y el tamaño medio es pequeño (entre 100 y 320 bytes). Por otro lado, los retardos añadidos a los paquetes TCRTP son mayores: a pesar de que sólo se pueden almacenar 50 paquetes, éstos son de un tamaño mayor (entre 465 y 650 bytes). Podemos concluir diciendo que RTP tiene una ventaja en términos de retardo cuando se utiliza este *buffer*.

Pero si observamos las pérdidas (Fig. 9.13), vemos que este parámetro se comporta significativamente peor para el tráfico RTP nativo: se alcanzan valores de hasta un 35%, mientras que para TCRTP siempre se mantienen por debajo del 14%. La razón es que la cantidad de paquetes generada por TCRTP es significativamente inferior, como se ha visto anteriormente. Un paquete TCRTP cuenta como uno solo en el *buffer*, a pesar de incluir un número de paquetes RTP comprimidos, por lo que, si el *buffer* tiene esta implementación, la multiplexión puede representar una ventaja importante. Este es un caso en el que la reducción en términos de paquetes por segundo representa una clara ventaja.

Buffer de alta capacidad

En este apartado estudiaremos el efecto de la multiplexión cuando se usa un *buffer* de alta capacidad. Para emularlo en el *testbed* se ha utilizado una política *thf* con 800 ms de límite de retardo. El ancho de banda es de 1 Mbps. Si se usa este *buffer*, al superar el límite de ancho de banda, el retardo crece indefinidamente, muy por encima de los límites aceptables para VoIP.

Número de flujos

La Fig. 9.14 muestra el Factor R en función del tráfico de fondo, con diferentes números de flujos. Se puede observar que el comportamiento es similar al

obtenido con un ancho de banda dedicado: una vez alcanzado el límite del ancho de banda, el Factor R se vuelve inaceptable. En este caso, se ha usado en la aplicación destino un *buffer* de *dejitter* con $b = 3$ muestras.

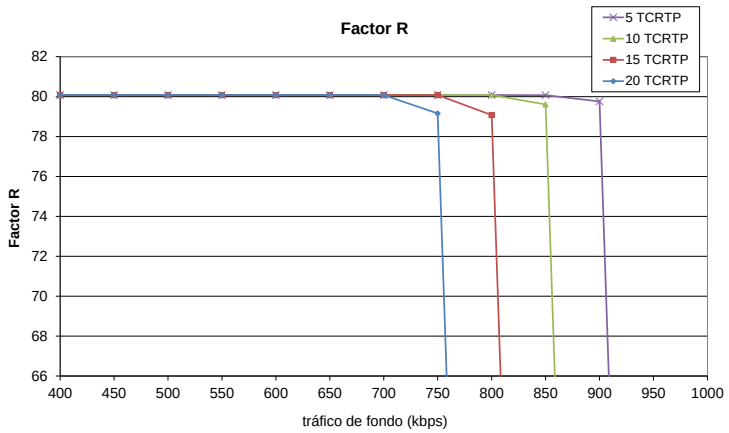


Fig. 9.14. Factor R con diferentes valores de *k* para *buffer* de alta capacidad

Número de muestras por paquete

Estudiaremos ahora la influencia del número de muestras que se incluyen en cada paquete RTP. De nuevo nos encontramos con un compromiso entre el retardo y la eficiencia. Por un lado, si usamos tráfico RTP nativo con una sola muestra por paquete, ahorramos 10 ms de retardo de paquetización, además de otros 10 ms de tiempo de retención, pero con el coste de enviar el doble de paquetes, y por tanto gastando un mayor ancho de banda a causa de las cabeceras de 40 bytes. Por otro lado, si aumentamos a tres el número de muestras, mejora la eficiencia pero se añaden retardos de paquetización y retención.

La Fig. 9.15 muestra estos dos efectos simultáneos: al usar TCRTP, los retardos adicionales empeoran ligeramente el Factor R para pequeñas cantidades de tráfico de fondo. Pero por otro lado, la multiplexión permite tener una buena calidad de conversación en presencia de mayores cantidades de tráfico de fondo. Por ejemplo, si se usa RTP nativo con una muestra por paquete, a pesar de que la calidad es buena, el *overhead* será grande, por lo que sólo se podrán soportar 450 kbps de tráfico de fondo. Si multiplexamos, el tráfico de fondo puede aumentar hasta 800 kbps manteniendo una calidad aceptable en las llamadas.

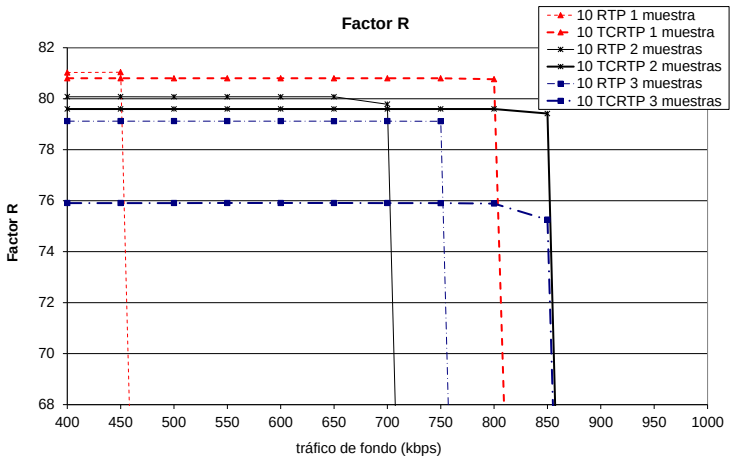


Fig. 9.15. Factor R con 10 flujos multiplexados usando diferentes números de muestras por paquete, para el *buffer* de alta capacidad

Comparativa entre las diferentes soluciones

La Fig. 9.16 compara los resultados para RTP nativo, TCRTP y Sze, para 15 flujos, utilizando el *buffer* de alta capacidad. Vemos que el comportamiento es del tipo "escalón": cuando se alcanza el límite del ancho de banda, la calidad disminuye bruscamente. En este caso, los resultados para TCRTP y Sze son casi idénticos, llegando a soportar un tráfico de fondo de 800 kbps, y mejoran en mucho a los de RTP nativo, que sólo soporta 550 kbps.

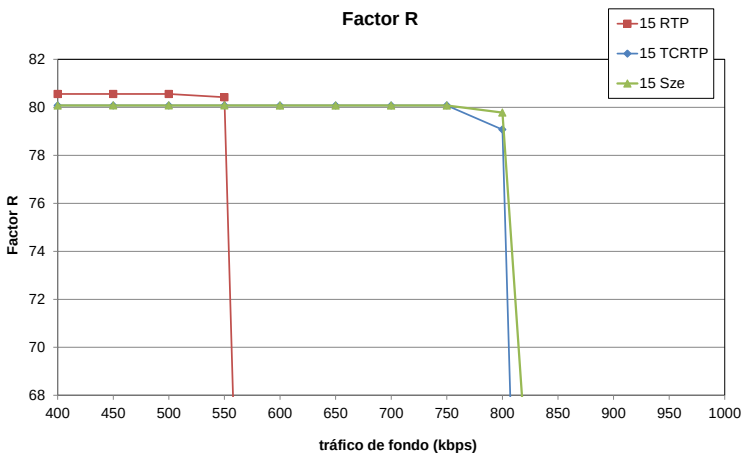


Fig 9.16. Factor R para 15 flujos usando RTP nativo, TCRTP y Sze, con *buffer* de alta capacidad

Los resultados anteriores se han obtenido utilizando un retardo de red de 40 ms: 20 fijos y otros 20 con varianza 5 y distribución log-normal. Ahora estudiaremos qué pasa si los retardos cambian, puesto que dependen de muchos factores, como la distancia geográfica entre los extremos de la comunicación [KPLK+09]. La Fig. 9.17 muestra el OWD para retardos de red con media entre 40 y 100 ms. Vemos que el retardo no cambia con el tráfico de fondo mientras hay suficiente ancho de banda, pero cuando se alcanza el límite, crece hasta valores inaceptables.

La Fig. 9.18 muestra el Factor R, y puede observarse que cuando el OWD pasa de 177,3 ms, lo que ocurre en el caso de los retardos de 80 y 100 ms, R empeora significativamente.

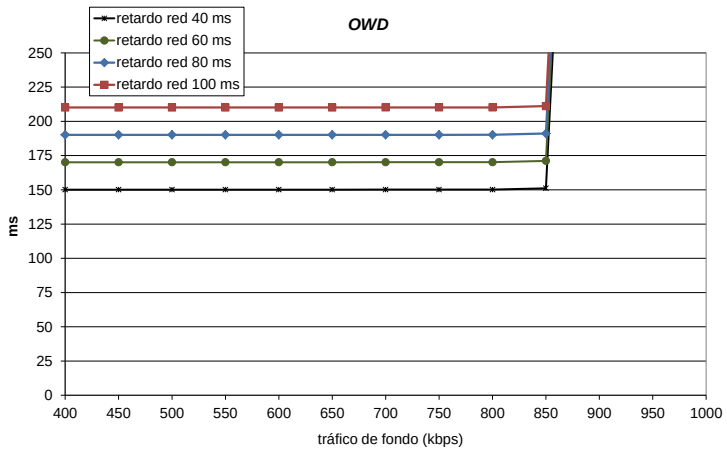


Fig. 9.17. OWD para 10 llamadas multiplexadas, con diferentes retardos de red, para el *buffer* de alta capacidad

Si queremos combinar el efecto de los retardos de red y el número de muestras por paquete, debemos tener en cuenta los retardos fijos. En la Fig. 9.19 se representa el Factor R para 40 y 100 ms de retardo de red, con una, dos y tres muestras por paquete. Vemos que en el caso de valores grandes del retardo, nos vemos forzados a evitar el uso de tres muestras por paquete, ya que R está por debajo de 70, debido a los retardos fijos.

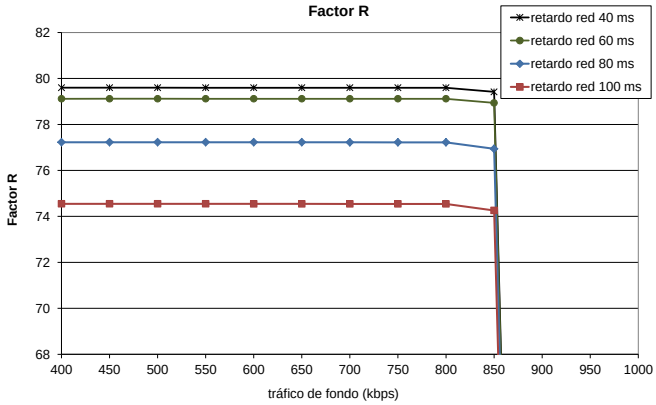


Fig. 9.18. Factor R para 10 llamadas multiplexadas, con diferentes retardos de red, para el *buffer* de alta capacidad

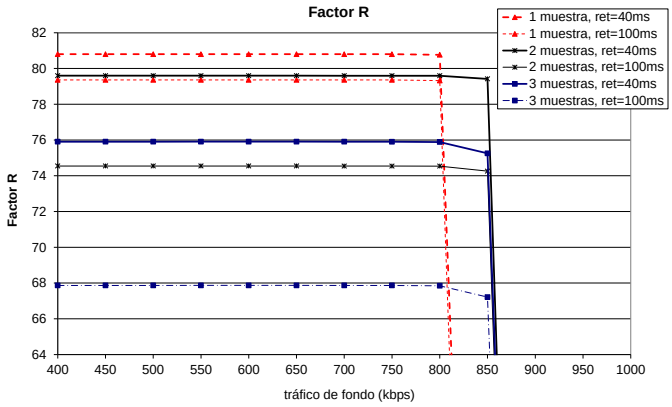


Fig. 9.19. Factor R para 10 llamadas multiplexadas, con diferentes retardos de red y diferente número de muestras por paquete, para el *buffer* de alta capacidad

Modo de agrupar los flujos

Ahora estudiaremos cómo la calidad puede verse modificada según el modo en que se agrupen los flujos TCRTP en diferentes números de túneles. Para estas pruebas se enviarán 40 llamadas utilizando el *codec* G.729a con dos muestras por paquete, y compartiendo el mismo enlace entre dos extremos. Estudiaremos la variación del Factor R según los diferentes valores de k, que es el número de flujos multiplexados en cada túnel, y l, el número de túneles, cumpliendo siempre $l \times k = 40$ flujos.

En este caso el *testbed* se utilizará para enviar el tráfico deseado y el de fondo compartiendo el mismo enlace de 2 Mbps. La Tabla 9.3 muestra el ancho de

banda que ocupan las diferentes opciones, así como el tamaño medio del paquete a nivel IP. Los parámetros usados en las medidas son los mismos que en secciones anteriores, pero en este caso el tamaño del *buffer* de *dejitter* se ha fijado a $b = 2$.

1 x k	1 x 40	2 x 20	4 x 10	5 x 8	8 x 5	No mux
Tamaño medio paq (bytes)	1.081	553	289	236	157	60
Ancho de banda (kbps)	432	442	462	472	502	960

Tabla 9.3. Tamaño medio de paquete a nivel IP (en bytes), y ancho de banda en kpbs

La Fig. 9.20 muestra el Factor R obtenido para distintos valores de k. El comportamiento es bueno hasta que se alcanza el límite del ancho de banda, y entonces cae bruscamente. Puede observarse que el mejor comportamiento se obtiene para $k = 20$ y $k = 40$ flujos. El comportamiento es muy similar, puesto que el ancho de banda que utilizan es prácticamente el mismo.

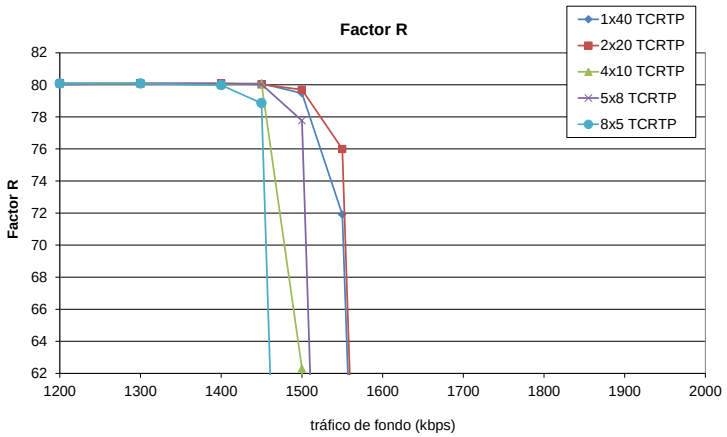


Fig. 9.20. Factor R para diferentes valores de k usando el *buffer* de alta capacidad

Buffer limitado en tiempo

Finalmente, se han realizado pruebas con un *buffer* limitado en tiempo. Lo estudiaremos con mayor profundidad, puesto que lo consideramos muy útil para servicios interactivos en tiempo real, como VoIP, donde el retardo tiene que mantenerse por debajo de un determinado valor si se quiere ofrecer un servicio con calidad. Este tipo de *buffer* también contribuye a reducir el *jitter*, evitando así

nuevas pérdidas causadas por el *buffer* de *dejitter*. En las pruebas se utilizará un ancho de banda de 1 Mbps. El *buffer* tiene una sola cola y descarta todos los paquetes que permanecen en ella más de 80 ms, y se ha implementado nuevamente mediante la disciplina de cola *tbf*.

Lógicamente, esta política hace que los paquetes grandes tengan una probabilidad mayor de ser descartados, cosa que ocurre también con los *buffer* de tamaño limitado en bytes. Esto supone una ventaja para los paquetes RTP nativos, pero es un inconveniente para los paquetes multiplexados que, al ser más grandes, serán descartados en un porcentaje mayor. Utilizando esta política, es de esperar que el Factor R descienda más despacio que en el caso de *buffer* grande, como ocurría en las pruebas presentadas en el capítulo anterior. Habrá por tanto dos efectos simultáneos: al multiplexar se ahorra ancho de banda, pero con la contrapartida de generar paquetes más grandes, lo que incrementará su probabilidad de pérdidas. Por tanto, puede resultar interesante estudiar en qué casos un efecto influye más que el otro.

Número de flujos

La Fig. 9.21 muestra el Factor R usando distintos valores de *k*. Si la comparamos con la Fig. 9.14, podemos darnos cuenta de que los valores obtenidos para valores pequeños de tráfico de fondo coinciden con los obtenidos para el *buffer* de alta capacidad. Pero las gráficas han dejado de tener una forma de "escalón", presentando ahora una pendiente, y consiguiendo así valores aceptables (por encima de R = 70) para mayores cantidades de tráfico de fondo. La causa es que esta política penaliza los paquetes grandes, así que los más perjudicados son los de 1.500 bytes de tráfico de fondo.

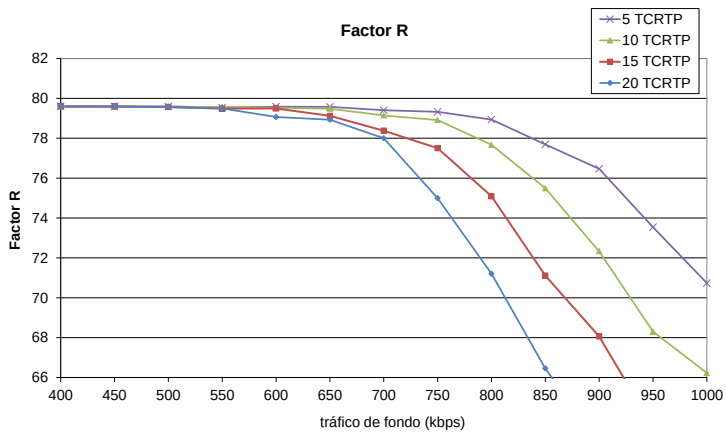


Fig. 9.21. Factor R con distintos valores de *k* para el *buffer* limitado en tiempo

Hemos realizado también algunas comparativas entre el comportamiento de RTP nativo, TCRTP y Sze usando el *buffer* limitado en tiempo. La Tabla 9.4 muestra el ancho de banda requerido por cada una de las soluciones. La Fig. 9.22 muestra el Factor R para 10 flujos de voz utilizando las distintas opciones. Vemos que RTP comienza teniendo valores mejores, porque carece de los retardos de retención y procesado. Pero los valores pasan a ser peores que los de las soluciones multiplexadas a partir de 650 kbps. La causa es que TCRTP y Sze están gastando sólo 114 y 99 kbps respectivamente, mientras que RTP ya está al 90% del ancho de banda, porque usa 240 kbps. Vemos que también con este buffer, la solución de Sze presenta una ligera ventaja frente al estándar TCRTP.

Número de flujos	**5**	**10**	**15**	**20**
RTP	120	240	360	480
TCRTP	62	114	166	216
Sze	55	99	143	185

Tabla 9.4. Ancho de banda requerido por RTP nativo, TCRTP y Sze a nivel IP en kbps

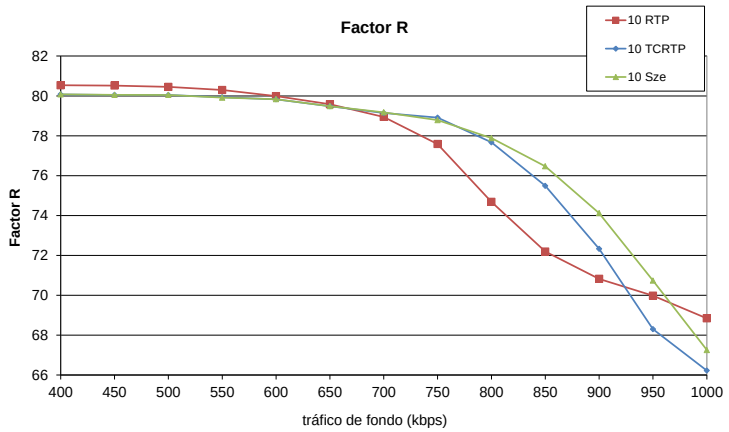


Fig. 9.22. Factor R para 10 flujos usando RTP nativo, TCRTP y Sze para el *buffer* limitado en tiempo

La Fig. 9.23 muestra la mejora del Factor R al usar TCRTP en lugar de RTP nativo. En primer lugar, podemos observar que para valores pequeños de tráfico de fondo, el empeoramiento es inferior al 1%. Esto está causado por los retardos adicionales debidos a la multiplexión. Pero cuando el tráfico de fondo crece, vemos que por encima de 5 flujos multiplexados, la multiplexión representa una interesante mejora, ganando hasta un 21%. Este efecto se debe principalmente al

ahorro de ancho de banda. Finalmente, se observa que cuando el tráfico de fondo llega al 95% del límite, el tráfico RTP nativo vuelve a ser mejor que el multiplexado. La causa es que los paquetes multiplexados, al ser más grandes, tienen una mayor probabilidad de descarte.

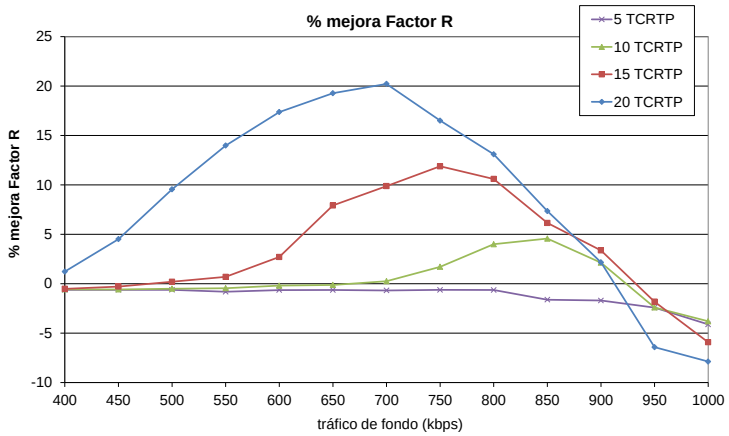


Fig. 9.23. Mejora del Factor R con $S = 20$ y distintos valores de k para el *buffer* limitado en tiempo

La Fig. 9.24 muestra el porcentaje de pérdidas de paquetes del tráfico de fondo. Una vez más, observamos que el ahorro de ancho de banda que supone la multiplexión, nos puede ayudar a disminuir en un gran porcentaje las pérdidas de paquetes.

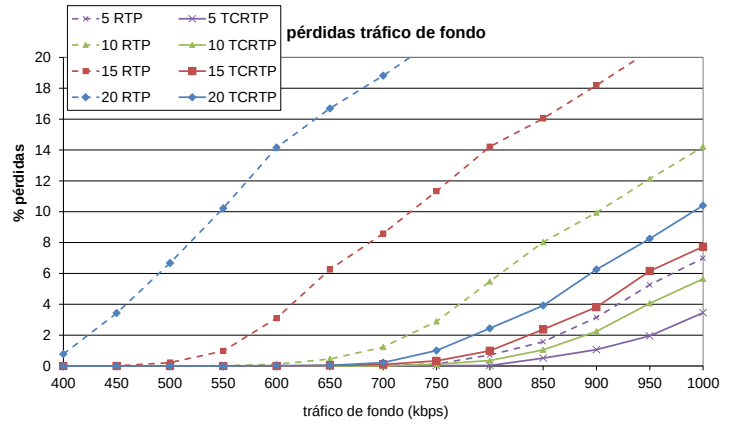


Fig. 9.24. Probabilidad de pérdidas para el tráfico de fondo, para el *buffer* limitado en tiempo

Número de muestras por paquete

Al usar multiplexión unimos *k* paquetes dentro de uno, por lo que el tamaño del paquete multiplexado también se modifica al cambiar el número de muestras por paquete. Aunque el ancho de banda requerido aumenta al reducir el número de muestras por paquete, otro efecto de esta reducción es que los paquetes se hacen más pequeños, por lo que este tráfico puede tener una ventaja según cuál sea el comportamiento del *buffer*. La Fig. 9.25 muestra el efecto del número de muestras por paquete para el *buffer* limitado en tiempo. Para valores pequeños del tráfico de fondo, se obtiene el mejor resultado para TCRTP con una muestra por paquete, ya que es la solución con el retardo más pequeño. Pero, al no ser la solución que ahorra más ancho de banda, el tráfico de fondo tendrá una probabilidad de pérdidas mayor, como vemos en la Fig. 9.26. La opción de dos muestras por paquete es capaz de mantener el Factor R por encima de 70, incluso con un tráfico de fondo del 90% del ancho de banda disponible. También vemos que RTP nativo tiene un comportamiento significativamente peor que TCRTP con respecto al tráfico de fondo, debido al ancho de banda que requiere.

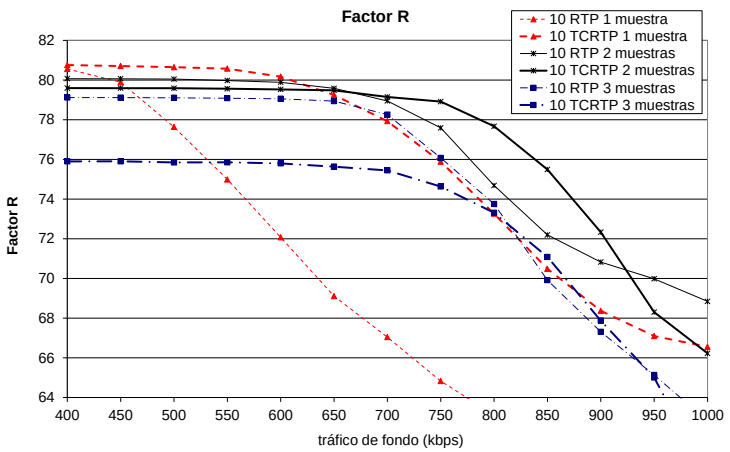


Fig. 9.25. Factor R con 10 flujos nativos y multiplexados, usando diferente número de muestras por paquete, para el *buffer* limitado en tiempo

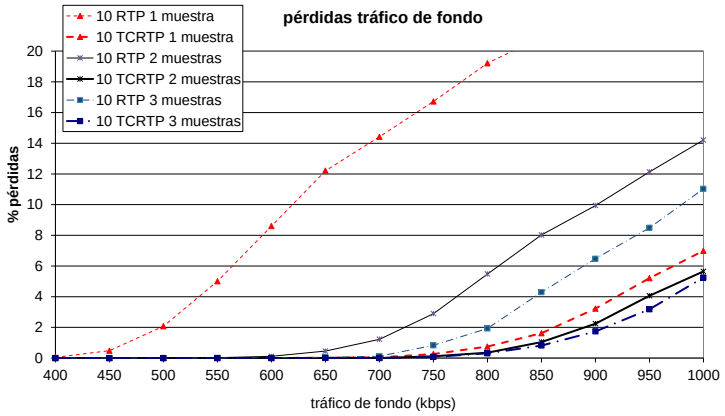


Fig. 9.26. Pérdidas del tráfico de fondo con 10 flujos nativos y multiplexados, usando diferente número de muestras por paquete, para el *buffer* limitado en tiempo

Variación del retardo de red

Las Fig. 9.27 y 9.28 ilustran el efecto del retardo de red en la calidad percibida, con 10 flujos multiplexados. Cuando el OWD llega al límite de 177,3 ms, el Factor R empeora. Esto confirma que el retardo de red es un parámetro importante que debe tenerse en cuenta.

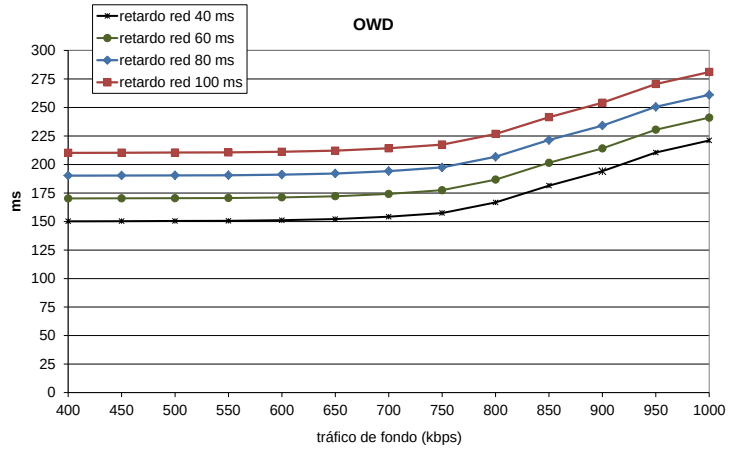


Fig. 9.27. OWD con 10 flujos multiplexados y diferentes retardos de red, con dos muestras por paquete, para el *buffer* limitado en tiempo

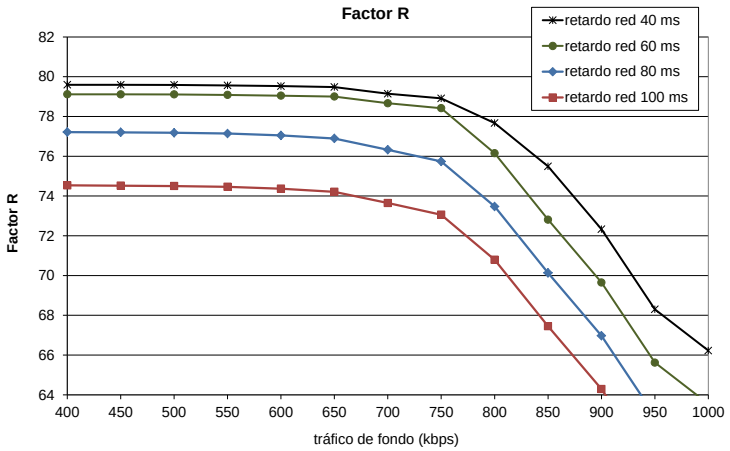


Fig. 9.28. Factor R con 10 flujos multiplexados y diferentes retardos de red, con dos muestras por paquete, para el *buffer* limitado en tiempo

La Fig. 9.29 presenta el efecto combinado del retardo de red y el número de muestras. Observamos que para valores altos del retardo se debe evitar el uso de tres muestras por paquete, ya que el efecto combinado del tiempo de paquetización y retención hace que los retardos crezcan por encima del límite recomendable.

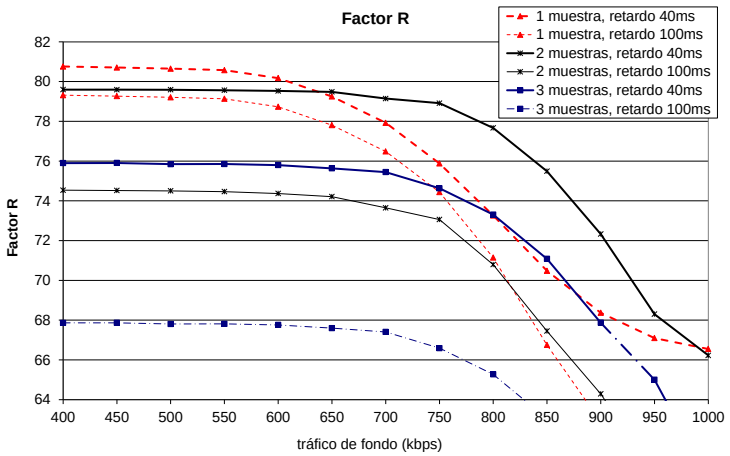


Fig. 9.29. Factor R para 10 flujos multiplexados, diferentes retardos de red y número de muestras por paquete, para el *buffer* limitado en tiempo

Modo de agrupar los flujos

Con respecto al problema de cómo distribuir un número de flujos en varios túneles, hemos construido la Fig. 9.30 de una manera similar que la Fig. 9.20, pero usando un *buffer* limitado en tiempo de 80 ms, y con 2 Mbps de ancho de banda. De nuevo observamos la ventaja de este *buffer* con respecto al de alta capacidad: las gráficas presentan una pendiente más suave, por lo que se puede conseguir una calidad aceptable para cantidades mayores de tráfico de fondo. La gráfica de 40 flujos nativos RTP también se ha incluido, pero muestra un comportamiento inaceptable para estos valores de tráfico de fondo, ya que el tráfico total ofrecido está por encima de la capacidad del enlace: el ancho de banda de 40 flujos RTP es 1.160 kbps, y el tráfico de fondo representado en la figura empieza a partir de 1.200 kbps.

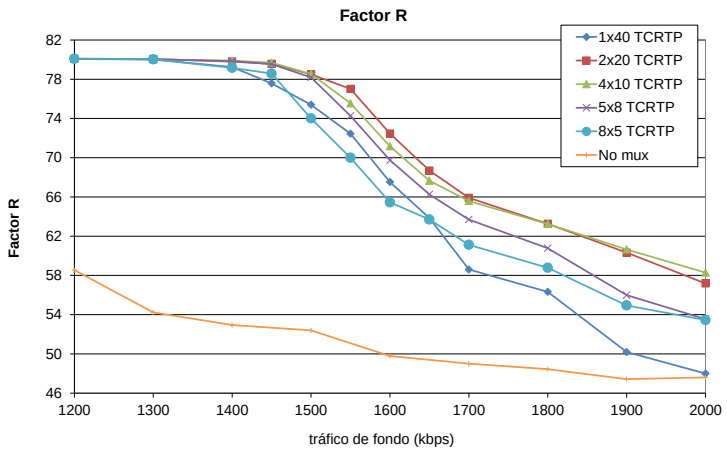


Fig. 9.30. Factor R para diferentes valores de *k* para el *buffer* limitado en tiempo

Observando la Fig. 9.30, podemos darnos cuenta de un resultado interesante: al usar un *buffer* limitado en tiempo, vemos que incluir todos los flujos en un solo túnel *(1x40)* no es la mejor solución. Hay tres distribuciones que logran mejores resultados *(2x20, 4x10* y *5x8)*.

Por claridad, se incluyen en la Tabla 9.5 los valores más interesantes de la Fig. 9.30, es decir, los correspondientes a 1.500 y 1.600 kbps de tráfico de fondo: vemos que el mejor resultado se obtiene usando $l = 2$ túneles de $k = 20$ flujos multiplexados, que da resultados muy similares a $l = 4$ túneles de $k = 10$ flujos.

1 x k	1 x 40	2 x 20	4 x 10	5 x 8	8 x 5	No mux
Tr. fondo 1.500 kbps	72,44	77,01	75,54	74,26	69,99	52,39
Tr. fondo 1.600 kbps	67,51	72,46	71,16	69,74	65,44	49,78

Tabla 9.5. Valores de R para 40 flujos multiplexados con distintos valores de *k*

Por un lado, multiplexar todos los flujos en un solo túnel (*k* = 40) es la solución que ahorra más ancho de banda, aunque la diferencia no es muy grande: pasar de *2x20* a *1x40* solamente ahorra 10 kbps (en torno al 2% del ancho de banda), pero hace que los paquetes casi doblen su tamaño (Tabla 9.3). Como se ha visto en la parte analítica, el ahorro de ancho de banda tiene una asíntota, así que a partir de 20 flujos la ganancia es despreciable. Pero por otro lado los paquetes serán más grandes, lo que aumenta la probabilidad de descarte si, como ocurre en este caso, el *buffer* penaliza los paquetes grandes. Por tanto, será más interesante agrupar los flujos en un número de túneles, generando así paquetes más pequeños.

La Fig. 9.31 muestra el porcentaje de mejora de ancho de banda que se puede obtener multiplexando, respecto a los valores de RTP nativo. Esta figura ilustra el hecho de que multiplexar todas las llamadas en un solo túnel no es siempre la mejor solución. Los mejores resultados se logran para *2x20* y para *4x10*. Se debe destacar otro efecto: aunque la curva *8x5* es la primera que baja, lo hace con una pendiente menor al resto de curvas, ya que usa paquetes más pequeños. Por eso consigue mejores resultados que *1x40* a partir de 1.700 kbps de tráfico de fondo.

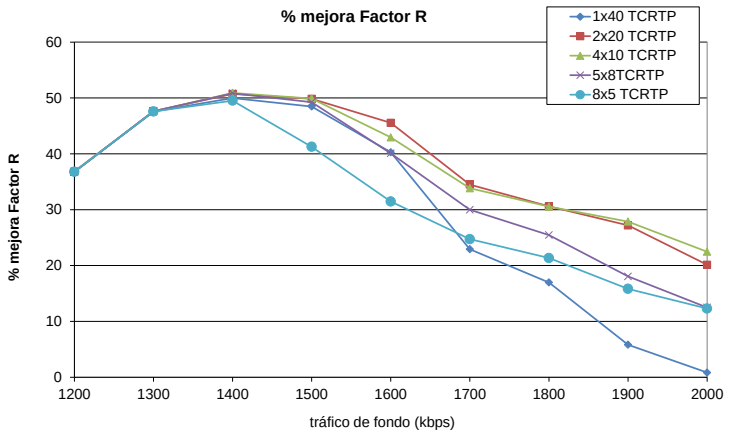


Fig. 9.31. Porcentaje de mejora del Factor R respecto a 40 flujos nativos RTP para el *buffer* limitado en tiempo

Otro asunto que debemos destacar es que el uso de valores grandes de *k* ayuda a evitar las pérdidas en el tráfico de fondo. Como se puede ver en la Fig. 9.32, a mayor número de llamadas multiplexadas, menor es el porcentaje de pérdidas para el tráfico de fondo. Esto ocurre porque multiplexar todos los flujos en un solo túnel es la opción que logra el mayor ahorro de ancho de banda, aunque produzca los paquetes de mayor tamaño. Por otro lado, usando *8x5* generaremos paquetes más pequeños, pero ahorrando menos ancho de banda.

Como resumen se puede decir que hay un compromiso: si queremos dar prioridad al tráfico de voz, tenemos que usar el valor de *k* que maximiza el Factor R, pero si simplemente queremos ahorrar ancho de banda, deberemos multiplexar todas las llamadas en un solo túnel, consiguiendo así los mejores resultados para el tráfico de fondo.

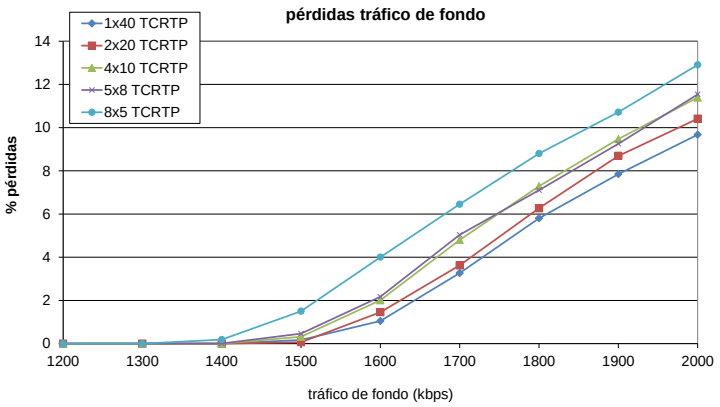


Fig. 9.32. Porcentaje de pérdidas de tráfico de fondo con diferentes valores de *k* para el *buffer* limitado en tiempo

Discusión de los resultados

En este apartado analizaremos los resultados obtenidos, tratando de extraer algunas conclusiones sobre el mejor modo de usar la multiplexión en escenarios reales. Como ya se ha visto, hay unas decisiones que debemos tomar en primer lugar, para establecer los parámetros principales de nuestro sistema de VoIP, como el número de muestras por paquete, y si vamos a multiplexar o no. Estas decisiones modificarán el ancho de banda total, el tamaño de paquete y los paquetes por segundo de nuestro tráfico.

El tráfico nativo RTP siempre utiliza el mismo tamaño, y su ancho de banda aumenta linealmente con el número de flujos. El ancho de banda usado por TCRTP también aumenta linealmente con *k* (ver ecuación 9.2), pero el tamaño de paquete aumenta con el número de flujos. Como hemos visto en los resultados, el aumento de tamaño no es siempre beneficioso para la calidad percibida, ya que los retardos aumentan, y además las pérdidas pueden incrementarse dependiendo de la implementación del *buffer*. Por tanto, hay un compromiso, que debe resolverse teniendo en cuenta la calidad percibida, que se mide en términos del Factor R.

Dependencia del Factor R con el número de flujos

Según vamos incrementando el número de flujos multiplexados, vemos que podemos ahorrar más ancho de banda, pero tenemos que tener en cuenta el comportamiento asintótico de la ganancia de multiplexión, ya que por encima de 15 o 20 flujos se hace despreciable.

Si el *buffer* tiene un número fijo de paquetes, el uso de TCRTP le hace capaz de almacenar un mayor número de bytes, puesto que los paquetes TCRTP son más grandes. Esto es beneficioso para evitar pérdidas. Aunque el retardo se incremente ligeramente al usar paquetes más grandes, lo que realmente empeora el Factor R es la alta tasa de pérdidas.

Si se usa un *buffer* de alta capacidad, entonces el comportamiento es muy simple: la calidad es aceptable hasta que se alcanza el límite de ancho de banda. En este caso, multiplexar interesa por el ahorro de ancho de banda.

Pero si se usa un *buffer* limitado en tiempo, el comportamiento cambia, como ya se ha visto. Hay que tener en cuenta el tamaño de los paquetes, puesto que los de VoIP tienen que competir con el tráfico de fondo. La Fig. 9.33 muestra el máximo tráfico de fondo tolerable para tener un valor definido de R. Hemos representado los valores para R = 70, que es el límite normalmente aceptado, y también para R = 65 y R = 75.

Vemos que no hay ganancia de multiplexión para *k* = 5 o 10 (por ejemplo, para *k* = 10 el resultado es casi el mismo para R = 70), pero con *k* = 15 o 20, hay una mejora interesante. También hemos visto que las pérdidas para el tráfico de fondo aumentan con el ancho de banda usado por los flujos de voz. Por tanto, la mejor elección para maximizar el Factor R puede no ser la misma que minimice las pérdidas del tráfico de fondo.

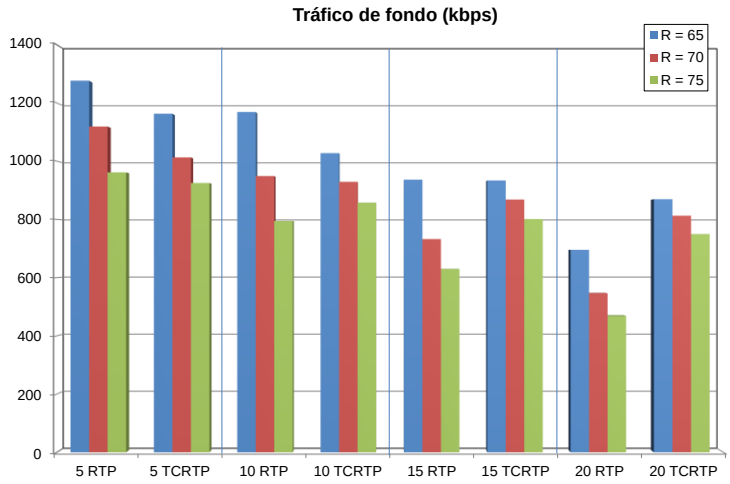


Fig. 9.33. Máximo tráfico de fondo tolerado para R = 65, 70 y 75, con dos muestras por paquete, para el *buffer* limitado en tiempo

Distribución de los flujos

En el apartado anterior hemos estudiado la influencia de la distribución de un número fijo de flujos RTP en la calidad percibida. No hemos estudiado esto para el *buffer* con un número fijo de paquetes porque, como se ha visto en los resultados, el incremento del tamaño del paquete será siempre beneficioso en este caso, por lo que la mejor elección será agrupar todos los flujos en un solo túnel. Esto también ocurre con el *buffer* de alta capacidad. Sin embargo, en el caso del *buffer* limitado en tiempo, el uso de un solo túnel no es necesariamente la mejor solución: en algunos casos se pueden obtener mejores valores del Factor R dividiendo el total de flujos en varios túneles.

La Fig. 9.34 muestra el Factor R para valores fijos de tráfico de fondo, usando diferentes distribuciones de 40 flujos. Como puede verse, los valores obtenidos con *2x20, 4x10* y *5x8* son mejores que los obtenidos para un solo túnel. La razón es la pequeña diferencia de ancho de banda, a causa de la proximidad con la asíntota, y por otro lado la gran diferencia en tamaño de paquete.

También podemos observar que mediante la multiplexión se pueden obtener mejores valores del Factor R respecto al tráfico RTP nativo, principalmente a causa del ahorro de ancho de banda. Por tanto habrá que tomar una decisión dependiendo del comportamiento de nuestro *router* y del ancho de banda de nuestro acceso.

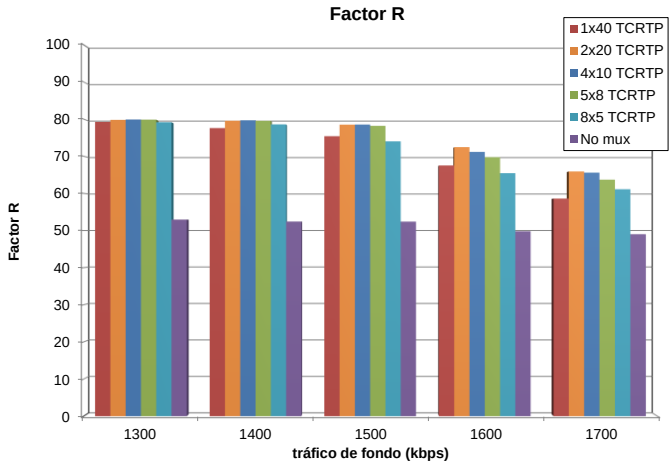


Fig. 9.34. Factor R para tráfico de fondo fijo con diferentes valores de *l*, para 40 flujos, para el *buffer* limitado en tiempo

Conclusiones

Como conclusión del presente capítulo se puede destacar que el comportamiento del *buffer*, causado por su implementación, tiene una gran influencia en la multiplexión RTP, viéndolo desde la perspectiva de la calidad percibida. Se ha visto que, por un lado, TCRTP requiere menos ancho de banda, pero por otro introduce nuevos retardos, principalmente tiempo de retención en el multiplexor y también tiempos de procesado en ambos lados de la comunicación. También modifica el tamaño de los paquetes, pues lógicamente al multiplexar se hacen más grandes. Esto puede aumentar la probabilidad de ser descartados, según cuál sea el comportamiento del *buffer* del *router*.

Dependiendo de la implementación del *buffer* habrá que aplicar una u otra solución. Si el *buffer* está diseñado para poder almacenar un número determinado de paquetes, la multiplexión representa una ventaja puesto que evita en gran medida las pérdidas, al disminuir el ancho de banda de llenado del *buffer*. Si se usa un *buffer* de alta capacidad, el Factor R muestra un comportamiento muy simple: es aceptable hasta que se alcanza el límite del ancho de banda. El mismo comportamiento se observa cuando se reserva un ancho de banda fijo solamente para el tráfico de voz. Pero si se utiliza un *buffer* limitado en tiempo, el Factor R puede dar resultados aceptables incluso con un tráfico total ofrecido por encima del límite de ancho de banda, ya que los paquetes de voz consiguen una ventaja respecto a los paquetes grandes del tráfico de fondo. Por eso esta

implementación del *buffer* parece especialmente adecuada para mantener acotado el retardo, y así proporcionar una mejor calidad.

Se han encontrado situaciones en las que el tráfico multiplexado obtiene mejor calidad que el tráfico RTP nativo. El uso de un túnel TCRTP no empeora la calidad, aunque lleva asociado un coste de ancho de banda y también añade retardos. El número de muestras por paquete también se ha estudiado, y se ha visto que este parámetro también afecta a la calidad percibida. Si los retardos de transmisión en la red son grandes, deberemos evitar el uso de un número alto de muestras por paquete.

Se ha visto también que la solución propuesta por Sze en [SLLY02] supone solamente una ligera mejora respecto a TCRTP, por lo que se ha descartado su uso al no tratarse de una solución estándar.

Por otro lado, se ha estudiado la influencia de la distribución de un número de flujos en diferentes números de túneles, y la conclusión es que agrupar todos los flujos en un solo túnel no será siempre la mejor solución, ya que el incremento del número de flujos no aumenta indefinidamente la eficiencia. Si el *buffer* penaliza los paquetes grandes, será mejor agrupar los flujos en un número de túneles. Finalmente, la capacidad de proceso del *router* se debe tener en cuenta, puesto que el límite de paquetes por segundo que puede gestionar no se debe exceder.

La implementación del *router* y el tráfico que queramos priorizar serán los parámetros usados para decidir la mejor distribución de los flujos RTP en un número de túneles. El número de paquetes por segundo también se debe tener en cuenta, para evitar que aumenten las perdidas.

Como última observación, podemos decir que la multiplexión parece una buena solución para aumentar la calidad del servicio VoIP en escenarios donde un número de flujos comparten el mismo camino, pero debemos conocer el comportamiento del *router,* en ancho de banda, implementación de su *buffer* y límite de paquetes por segundo, para tomar la decisión correcta.

Con estos resultados, vemos interesante integrar la multiplexión en el sistema de telefonía que estamos estudiando, puesto que en este escenario se da la circunstancia de que aparecerán llamadas simultáneas entre el mismo par de oficinas, que podrán por tanto ser multiplexadas. Esta integración se estudia en el siguiente capítulo.

Capítulo 10

INTEGRACIÓN DE ESQUEMAS DE OPTIMIZACIÓN DE TRÁFICO DE VOZ EN EL SISTEMA DE TELEFONÍA IP

Partes de este capítulo se publicaron en el artículo **"Improving Quality in a Distributed IP Telephony System by the use of Multiplexing Techniques,"** International Symposium on Performance Evaluation of Computer and Telecommunication Systems **SPECTS 2011**, La Haya, Holanda, Jun. 2011, cuyos autores son Jenifer Murillo, José Mª Saldaña, Julián Fernández-Navajas, José Ruiz-Mas, Eduardo Viruete y José I. Aznar.

En el capítulo anterior se han estudiado algunos esquemas de optimización del tráfico de voz, y concretamente se ha visto que la multiplexión y compresión de cabeceras TCRTP presenta ventajas para el sistema de telefonía que estamos estudiando. En este escenario es posible que existan llamadas telefónicas simultáneas entre dos sucursales, con lo que se podría establecer un túnel TCRTP que agrupase el tráfico entre cada par de oficinas.

Por tanto, en este capítulo aplicaremos los resultados del capítulo 9 al sistema de telefonía presentado en los capítulos 7 y 8. Se realizarán simulaciones para obtener los valores de los estimadores de calidad subjetiva en un escenario con un número de sucursales. Previamente, será necesario realizar una batería de pruebas en el *testbed,* para obtener los valores de los parámetros de red objetivos en cada caso.

Introducción

El escenario en el que nos situamos es el mismo usado anteriormente. Recordemos sus principales características: está pensado para una empresa con diferentes sucursales en varias áreas geográficas, que pueden ser países distintos. Cada oficina tiene un conjunto de usuarios, y se conecta a RTC mediante un *gateway*. Cada sucursal dispone de un acceso a Internet con un ancho de banda limitado, y las llamadas de VoIP son controladas por un sistema CAC. En el centro de datos hay una PBX *software* que también está conectada a la red IP. El plan de numeración se almacena solamente en la PBX, reduciendo así gastos de gestión. Se utiliza Internet para el tráfico telefónico, evitando de esta manera los costes de las líneas dedicadas. Se ha asumido que el sistema no utiliza ningún protocolo de reserva de recursos y que el único tráfico en tiempo real del que nos ocuparemos de un modo especial será el de VoIP. Para evitar cuellos de botella

en el centro de datos, la PBX sólo toma parte en la señalización de las llamadas, mientras que el tráfico RTP se envía directamente entre sucursales, con un esquema en estrella. Además, el sistema CAC controla las líneas de los *gateway* de cada oficina, permitiendo así la posibilidad de compartir las líneas entre ellas.

En este escenario, si el número de usuarios es suficiente, ocurrirá con frecuencia que varias llamadas tendrán el mismo origen y destino (Fig. 10.1). Por tanto, el uso de técnicas de multiplexión puede suponer una mejora. Como hemos visto en el capítulo anterior, si introducimos en el mismo paquete IP las muestras de diferentes llamadas, comprimiendo cabeceras, podremos ahorrar ancho de banda añadiendo solamente pequeños retardos, que no empeoran la calidad percibida por el usuario, y que en determinados casos pueden incluso aumentarla, a causa del ahorro de ancho de banda.

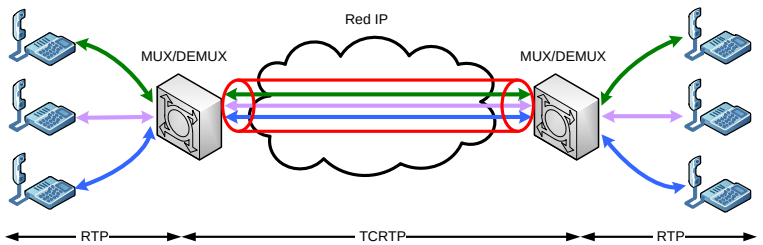


Fig. 10.1. Llamadas que comparten las mismas sucursales origen y destino

En este capítulo consideraremos el sistema de telefonía trabajando en dos posibles modos: el modo *original*, usado hasta ahora, en el que se usa RTP nativo, por lo que cada flujo RTP es independiente del resto, y el modo *multiplexión*, que aprovecha el hecho de que muchas llamadas tengan el mismo origen y destino para multiplexar los flujos usando TCRTP.

Cuando se usa el modo *multiplexión*, el ancho de banda ocupado por las llamadas será menor que el que ocuparían si se enviasen por separado. El sistema CAC debe tener esto en cuenta para tomar las decisiones de admisión, puesto que una llamada no ocupa lo mismo cuando está multiplexada, que cuando se usa el modo *original*.

Veamos ahora qué ancho de banda ocupan las llamadas multiplexadas. En la ecuación (9.2) del capítulo anterior habíamos calculado el valor esperado del tamaño de un paquete multiplexando k paquetes nativos RTP como:

$$E[PS_{k\ flujos}] = CH + k\,(MH + E[RH] + S)$$

Si ahora dividimos por el tiempo entre paquetes *(Inter Packet Time, IPT)*, obtenemos el ancho de banda de un túnel que incluye *k* flujos:

$$BW_{kflujos} = \frac{E[PS_{kflujos}]}{IPT} = \frac{CH}{IPT} + \frac{k(MH + E[RH] + S)}{IPT} \qquad (10.1)$$

Recordemos que *CH* es el tamaño de la cabecera compartida; *MH* el de la cabecera de cada paquete multiplexado; *RH* la cabecera comprimida; y *S* el tamaño de las muestras.

El tiempo entre paquetes *IPT* valdrá 10, 20 o 30 ms según se usen una, dos o tres muestras por paquete respectivamente. En este capítulo usaremos exclusivamente dos muestras por paquete, dado que es el valor más comúnmente utilizado, y que ha dado buenos resultados en el capítulo anterior.

Podemos ver que el ancho de banda calculado (10.1) tiene una dependencia lineal con el número de flujos *k,* así que no hay inconveniente en que el sistema CAC siga basándose en el conteo del número de llamadas establecidas. Lógicamente, es de esperar que los valores que se establezcan como límite de llamadas simultáneas en el CAC, puedan ser mayores en el caso de estar en el modo *multiplexión.*

Metodología de las pruebas

El objetivo de las pruebas es mostrar las mejoras en términos de probabilidad de admisión y Factor R cuando se comparten las líneas de las sucursales y se usa multiplexión RTP. Ambos parámetros influyen en la calidad percibida por los usuarios.

Medidas en la plataforma de pruebas

Se ha usado el método híbrido ya empleado anteriormente (Fig. 8.10). En primer lugar, se ha usado el *testbed* para obtener el retardo y las pérdidas de paquetes en dos situaciones: por un lado, enviando un número de flujos RTP y por otro enviando esos mismos flujos multiplexados entre cada par de oficinas. Como para el tráfico de voz se utiliza un sistema en estrella, en el acceso de una oficina coincidirán los túneles con todas las demás. Por ejemplo (Fig. 10.2), si tenemos cuatro sucursales, en el *router* de acceso de cada una habrá tres túneles: uno con cada una de las otras tres sucursales.

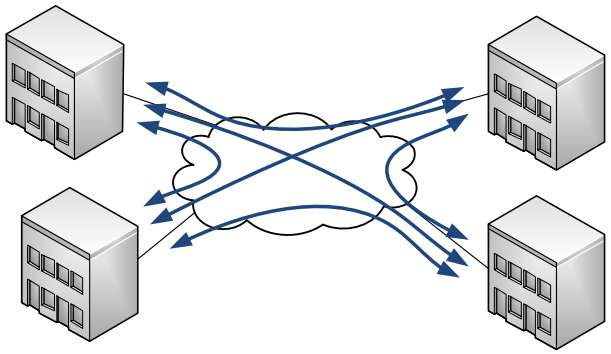


Fig. 10.2. Túneles entre cuatro sucursales

Si denominamos *C* al límite del CAC y *O* al número de oficinas, entonces cada uno de los túneles tendrá como mínimo ninguna, y como máximo *C* llamadas, así que necesitaremos un número de pruebas que se corresponda con las combinaciones con repetición de *C+1* valores para las llamadas (puede ocurrir que no haya ninguna llamada) y *O-1* valores para el número de sucursales. Por tanto, el número de pruebas requeridas será:

$$CR_{C+1,O-1} = \binom{C+O-1}{O-1}$$

Por ejemplo, si tenemos un límite del CAC de 15 llamadas, y 4 sucursales, para el caso de modo *original* se necesitarán 16 pruebas, porque el tráfico RTP es idéntico aunque vaya dirigido a una sucursal o a otra. Pero si queremos establecer un límite para el caso *multiplexión,* podría ocurrir que hubiese un túnel con 3 llamadas dirigido a una sucursal, otro con 5 llamadas a otra sucursal, y un tercero con 2 llamadas a la otra sucursal. Por tanto, tendríamos que hacer 815 pruebas, correspondiendo a las combinaciones con repetición de 16 valores tomados de 3 en 3. Lógicamente, para las situaciones de pocas llamadas puede no ser necesario realizar las pruebas, pues se asume que tendrán una calidad aceptable. De esta manera puede reducirse el número de pruebas a realizar. Utilizaremos estos parámetros para calcular el Factor R en los diferentes momentos de la llamada.

Simulación

Una vez obtenidos los valores del retardo y pérdidas en el *testbed,* se ha usado el entorno de simulación desarrollado en Matlab para generar escenarios con un número de sucursales elevado. Así se han podido obtener realizaciones con diferentes parámetros: número de oficinas, usuarios, líneas de los *gateway*, países,

zonas geográficas, retardos de establecimiento, etc. Cada realización obtendrá la información más importante de los diferentes tramos de cada llamada: instante de comienzo, duración y extensiones implicadas.

Al final de cada simulación, se usan los parámetros de QoS obtenidos en las pruebas del *testbed* para estimar los valores del Factor R. Aunque la simulación no distingue cada paquete enviado, sí puede distinguir cada parte de la llamada, que hemos denominado *intervalo*. Se define como el tiempo en que coinciden el mismo número de llamadas en el *router* de acceso de la sucursal. En cuanto una llamada nueva se establece, o termina otra, termina un intervalo y comienza el siguiente.

El valor de R en cada intervalo depende del número de llamadas simultáneas en un enlace. Por eso, podemos estimar no sólo el Factor R de cada llamada, sino también sus valores a lo largo de los diferentes intervalos. Si un intervalo no influye en los siguientes, esto significaría que la ocupación del *buffer* es independiente entre intervalos. Se ha asumido esto porque el límite de retardo del *buffer* es de decenas de milisegundos, mientras que los intervalos de las llamadas suelen durar decenas de segundos.

Resultados

Se han realizado simulaciones para estudiar la calidad de las llamadas, en términos del Factor R, estimado a partir de los parámetros de calidad obtenidos en el *testbed*. Sólo nos interesan las llamadas que atraviesan la red IP, porque son las que se ven afectadas por la multiplexión RTP, ya que las llamadas entre teléfonos de la misma oficina tendrán una buena calidad al estar en una red local con suficiente ancho de banda.

Se ha configurado un escenario con cuatro zonas en el mismo país, y una sucursal en cada una. Existen 25 usuarios en cada oficina que generan llamadas a los *gateway* de otras oficinas. Se ha usado un tráfico de fondo que satura el acceso en un 80%, para poder hacer las comparaciones en un caso en que la calidad no sea fácil de obtener. Los parámetros variables son la tasa de generación de llamadas y el límite del CAC de las sucursales.

La Fig. 10.3 muestra el Factor R medio para todos los intervalos de cada llamada de una realización, en los modos *original* y *multiplexado*. Se puede ver que en este último caso los valores obtenidos mejoran al multiplexar, pasando de una media de 62 a una de 78. También se observa que la variabilidad del Factor R es menor en el caso multiplexado.

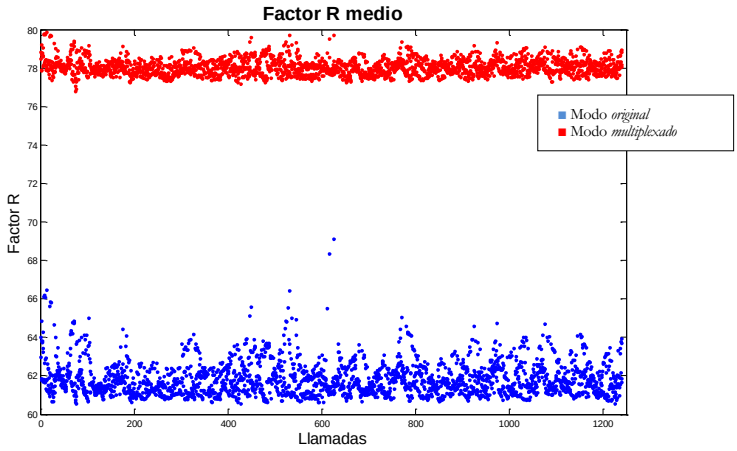


Fig. 10.3. Factor R medio para todas las llamadas de una realización

La Fig. 10.4 compara los dos modos en términos de Factor R en función del límite del CAC. Podemos sacar varias conclusiones de la figura. En primer lugar, puede verse que la multiplexión supone una mejora para los mismos valores de λ y del límite del CAC. Este hecho se debe a que la cantidad de tráfico enviado en el modo *multiplexión* es menor que en el *original*. Por tanto, usando multiplexión, el límite del CAC puede ser mayor si se desea obtener el mismo valor del Factor R, o sea, garantizar la misma calidad para las llamadas.

Por otra parte, si mantenemos λ fija, la figura muestra que a mayor límite del CAC, peor será la calidad en términos de R. Obviamente, incrementar el número de llamadas simultáneas sin aumentar el ancho de banda hará que la calidad disminuya.

Concretamente, se puede observar que si λ = 10 llamadas por hora, el Factor R usando multiplexión está en torno a 80, mientras que en el modo *original* no llega a 70, siendo por tanto, inaceptable. También puede observarse que con multiplexión, el valor del CAC puede incrementarse hasta 25 llamadas simultáneas, manteniendo el Factor R en valores aceptables, y mejorando así la probabilidad de admisión.

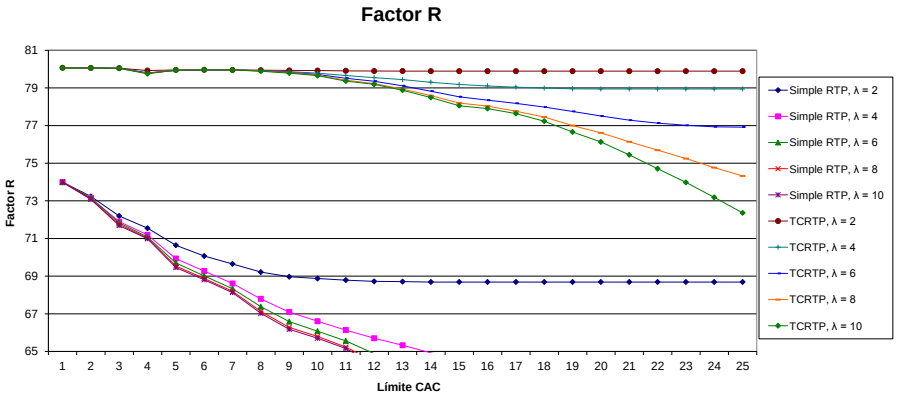


Fig. 10.4. Factor R en modos *original* y *multiplexado* en función del límite del CAC. No se representan los valores bajos del Factor R, porque se considera inaceptable por debajo de 70

Teniendo en cuenta estos resultados y los de probabilidad de admisión, obtenidos en el capítulo 8, vemos que hay un compromiso entre la probabilidad de admisión y la calidad de las llamadas. Por tanto, se podría sacrificar parte del margen del Factor R que nos aporta la multiplexión, para lograr así mejores valores de probabilidad de admisión.

Vemos finalmente que los resultados obtenidos nos permiten calcular el límite del CAC que nos dará valores aceptables para la QoS de las llamadas y la probabilidad de admisión.

Conclusiones

En este capítulo se ha estudiado la inclusión de la multiplexión TCRTP en el sistema de telefonía presentado anteriormente. Se ha utilizado un sistema híbrido de pruebas, obteniendo en un *testbed* en primer lugar los parámetros de QoS, y usándolos después como entradas para las simulaciones del escenario completo.

El objetivo era comparar el comportamiento del sistema en términos de calidad de las llamadas, según se use RTP nativo o túneles TCRTP. Se han realizado simulaciones del sistema en diferentes situaciones. Los resultados obtenidos muestran mejoras en el Factor R cuando se usa multiplexión TCRTP, que se pueden traducir también en aumento del límite del CAC, lo que llevará a un aumento de la probabilidad de admisión.

El entorno de simulación desarrollado se podría también usar para la planificación de un sistema antes de su puesta en marcha, pudiendo conocer de antemano la probabilidad de admisión y los parámetros de QoS esperados, así como el retardo, las pérdidas o el Factor R de las llamadas.

Sección C: Optimización del tráfico de juegos online

En la anterior sección se ha presentado un sistema de telefonía IP. Su diseño incluía además un sistema de control de admisión. Se ha estudiado en una plataforma de pruebas, y también mediante simulación. Esto nos ha permitido conocer su comportamiento desde varios puntos de vista: los tiempos de establecimiento, los parámetros que determinan la QoS, como son el retardo, las pérdidas o el Factor R, así como la probabilidad de admisión. Una mejora importante introducida ha sido la inclusión de métodos de optimización del tráfico, principalmente multiplexión y compresión de cabeceras. Se ha visto cómo estas técnicas permiten obtener una mejor calidad con los mismos anchos de banda, y cómo esta ventaja se puede traducir también en una mejora de la probabilidad de admisión.

En la sección que ahora comenzamos se va tratar sobre la adaptación de las técnicas empleadas para el tráfico de voz, para utilizarlas también en otros servicios de tiempo real. Se debía por tanto seleccionar otro servicio con características similares a las de VoIP, en cuanto a las altas tasas de paquetes pequeños. Se ha seleccionado el tráfico de juegos *online,* y en concreto el de juegos FPS por las siguientes razones: en primer lugar, es un servicio que está creciendo mucho en los últimos años; también es un servicio con unos requerimientos temporales muy estrictos, como ocurría con en el caso de VoIP; finalmente, la existencia de escenarios en los que varios flujos comparten el mismo camino, al igual que ocurría con los flujos de voz en el sistema de telefonía. Decidirnos por este servicio permite mostrar que la multiplexión se puede generalizar, para utilizarse en muchos otros servicios además de VoIP. Aunque el método TCRTP del IETF está pensado para multiplexar flujos RTP, si lo generalizamos puede ser válido también para multiplexar paquetes UDP, que son utilizados por estos juegos.

Para el caso de los juegos *online* no se estudiarán los mismos parámetros de calidad que para la voz. Por ejemplo, la probabilidad de admisión es un parámetro que se puede estudiar para el sistema de telefonía IP, pero que no tiene una aplicación directa en los juegos. Sí se puede estudiar cuántos jugadores podrán tener buena calidad teniendo en cuenta las limitaciones de la red, pero para este servicio tiene mayor importancia la capacidad de proceso del servidor. En definitiva, se estudiarán los parámetros que determinan la calidad subjetiva, y su variación si se usan técnicas de multiplexión y compresión, pero sin construir un sistema completo, como se ha hecho en el caso de la telefonía IP.

Capítulo 11

ADAPTACIÓN DEL ESQUEMA DE OPTIMIZACIÓN PARA SU USO EN JUEGOS *ONLINE*

Partes de este capítulo, especialmente la descripción de escenarios y la comparativa entre los diferentes juegos, han sido publicadas en el artículo **"First Person Shooters: Can a Smarter Network Save Bandwidth without Annoying the Players?," IEEE Communications Magazine**, Consumer Communications and Networking Series, vol. 49, no. 11, pp. 190-198, Nov. 2011, cuyos autores son José Mª Saldaña, Julián Fernández-Navajas, José Ruiz-Mas, José I. Aznar, Eduardo Viruete y Luis Casadesus.

La comparativa entre las diferentes políticas de multiplexado ha sido publicada en el artículo **"Comparative of Multiplexing Policies for Online Gaming in terms of QoS Parameters," IEEE Communications Letters**, vol. 15, no. 10, pp. 1132-1135, Oct. 2011, cuyos autores son José Mª Saldaña, Julián Fernández-Navajas, José Ruiz-Mas, José I. Aznar, Luis Casadesus y Eduardo Viruete.

Un estudio detallado sobre los beneficios de la técnica TCM se publicó en el artículo **"Bandwidth Efficiency Improvement for Online Games by the use of Tunneling, Compressing and Multiplexing Techniques,"** Proc. International Symposium on Performance Evaluation of Computer and Telecommunication Systems **SPECTS 2011**, La Haya, Holanda, Jun. 2011, cuyos autores son José Mª Saldaña, Jenifer Murillo, Julián Fernández-Navajas, José Ruiz-Mas, José I. Aznar y Eduardo Viruete.

Introducción

En los capítulos introductorios se ha visto que los juegos *online*, y especialmente los FPS, tienen unos requerimientos de tiempo real muy estrictos, y que utilizan una arquitectura cliente-servidor. Las acciones de los usuarios se deben transmitir con rapidez al servidor, que calcula el nuevo estado del juego y a su vez lo comunica a todos los usuarios. Esto puede hacer que el *router* de acceso, que conecta al usuario con Internet, deba transmitir un gran número de paquetes por segundo con poco retardo hasta el servidor.

Existen escenarios en los que muchos jugadores comparten la misma ruta desde la red de acceso hasta el servidor del juego: por ejemplo, los denominados *cibercafés,* muy populares en algunos países [BM10], disponen frecuentemente de ordenadores capaces de ejecutar juegos *online,* y grupos de jugadores suelen acudir a estos establecimientos [GS07]. También el tráfico entre los servidores de un mismo juego es otro escenario donde se comparte la misma ruta.

Muchos juegos utilizan patrones de tráfico similares, generando altas tasas de paquetes pequeños y presentando por eso un gran *overhead*. Los tráficos son diferentes en el sentido cliente-a-servidor y en el servidor-a-cliente. El tráfico generado por un grupo de usuarios se podría comprimir para ahorrar ancho de banda, teniendo en cuenta que la red de acceso suele constituir el cuello de botella más restrictivo. Además, en el caso de conexiones asimétricas, como el ADSL, el *uplink* tiene normalmente menos ancho de banda que el *downlink*, y por eso influye de manera diferente en el tráfico del juego.

Podríamos aplicar las técnicas de optimización de tráfico, estudiadas para VoIP en capítulos anteriores, a este otro servicio. Mediante el uso de un elemento multiplexor (Fig. 11.1) encargado de retener los paquetes, comprimir las cabeceras y utilizar multiplexión para su envío, se podría ahorrar ancho de banda, con el coste de incluir un nuevo retardo, que estará causado principalmente por el tiempo de espera en la cola del multiplexor.

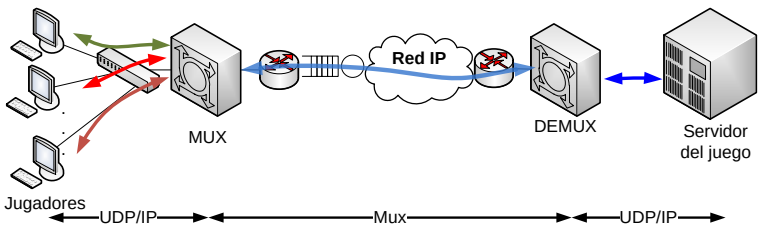


Fig. 11.1. Esquema del sistema de multiplexión

En el otro lado de la comunicación, el servidor debería implementar el demultiplexor y el descompresor, lo que conllevaría cierta capacidad de proceso, y espacio para almacenar el *contexto* de cada flujo, es decir, la información necesaria para reconstruir las cabeceras comprimidas (algunas decenas de bytes [DNP99]). Esto no produce un problema de escalabilidad, ya que el servidor, de hecho, ya almacena el estado del juego de cada usuario. Por otra parte, el ahorro de ancho de banda y de paquetes por segundo sería beneficioso para el servidor.

Nos centraremos en el tráfico cliente-a-servidor, ya que es donde se puede conseguir un mayor ahorro y, en muchos escenarios, como en un acceso ADSL, se trata del enlace más restrictivo. Además, tendremos que medir la degradación de los parámetros que determinan la calidad experimentada por los usuarios para ajustar correctamente el método, encontrando un compromiso entre el ahorro de ancho de banda y la calidad.

En los últimos años se está dando una tendencia encaminada al desarrollo de nuevas arquitecturas y técnicas para trasladar inteligencia desde el servidor hasta los extremos de la red. Esta tendencia se puede observar en las redes de nueva generación y también en nuevos servicios emergentes. Los servicios tradicionales, como la navegación web o la VoIP han usado con frecuencia el concepto de *proxy* como un componente clave para transferir carga de trabajo desde el núcleo de la red hasta los extremos. Vemos que este concepto coincide con lo que nos proponemos en esta tesis, y que también se puede aplicar a los juegos *online*. De hecho, Mauve y otros [MFW02] estudiaron las ventajas del uso de *proxy* para dar soporte a los juegos. También Bauer y otros [BRS02] propusieron en la misma línea el uso de *booster boxes*, que estarían dentro del concepto de *middleboxes* propuesto por el IETF en el RFC 3303 [SKRM+02]. Podrían ubicarse junto al *router* de acceso, permitiendo a la red asumir algunas funciones de la aplicación. Si la red es más inteligente, se podrán implementar mecanismos de QoS para mejorar la experiencia del usuario. Un *proxy* podría detectar y multiplexar automáticamente flujos con el mismo origen y destino.

Uno de los grandes problemas que se plantean a la hora de poner en marcha este servicio es el de la escalabilidad, pues el tráfico del servidor a los clientes aumenta con el cuadrado del número de jugadores, ya que la aplicación de cada jugador debe conocer los movimientos del resto de jugadores. En la Fig. 11.2 se puede observar un esquema simplificado con sólo tres jugadores: cada jugador debe enviar la información de sus movimientos, pero debe recibir la información del resto de jugadores. La información recibida por el servidor es *N* veces la de un jugador, pero la total enviada por el servidor es *N(N-1)* veces la de un jugador.

Los paquetes generados por el servidor aumentan de tamaño con el número de jugadores, mientras que los generados por los clientes mantienen su tamaño. En consecuencia, si un número de jugadores se conecta a través de un *proxy*, el servidor podría enviarle un solo paquete, y el *proxy* reenviaría esta información a todos los participantes. Esta podría ser una primera utilidad del *proxy*.

Otros problemas relacionados con el tráfico del cliente al servidor son, por una parte, que consiste en altas tasas de pequeños paquetes UDP, lo que conlleva baja eficiencia; por otra parte, el número de paquetes que el *router* tendrá que gestionar será elevado. En [FCFW05] se estudió el problema de la infraestructura para el soporte de este servicio, y una de las conclusiones fue que hay casos en que el número de paquetes por segundo puede ser un cuello de botella más exigente que el límite de ancho de banda.

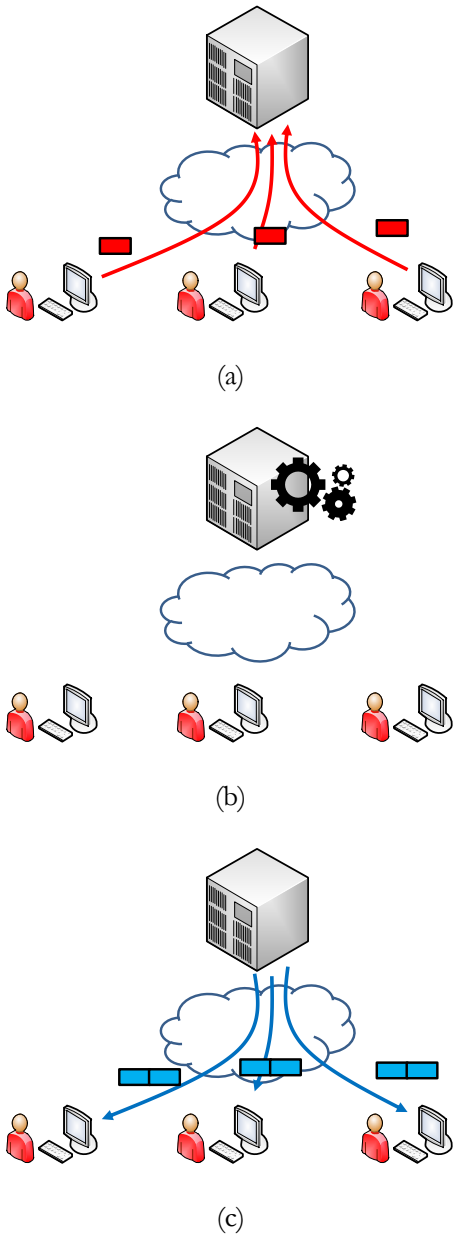


Fig. 11.2. Fases del envío de tráfico en un juego FPS: a) los jugadores informan al servidor; b) el servidor calcula el nuevo estado del juego; c) el servidor informa a los jugadores

Los mismos *proxy* se encargarían de implementar las técnicas de compresión y multiplexión cuando cierto número de jugadores compartan la misma conexión,

proporcionando así dos ventajas: la reducción del ancho de banda y la de los paquetes por segundo.

Presentaremos por tanto en este capítulo un método, que hemos denominado TCM (Tunelado, Compresión, Multiplexión), para multiplexar el tráfico de juegos *online* cuando varios jugadores comparten el mismo enlace. Este método se ha adaptado de la técnica TCRTP para flujos RTP explicada en capítulos anteriores. Se probará con el tráfico de cliente-a-servidor de juegos FPS, ya que este tráfico representa un buen ejemplo de las tres características básicas: requerimientos estrictos de retardo, baja eficiencia y altas tasas de paquetes por segundo. Se deberá realizar un estudio teórico exhaustivo del método propuesto, viendo, por un lado, qué ahorro de ancho de banda se puede obtener y, por otro, qué inconvenientes tiene, es decir, cómo influye el método en los retardos, las pérdidas de paquetes, el *jitter*, etc. También se deberán comparar las diferentes políticas que se pueden usar para la multiplexión.

Escenarios de aplicación

Existen diferentes infraestructuras de red en las que se podrían utilizar *proxy* para juegos, que podrían ser gestionados por el proveedor de acceso a Internet, los usuarios finales o bien por el proveedor del juego. Lógicamente, se podrían dar diferentes combinaciones aunque, por claridad, los estudiaremos por separado.

Proveedor de acceso y usuarios finales

La Fig. 11.3 muestra un *proxy* junto al *router* de acceso, y gestionado también por el proveedor. Esta ubicación le permitiría obtener información del estado de la red, y tomar decisiones para mejorar la calidad. Este *proxy* podría implementar funciones estándar que fueran válidas para juegos de distintos fabricantes, dando así más oportunidades de negocio. El proveedor de acceso podría establecer un contrato con el usuario para optimizar el tráfico de juegos que genere.

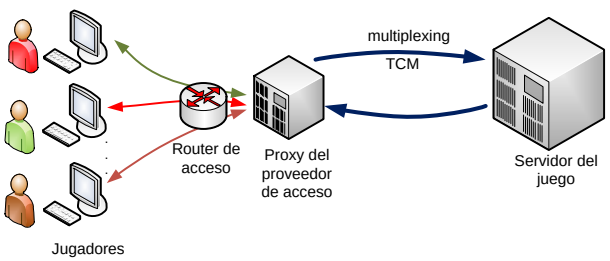


Fig. 11.3. Escenario de aplicación de TCM: *proxy* gestionado por el proveedor de acceso

Como ejemplo de un escenario real con varios jugadores compartiendo el mismo acceso, se puede considerar un *cibercafé*. Hoy en día constituyen un modo popular de acceder a Internet en muchos países. También es muy frecuente su uso por parte de jugadores *online*. En [GS07] se presentó un estudio del comportamiento de los usuarios de estos establecimientos, y una de las conclusiones fue que los juegos *online* son uno de los servicios más utilizados. Los *cibercafés* están presentes en todo el mundo, pero tienen una especial relevancia en países en vías de desarrollo [BM10], [FKW08]. Las infraestructuras de telecomunicaciones varían mucho de un lugar a otro, y en muchas ocasiones no se dispone de las últimas tecnologías de acceso, por lo que las técnicas de ahorro de ancho de banda pueden tener un gran interés.

Otra cuestión a considerar es la asimetría de muchas de las tecnologías de acceso actuales, como ocurre con el ADSL: aunque proporcionan un ancho de banda aceptable en el *downlink,* muchas veces éste es escaso en el *uplink*. Por tanto, ahorrar ancho de banda en el sentido cliente-a-servidor puede tener un mayor interés.

Otra opción sería utilizar un *proxy* por *software*, que un grupo de usuarios o bien el dueño del *cibercafé* podrían instalar en su red local (Fig. 11.4). Se podría distribuir con el juego, del mismo modo que se distribuyen servidores para redes locales.

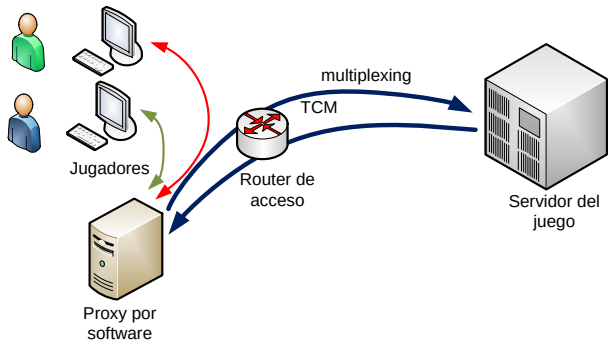


Fig. 11.4. Escenario de aplicación de TCM: *proxy* por *software*

Otro ejemplo de escenario donde aplicar TCM sería el de una red de acceso basada en WiMAX. Esta tecnología se puede emplear para proporcionar acceso en zonas rurales, donde el despliegue de una infraestructura cableada resultaría muy caro. En este caso, el tráfico de varios jugadores puede compartir también la misma ruta, por lo que el uso de TCM podría resultar interesante (Fig. 11.5).

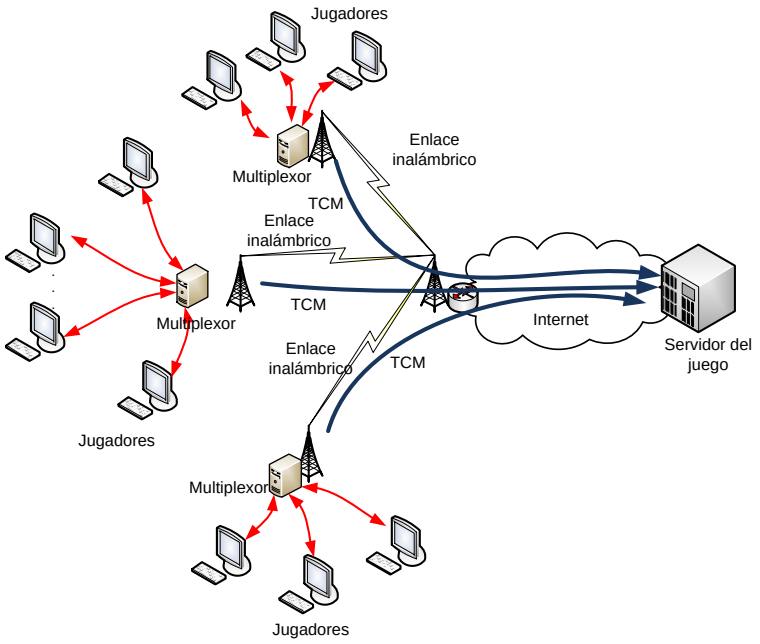


Fig. 11.5. Escenario de aplicación de TCM: red de acceso WiMAX

Proveedor del juego

También podemos considerar los *proxy* como extensiones del servidor, que delega en ellos algunas tareas (Fig. 11.6), El proveedor del juego los podría utilizar para compartir información e *inteligencia* por la red, ahorrando de este modo carga de trabajo al servidor, y evitando así el cuello de botella que éste puede suponer.

Un *proxy* debería ser capaz de interactuar con otros, y así en este caso el tráfico de varios jugadores compartiría la misma ruta, por lo que un algoritmo que ahorre ancho de banda podría también ser interesante para el tráfico *proxy* a *proxy*.

Los *proxy* podrían proporcionar algunas otras ventajas. Si observamos la Fig. 11.6, vemos que Bob, que tiene una conexión inalámbrica, quizá no pueda jugar una partida con Alice o Dash, pero es posible que el *proxy* más cercano pueda gestionar una partida con Helen y Jack. Los *proxy* podrían servir juegos a los jugadores cercanos, independientemente del servidor central.

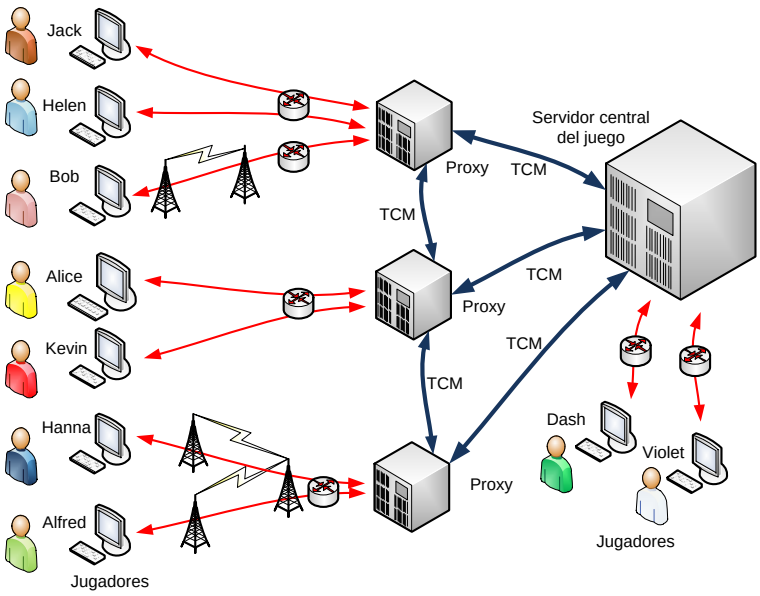


Fig. 11.6. Escenarios de aplicación de TCM: infraestructura de *proxy* del proveedor del juego

Características del tráfico de los juegos FPS

Los juegos FPS generan diferentes tráficos según el momento de la partida en que se encuentren. Por ejemplo, puede ocurrir que se genere tráfico TCP mientras el juego está comenzando, para la descarga de los mapas de los escenarios, o la personalización de algunas imágenes de los personajes. De todas formas, en este estudio nos centraremos sólo en el tráfico *activo*, que es el que aparece mientras la partida está en marcha.

Este tráfico tiene dos comportamientos diferentes: por un lado, los clientes generan paquetes UDP pequeños (típicamente de algunas decenas de bytes) con diferentes distribuciones para el tiempo entre paquetes. Estos flujos comunican al servidor la información de las acciones de los jugadores. La cantidad de tráfico generada por un jugador no depende del número de jugadores de la partida. Solamente algunas consolas permiten hasta cuatro jugadores en la misma máquina (por ejemplo *Halo 2* para la consola *Xbox*). En este caso, el tráfico generado por una sola máquina sí podría variar con el número de jugadores.

El servidor debe mantener el estado del juego y comunicar los movimientos de cada jugador al resto, así que el tráfico generado por el servidor aumenta con el

cuadrado del número de jugadores. Estos paquetes son típicamente más grandes que los generados por los clientes.

En [RHS10] se presentó un estudio de las distribuciones de tráfico para 17 juegos FPS diferentes, recapitulando los modelos desarrollados en la literatura. Cada juego se caracterizó mediante cuatro distribuciones estadísticas diferentes: tiempo entre paquetes y tamaño de los paquetes, en el sentido cliente-a-servidor y en el servidor-a-cliente. Es frecuente que sus autores comparen los modelos propuestos con el tráfico real, usando el *Gráfico Q-Q* u otras herramientas matemáticas para validarlos ([ZA05], [LABC03], [BA06]).

Algoritmo de Tunelado, Compresión y Multiplexión (TCM)

Como se ha visto en apartados anteriores, en el RFC 4170 [TKW05], el IETF aprobó TCRTP para comprimir y multiplexar flujos RTP. En primer lugar se utiliza ECRTP para comprimir las cabeceras. Después, varios paquetes se incluyen en uno usando PPPMux. Finalmente, se usa un túnel L2TP para enviar el paquete multiplexado extremo a extremo. Este método es adecuado para el tráfico de VoIP, ya que éste utiliza el protocolo RTP.

En el caso de los juegos *online* el tráfico no es RTP, por lo que sólo podemos comprimir las cabeceras IP/UDP. Por tanto, no podemos usar ECRTP para comprimir las cabeceras. Pero los métodos IPHC y ROHCv2 sí permiten comprimir cabeceras IP/UDP, así que deberemos recurrir a ellos para adaptar TCRTP a nuestras necesidades. La Fig. 11.7 muestra la pila de protocolos, y la Fig. 11.8 la estructura de un paquete TCM.

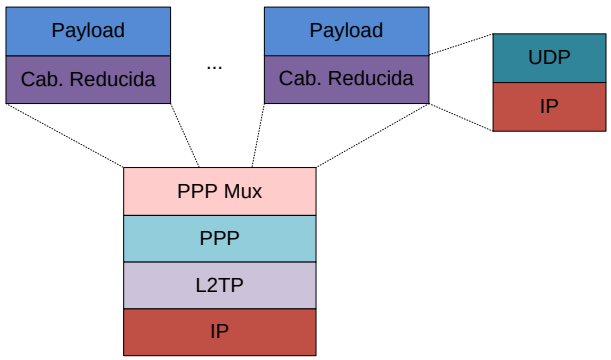


Fig. 11.7. Pila de protocolos de TCM

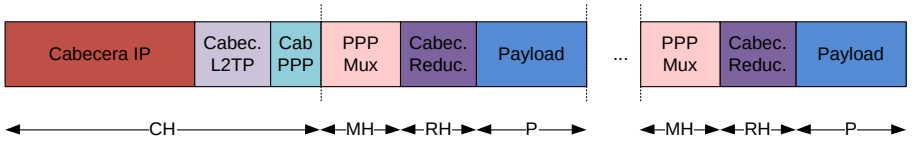


Fig. 11.8. Esquema de un paquete multiplexado

Igual que se hizo en capítulos anteriores para los paquetes de voz, podemos dividir un paquete multiplexado en las siguientes partes:

- **CH:** Cabecera Común *(Common Header)*: Corresponde a las cabeceras IP, L2TP y PPP.
- **MH:** Cabecera PPPMux *(Multiplexed Header):* Se incluye al principio de cada paquete comprimido.
- **RH:** Cabecera reducida *(Reduced Header)*: Corresponde a la cabecera comprimida IP/UDP del paquete original.
- **P:** Contenido *(Payload):* Contenido de los paquetes originales generados por la aplicación.

Para ilustrar el problema de la eficiencia en los flujos de servicios de tiempo real, se ha incluido la Fig. 11.9, que está hecha a escala. En primer lugar se representa un paquete TCP de 1.500 bytes, que es el típico usado por aplicaciones de descarga de ficheros, puesto que es el tamaño máximo en muchas de las redes cableadas actuales. Este tipo de paquetes se usa típicamente en servicios no de tiempo real, como el correo, la *web*, el FTP o las aplicaciones P2P. Como se puede ver, su eficiencia es muy alta. Hemos representado la eficiencia (la relación entre el *payload* y el tamaño total) con la letra η. Como ejemplo de poca eficiencia (33%) se puede ver un paquete de VoIP.

En la misma figura, si miramos al paquete servidor-a-cliente de un juego FPS, vemos que la eficiencia empeora, bajando hasta un 85%, aunque sigue siendo aceptable. Los requerimientos de tiempo real hacen que la aplicación deba enviar información con una frecuencia alta, por lo que el *payload* es pequeño, mientras que la cabecera IP/UDP estándar requiere 28 bytes. Finalmente, si observamos el paquete cliente-a-servidor del juego FPS, podemos ver que la eficiencia ha descendido hasta un 68%. Si aplicamos el método TCM, podemos conseguir que suba hasta un 83%. La figura muestra el ahorro que se consigue en el caso de cuatro paquetes, pero este número podría ser significativamente mayor

Hoy en día, la nueva versión del protocolo de Internet, IPv6, está comenzando a reemplazar a su predecesor IPv4, por lo que consideramos importante estudiarlo también. Para más información al respecto, se puede acudir a [CGKR10], donde se presentó un estudio sobre la adopción de IPv6 en la actualidad. Uno de los inconvenientes de esta nueva versión es que introduce un *overhead* mayor, puesto que la cabecera es como mínimo 20 bytes más grande que la de IPv4.

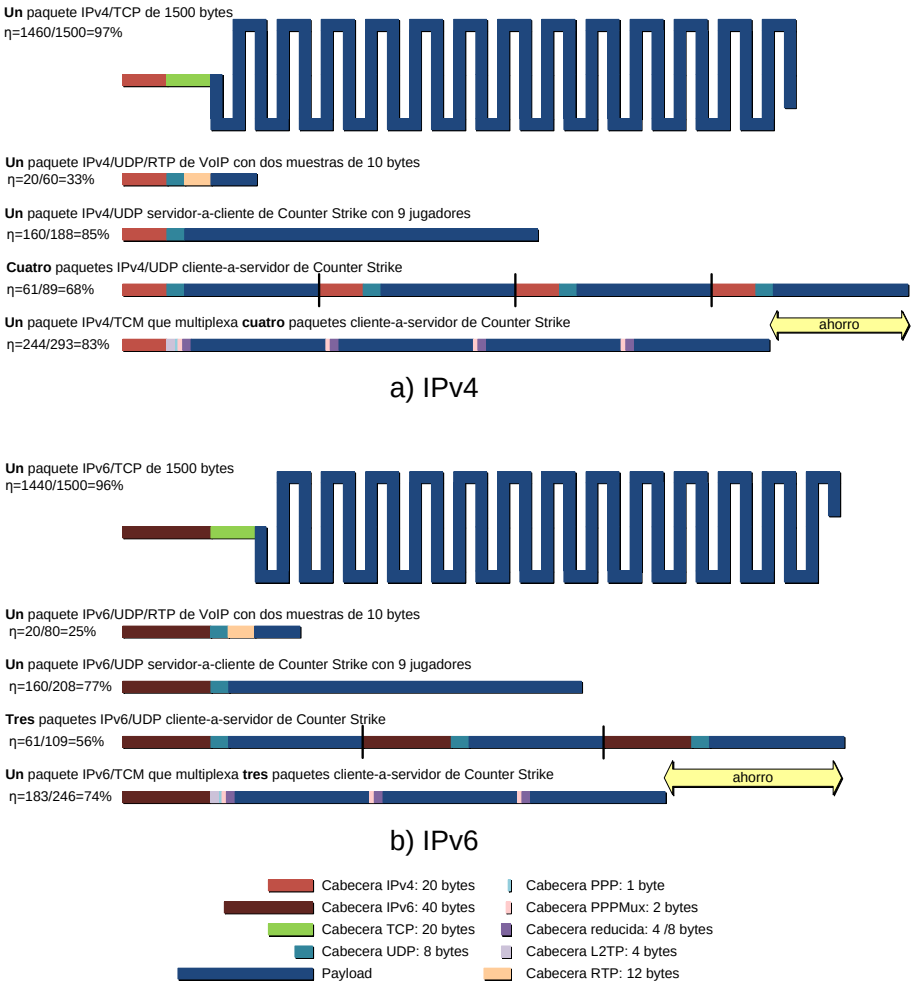


Fig. 11.9. Eficiencia de los paquetes para a) IPv4 b) IPv6. Los tamaños de los paquetes están a escala

La Fig. 11.9 b) ilustra el problema de la eficiencia para el caso de IPv6. Se puede observar que, aunque el *overhead* añadido no es significativo para el caso de paquetes grandes TCP, sí lo es para el tráfico de los juegos: la eficiencia de los paquetes cliente-a-servidor cae hasta un 56%. Por eso, en el escenario que nos

encontraremos dentro de unos años, la importancia de las técnicas de ahorro de ancho de banda será aún mayor, debido a la implantación de IPv6.

Análisis teórico del método propuesto

A continuación expondremos un análisis del ahorro de ancho de banda que se puede lograr con esta técnica. Asumiremos que el tamaño del paquete multiplexado nunca excederá los 1.500 bytes, cosa cierta para los tráficos usados.

En primer lugar, deberemos definir una política que decida cuántos paquetes nativos se multiplexan, con qué periodicidad, etc. En este trabajo se han considerado cuatro de entre las posibles:

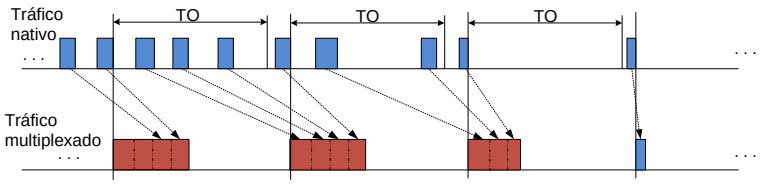
- *Fixed Number* (número fijo): Cada paquete multiplexado incluiría un número fijo *FN* de paquetes nativos (*FN* > 1).

- *Size Limit* (límite de tamaño): En algún caso nos puede interesar que los paquetes multiplexados tengan un tamaño determinado. Por tanto, con esta política se enviará un paquete multiplexado cada vez que se alcance el tamaño predefinido *SL*.

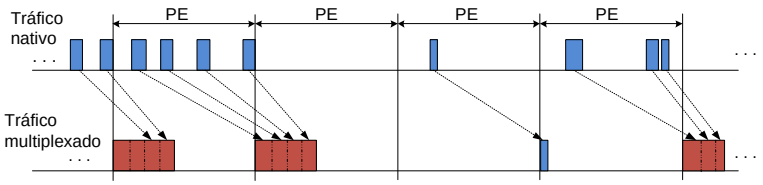
- *Timeout* (límite de tiempo): Como se observa en la Fig. 11.10 a), si al llegar un paquete ha pasado un tiempo *TO* desde el envío del último, se envía uno nuevo. En el caso de que haya sólo un paquete a enviar, se manda en su forma nativa, ya que el uso de un túnel aumentaría su tamaño.

- *Period* (periodo): Como se puede ver en la Fig. 11.10 b), se define un periodo *PE* y se envía un paquete multiplexado al final de cada periodo, incluyendo todos los que hayan llegado. En este caso se harán dos excepciones: si no ha llegado ningún paquete, no se envía nada. Si hay un solo paquete, se envía en su forma nativa.

La decisión sobre la política a utilizar depende de muchos factores: las dos primeras son las que presentarán menores requerimientos computacionales, puesto que el multiplexor sólo debe contar paquetes o bytes. Las otras dos son más adecuadas para controlar el retardo añadido, que tendrá influencia en la QoS. Aunque aquí estudiaremos separadamente cada política, se podrían combinar de diferentes maneras.



(a)



(b)

Fig. 11.10. Comportamiento de la política de multiplexión a) *timeout* b) *period*

Nos referiremos a los paquetes generados por la aplicación como *nativos*, en contraste con los multiplexados (*mux*). Al igual que hemos hecho para voz, usaremos la relación de anchos de banda *(BWR)*, que es un parámetro interesante para conocer los beneficios obtenidos con la multiplexión, y se calcula como la división de los anchos de banda de los tráficos *mux* y *nativos*.

$$BWR = \frac{BW_{mux}}{BW_{nativo}} \tag{11.1}$$

El ahorro de ancho de banda será por tanto:

$$Ahorro = \frac{BW_{nativo} - BW_{mux}}{BW_{nativo}} = 1 - \frac{BW_{mux}}{BW_{nativo}} = 1 - BWR \tag{11.2}$$

Calcularemos *BWR* para cada política como el cociente entre el número de bytes correspondientes al tráfico nativo y al multiplexado, puesto que ambos tráficos se envían en el mismo intervalo de tiempo. Denominaremos k al número de paquetes multiplexados. *NH* (*Normal Header)* se refiere al tamaño de una cabecera normal IP/UDP. Para obtener *BWR*, calcularemos primero el número de bytes de un paquete multiplexado, distinguiendo entre tener uno o varios paquetes:

$$bytes_{mux} = \Pr(k=1)(NH + E[P])$$

$$+ \Pr(k>1)[CH + E[k \mid k>1](MH + E[RH] + E[P])] \tag{11.3}$$

Y el tamaño de los paquetes nativos será:

$$bytes_{nativo} = E[k]\ (NH + E\ [P]) \tag{11.4}$$

Dividiendo (11.3) entre (11.4) y simplificando, obtenemos el valor de *BWR*:

$$BWR = \frac{\Pr(k=1)}{E[k]} + \Pr(k>1)\frac{CH}{E[k](NH + E[P])} +$$

$$+ \Pr(k>1)\frac{E[k \mid k>1]}{E[k]}\frac{MH + E[RH] + E[P]}{NH + E[P]} \tag{11.5}$$

El primer término está causado por la decisión de no multiplexar cuando hay un solo paquete. El segundo expresa cómo la cabecera común es compartida por todo el paquete, y se va haciendo menor según aumenta el número de paquetes multiplexados. El tercer término depende del algoritmo de compresión y del tamaño medio de paquete generado por la aplicación.

Vemos que si tenemos un gran número de usuarios y un número de paquetes multiplexados también grande, el primer y segundo términos tenderán a ser despreciables. Con respecto al tercero, Pr ($k > 1$) será casi la unidad, y lo mismo ocurrirá con $E[k \mid k > 1] / E[k]$. De esta manera, podemos obtener una expresión para la asíntota de *BWR:*

$$BWR_a = \frac{MH + E[RH] + E[P]}{NH + E[P]} \tag{11.6}$$

Podemos observar que cuanto menor sea $E[P]$, menor será el valor de la asíntota. Por eso, la técnica presentada tendrá un buen comportamiento para aplicaciones que generen un gran número de paquetes pequeños, como ocurre en los juegos FPS. Lógicamente, lo esperado es que a mayor número de jugadores se consiga un mayor ahorro, ya que el mismo número de paquetes podrá ser multiplexado añadiendo menores retardos.

Estimación del ahorro máximo que se puede obtener

Como se ha visto, el valor de la asíntota BWR_a es el que nos dará el máximo ahorro que se podrá conseguir para cada juego. Este valor es independiente de la política que se utilice. Utilizaremos parámetros reales para obtener este valor usando algunos juegos comerciales. Los parámetros de los protocolos usados serán:

- *NH*: 28 bytes para IPv4 y 48 para IPv6.

- *CH*: 25 bytes para IPv4: 20 corresponden a la cabecera IP, 4 a L2TP y 1 a la cabecera PPP. Para IPv6, *CH* = 45 bytes.

- *MH*: 2 bytes, correspondientes a PPPMux.

- *E*[*P*]: El valor del *payload* UDP depende de la aplicación usada.

- *E*[*k*]*:* El número de paquetes de los *N* jugadores que se van a multiplexar.

- *E*[*RH*]: En este ejemplo, para situarnos en el peor caso, hemos considerado IPHC comprimiendo las cabeceras UDP a 2 bytes, usando sólo 8 bits para el campo CID, y evitando el *checksum* opcional. Las cabeceras IPv4 e IPv6 se pueden también comprimir a 2 bytes. Así que consideraremos una media de 4 bytes para las cabeceras comprimidas y 28 o 48 bytes para las cabeceras completas IP/UDP, que se envían cada 5 segundos (el parámetro F_MAX_TIME que usa por defecto IPHC).

Usando estos valores, obtenemos los resultados que se muestran en la Tabla 11.1. Son valores de la asíntota, o sea, el mejor *BWR* que se puede obtener si el número de jugadores y el periodo son lo suficientemente grandes. Como puede observarse, se han obtenido para IPv4 e IPv6. Hemos seleccionado algunos juegos populares, y los valores concretos de paquetes por segundo y tamaño de paquete se han obtenido de [FCFW05] y [RHS10] [1].

Juego	**Motor gráfico**	**E[P]**	**λ**	**BWR_a IPv4**	**BWR_a IPv6**
Unreal T 2003	Unreal 2.0	29,5	25	62%	46%
Quake III	Id Tech 3	36,15	93	65%	50%
Quake II	Id Tech 2	37	26,38	66%	51%
Counter Strike 1	GoldSrc	41,09	24,65	68%	53%
Halo 2	Halo2	43,2	25	69%	54%

Tabla 11.1. Valores de la asíntota de *BWR* para diferentes juegos

[1] Los valores para *Halo 2* se refieren a una consola con un solo usuario [ZA05]. El valor de λ para *Quake III* es el que se obtiene con las tarjetas gráficas más rápidas.

Observamos que los valores de *BWR* obtenidos son significativos: todos los juegos permiten un ahorro de un 30% en ancho de banda para IPv4, y este ahorro puede llegar hasta el 54% si se usa IPv6 con algunos títulos.

Parámetros que dependen de las políticas

Se presentan ahora los cálculos necesarios para obtener una expresión analítica de Pr (k=0), Pr (k=1), Pr (k>1), $E[k]$ y $E[k|k>1]$, que se requieren para construir las gráficas de *BWR* y también los paquetes por segundo correspondientes a cada política. Como se verá, estos valores varían según el comportamiento estadístico del tiempo entre paquetes de cada juego. Las distribuciones estadísticas más usadas son la exponencial, determinista, normal, log-normal y extrema [ZA05]. En este trabajo se realizarán los cálculos teóricos para las distribuciones exponencial y determinista.

En primer lugar, podemos calcular $E[k]$ como:

$$E[k] = \Pr(k=0)\ E[k|k=0] +$$

$$+\Pr(k=1)\ E[k|k=1] + \Pr(k>1)\ E[k|k>1] \tag{11.7}$$

Pero si tenemos en cuenta que $E[k|k=0]=0$ y que $E[k|k=1]=1$, obtenemos:

$$E[k|k>1] = \frac{E[k] - \Pr(k=1)}{\Pr(k>1)} \tag{11.8}$$

Si utilizamos la política *Fixed Number*, entonces tenemos que $E[k]=FN$, y que $\Pr(k>1)=1$.

En el caso de la política *Size Limit*, si asumimos que $\Pr(k>1)=1$, podemos hallar k, teniendo en cuenta que el número de paquetes debe ser un número entero, como:

$$k = \left\lceil \frac{SL - CH}{MH + RH + P} \right\rceil \tag{11.9}$$

Y por tanto E[k] será:

$$E[k] = \frac{SL - CH}{MH + E[RH] + E[P]} + \frac{1}{2} \tag{11.10}$$

Política timeout

Definiremos λ como la tasa de generación de paquetes de cada uno de los jugadores, y N como el número de jugadores.

Si utilizamos la política *timeout*, definimos como TO el intervalo que se usa como plazo para decidir si se envía un nuevo paquete. Hemos denominado k al número de paquetes multiplexados. Si consideramos que las llegadas de paquetes de los N jugadores son independientes, tenemos que durante el intervalo TO llegarán en media $N\lambda TO$ paquetes. Para calcular el número de paquetes multiplexados, tendremos que añadir el paquete que llega después del *timeout* y causa el envío del paquete multiplexado. Por tanto, tenemos:

$$E[k] = N\,\lambda\,TO + 1 \tag{11.11}$$

Podemos calcular también la tasa de paquetes multiplexados por segundo que se generará. Esta tasa es el inverso del tiempo entre paquetes. Este tiempo corresponderá al intervalo TO más el tiempo que pasa hasta que llega el siguiente paquete, que será en media $1/N\lambda$:

$$E[\textit{Tiempo entre paquetes}] = TO + \frac{1}{N\lambda} \tag{11.12}$$

Por tanto, la tasa de generación de paquetes pps_{TO} será:

$$pps_{TO} = \frac{1}{TO + \frac{1}{N\lambda}} \tag{11.13}$$

Se observa que esta tasa tenderá a ser el inverso de TO si el número de jugadores o la tasa de generación de cada jugador crecen.

Seguidamente, calcularemos los valores de Pr (k=0), Pr (k=1), Pr (k>1), $E[k]$ para esta política. El número de paquetes multiplexados nunca puede ser nulo, puesto que al menos se envía el paquete que llega una vez pasado el *timeout*. Por tanto, tenemos:

$$\Pr(k = 0) = 0$$

$$\Pr(k = 1) = 1 - \Pr(k > 1) \tag{11.14}$$

En el caso de tener una distribución exponencial, podemos calcular las probabilidades:

$$\Pr(k = 1) = e^{-N\lambda TO}$$

$$\Pr(k > 1) = 1 - e^{-N\lambda TO} \tag{11.15}$$

Algunos juegos, como por ejemplo *Half Life Counter Strike 1* en modo *OpenGL* [RHS10], [LABC03] presentan un tiempo entre paquetes determinista, pero con dos valores posibles para ese tiempo. En este caso consideraremos t_1 como el tiempo menor y t_2 como el tiempo mayor. Las probabilidades de cada tiempo serán p_1 y p_2, cumpliendo $p_1+p_2=1$. Se presentan ahora los cálculos correspondientes a esta distribución. El caso de una distribución con un solo valor del tiempo entre paquetes será un caso particular en el que $p_1=1$ y $p_2=0$.

Por tanto, podemos calcular la tasa de llegada de paquetes de un usuario como:

$$\lambda = \frac{1}{p_1 t_1 + p_2 t_2} \tag{11.16}$$

Ahora definiremos l como el número de paquetes de un solo jugador que llegan durante *TO*. Necesitamos calcular el valor de $\Pr(k=1)$, es decir, la probabilidad de multiplexar un solo paquete, que en este caso será el que llega después de *TO*. Para que esto ocurra, el número de paquetes que llegan de cada usuario debe ser nulo, es decir:

$$\Pr(k = 1) = [\Pr(l = 0)]^N \tag{11.17}$$

Podemos distinguir dos casos: si $TO < t_1$ entonces no pueden haber llegado dos paquetes del mismo usuario, es decir $\Pr(l=2)=0$. Por eso, la probabilidad de que llegue un paquete será igual al valor esperado de l, y así podemos obtener $\Pr(l=1)$ y $\Pr(l=0)$:

$$\Pr(l = 1) = E[l] = \lambda TO$$

$$\Pr(l = 0) = 1 - \Pr(l = 1) = 1 - \lambda TO \tag{11.18}$$

En caso de tener $t_1 \leq TO < t_2$ entonces podemos obtener $\Pr(l=0)$ como la probabilidad de que el intervalo de duración *TO* comience durante los primeros t_2-*TO* segundos de un tiempo entre paquetes de duración t_2, como se ve en la Fig. 11.11. Por tanto:

$$\Pr(l = 0) = p_2 \lambda (t_2 - TO) \tag{11.19}$$

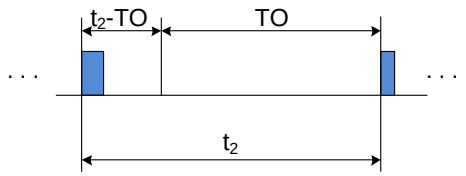


Fig. 11.11. Probabilidad de que no llegue ningún paquete durante *TO*

Política period

En este caso, el valor esperado del número de paquetes generados será $E[k]=N\lambda PE$. La tasa de paquetes multiplexados pps_{PE} será:

$$pps_{PE} = \frac{\Pr(k > 0)}{PE} \tag{11.20}$$

Esta cantidad tiende a ser el inverso del periodo según crece el número de usuarios y la tasa de generación de paquetes, puesto que en ese caso la probabilidad de recibir algún paquete tenderá a la unidad.

Consideraremos, como efectivamente ocurre en el juego considerado en esta sección, que $T < 2t_1$, y $t_2 < 2t_1$.

Tenemos que hallar Pr (k=0), que se corresponderá con la probabilidad de no haber recibido ningún paquete de ningún usuario:

$$\Pr (k = 0) = [\Pr (l = 0)]^{N} \tag{11.21}$$

La probabilidad de haber recibido un solo paquete será la correspondiente a que un jugador haya enviado uno, y el resto ninguno:

$$\Pr (k = 1) = \binom{N}{1} \Pr (l = 1)[\Pr (l = 0)]^{N-1} \tag{11.22}$$

Ahora distinguiremos dos casos diferentes. En primer lugar, si tenemos $PE < t_1$, tenemos $\Pr (l = 2) = 0$ y por tanto:

$$\Pr (l = 1) = E[l] = \lambda PE$$

$$\Pr (l = 0) = 1 - \Pr (l = 1) = 1 - \lambda PE \tag{11.23}$$

Y en el caso de $t_1 \leq PE < t_2$, deberemos encontrar $\Pr (l = 0)$, es decir, la probabilidad de que no hayan llegado paquetes en el periodo *PE,* que se

corresponde con la probabilidad de que el periodo comience en los primeros t_2-PE segundos de un tiempo entre paquetes de duración t_2. Por tanto, considerando que los tiempos entre paquetes consecutivos son independientes:

$$\Pr(l = 0) = p_2 \lambda (t_2 - PE) \tag{11.24}$$

Y ahora, sabiendo que la suma de las probabilidades debe ser la unidad, y que el valor esperado de l es:

$$E[l] = \Pr(l = 1) + 2 \Pr(l = 2) = \lambda PE \tag{11.25}$$

Podemos obtener:

$$\Pr(l = 1) = \lambda [PE - 2p_1 (PE - t_1)]$$

$$\Pr(l = 2) = p_1 \lambda (PE - t_1) \tag{11.26}$$

Comparativa entre las políticas

Seguidamente nos ocuparemos de seleccionar una de las políticas de multiplexión, para utilizarla en el estudio que realizaremos sobre el ahorro que se puede conseguir con diferentes juegos FPS. Estudiaremos el ahorro en ancho de banda y en paquetes por segundo. Para ello se utilizarán representaciones gráficas, y también se usará el tráfico de un juego representativo: *Half Life Counter Strike 1*. Se presentarán primero resultados analíticos, y posteriormente usaremos simulación.

En primer lugar se ha tomado la decisión de dejar de lado las políticas *Fixed Number* y *Size Limit*. Aunque serían sencillas de implementar, no consideran el retardo como el parámetro principal, que es lo que hacen las políticas *timeout* y *period*. Por tanto, dada la sensibilidad de los jugadores al retardo, nos centraremos en el estudio de estas dos últimas.

Utilizaremos para esta comparativa el tráfico cliente-a-servidor, por dos razones: en primer lugar, porque es el que produce un mayor número de paquetes, y de menor tamaño, así que será el que obtenga mayores ahorros de ancho de banda. En segundo lugar, dado que muchos de los accesos a Internet en la actualidad son asimétricos (como el ADSL), el principal cuello de botella se da en el enlace de subida, y no en el de bajada. Así que nos interesa más el ahorro en la dirección cliente a servidor.

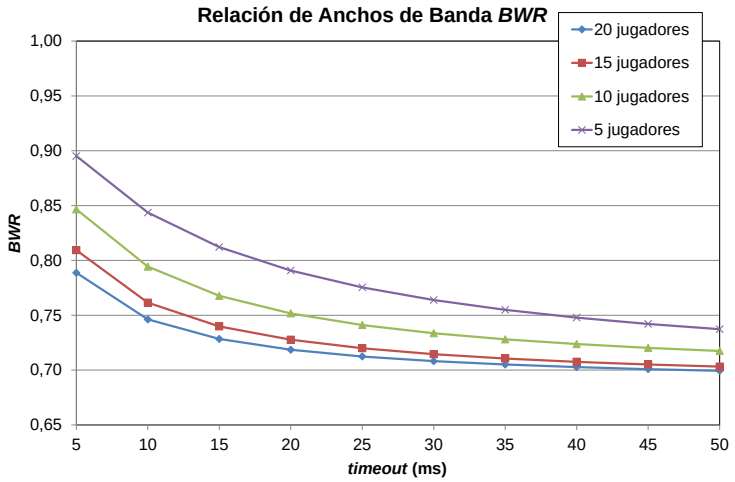
Resultados analíticos

Presentaremos en este subapartado las gráficas que corresponden a los resultados analíticos de la multiplexión del tráfico cliente-a-servidor del juego estudiado. Éste presenta características diferentes según el modo gráfico utilizado: en modo *OpenGL*, que es el que estudiaremos, genera paquetes separados por 33 o 50 ms, con una probabilidad del 50 % para cada valor. Por tanto, tendremos t_1=33 ms, t_2=50 ms, p_1=p_2=0,5. El valor de λ será en este caso 24 paquetes por segundo. Si se utilizase el modo *Direct3D,* el juego generaría paquetes con un intervalo fijo de 41 ms. Por último, en modo *software,* los paquetes se generan con intervalos variables [LABC03].

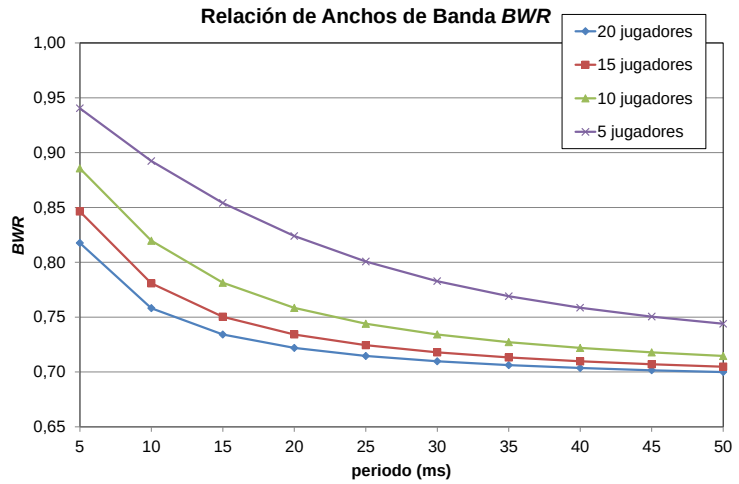
Las Fig. 11.12 y 11.13 representan los valores de *BWR* para ambas políticas, en función del número de jugadores y del valor del intervalo usado para multiplexar. El comportamiento asintótico se puede observar para ambos parámetros, y por eso consideramos que la zona más interesante se da cuando *BWR* está entre 0,70 y 0,75. Por ejemplo, si observamos la gráfica de 20 jugadores de la Fig. 11.12.b), una vez que se alcanza el valor de 0,75, el incremento del retardo para mejorar el ahorro de ancho de banda supondrá un beneficio muy pequeño.

También el número de jugadores influye. Lógicamente, si hay más jugadores, se podrá conseguir el mismo valor de $E[k]$ para valores más pequeños de *TO* o *PE*. Por tanto, confirmamos que el aumento del número de jugadores es siempre beneficioso. De hecho, en el caso de tener solamente 5 jugadores, quizá lo mejor sea mantener el valor de *BWR* en torno a 0,80. Lógicamente, el valor del retardo de red tendrá una cierta influencia en el valor elegido para el intervalo de multiplexión. Si la red es rápida, podremos permitirnos añadir un retardo mayor para así mejorar el ahorro de ancho de banda.

Hasta ahora hemos mostrado los resultados de *BWR*. Se podría haber mostrado de la misma manera el ahorro del ancho de banda, puesto que ambos parámetros están relacionados directamente, pues su suma es la unidad (ecuación 11.2). Como resumen, la Fig. 11.14 presenta el ahorro de ancho de banda en función de ambos parámetros: el número de jugadores y el *timeout* o *period*. Se observa que la política *timeout* obtiene mejores valores para intervalos y números de jugadores pequeños. Esta ventaja de *timeout* se debe a que con esta política se inserta un paquete más: el que desencadena el envío del paquete multiplexado. De todas maneras, se observa que según crece el número de jugadores o el periodo, las políticas se igualan puesto que el tiempo entre el final de *TO* y la llegada de un paquete se va reduciendo.

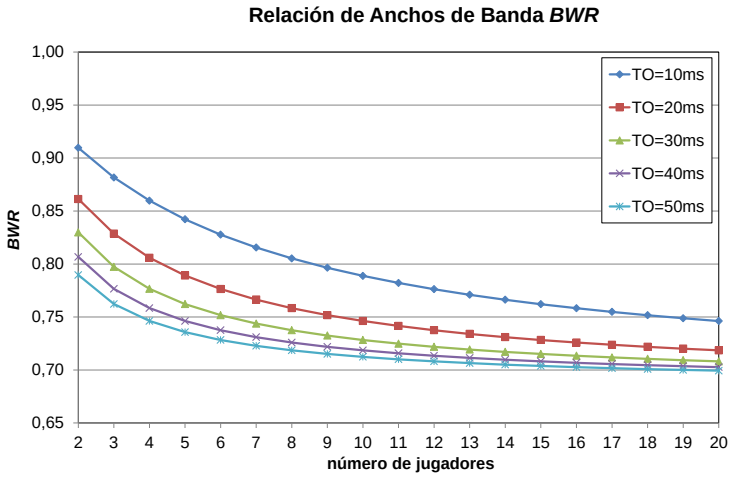


(a)

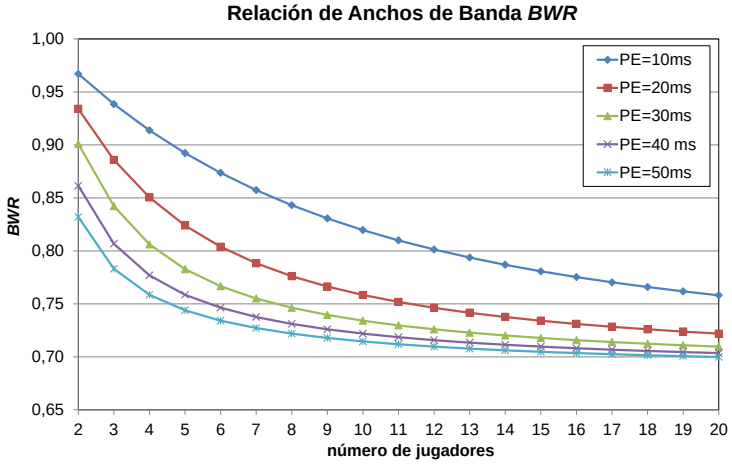


(b)

Fig. 11.12. Relación teórica de anchos de banda para las políticas a) *timeout* y b) *period,* según el valor de *TO* y *PE*

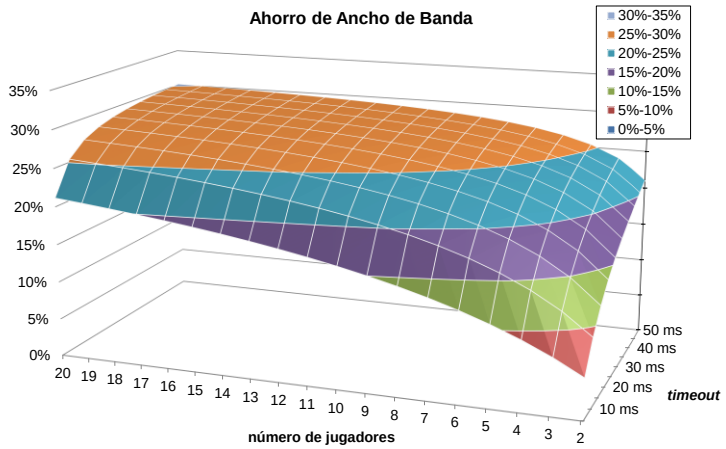


(a)

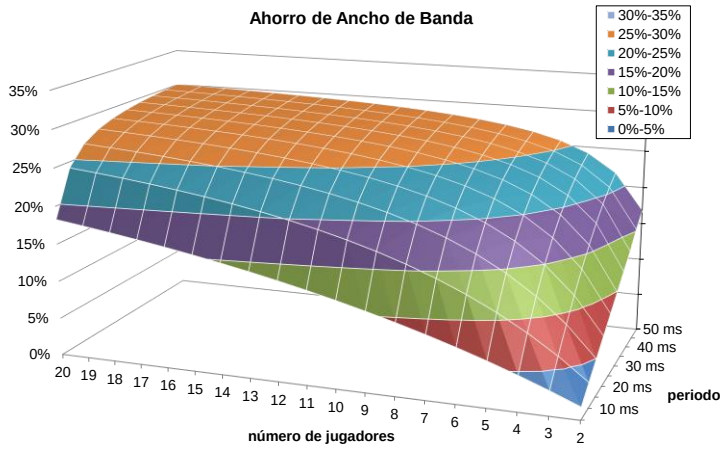


(b)

Fig. 11.13. Relación teórica de anchos de banda para las políticas a) *timeout* y b) *period,* según el número de jugadores



(a)



(b)

Fig. 11.14. Ahorro de ancho de banda teórico para las políticas a) *timeout* y b) *period* en función del intervalo y del número de jugadores

Seguidamente nos ocuparemos de estudiar la diferencia en términos de paquetes por segundo. La Fig. 11.15 representa los valores teóricos para ambas políticas. Se puede observar que con *timeout* se envían menos paquetes, y que las diferencias se reducen según aumentan el número de usuarios y el intervalo. Podemos hacer una observación: como se ha dicho en las secciones previas, una limitación de los *router* comerciales es el número de paquetes por segundo que pueden gestionar. El

método presentado también reduce notablemente esta cantidad. En la figura podemos ver cómo se pasa de 493 paquetes por segundo con 20 jugadores y tráfico nativo, a tan sólo 20 paquetes por segundo si se usa un intervalo de multiplexión de 50 ms.

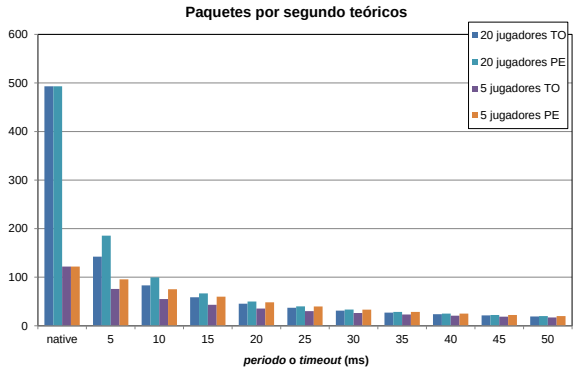


Fig. 11.15. Paquetes por segundo teóricos para las políticas *timeout* y *period*

Resultados obtenidos con simulación

Para comprobar que los resultados obtenidos analíticamente son válidos, y dado que hemos hecho algunas suposiciones sobre el tráfico, presentamos ahora resultados obtenidos mediante simulación. La simulación también nos servirá para poder hallar resultados con diferentes juegos, sea cual sea su distribución de tráfico. Para ello necesitaremos trazas de tráfico originales del juego o bien un modelo estadístico de los existentes en la literatura [RHS10].

Explicaremos seguidamente cómo se ha trabajado con las trazas del juego. También usaremos esas trazas para comprobar que se corresponden con los cálculos teóricos. Para poder comparar los resultados de simulación con los teóricos, se ha usado el mismo juego: *Half Life Counter Strike 1*. Las trazas de tráfico se han obtenido del proyecto CAIA (por ejemplo, la traza para 5 jugadores se puede encontrar en [SB06]). Hay disponibles trazas desde 2 hasta 9 jugadores. Los primeros 10.000 paquetes no se consideran, y sólo se incluyen los siguientes 5.000 * (número de jugadores). Así se asegura que el tráfico capturado sólo se corresponde con tráfico de juego activo, que es el que queremos estudiar.

Para obtener trazas correspondientes a un mayor número de jugadores, se han sumado las que hay disponibles. Por ejemplo, la traza de 20 jugadores se ha obtenido sumando las de 9, 6 y 5 jugadores. Esto es posible gracias a la naturaleza del tráfico del cliente al servidor, cuya distribución es independiente del número

de jugadores [BA06], [LABC03]: el tráfico cliente a servidor de una partida de 20 jugadores es similar al que generarían una partida de 9, otra de 6 y otra de 5 jugadores. Lógicamente, se ha cortado la duración de las trazas a la más corta, obteniendo 110 segundos de tráfico activo.

Se han realizado simulaciones con Matlab para obtener trazas comprimidas y multiplexadas, como se muestra en la Fig. 11.16. En primer lugar, se separan las trazas de los distintos jugadores, y se elimina el tráfico del servidor a los clientes. Para generar el tráfico en las pruebas, solamente necesitamos la información del instante de generación, el usuario y el tamaño del paquete. Después, se aplica la compresión a las cabeceras IP/UDP de cada flujo. Finalmente, usando el intervalo *TO* o *PE* se obtienen los tamaños e instantes de los paquetes multiplexados. El histograma de la traza obtenida se corresponde con lo esperado: dos posibles valores del tiempo entre paquetes, en torno a 33 y 50 ms, con igual probabilidad (Fig. 11.17).

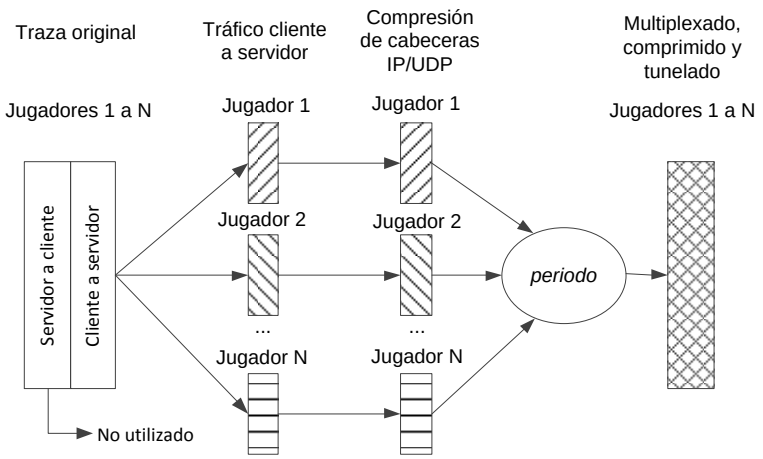


Fig. 11.16. Método utilizado para construir las trazas

Veamos ahora las gráficas obtenidas con simulación para el *BWR*. Las mostraremos comparándolas con las obtenidas analíticamente (Fig. 11.12 y 11.13). Las curvas de la simulación van en línea continua, mientras que las de los resultados teóricos tienen trazos discontinuos. La Fig. 11.18 muestra el *BWR* según el intervalo de multiplexión. Se observa que los resultados son prácticamente iguales, salvo pequeñas diferencias (en torno al 1%) para intervalos y números de jugadores pequeños, que se deben a que el tiempo entre paquetes (como se ve en el histograma de la Fig. 11.17) tiene pequeñas variaciones en torno a los valores teóricos.

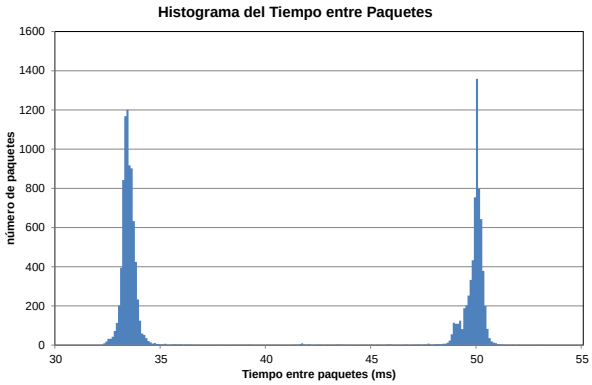
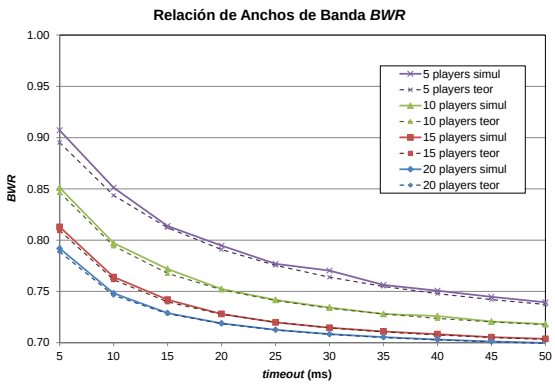
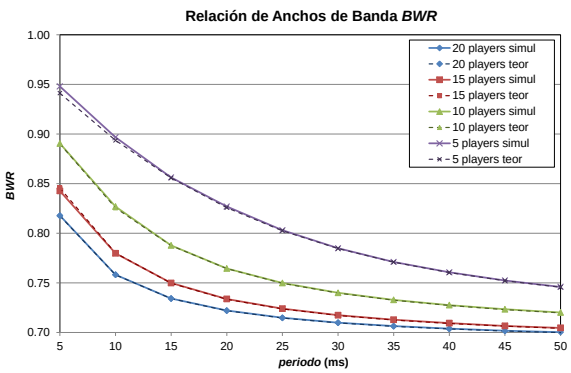


Fig. 11.17. Histograma del tiempo entre paquetes en ms



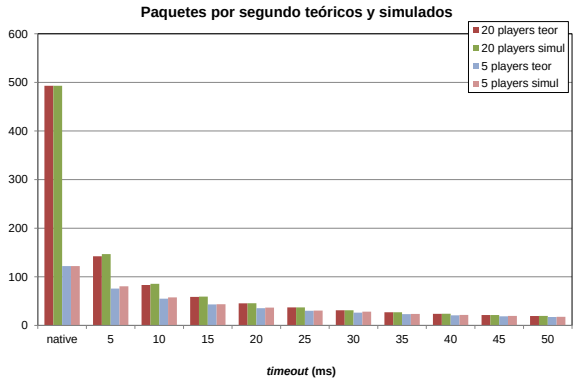
(a)



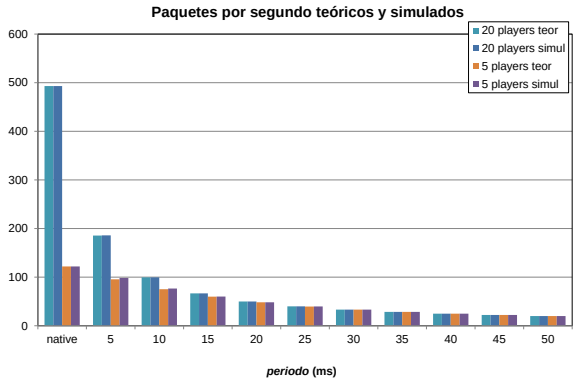
(b)

Fig. 11.18. *BWR* teórico vs simulado para las políticas a) *timeout* y b) *period*

También podemos comparar los resultados teóricos con los de las simulaciones en el caso de las cantidades de paquetes por segundo. En la Fig. 11.19 se puede ver que las gráficas teóricas y simuladas son casi idénticas.



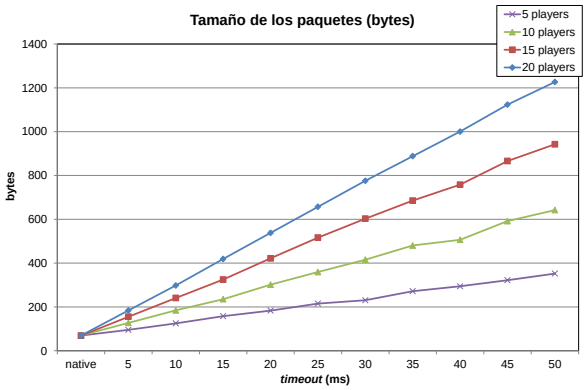
(a)



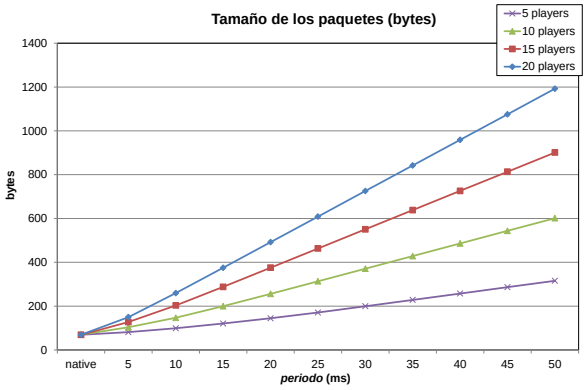
(b)

Fig. 11.19. Cantidad de paquetes por segundo teórica vs simulada para las políticas a) *timeout* y b) *period*

Otro parámetro interesante a considerar es el tamaño de los paquetes generados al multiplexar. El tamaño de los paquetes puede producir distintos comportamientos en el *router*, especialmente en lo que se refiere a pérdidas. La Fig. 11.20 muestra el tamaño de los paquetes. Se puede observar que para este juego nunca se alcanzan los 1.500 bytes, que es el tamaño máximo en muchas redes de datos. Más adelante veremos lo que ocurre con otros juegos en los que sí se llega a ese límite.



(a)



(b)

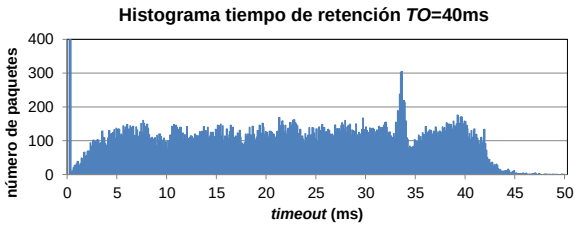
Fig. 11.20. Tamaño de los paquetes simulados para las políticas a) *timeout* y b) *period*

Además del retardo y las pérdidas de paquetes, otro parámetro importante que influye en la calidad experimentada por el usuario de un juego es el *jitter*, o variación del retardo. Por eso consideraremos cómo influyen las dos políticas en este parámetro. Como se ha visto anteriormente, este parámetro se puede medir de varias maneras, una de las cuales es considerarlo como la desviación estándar del retardo [WKVA06].

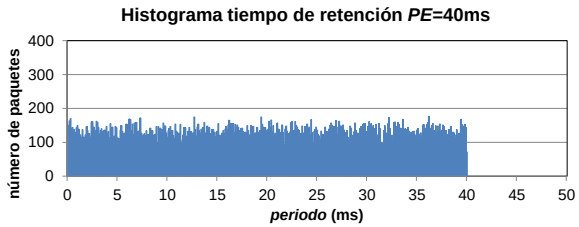
El retardo añadido al multiplexar variará para cada paquete, pues éste puede llegar en distintos momentos del intervalo (*PE* o *TO*). Por tanto, el retardo añadido variará entre 0 y *PE* para la política *period*, y podrá ser algo mayor que *TO* para la

política *timeout*. Esto añadirá un *jitter*, es decir, la desviación estándar del retardo será mayor a la salida del multiplexor.

Veamos ahora los histogramas del tiempo de retardo en el multiplexor correspondiente a cada paquete (Fig. 11.21), para las dos políticas. Se han obtenido utilizando un intervalo de multiplexión de 40 ms. En el de la política *timeout* se observa, por un lado, un pico correspondiente al retardo nulo, que se debe a los paquetes que desencadenan el envío de cada paquete multiplexado. En la gráfica se ha recortado ese pico, que tiene un valor de 2.641 paquetes, para evitar que el resto de valores aparecieran demasiado pequeños. También se observa otro pico en torno a 33 ms, que se corresponde con uno de los posibles valores del tiempo entre paquetes. Finalmente, se puede observar que aparece una cola correspondiente a los retardos superiores a 40 ms. En la Fig. 11.21 b) se observa que la política *period* presenta una distribución más uniforme, y ya no aparecen los picos ni la cola por encima de 40 ms.



(a)



(b)

Fig. 11.21. Histograma del tiempo de retención en el multiplexor, obtenido en las simulaciones, para las políticas a) *timeout* y b) *period*

La Fig. 11.22 muestra la desviación estándar del retardo para las dos políticas. En primer lugar, se observa que para la política *period* el valor de la desviación es independiente del número de jugadores, por lo que las gráficas aparecen superpuestas. Podríamos haber esperado este resultado, pues al tener una

distribución uniforme, la varianza sólo depende del valor de *PE*. La fórmula de la varianza para una distribución uniforme en el intervalo (*a,b*) es:

$$\text{var} = \frac{(b-a)^2}{12} \tag{11.27}$$

Por tanto, la desviación estándar en el caso de la política *period* será:

$$stdev = \frac{PE}{\sqrt{12}} = 0{,}286PE \tag{11.28}$$

Observamos en la Fig. 11.22 que se cumple esta igualdad.

En cambio, para la política *timeout* aparece una mayor variación, que es debida a lo ya visto en la Fig. 11.21, en la que el histograma para *timeout* tiene un mayor rango de valores, mientras que el de *period* es más uniforme y con un valor acotado. La pequeña disminución que se observa en la gráfica *5 players* en torno a 30 ms, se debe a la presencia del pico de 33 ms en el tiempo de retención.

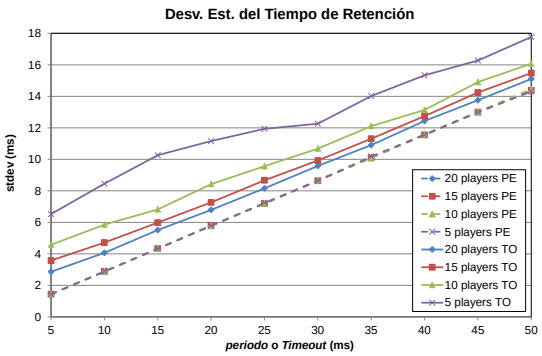


Fig. 11.22. Desviación estándar del tiempo de retención en el multiplexor, obtenido en las simulaciones, para ambas políticas

Podemos extraer algunas consecuencias a la vista de estas últimas gráficas: la ventaja de la política *timeout* es que nos proporciona un mayor ahorro en ancho de banda y paquetes por segundo, cuando el número de jugadores o el intervalo son pequeños. Pero tiene el inconveniente, respecto a la política *period* de añadir un *jitter* mayor al tráfico de los juegos. La diferencia en cuanto a ahorro de ancho de banda es pequeña, y además se da cuando hay pocos jugadores, que es precisamente la circunstancia en que menos necesario es ese ahorro. Por tanto,

teniendo en cuenta que hay juegos en los que la percepción del usuario es muy sensible al *jitter*, se ha tomado la opción de usar a partir de ahora la política *period*.

Ampliación del estudio a más juegos

Una vez estudiado el método de multiplexión para un juego, realizaremos en este apartado un estudio de otros títulos disponibles en el mercado. Se han seleccionado de forma que se incluya una gran variedad de comportamientos en cuanto a tamaño de paquete y tiempo entre paquetes.

Este estudio se realizará mediante simulación. Para ello, en los casos en que sea posible, se utilizarán trazas reales del tráfico del juego correspondiente *(Counter Strike 1, Counter Strike 2, Quake III, Quake IV* y *Wolfenstein: Enemy Territory)*, que están disponibles en la web del proyecto CAIA [SB06], al igual que las usadas en los apartados anteriores. Estas trazas disponen de abundante documentación que explica las condiciones en que se han obtenido: características de las máquinas y de la red, etc. En estos cinco casos, se ha trabajado con las trazas, para separar el tráfico cliente-a-servidor y combinarlo para obtener trazas de 5, 10, 15 y 20 jugadores. Para otros tres juegos *(Halo 2, Quake II* y *Unreal Tournament 1.0)* las trazas se han generado a partir de modelos obtenidos de la literatura, que están referenciados en [RHS10]. En el caso de *Halo 2* se ha generado el tráfico de un solo jugador en una consola, aunque este juego permite hasta cuatro jugadores en la misma máquina.

Posteriormente se separa el tráfico de cada jugador, y se le aplica el algoritmo de compresión. Finalmente, todos los flujos se multiplexan utilizando diferentes valores de *PE*. Recordemos que en este apartado sólo se utilizará la política *period*. También se presentarán resultados de IPv6, para los cuales los tamaños de las cabeceras se han modificado en la manera que les corresponde.

Las Fig. 11.23 a 11.30 presentan cuatro gráficas para cada juego: en primer lugar, se incluyen los histogramas del tamaño del paquete para IPv4 (incluyendo las cabeceras IP y UDP) y del tiempo entre paquetes. Posteriormente, se representan los ahorros de ancho de banda que se pueden obtener para IPv4 e IPv6, en función de *PE*. Para cada juego, también se muestra su año de presentación, y el motor gráfico que usa, así como el tamaño medio de paquete para IPv4, y el número de paquetes por segundo que genera. Se incluirá también el juego *Counter Strike 1*, que ha sido analizado anteriormente.

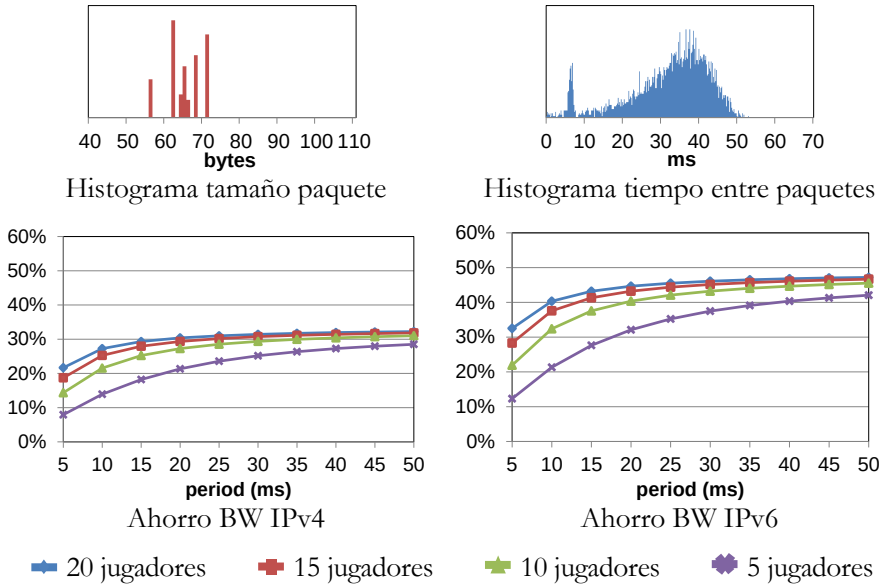


Fig. 11.23. Comportamiento de *Quake II* (1997, idTech2)

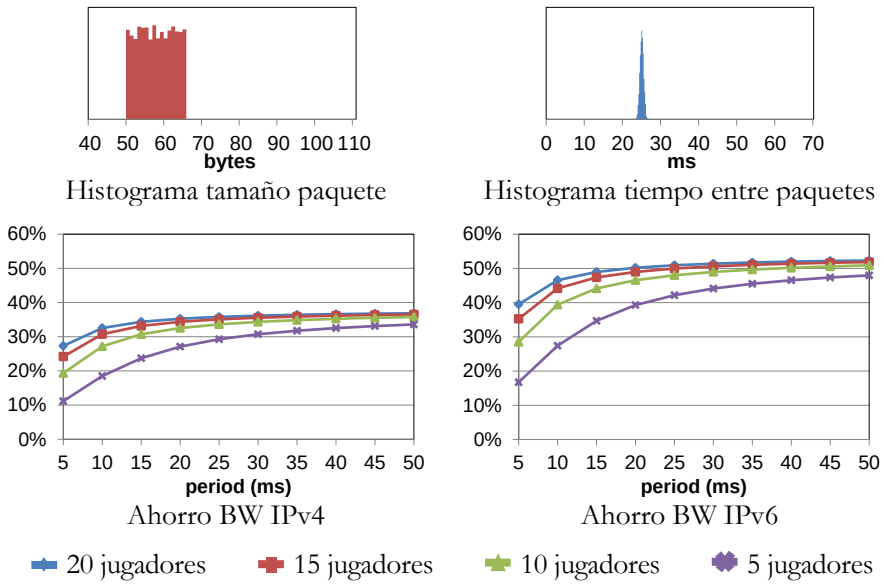


Fig. 11.24. Comportamiento de *Unreal Tournament* (1999, unreal 1.0)

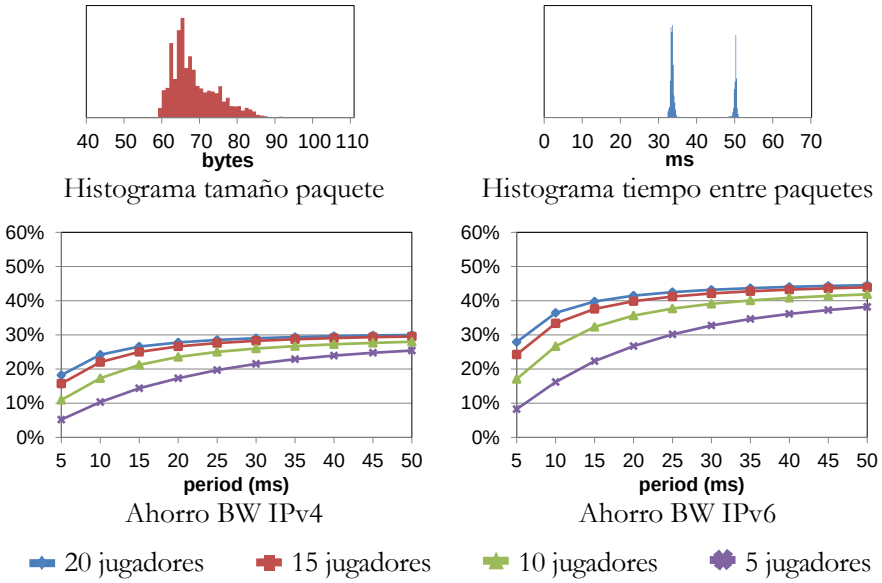


Fig. 11.25. Comportamiento de *Half Life Counter Strike 1* (1999, GoldSrc)

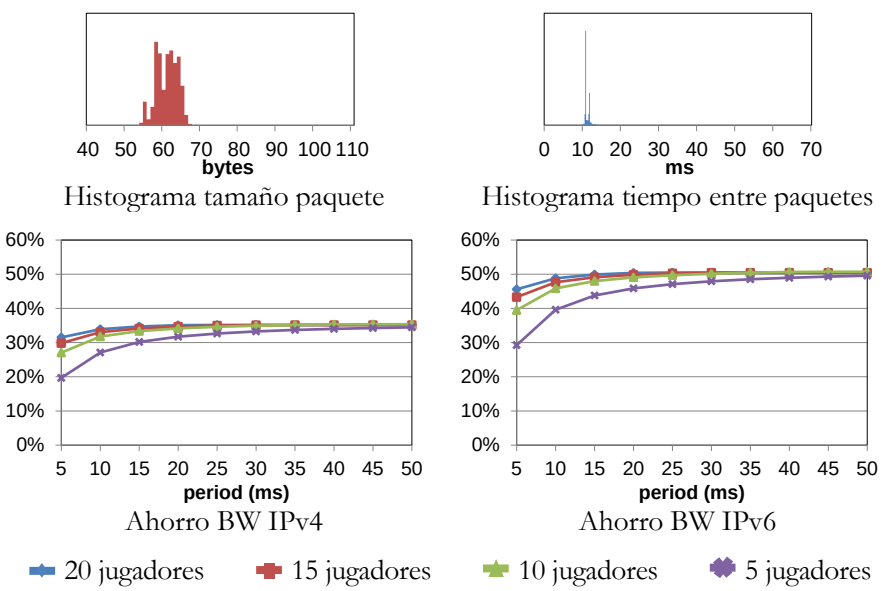


Fig. 11.26. Comportamiento de *Quake III* (1999, idTech3)

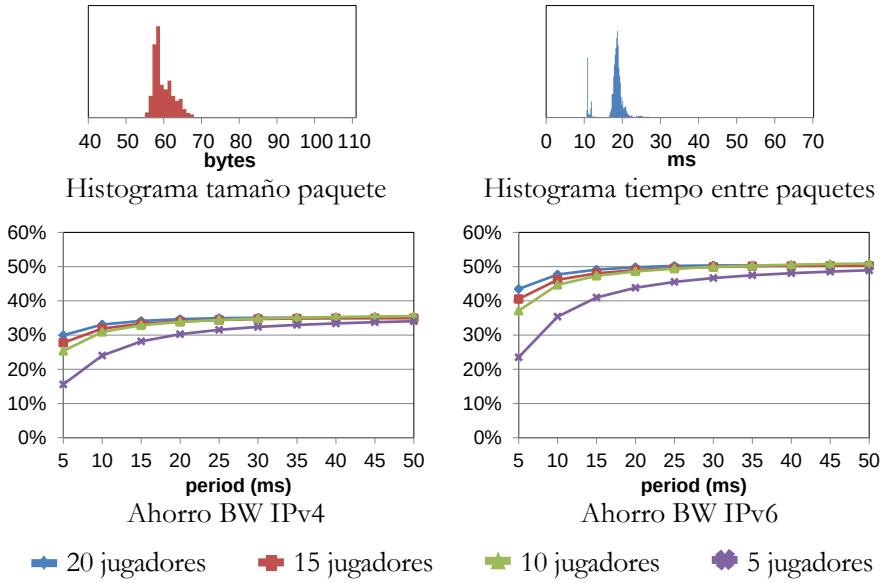


Fig. 11.27. Comportamiento de *Wolfenstein: Enemy Territory* (2003, idtech3 mod)

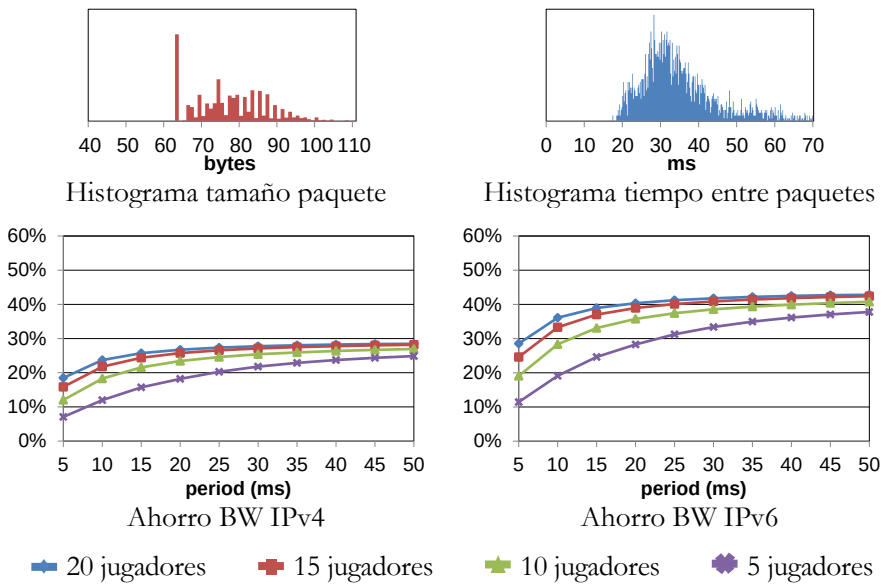


Fig. 11.28. Comportamiento de *Half Life Counter Strike 2* (2004, Source)

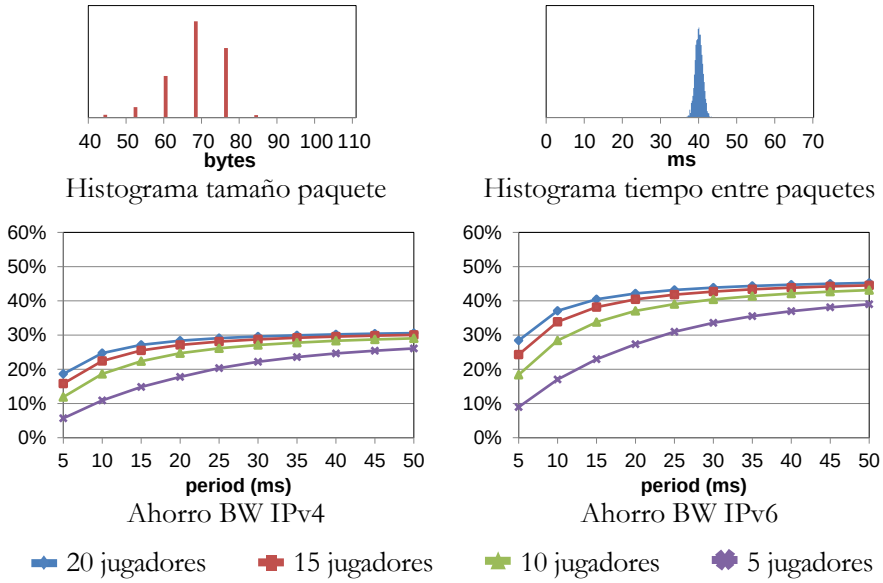


Fig. 11.29. Comportamiento de *Halo 2* (2004, Halo2)

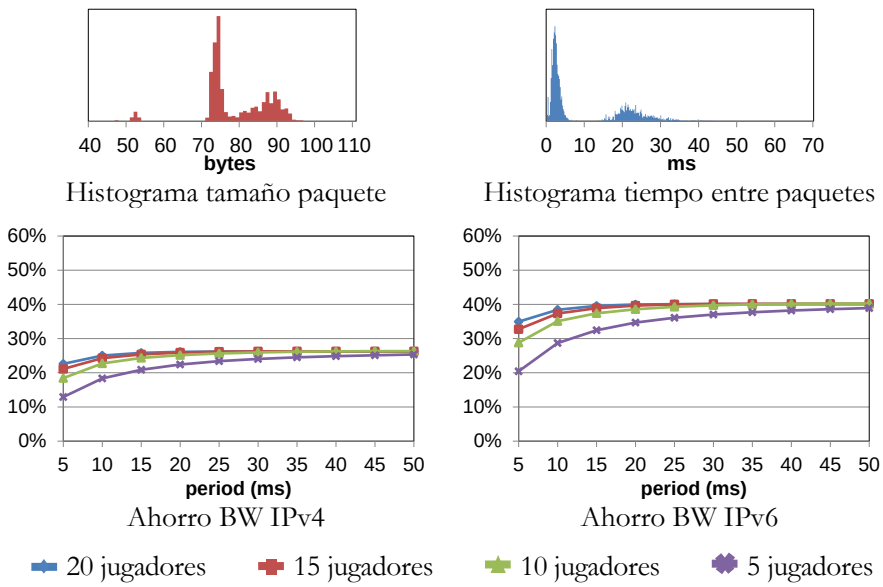


Fig. 11.30. Comportamiento de *Quake IV* (2005, idTech4)

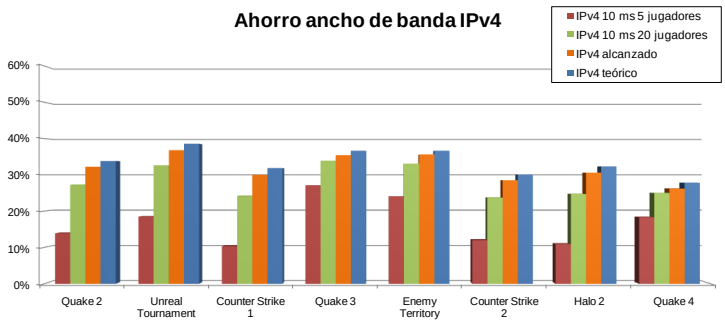
Podemos observar una gran variedad en los histogramas de tamaño de paquete: algunos, como el de *Halo 2*, solamente generan paquetes con unos tamaños determinados; otros, como *Unreal Tournament* o *Quake III*, tienen unos márgenes de tamaños muy pequeños. Pero todos coinciden en el hecho de generar paquetes muy pequeños, con la media variando entre 57 y 79 bytes.

Si miramos los histogramas de tiempo entre paquetes, observaremos que algunos juegos tienen un comportamiento muy regular, como por ejemplo *Unreal Tournament*, que genera un paquete cada 25 ms, o *Counter Strike 1* que, en modo gráfico *OpenGL*, tiene dos posibles valores para el tiempo entre paquetes: 33 o 50 ms. Otros títulos, como *Counter Strike 2* o *Quake IV* tienen un mayor rango de variación.

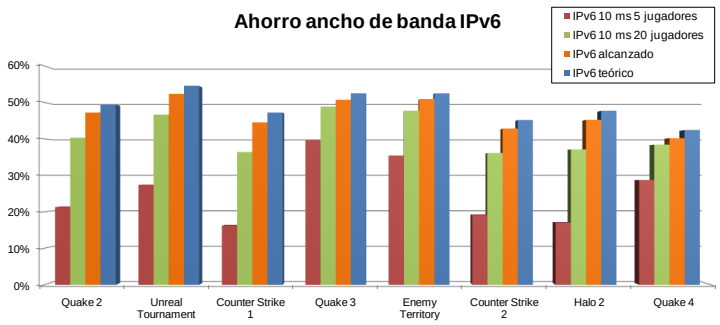
Con respecto a las gráficas de ahorro de ancho de banda, haremos dos observaciones: en primer lugar, los títulos que consiguen mayores ahorros son los que generan paquetes más pequeños, como *Unreal Tournament*, dado el gran *overhead* que tienen. En segundo lugar, podemos ver que los juegos con mayor cantidad de paquetes por segundo son los que consiguen más rápidamente buenos resultados, es decir, para pequeños valores del periodo. Por ejemplo, si miramos a la gráfica de IPv6 para *Quake III*, podemos observar que para 5 jugadores y *PE*=5 ms, el ahorro de ancho de banda está ya en torno al 30%.

Para resumir los resultados, se han incluido en la Fig. 11.31 cuatro valores diferentes de ahorro para cada título: en primer lugar, el ahorro obtenido con un periodo de 10 ms para 5 y 20 jugadores. Después, el ahorro máximo obtenido en las simulaciones y, finalmente, el ahorro que se conseguiría en la asíntota. Comentemos ahora esos resultados.

Una primera cuestión que se debe aclarar es la diferencia entre la tercera y cuarta columnas. Estas diferencias son pequeñas (alrededor del 2%), y están principalmente causadas porque nunca se alcanza el valor asintótico. Pero en algunos casos, especialmente en los juegos que generan un gran número de paquetes por segundo, existe otra limitación: si al multiplexar se alcanza el valor máximo de un paquete (1.500 bytes), se envía inmediatamente un paquete y se comienza un nuevo periodo. Este fenómeno sólo ocurre con *Wolfenstein: Enemy Territory, Quake III* y *Quake IV*. La Fig. 11.32 muestra el tamaño de paquete para *Quake IV*. Puede verse cómo, para números de jugadores grandes, se alcanza una saturación en el tamaño de forma que, aunque se aumente el periodo, éste nunca llega a alcanzarse. Por tanto, para esos juegos no tendría sentido aumentar el periodo indefinidamente, según el número de jugadores.



(a)



(b)

Fig. 11.31. Resumen del ahorro que se puede alcanzar en cada juego a) IPv4; b) IPv6

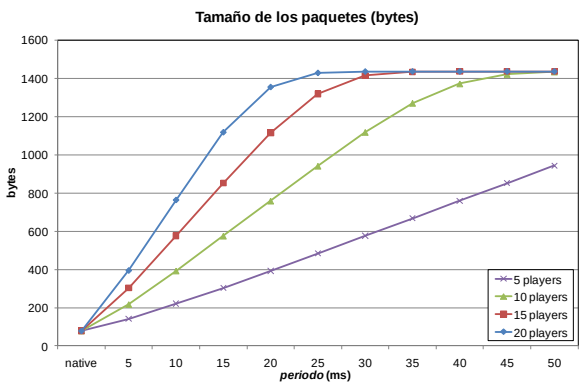


Fig. 11.32. Tamaño de paquete para *Quake IV*

Con respecto a la segunda columna de la Fig. 11.31, que se corresponde con un valor pequeño del periodo, puede verse que todos los juegos obtienen buenos resultados para 20 jugadores. Esto significa que si hay un número suficiente, el comportamiento del sistema será bueno. Pero si miramos a la primera columna, que es el mismo valor para 5 jugadores, observamos importantes diferencias entre los títulos. Una vez más, los juegos con mayores cantidades de paquetes por segundo, como *Quake III*, consiguen buenos resultados incluso para un número de jugadores pequeño.

Para IPv4 los ahorros máximos varían entre el 26% y el 36%, mientras que para IPv6 lo hacen desde el 40% hasta el 52%. La diferencia de ahorro entre IPv4 e IPv6 está en torno al 15%. Lógicamente, el *overhead* extra introducido por IPv6 hace que el método obtenga mejores resultados.

Evaluaciones subjetivas

Hemos mostrado cómo el método TCM puede lograr ahorros de ancho de banda pero, obviamente, hay una contrapartida, que en nuestro caso es el incremento en el retardo y el *jitter*. El retado se produce principalmente en el multiplexor y, como hemos visto, su valor medio será de la mitad del periodo utilizado. También se añadirá un pequeño tiempo de procesado. Pero por otra parte es de esperar que el retardo se reduzca a causa del ahorro de ancho de banda, por ejemplo en el tiempo de encolado en el *buffer*. El *jitter* se deberá a que unos paquetes se retardan más que otros al esperar en el multiplexor.

Otros parámetros, como el porcentaje de paquetes perdidos, no se ven directamente afectados por TCM, aunque las pérdidas se pueden modificar debido al incremento en el tamaño medio de paquete. Habrá otro efecto: el ahorro de ancho de banda reducirá la saturación del *router*, por lo que puede ocurrir que se reduzca la probabilidad de pérdidas. Pero estos dos efectos no son directos, por lo que no se estudiarán aquí.

Como hemos visto en los capítulos iniciales, existen estudios realizados con evaluaciones subjetivas de los jugadores, que recomiendan que el retardo no sea superior a 200 o 225 ms [ZA04]. Otros estudios han desarrollado modelos de la calidad percibida similares a los que existen desde hace años para VoIP. Usaremos uno de ellos, como un ejemplo para comprobar si la calidad subjetiva se vería comprometida al usar TCM: el G-Model para *Quake IV* [WKVA06]. En este estudio se obtuvo una fórmula para el MOS a partir de la realización de pruebas de calidad subjetiva con usuarios reales. Incluye un factor que expresa los

inconvenientes de la red, y después usa una función polinómica para obtener el MOS. Se basa en el retardo y el *jitter*, ya que consideran que el juego estudiado tiene un sistema para que las pérdidas de paquetes no afecten hasta que lleguen al 35% [ZA04].

Hemos realizado unos sencillos cálculos usando el modelo para *Quake IV*, y se han obtenido resultados para el MOS por encima de 3,5 usando un periodo de 20 ms para 20 jugadores, si el retardo de la red es también de 20 ms. Pero si el retardo de la red es de 40 ms, el periodo se debe reducir a 15 ms. El MOS puede mantenerse por encima de 3 con 40 ms de retardo de red, incluso con un periodo de 45 ms.

Conclusiones

En este capítulo se ha estudiado el tráfico de los juegos *online* FPS, y se ha visto que tiene características en parte similares al de VoIP: paquetes pequeños con una frecuencia elevada, y requerimientos temporales muy estrictos.

Se ha visto que existen escenarios donde los flujos de un número de jugadores comparten el mismo camino, por lo que nos hemos planteado la adaptación de las técnicas de multiplexión, ya usadas con tráfico de voz, para su uso con esta otra aplicación de tiempo real. Nos hemos centrado en el tráfico cliente-a-servidor, por ser susceptible de mayor ahorro que el de servidor-a-cliente.

Hemos adaptado el método TCRTP, para poder usarlo cuando el tráfico nativo no es RTP, sino simplemente IP/UDP. Para ello, ha sido necesario sustituir la compresión de cabeceras ECRTP por IPHC o ROHCv2.

Se han comparado diferentes políticas para decidir cuál es el método óptimo para decidir qué paquetes se multiplexan, optando por un método que define un periodo fijo, al final del cual se envían todos los paquetes recibidos.

El análisis teórico del método ha mostrado que el ahorro de ancho de banda tiene un comportamiento asintótico, es decir, que no se consigue aumentar indefinidamente el ahorro, aunque aumenten el número de jugadores o el periodo. El ahorro de ancho de banda puede llegar a un 30% en el caso de IPv4, y a un 50% si se usa IPv6. También se consiguen ahorros significativos en paquetes por segundo generados.

Se han realizado simulaciones, utilizando trazas de tráfico extraídas de partidas reales de diferentes juegos, y se ha comprobado que el comportamiento de TCM

se asemeja al obtenido analíticamente. Se ha observado que el método introduce retardos, y también incrementa el *jitter*, debido a que unos paquetes son retardados más que otros. Por eso, en el siguiente capítulo estudiaremos la modificación de los parámetros de calidad, teniendo también en cuenta el retardo introducido por el *buffer* del *router*, y por la red.

ESTUDIO DE LA INFLUENCIA DEL ESQUEMA DE OPTIMIZACIÓN EN LOS PARÁMETROS DE CALIDAD PARA EL TRÁFICO DE JUEGOS *ONLINE*

Partes de este artículo han sido publicadas como **"Influence of the Router Buffer on Online Games Traffic Multiplexing,"** Proc. International Symposium on Performance Evaluation of Computer and Telecommunication Systems **SPECTS 2011**, La Haya, Holanda, Jun. 2011, cuyos autores son José Mª Saldaña, Julián Fernández-Navajas, José Ruiz-Mas, José I. Aznar, Eduardo Viruete y Luis Casadesus.

Introducción

En el capítulo anterior se ha expuesto la adaptación de un método de multiplexión (inicialmente desarrollado para RTP) para su uso con tráfico de juegos *online*. Se han estudiado los ahorros de ancho de banda que se pueden conseguir para el tráfico cliente-a-servidor. En el capítulo que ahora comenzamos se estudiará el efecto de la multiplexión sobre la calidad percibida por el usuario.

Para ello, se tendrá en cuenta que el usuario se conecta normalmente a través de un *router* de acceso, y su tráfico llega al servidor a través de una red pública como es Internet. Por tanto, debemos tener en cuenta el efecto del *router* y de los retardos y pérdidas introducidos por la red, para así realizar un estudio más completo del efecto de la multiplexión.

Si observamos la Fig. 12.1, vemos el camino completo que debe recorrer un paquete para llegar desde la máquina del jugador hasta el servidor. En el capítulo anterior se ha estudiado sólo el retardo desde que el paquete sale de la máquina del jugador, hasta que sale del multiplexor. En este capítulo añadiremos el efecto del *buffer* del *router*, y el de la red.

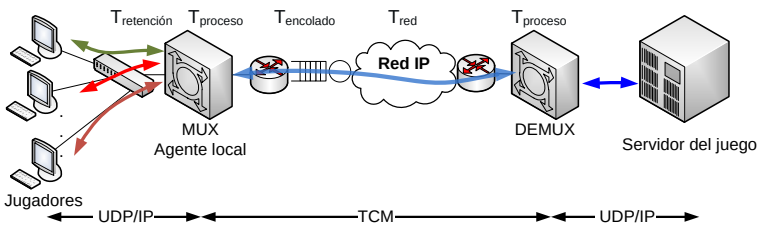


Fig. 12.1. Retardos del sistema

Esos tiempos son los siguientes:

- $T_{retención}$ es el tiempo que un paquete pasa en la cola del multiplexor. Como se ha visto, su valor medio es la mitad del periodo.

- $T_{proceso}$ representa el tiempo que el paquete pasa en el multiplexor y demultiplexor.

- $T_{encolado}$ es el tiempo de espera de la trama ya multiplexada en la cola del *router* de acceso. Como veremos, se modifica al multiplexar, puesto que las características del tráfico que recibe cambian.

- T_{red} es el retardo de red. Trabajaremos en el caso en que este retardo no depende del tamaño del paquete, y con una red que no añade más pérdidas.

Teniendo en cuenta que los jugadores *online* son clientes muy exigentes, debemos estudiar el efecto de la técnica TCM en los distintos parámetros que van a determinar la calidad percibida, que principalmente son el retardo y las pérdidas. Una manera que tienen los jugadores de medir el retardo es usando el tiempo de respuesta del sistema (SRT, *System Response Time)* [ZA04], que ellos habitualmente denominan *"ping"*. Este tiempo se refiere al intervalo desde que el jugador genera una acción, hasta que el servidor la procesa y produce el correspondiente cambio en el dispositivo de visualización del jugador. Los fabricantes de juegos dan mucha importancia a este parámetro. Como se ve en la Fig. 12.2, que corresponde a la pantalla de puntuaciones del juego *Counter Strike 1*, aparecen dos parámetros que se refieren a la puntuación de cada jugador, y el tercer parámetro (en la columna derecha) es el denominado *latency*, que es al que nos estamos refiriendo. Se suele medir en milisegundos.

En [Hen01] se llevó a cabo un estudio sobre los usuarios de *Half Life*, y se llegó a la conclusión de que los jugadores no aceptarían retardos mayores de 225-250 ms. En [ZA04] se concluye que el retardo tolerado para *Quake III* está entre 150 y 180 ms, mientras que para *Counter Strike* está por encima de 200 ms.

Existen por tanto una serie de compromisos a tener en cuenta, muchos de los cuales son similares a los que ya hemos visto para VoIP en capítulos anteriores. Hemos tratado de resumirlos en la Fig. 12.3. Por un lado, con respecto a las pérdidas, si aumentamos el periodo de multiplexión, estaremos aumentando también el tamaño de los paquetes generados. Esto puede producir un aumento

de las pérdidas en los casos en que el *buffer* del *router* tenga un tamaño fijo en número de bytes, ya que los paquetes grandes tendrán una mayor probabilidad de no caber en el *buffer*. Pero por otro lado, al multiplexar ahorramos ancho de banda y, por tanto, el tráfico ofrecido a la entrada del *router* será menor, hecho que puede reducir las pérdidas.

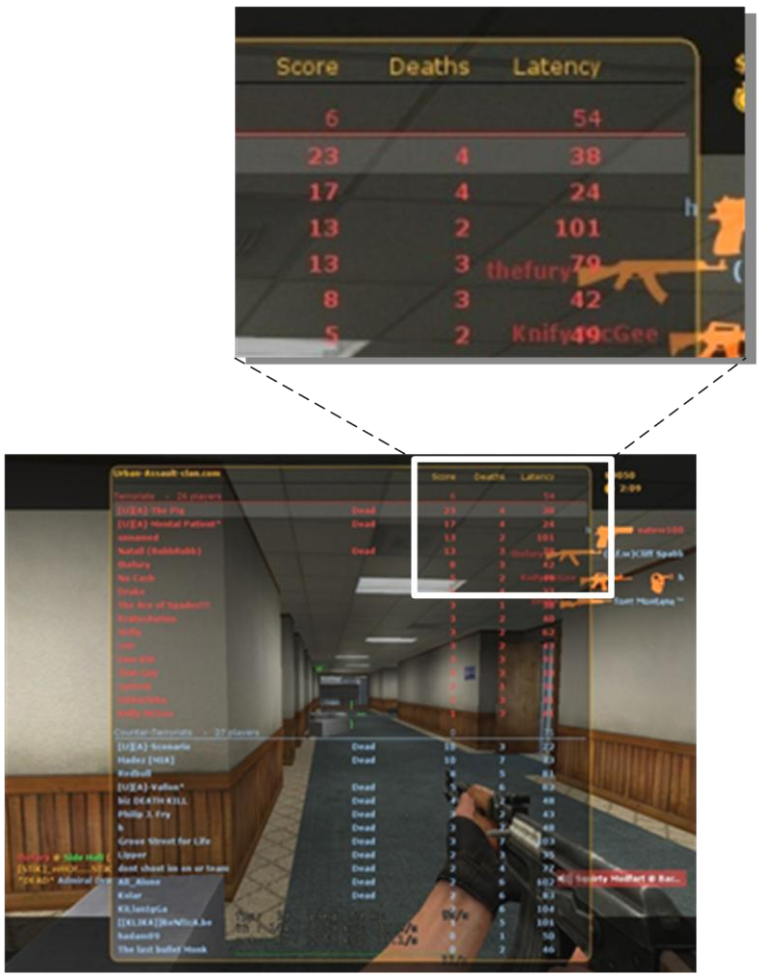


Fig. 12.2. Captura de pantalla de *Counter Strike 1* con las puntuaciones y el parámetro *latency*

Vemos que también aquí la política de encolado del *router* tiene importancia. Puede ocurrir que al aumentar el tamaño de los paquetes se modifiquen las pérdidas o no.

Si nos fijamos en los retardos, tenemos también dos efectos contradictorios: si aumentamos el periodo, generamos menos tráfico, por lo que también puede disminuir el retardo de encolado. Pero a la vez añadimos un mayor retardo de multiplexión. Así pues, debemos estudiar el sistema completo para ver qué efectos predominan en las diferentes situaciones.

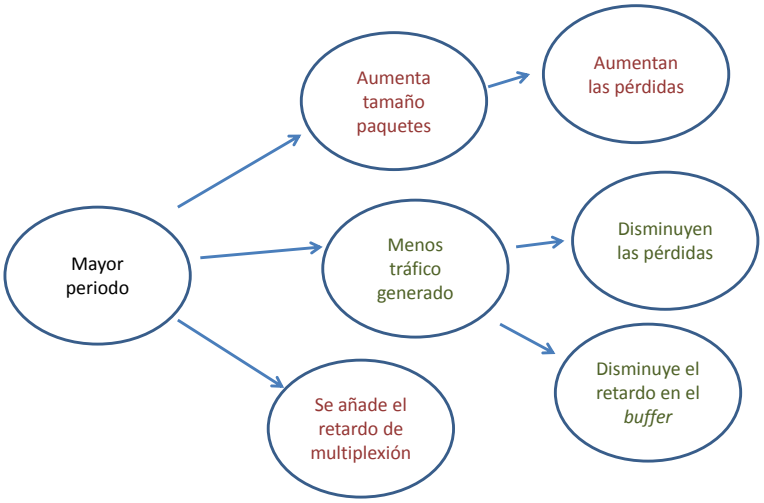


Fig. 12.3. Compromisos que aparecen al multiplexar

Por tanto, en este capítulo se mostrarán las pruebas realizadas para evaluar la influencia del esquema de multiplexión propuesto, variando los tiempos de multiplexión, y para diferentes políticas de *buffer* y cantidades de tráfico de fondo. Así pues, se enviará tráfico de juegos, nativo y multiplexado, para comparar los parámetros de calidad, como el retardo o la probabilidad de pérdidas. Se usará el tráfico de juegos comerciales, para ver el ahorro que se puede lograr en cada caso, buscando la relación entre el ahorro logrado y las características del tráfico generado.

Metodología de las pruebas

Las trazas de tráfico nativo son las mismas utilizadas en el capítulo anterior, que incluyen solamente tráfico de juego activo. Usando el mismo método, se han combinado distintas trazas para obtener la de 20 usuarios. Este número se ha usado por ser lo suficientemente grande como para que los efectos de la multiplexión se noten, a la vez que es un número de usuarios realista, ya que algunos juegos establecen un límite de usuarios en torno a 20 o 25 [FCFW02].

Se ha utilizado para generar las trazas de tráfico multiplexado el mismo programa con el que se han realizado las simulaciones presentadas en el capítulo anterior. Para cada valor del periodo se han obtenido dos ficheros de texto: uno con el tamaño de cada paquete y otro con el tiempo (en microsegundos) en que sale del multiplexor.

La Fig.12.4 muestra el esquema usado en la realización de las pruebas. Primero, el tráfico del juego y el de fondo se envían usando el generador JTG [Man11], que es capaz de enviar las trazas exactamente como eran originalmente, ya que lee los tiempos y tamaños de sendos ficheros de texto. Por tanto, no se ha necesitado usar un modelo teórico para el tráfico. La distribución de los tamaños de paquetes para el tráfico de fondo es la usada en capítulos anteriores: el 50% son de 40 bytes, el 10% de 576, y el 40% restante de 1.500 [Nas05].

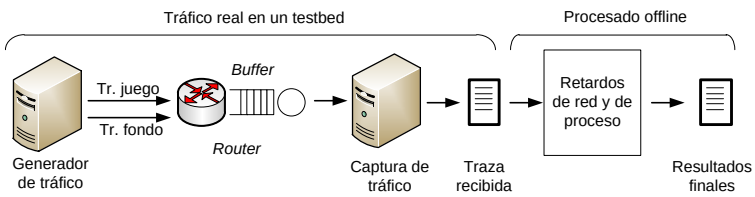


Fig 12.4. Esquema de las pruebas

Ambos tráficos comparten el mismo enlace de acceso, emulado de nuevo mediante la herramienta Linux *tc (traffic control)*. El límite de ancho de banda se ha establecido en 1 Mbps. El parámetro *burst* se ha establecido en 5.000 bytes. El tamaño del *buffer* se define según el tiempo máximo de un paquete en él. Se han usado dos diferentes: un *buffer* de alta capacidad, con un tiempo máximo de 500 ms, y otro limitado en tiempo, de 50 ms.

El tráfico, una vez recogido, se procesa *offline* para añadir un retardo de red, resultante de la suma de uno fijo de 20 ms, asociado a la distancia geográfica, y otro log-normal de media 20 ms y varianza 5. Esta distribución se ha tomado de [KPLK+09], usando retardos típicos en Internet [ATT11]. También se añade un retardo de procesado de 5 ms, que da cuenta de los retardos en el multiplexor y demultiplexor. En [SLLY02] se implementó en Linux un multiplexor para VoIP, y el retardo estaba por debajo de 1 ms, así que los 5 ms supondrían un valor pesimista para los retardos de procesado.

En las gráficas de retardo se ha usado como límite el valor de 200 ms, porque, como hemos visto, algunos estudios lo consideran el máximo retardo aceptable.

Con respecto a las pérdidas, su comportamiento depende del juego: algunos dejan de funcionar a partir de un 4% de pérdidas, mientras que otros funcionan bien incluso con un 35 % [ZA04], porque el juego incorpora un algoritmo para ocultar las pérdidas al usuario.

Pruebas y resultados

A continuación presentaremos algunas gráficas del retardo en un sentido (OWD) y pérdidas, usando diferentes valores de tráfico de fondo para saturar el *router*. Lógicamente, la multiplexión interesará sólo cuando el tráfico del juego tenga que competir con tráficos de fondo significativos. Para cada *buffer* se han usado tres tráficos: el *nativo*, en el que no hay multiplexión, y otros dos con *PE* = 25 ms y *PE* = 50 ms respectivamente.

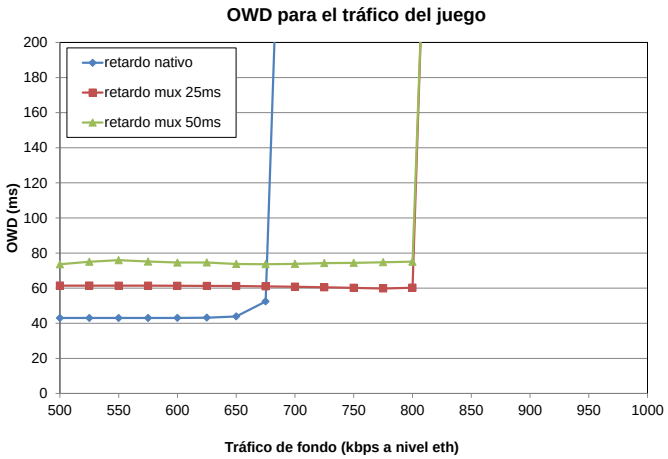
La Fig. 12.5 muestra los resultados del *buffer* de alta capacidad. Se puede ver que aparece un pequeño incremento en el OWD al multiplexar, debido al tiempo de retención (la mitad del periodo) y al de proceso. El ancho de banda del tráfico nativo es de 319 kbps a nivel *Ethernet*. Cuando el tráfico total rebasa el límite de ancho de banda, los retardos aumentan considerablemente. También se puede ver que el ahorro de ancho de banda (unos 120 kbps) se traduce en una mayor cantidad de tráfico de fondo que se puede soportar manteniendo los retardos en un nivel aceptable.

La Fig. 12.6 se ha obtenido usando el *buffer* limitado en tiempo. Los efectos del retardo son los mismos que para la Fig. 12.5, pero el uso de este *buffer* tiene la ventaja de mantener el OWD por debajo de 160 ms independientemente del tráfico de fondo.

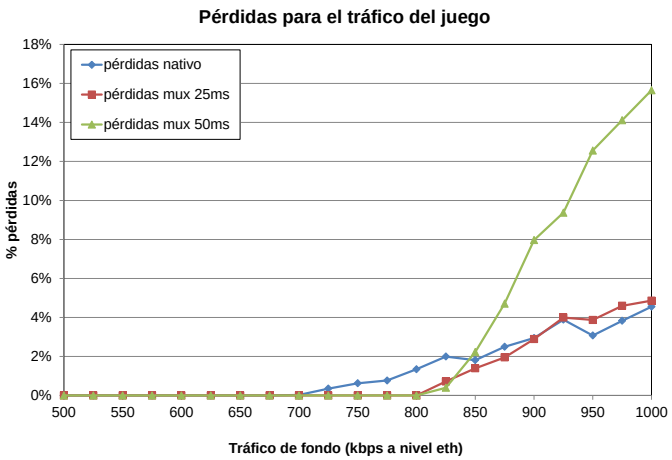
Hay otro efecto interesante relacionado con las pérdidas: la utilización del periodo de 25 ms da mejores resultados que el uso de tráfico nativo a causa del ahorro de ancho de banda, siendo además mejor que usar un periodo de 50 ms. Podemos discutir este resultado mirando a la gráfica de *20 jugadores* de la Fig. 11.18 b): los valores de *BWR* para 25 y 50 ms son muy similares, por estar muy cerca de la asíntota. De hecho, la diferencia en términos de ancho de banda es de 6 kbps. Pero si calculamos el tamaño medio de paquete, vemos que en el primer caso son 608 bytes, mientras que en el segundo son 1.192. Por tanto, si la política del *buffer* penaliza los paquetes grandes, será mejor no usar un valor grande para el periodo.

Pero a partir de 925 kbps de tráfico de fondo se puede ver que el tráfico nativo tiene menos pérdidas que el multiplexado, para los dos *buffer*. La causa es que, en este caso particular, los paquetes pequeños tienen menos probabilidad de

descarte. Vemos, por tanto, que hay situaciones en que multiplexar puede aumentar las pérdidas.



(a)



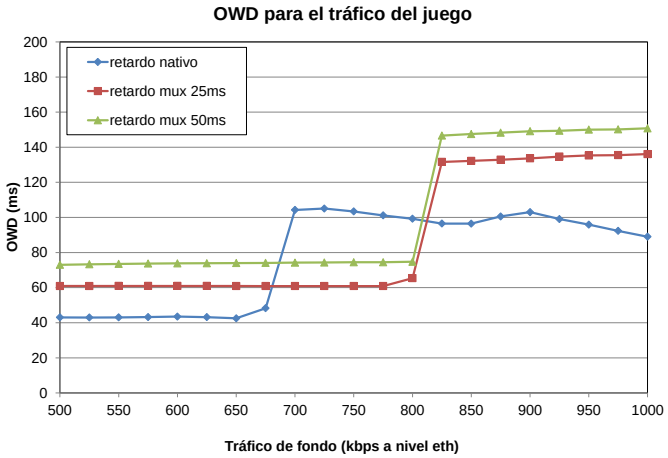
(b)

Fig 12.5. Retardo y pérdidas para el *buffer* de alta capacidad

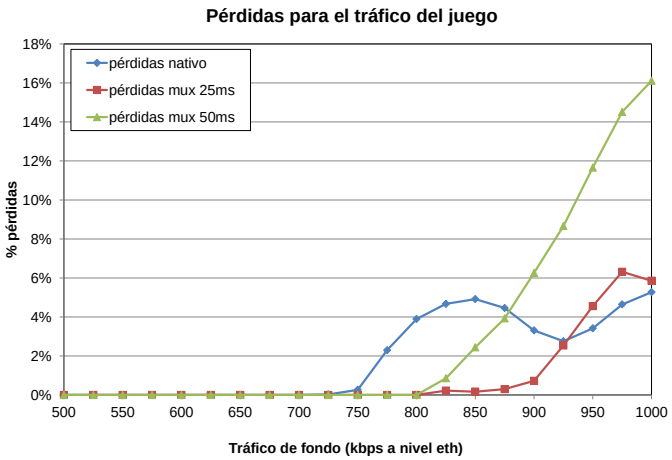
Una primera conclusión es que la multiplexión afecta de distinta manera al retardo y las pérdidas del tráfico del juego según sea la política del *buffer* del *router*.

Finalmente, analizaremos los resultados para el tráfico de fondo. La Fig. 12.7 muestra las pérdidas para el tráfico de fondo usando ambos *buffer*. Las líneas

discontinuas representan los valores del *buffer* de alta capacidad. Vemos que los resultados son muy similares. El ahorro de ancho de banda se traduce en una probabilidad de pérdidas menor, por lo que observamos que multiplexar es siempre beneficioso para el tráfico de fondo. Podemos concluir que estas políticas del *buffer* no tienen una influencia directa en las pérdidas para el tráfico de fondo.



(a)



(b)

Fig. 12.6. Retardo y pérdidas para el *buffer* limitado en tiempo

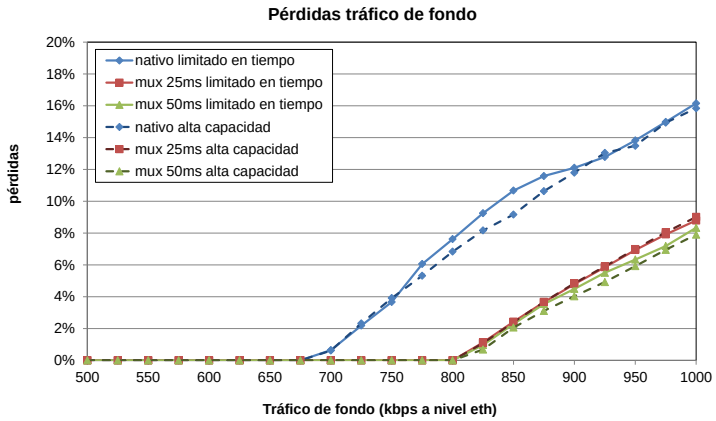


Fig. 12.7. Pérdidas del tráfico de fondo para ambos *buffer*

Conclusiones

Se ha presentado una comparativa del comportamiento de flujos TCM en función de las políticas de *buffer*, mostrando que la mejor solución no es siempre la que ahorra más ancho de banda. El tamaño de los paquetes se debe tener en cuenta, ya que algunas políticas penalizan los paquetes grandes.

Teniendo en cuenta la gran variedad de *router* que se pueden encontrar en el escenario considerado, las medidas presentadas ilustran la necesidad de particularizar la solución para cada caso concreto: cada red tendrá distintos retardos, diferentes comportamientos respecto a las pérdidas, variadas distribuciones de tráfico de fondo y un límite de paquetes por segundo que el *router* es capaz de gestionar. Por tanto, la técnica TCM nos puede ayudar a adaptar el tráfico al comportamiento de la red. Si dicha red está mejor preparada para bajas tasas de paquetes grandes, podemos modificar el tráfico para adaptarlo a esa circunstancia. TCM puede por tanto proporcionar una técnica flexible, útil para mejorar la experiencia del usuario en escenarios en que un número de jugadores comparten la misma ruta.

CONCLUSIONES

El diseño inicial de Internet no contemplaba su uso para el transporte de servicios de tiempo real. No obstante, con el paso de los años y debido a su gran éxito, cada día son más los servicios de este tipo que se utilizan. Esto ha llevado al sector, y en especial a los investigadores a la búsqueda de técnicas para ofrecer unas ciertas garantías temporales en la entrega de la información.

Esta tesis se enmarca dentro de las propuestas para solucionar este complejo problema: el uso de una red de conmutación de paquetes sin garantías de retardo máximo, como es Internet, para la provisión de servicios con requerimientos estrictos de tiempo real.

Selección de los servicios a estudiar

En los primeros capítulos se abordó el problema de estos servicios, y se seleccionaron dos de ellos especialmente significativos, como la VoIP y los juegos *online* FPS, para estudiar técnicas de optimización del tráfico, y ver hasta dónde se puede mejorar. Son dos aplicaciones que comparten algunas características, principalmente referidas a la naturaleza del tráfico.

El servicio de VoIP, cuando se usa en entornos empresariales, se suele integrar dentro de sistemas denominados de telefonía IP, que establecen una estructura centralizada para comunicar a todos los empleados entre sí y con el exterior. Por ello, se vio conveniente disponer de un sistema de este tipo en el que poder desarrollar nuestras pruebas. Muchos de los sistemas de telefonía actuales incluyen un sistema de control de admisión de llamadas (CAC), que rechaza las que no van a poder disfrutar de la suficiente calidad, o que van a perjudicar las que ya están en curso. Estos sistemas pueden ser más o menos inteligentes, pudiendo también estar distribuidos para dar un mejor servicio.

El servicio de juegos *online* se suele estructurar en torno a un servidor central al que se conectan todos los clientes. El servidor es quien controla la partida: recibe de cada cliente los paquetes que especifican sus acciones, después calcula el nuevo estado del juego y se lo comunica al resto de jugadores.

En cuanto al concepto de *Calidad de Servicio* (QoS) ha estado presente a lo largo de todo el trabajo. Existen unos parámetros objetivos medibles en la red, como pueden ser el retardo o las pérdidas, que modifican el tráfico, pero debemos tener

en cuenta que lo más importante es el grado de satisfacción que generan en el usuario final de la aplicación. Por eso, existen modelos para estimar la calidad percibida por el usuario en función de los parámetros de la red. Estos modelos buscan resumir en un solo parámetro el efecto de los diferentes elementos de la comunicación, especialmente la red.

Redes de acceso

En la actualidad los usuarios se conectan mayoritariamente a Internet a través de redes de acceso. Puede ocurrir que el usuario disponga de una red local de alta velocidad, y también sucede que la red troncal funciona a velocidades aún mayores, pero la red de acceso, debido al estado actual de la tecnología, funciona a velocidades mucho menores, que en ocasiones están varios órdenes de magnitud por debajo. Esto provoca que la red de acceso constituya el principal cuello de botella que restringe la calidad de las aplicaciones.

En concreto el *router* de acceso, que conecta al usuario a la red, se muestra como un elemento clave cuyo comportamiento se debe tener en cuenta a la hora de estudiar el funcionamiento de las aplicaciones de tiempo real que usan Internet. En concreto, el dimensionado del *buffer* del *router* ha dado lugar a un gran número de trabajos de investigación, que han puesto de manifiesto un compromiso entre tamaño y retardo. Si el *buffer* es muy grande, se facilita una utilización elevada del enlace, pero introduce retardos, que afectan especialmente a los servicios de tiempo real. Pero si es muy pequeño puede aumentar las pérdidas de paquetes, pues en muchas ocasiones serán descartados al no tener lugar donde ser almacenados. En la literatura se encontró la propuesta de un *buffer* limitado en tiempo, que se ha mostrado especialmente adecuado para limitar el retardo total en las aplicaciones de tiempo real.

Los flujos de los servicios de tiempo real escogidos, que utilizan esta red de acceso, dividen la información en paquetes muy pequeños, que se deben enviar con frecuencias altas. Al usar una red IP de conmutación de paquetes, cada elemento de información enviado debe ir precedido de unas cabeceras en las que se indica la dirección origen, destino, puerto, etc. Esto provoca una disminución de la eficiencia en el tráfico.

Optimización del tráfico

Muchos de los campos de estas cabeceras, que son imprescindibles, son idénticos para todos los paquetes de un mismo flujo, o varían muy poco entre un paquete y el siguiente. Aprovechando esta circunstancia, existen métodos de compresión de

cabeceras que permiten reducir drásticamente su número de bytes. Pero estos métodos sólo pueden emplearse salto a salto, pues un paquete con una cabecera comprimida no puede circular por Internet. Este problema se puede solucionar mediante técnicas de tunelado, que introducen un flujo dentro de un túnel, que de este modo puede funcionar extremo a extremo.

Otra posible mejora para el tráfico es la multiplexión, un concepto que ha sido ampliamente utilizado, y que consiste en unir el contenido de varios paquetes que comparten la misma cabecera. Si se aplica a las muestras de un mismo flujo en servicios de tiempo real, pueden aparecer retardos intolerables para el usuario. Pero en un escenario en el que varios flujos comparten un mismo camino, sí podemos hacer que un paquete de cada flujo se incluya dentro de uno multiplexado, añadiendo solamente un pequeño retardo necesario para disponer de todos los paquetes y poder así agruparlos.

En esta línea, el IETF, el organismo que define los protocolos que se usan en Internet, definió TCRTP como un método de multiplexión para unir en un túnel varios flujos de tiempo real que utilicen el protocolo RTP. Este método no define un nuevo protocolo, sino que utiliza otros ya existentes. Incluye multiplexión, compresión de cabeceras y tunelado para poder funcionar extremo a extremo. Existen otras propuestas, pero en este trabajo se ha optado por TCRTP principalmente por ser un estándar del IETF.

Entornos de pruebas

Así pues, una vez seleccionados los dos servicios de tiempo real, y el método de multiplexión que se quiere utilizar, era necesario disponer de entornos de pruebas adecuados para las medidas. Para aquellas que requieren el funcionamiento de aplicaciones reales, se ha implementado una plataforma de pruebas *(testbed)* basada en virtualización, que permite incluir un escenario de red completo dentro de una sola máquina física. Se ha optado por la solución de paravirtualización Xen, por su buena eficiencia y capacidad de aislamiento entre las máquinas. Se han buscado también métodos para emular el comportamiento de la red en cuanto a retardos, pérdidas, *buffer*, etc.

También se han utilizado generadores de tráfico, que permiten enviar paquetes siguiendo diferentes distribuciones estadísticas. Especialmente útil ha resultado el envío a partir de un fichero con tiempos de salida y tamaños de paquete, pues nos ha permitido enviar trazas de partidas reales capturadas por otros grupos de

investigación para diferentes juegos *online,* y también esas mismas trazas una vez multiplexadas en el entorno de simulación.

Para algunas de las pruebas se ha recurrido a la simulación, en este caso utilizando Matlab. Esto ha permitido la realización de pruebas con escenarios más grandes. Se ha utilizado un esquema híbrido, en el que los parámetros de calidad se obtenían en el *testbed*, y luego se incluían como parámetros de entrada en el entorno de simulación.

Sistema de telefonía IP

Para el estudio del servicio de VoIP, el primer paso fue disponer de un sistema de telefonía similar al empleado en soluciones comerciales. Nos propusimos implementarlo utilizando solamente aplicaciones de *software* libre, de forma que tuviésemos acceso al código fuente, ya que esto nos proporcionaba una mayor flexibilidad a la hora de modificar parámetros para las diferentes pruebas. También se seleccionó el protocolo de señalización SIP por tratarse de un estándar abierto y ampliamente utilizado en la actualidad. Se pensó en un sistema distribuido, adecuado para una empresa con sucursales en diferentes países, y agrupadas en zonas. Cada ubicación dispondría de un agente local, encargado de controlar todas sus llamadas, y conectado con una PBX que se encontraría en el centro de datos.

El agente local, situado en cada oficina, a través de un *proxy* SIP que tiene incluido, se encarga de implementar el control de admisión de las llamadas, y también controla las líneas que conectan la oficina a RTC. De este modo puede permitir que el sistema completo comparta todas las líneas entre todas las sucursales, aumentando así la probabilidad de admisión. Estas llamadas se pueden establecer en dos tramos: uno a través de Internet hasta la sucursal destino, y otro a través de RTC hasta el usuario final.

El sistema se implementó en la plataforma de pruebas, eligiendo para ello aplicaciones adecuadas. Una vez en funcionamiento, se realizaron en primer lugar diferentes medidas de los tiempos de procesado, mostrando que eran lo suficientemente pequeños como para no afectar a la experiencia del usuario. Se comprobó que el hecho de introducir los *proxy* SIP en el camino de la señalización no suponía un inconveniente.

Para medir los parámetros de calidad de las llamadas, se recurrió al envío de diferentes flujos a través de un *router* emulado, y se calcularon después los resultados en términos de retardo, pérdidas, *jitter* y Factor R. Se compararon

diferentes tamaños de *buffer*, observándose que los *buffer* de alta capacidad funcionan bien mientras no se supera el límite de ancho de banda. Pero a partir de ese momento introducen retardos inaceptables para el usuario de un servicio de tiempo real. Por otro lado, se estudió el *buffer* limitado en tiempo, que mostró un comportamiento diferente: cuando el ancho de banda no es suficiente, los paquetes de mayor tamaño son los más perjudicados, por lo que el tráfico de voz, dado su pequeño tamaño, era capaz de obtener buenos resultados en presencia de cantidades mayores de tráfico de fondo.

También se realizaron simulaciones, para obtener en primer lugar tiempos de establecimiento, que mostraron una dependencia casi exclusiva del retardo de red. En segundo lugar se midió la probabilidad de admisión, observándose que el hecho de compartir las líneas de todas las sucursales permite mejorarla significativamente. También se vio la dependencia mutua entre la probabilidad de admisión del sistema CAC, que depende principalmente del ancho de banda de la conexión a Internet, y la de las líneas de conexión a RTC. Si aumenta el límite del CAC, se pueden compartir más las líneas, lo que aumenta la probabilidad de que una llamada encuentre una línea disponible.

Multiplexión de voz

Con posterioridad, se estudió analíticamente, y mediante pruebas con tráfico real, el método de multiplexión TCRTP, con la idea de poder luego incluirlo en el sistema de telefonía ya diseñado. En primer lugar se realizó un estudio matemático, y se observó que el ahorro de ancho de banda tiene una asíntota. Por tanto, aumentar indefinidamente el número de flujos multiplexados no interesa, pues la ganancia va siendo cada vez menor.

Se observó que otro parámetro que disminuye al utilizar la multiplexión es el número de paquetes por segundo enviados, pues los flujos se agrupan. Esto puede ser interesante si el *router* presenta un límite de paquetes por segundo que es capaz de gestionar. En la multiplexión aparece por tanto un compromiso: se pueden reducir el ancho de banda y los paquetes por segundo, pero simultáneamente aumenta el tamaño medio de los paquetes, lo que puede ser un inconveniente según sea el comportamiento de nuestro *router* y la distribución de tamaños de paquete del tráfico de fondo.

Para valorar el efecto de la multiplexión del tráfico de voz, se enviaron diferentes flujos de tráfico RTP nativo y multiplexado, en presencia de tráfico de fondo, y utilizando diferentes implementaciones del *buffer* del *router*.

En primer lugar se observó que si se prioriza el tráfico de voz, mediante un ancho de banda exclusivo para él, el comportamiento es de tipo "escalón": funciona bien mientras hay ancho de banda. Para un *buffer* que mide su capacidad en número de paquetes, y no en bytes, se vio que la multiplexión supone una ventaja, pues el tamaño del paquete aumenta, así que en cada paquete se almacenan más muestras. Por tanto, al usar paquetes más grandes, cabe más información en la cola, y se reducen las pérdidas globales. Esto tiene una contrapartida, y es que al usar TCRTP se añade más retardo, pues los paquetes requieren más tiempo para ser enviados. De todas formas, el ahorro de ancho de banda conseguido por TCRTP hace que funcione correctamente en presencia de mayores cantidades de tráfico de fondo que para RTP nativo.

Con el *buffer* de alta capacidad volvemos a encontrar un comportamiento de tipo "escalón", también para TCRTP. En el momento en que se ocupa todo el ancho de banda, el retardo crece hasta límites inaceptables para el usuario. Si usamos el *buffer* limitado en tiempo, encontramos varios efectos simultáneos: por un lado, TCRTP ahorra ancho de banda, pero por otro genera paquetes más grandes y añade pequeños retardos adicionales. Como consecuencia, en las gráficas se observa una zona en la que es preferible utilizar TCRTP, y otra en la que su uso no aporta una mejora en la calidad.

Respecto al tráfico de fondo, se ha observado un efecto claro: se obtienen menores pérdidas cuanto mayor es el ahorro de ancho de banda. Es decir, la multiplexión siempre resulta adecuada para evitar pérdidas en el tráfico de fondo.

Se han realizado también pruebas modificando el número de muestras por paquete, y variando el retardo añadido por la red. Se ha visto cómo a partir de un cierto valor del retardo, la calidad empeora sustancialmente. Por eso, según sea el retardo de la red, deberemos evitar el uso de un número grande de muestras por paquete.

Por último, se estudió el modo de agrupar un número de flujos, y se vio que hay casos en que agrupar todos los flujos en un solo túnel no es la mejor solución. Si el *buffer* penaliza los paquetes grandes, se obtienen mejores resultados usando varios túneles que usando uno solo. Esto es debido al comportamiento asintótico del ahorro: a partir de cierto número de flujos, el hecho de agrupar más no supone un incremento significativo del ahorro. Y por otro lado, el aumento en el tamaño de los paquetes puede hacer que la mejor solución no sea usar un solo túnel.

Una vez estudiado el método de multiplexión, se decidió comprobar si integrarlo dentro del sistema de telefonía podría suponer una ventaja. Se realizaron simulaciones para comparar los resultados de calidad y de probabilidad de admisión. En primer lugar fue necesario el uso del *testbed* para obtener una batería de resultados, que luego se utilizaron en las simulaciones, para así conocer los parámetros de calidad esperados en cada caso. Los resultados mostraron que la inclusión de TCRTP aumentaba la calidad de las llamadas, y que este margen de calidad se podría sacrificar en parte, transformándolo en incremento de probabilidad de admisión, mediante el aumento del límite del CAC.

Adaptación del esquema para juegos FPS

En la última sección de la tesis se adaptó el esquema de optimización de tráfico, para su uso en juegos *online*. Se estudió el comportamiento del tráfico de los juegos FPS, y se adaptó el esquema TCRTP para poder utilizarlo en juegos, teniendo en cuenta que el tráfico que generan no es RTP. Por ello, se recurrió a otros esquemas de compresión de cabeceras diferentes de ECRTP. El método adaptado se ha denominado TCM (Tunelado, Compresión, Multiplexión).

Se comprobó que existen escenarios, como por ejemplo los denominados *cibercafés*, en los que el tráfico de un número de usuarios comparte el mismo camino hasta el servidor. Se decidió optimizar el tráfico cliente-a-servidor, por ser el que presenta mayores problemas a causa del *overhead*.

Mediante un análisis teórico se compararon las diferentes políticas que ayudan a decidir qué paquetes multiplexar y con qué frecuencia. De nuevo, igual que había pasado con el servicio de voz, se pudo observar un comportamiento asintótico, en el que el ahorro que se puede conseguir tiene un límite. En este caso, con IPv4 se puede llegar a ahorrar un 30% del ancho de banda, pero con IPv6 esta cifra puede alcanzar un 50%, debido al mayor *overhead* de la versión más reciente del protocolo.

Este comportamiento se ha corroborado mediante simulaciones, utilizando el tráfico extraído de partidas reales de diferentes juegos FPS. Dado que el sistema TCM añade retardo y *jitter*, se ha visto conveniente estudiar su efecto en los parámetros que determinan la calidad percibida. Para ello, se han realizado envíos de tráfico real del juego, multiplexado y nativo, para comparar su comportamiento. Por un lado, TCM añade retardos, pero por otro reduce el tráfico total ofrecido al *router*. En este caso, se repiten algunos de los compromisos que hemos encontrado para el tráfico de voz. Por ejemplo, si se

quiere aumentar el ahorro de ancho de banda, también aumenta el tamaño de los paquetes, por lo que el comportamiento del *buffer* del *router* vuelve a ser clave para decidir los parámetros de la multiplexión.

Resultados comunes para ambos servicios

En definitiva, se ha visto que los dos servicios estudiados en esta tesis: VoIP y juegos *online* FPS, han mostrado comportamientos muy similares, mostrando efectos muy parecidos. El comportamiento del *buffer* del *router* tiene una gran importancia para ambos, y se debe tener en cuenta a la hora de tomar decisiones relacionadas con la multiplexión. Esto se debe a que se trata de servicios en tiempo real que generan paquetes muy pequeños. Por eso, el *overhead* es clave a la hora de entender la problemática que aparece con la multiplexión: si una parte importante del tamaño del paquete está ocupada por las cabeceras, al comprimirlas y compartirlas entre un gran número de paquetes, se obtienen ahorros sustanciales en ancho de banda. También el ahorro en paquetes por segundo resulta significativo en ambos casos.

Líneas futuras de trabajo

En las pruebas presentadas se han utilizado modelos de tráfico de fondo siguiendo una distribución de Poisson. Podría resultar interesante realizar otras pruebas utilizando diferentes modelos de tráfico, como el de Pareto, que, según algunos estudios [VV08], se ajusta más al tráfico real existente en Internet.

En el modelo de red usado, todas las pérdidas consideradas tenían lugar en el *buffer* del *router* o en el *buffer* de *dejitter* . Sería bueno estudiar también otros *buffer* con un comportamiento diferente. Por otro lado, no se ha contemplado la posibilidad de que la red añada nuevas pérdidas. En futuros trabajos se debería considerar que la red pierde paquetes, y que lo hace con diferentes probabilidades en función de su tamaño.

En las pruebas del sistema de telefonía IP realizadas mediante simulación, se ha considerado que todas las sucursales generaban una tasa de llamadas similar. Se podrían realizar nuevas simulaciones encaminadas a comprobar qué ocurriría en el caso de que cada sucursal generase un tráfico diferente.

Hemos visto cómo en el servicio de VoIP multiplexado, la solución óptima no siempre es la inclusión de todas las llamadas en un solo túnel. Puede haber casos en que usar varios túneles dé mejores resultados, a causa del distinto trato que da el *buffer* a los paquetes según su tamaño, debido a su implementación. Para el caso

de juegos *online* esta problemática se plantea de un modo diferente, puesto que el parámetro que define el tamaño del paquete no es el número de conversaciones, sino el periodo con el que se envía la información multiplexada. Por tanto, en lugar de variar el número de túneles, habrá que encontrar en cada caso el valor del periodo para el que se optimiza la calidad percibida. Esta tarea queda como trabajo futuro.

Al estudiar el servicio de telefonía IP se ha utilizado el E-Model como estimador de la calidad percibida. Al tratarse de un parámetro ampliamente aceptado y con una utilización muy generalizada, se puede considerar que estima bien la calidad. Existen diferentes parámetros que lo modifican según el *codec*, el número de muestras por segundo, etc. Pero en el caso de los juegos *online*, no se cuenta con un consenso tan grande a la hora de estimar la calidad. Por un lado, cada juego es diferente y no genera el tráfico con la misma frecuencia. Algunos juegos se ven más afectados que otros por el retardo o por las pérdidas, así que la solución por la que se ha optado ha sido presentar el retardo y las pérdidas por separado, sin integrarlos en un estimador de calidad. Queda para futuros trabajos la tarea de buscar en la literatura estos estimadores, que variarán con los títulos, y realizar un estudio de los resultados de calidad percibida cuando se usa la técnica TCM presentada en esta tesis.

También se podría abordar en futuros trabajos la tarea de aplicar estas técnicas de optimización al tráfico generado por otras aplicaciones, como videoconferencia, televigilancia, otros géneros de juegos no FPS, aplicaciones de teleasistencia, etc.

[3GPP06] 3GPP TS 24.228 v5.15.0, "Signalling flows for the IP multimedia call control based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP)," Stage 3, R5, sep. 2006.

[AHT06] B. Athwal, F.C. Harmatzis, V.P. Tanguturi, "Replacing Centric Voice Services with Hosted VoIP Services: An Application of Real Options Approach," Proc. 16th International Telecommunications Society (ITS) European Regional Conference, Porto, Portugal, sep. 2006.

[AKM04] G. Appenzeller, I. Keslassy, N. McKeown. "Sizing router buffers," SIGCOMM '04, pp 281–292, ACM Press. New York, USA, 2004.

[Ale02] J. Alexander et. al. "Cisco CallManager Fundamentals," Cisco Press, 2002.

[Ast11] www.asterisk.org, Última visita 8/8/2011.

[ATT11] AT&T, "Global Network Latency Averages," http://ipnetwork.bgtmo.ip.att.net/pws/global_network_avgs.html, último acceso 1/9/2011.

[awk11] Free Software Foundation, Inc, "The GNU Awk User's Guide," http://www.gnu.org/software/gawk/manual/gawk.html, último acceso 1/9/2011.

[BA06] P. Branch, G. Armitage, "Extrapolating Server To Client IP traffic From Empirical Measurements of First Person Shooter games," Proc. 5th ACM SIGCOMM Workshop on Network and system support for games (NetGames '06). ACM, NY, USA, 2006.

[BBCC+ 04] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, M. Wawrzoniak, "Operating System Support for Planetary-Scale Network Services," Proc. 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04), San Francisco, CA, 2004.

[BCS94] R. Braden, D. Clark, S. Shenker, "Integrated Services in the Internet Architecture: an Overview," RFC 1633, 1994.

[BDP07] A. Botta, A. Dainotti, A. Pescapè, "Multi-protocol and multi-platform traffic generation and measurement," INFOCOM 2007 DEMO Session, Anchorage, Alaska, USA, May 2007.

[BG98] J. M. Boyce, R. D. Gaglianello, "Packet loss effects on MPEG video sent over the public Internet," Proc. Sixth ACM MULTIMEDIA '98. ACM, New York, NY, USA, 181-190, 1998.

[BJS00] L. Breslau, S. Jamin, S. Shenker, "Comments on the performance of measurement-based admission control algorithms," INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE , vol.3, pp.1233-1242 vol.3, 26-30 Mar. 2000.

[Bla98] S. Blake et al, RFC 2475, "An Architecture for Differentiated Services," 1998.

[BM10] S. H. Batool, K. Mahmood, "Entertainment, communication or academic use? A survey of Internet cafe users in Lahore, Pakistan," Information Development vol. 26. pp 141-147, May. 2010.

[Bra97] R. Braden et al, RFC 2205, "Resource ReSerVation Protocol (RSVP)," 1997.

[BRS02] D. Bauer, S. Rooney, P. Scotton, "Network Infrastructure for Massively Distributed Games," Proc. 1st workshop on Network and system support for games (NetGames'02), pp. 36-43. ACM, New York, 2002.

[BSPL+09] P. K. Biswas, C. Serban, A. Poylisher, J. Lee, S. Mau, R. Chadha, C. J. Chiang, R. Orlando, K. Jakubowski, "An integrated testbed for Virtual Ad Hoc Networks," 5th International Conference on Testbeds and Research Infrastructures for the Development of Networks Communities and Workshops Tridentcom 2009, pp.1-10, 2009.

[BSUB98] M. S. Borella, D. Swider, S. Uludag, G. B. Brewster, "Internet Packet Loss: Measurement and Implications for End-to-End QoS," Parallel Processing Workshops, International Conference on, p. 3, International Conference on Parallel Processing Workshops (ICPPW'98), 1998.

[CFSS05] C. Chambers, W. Feng, S. Sahu, D. Saha, "Measurement-based Characterization of a Collection of On-line Games," Proc. 5th ACM SIGCOM conference on Internet Measurement (IMC'05). USENIX Association, Berkeley, 2005.

[CGKR10] L. Colitti, S. H. Gunderson, E. Kline, T. Refice, "Evaluating IPv6 adoption in the Internet," Proc. 11th International Conference on Passive and Active Network Measurement, pp 141–150. Springer-Verlag, abr. 2010.

[CGPD07] E. Conchon, J. Garcia, T. Pérennou, M. Diaz, "Improved IP-level Emulation for Mobile and Wireless Systems," Proc. IEEE Wireless Communications & Networking Conference, Hong Kong, 2007.

[CGS05] P. Camarda, C. Guaragnella, D. Striccoli, "A New MBAC Algorithm for Video Streaming Based on Autoregressive Adaptive Filtering," ICME 2005, IEEE International Conference on Multimedia and Expo, pp.1512-1515, 2005.

[CHL05] K. Chen, P. Huang, C. Lei, "Game traffic analysis: An MMORPG perspective," Proc. international workshop on Network and operating systems support for digital audio and video (NOSSDAV'05), pp. 19-24. ACM, New York, 2005.

[Cis01A] "SIP: Measurement-Based Call Admission Control for SIP," http://www.cisco.com/en/US/docs/ios/12_2t/12_2t15/feature/guide/ftcacsip.pdf, última visita 29/8/2011.

[Cis01B] "VoIP Call Admission Control," http://www.cisco.com/en/US/docs/ios/solutions_docs/voip_solutions/CAC.pdf, última visita 29/8/2011.

[CJ99] S. Casner et V. Jacobson, RFC 2508, "Compressing IP/UDP/RTP Headers for Low-Speed Serial Links," feb. 1999.

[CK00] C. Cetinkaya, E. Knightly, "Egress Admission Control," Proc. IEEE INFOCOM 2000, mar. 2000.

[CLMR+06] A. Couvreur, L. M. Le-Ny, A. Minaburo, G. Rubino, B. Sericola, L. Toutain, "Performance analysis of a header compression protocol: The ROHC unidirectional mode," Telecommunication Systems, vol. 31, no. 6, pp. 85–98, 2006.

[CR00] R.G. Cole, J.H. Rosenbluth, "Voice over IP performance monitoring," SIGCOMM Comput. Commun. Rev. 31, 2 (abr. 2001), pp. 9-24.

[CWXL+03] X. Chen, C. Wang, D. Xuan, Z. Li, Y. Min, W. Zhao, "Survey on QoS Management of VoIP," Proc. 2003 International Conference on Computer Networks and Mobile Computing, IEEE Computer Society.

[DC02] C. Demichelis, P. Chimento, RFC 3393, "IP Packet Delay Variation," Nov. 2002.

[DD06] A. Dhamdhere, C. Dovrolis, "Open issues in router buffer sizing," Comput. Commun. Rev., vol. 36, no. 1, pp. 87–92, ene. 2006.

[DFK06] G. Dan, V. Fodor, G. Karlsson, "On the effects of the packet size distribution on FEC performance," Computer Networks, Volume 50, Issue 8, Selected Papers from the 3rd International Workshop on QoS in Multiservice IP Networks (QoS-IP 2005), pp 1104-1129, jun. 2006.

[DKP07] G. Dimitriadis, S. Karapantazis, F. N. Pavlidou, "Comparison of Header Compression Schemes over Satellite Links," Proc. International Workshop on IP Networking over Next-generation Satellite Systems (INNSS'07), Budapest, Hungary, jul. 2007.

[DNP99] M. Degermark, B. Nordgren, D. Pink, RFC 2507, "IP Header Compression," 1999.

[DWW05] M. Dick, O. Wellnitz, L. Wolf, "Analysis of factors affecting players' performance and perception in multiplayer games," Proc. 4th ACM SIGCOMM workshop on Network and system support for games (NetGames '05). ACM, New York, NY, USA, 1-7, 2005.

[EC04] E. Ertekin, C. Christou, "Internet protocol header compression, robust header compression, and their applicability in the global information grid," IEEE Communications Magazine, vol. 42, pp. 106-116, 2004.

[EGGMR05] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, T. Roughgarden. "Part III: routers with very small buffers," SIGCOMM Comput. Commun. Rev. 35, 3, pp 83-90, jul 2005.

[FCFW02] W. Feng, F. Chang, W. Feng, J. Walpole, "Provisioning On-line Games: A Traffic Analysis of a Busy Counter-Strike Server," SIGCOMM Comput. Commun. Rev. 32, p. 18, 2002.

[FCFW05] W. Feng, F. Chang, W. Feng, J. Walpole, "A Traffic Characterization of Popular On-Line Games," IEEE/ACM Trans. Networking, pp. 488-500, 2005.

[FKW08] B. Furuholt, S. Kristiansen, F. Wahid, "Gaming or gaining? Comparing the use of Internet cafes in Indonesia and Tanzania," The Intern. Inform. & Library Review, Volume 40, Issue 2, pp 129-139, jun. 2008.

[GD98] L. Gautier, C. Diot, "Design and Evaluation of MiMaze, a Multi-player Game on the Internet," Proc. IEEE International Conference on Multimedia Computing and Systems

(ICMS'98), IEEE Computer Society, pp. 233. Washington, 1998.

[GKFM+ 07] A. Gavras, A. Karila, S. Fdida, M. May, M. Potts, "Future internet research and experimentation: the FIRE initiative," ACM SIGCOMM Computer Communication Review, v.37 n.3, 2007.

[Gök07] E. Göktürk, "A stance on emulation and testbeds," Proc. 21st European Conf. on Modelling and Simulation ECMS 2007.

[GS07] M. Gurol, T. Sevindik, "Profile of Internet Cafe users in Turkey," Telematics and Informatics, Vol. 24, Issue 1, pp 59-68, feb. 2007.

[GT03] M. Grossglauser, D. Tse, "A Time-Scale Decomposition Approach to Measurement-Based Admission Control," IEEE/ACM Transactions on Networking, ago. 2003.

[GWCC+07] D. Gupta, D. Wu, C. C. Chen, C. Chuah, P. Mohapatra, S. Rungta, "Experimental Study of Measurement-based Admission Control for Wireless Mesh Networks," IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, pp. 1-9, 2007 IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems, 2007.

[Han06] M. Handley, "Why the Internet only just works," BT Technology Journal 24, 3, pp 119-129, jul. 2006.

[Hem05] S. Hemminger, "Network Emulation with NetEm," Proceedings of Linux Conference AU, Canberra, 2005.

[Hen01] T. Henderson, "Latency and User Behaviour on a Multiplayer Game Server," Lect. Notes Comp. Science, Springer Berlin/ Heidelberg, pp 1-13, vol 2233, 2001.

[HLR08] T. R. Henderson, M. Lacage, G. F. Riley, "Network Simulations with the ns-3 Simulator," demo paper at ACM SIGCOMM'08, ago. 2008.

[HTT99] T. Hoshi, K. Tanigawa, K. Tsukada, "Proposal of a method of voice stream multiplexing for IP telephony systems," Proc. IWS '99, pp. 182–188, feb. 1999.

[IK01] I. M. Ivars, G. Karlsson, "PBAC: Probe-Based Admission Control," Proc. QoFIS 2001, 2001, pp. 97–109.

[ITU96] International Telecommunication Union (ITU-T Recommendation G.114), "One-way transmission time," feb. 1996.

[ITU03] International Telecommunication Union (ITU-T Recommendation G.107), "The E-model, a computational model for use in transmission planning," Mar. 2003.

[Jac98] V. Jacobson, RFC 1144, "Compressing TCP/IP Headers for Low-Speed Serial Links," 1990.

[JENN04] Y. Jiang, P. J. Emstad, V. Nicola, A. Nevin, "Measurement-based admission control: A revisit," 17th Nordic Teletraffic Seminar, 2004.

[Jon11] M. Tim Jones, "Virtual Linux. An overview of virtualization methods, architectures, and implementations," http://www.ibm.com/developerworks/linux/library/l-linuxvirt/, última visita agosto 2011.

[JX03] X. Jiang, D. Xu, "vbet: a vm-based emulation testbed," Proc. ACM SIGCOMM workshop Models, methods and tools for reproducible network res. MoMeTools'03, ACM Press, pp. 95–104, New York, 2003.

[KCGT+03] T. Koren, S. Casner, J. Geevarghese, B. Thompson, P. Ruddy, RFC 3545, "Enhanced Compressed RTP (CRTP) for Links with High Delay, Packet Loss and Reordering," Jul. 2003.

[KPLK+09] S. Kaune, K. Pussep, C. Leng, A. Kovacevic, G. Tyson, R. Steinmetz, "Modelling the internet delay space based on geographical locations," 17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2009), feb. 2009.

[KW05] J. Korhonen, Y. Wang, "Effect of packet size on loss rate and delay in wireless links," Wireless Communications and Networking Conference, 2005 IEEE , vol.3, pp. 1608- 1613 Vol. 3, 13-17, mar. 2005.

[LABC03] T. Lang, G. Armitage, P. Branch, H. Choo, "A Synthetic Traffic Model for Half-Life," in Proc. Australian Telecom, Networks and Applications Conference (ATNAC), Melbourne, 2003.

[Liu07] J. Liu, "A Primer for Real-Time Simulation of Large-Scale Networks," Proc. 41st Annual Simulation Symposium (anss-41 2008) (ANSS-41 '08). IEEE Computer Society, Washington, DC, USA, 85-94.

[LKC04] K. Lee, B. Ko, S. Calo, "Adaptive Server Selection for Large Scale Interactive Online Games," Proc. 14th International Workshop on Network and operating systems support for digital audio and video (NOSSDAV'04), pp. 152-157. ACM, New York, 2004.

[Man11] J. Manner, JTG, http://www.cs.helsinki.fi/u/jmanner/software/jtg/, último acceso 7/9/2011.

[MFW02] M. Mauve, S. Fischer, J. Widmer, "A Generic Proxy System for Networked Computer Games," Proc. 1st workshop on Network and system support for games (NetGames'02), pp. 25-28. ACM, New York, 2002.

[MH02] G. Mao, D. Habibi, "Loss Performance Analysis for Heterogeneous ON-OFF Sources with Application to Connection Admission Control," IEEE/ACM Transactions on Networking, February 2002.

[MHKE01] M. Mauve, V. Hilt, C. Kuhmünch, W. Effelsberg, "RTP/I-Toward a Common Application Level Protocol for Distributed Interactive Media," IEEE Transactions on Multimedia, pp. 152-161, 2001.

[MMBK10] D. Mills, J. Martin, J. Burbank, W. Kasch, RFC 5905, "Network Time Protocol Version 4: Protocol and Algorithms Specification," Jun. 2010.

[MSM05] J. Van Meggelen, J. Smith, L. Madsen, "Asterisk, the future of telephony," 2ª edición. Capítulo 12. O'Reilly 2005.

[Nas05] Cooperative Association for Internet Data Analysis, "NASA Ames Internet Exchange Packet Length Distributions".

[Nic98] K. Nichols et al, RFC 2474, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," 1998.

[OH03] M. Oliveira, T. Henderson, "What online gamers really think of the Internet?," Proc. 2nd workshop on Network and system support for games (NetGames '03). ACM, New York, NY, USA, 185-193, 2003.

[Ope11] www.opensips.org, última visita 8/8/2011.

[PAF01] R. Pazhyannur, I. Ali, C. Fox, RFC 3153, "PPP multiplexing," ago. 2001.

[Per03] C. Perkins, "Rtp: Audio and Video for the Internet," Addison-Wesley Professional. 2003.

[Pin10] The PingER Project, http://www-iepm.slac.stanford.edu /pinger, última visita 7/9/2011.

[PJS11] www.pjsip.org, última visita 8/8/2011.

[PS08] G. Pelletier, K. Sandlund, RFC 5225, "RObust Header Compression Version 2 (ROHCv2)," 2008.

[QNC06] B. Quetier, V. Neri, F. Cappello, "Selecting A Virtualization System For Grid/P2P Large Scale Emulation," Proc. Workshop on Experimental Grid testbeds for the assessment of large-scale distributed applications and tools (EXPGRID'06), París, Francia, 2006.

[RHS10] S. Ratti, B. Hariri, S. Shirmohammadi, "A Survey of First-Person Shooter Gaming Traffic on the Internet," IEEE Internet Computing, pp. 60-69, sep./oct. 2010.

[RSCP+02] J. Rosenberg, H. Schulzrinne, G. Camarillo, J. Peterson, A. Johnston, E. Schooler, RFC 3261, "SIP: Session Initiation Protocol," jun. 2002.

[RSR08] M. Ries, P. Svoboda, M. Rupp, "Empirical study of subjective quality for Massive Multiplayer Games," Systems, Signals and Image Processing, 2008. IWSSIP 2008. 15th International Conference on, pp.181-184, 25-28, jun. 2008.

[SB06] L. Stewart, P. Branch: HLCS, Map: dedust, 5 players, 13Jan2006. Centre for Advanced Internet Architectures SONG Database, http://caia.swin.edu.au/sitcrc/hlcs_130106_1_dedust_5_fragment.tar.gz

[SCFV03] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, RFC 3550, "RTP: A transport protocol for real-time applications," jul. 2003.

[SCV07] R. Solange, P. Carvalho, V. Freitas, "Admission Control in Multiservice IP Networks: Architectural Issues and Trends," IEEE Communications Magazine, vol.45, no. 4, pp. 114-121, 2007.

[SERZ02] C. Schaefer, T. Enderes, H. Ritter, M. Zitterbart. "Subjective quality assessment for multiplayer real-time games," Proc. 1st workshop on Network and system support for games (NetGames '02). ACM, New York, NY, USA, 74-78. 2002.

[Sim94] W. Simpson, RFC 1661, "The Point-to-Point Protocol (PPP)," jul. 1994.

[SKRM+02] P. Srisuresh, J. Kuthan, J. Rosenberg, A. Molitor, A. Rayhan, RFC 3303, "Middlebox Communication Architecture and Framework," ago. 2002.

[SKR07] P. Svoboda, W Karner, M. Rupp, "Traffic Analysis and Modeling for World of Warcraft," ICC '07. IEEE International Conference on Communications, pp.1612-1617, 24-28, jun. 2007.

[SLLY02] H.P. Sze, S. C. Liew, J.Y.B. Lee, D.C.S.Yip, "A Multiplexing Scheme for H.323 Voice-Over-IP Applications," IEEE J. Select. Areas Commun, Vol. 20, pp. 1360-1368, sep. 2002.

[SPGW08] J. Sommers, P. Barford, A. Greenberg, W. Willinger, "An SLA perspective on the router buffer sizing problem," SIGMETRICS Perform. Eval. Rev. 35, 4, pp 40-51, mar. 2008.

[SS98] B. Subbiah, S. Sengodan. draft-ietf-avt-mux-rtp-00.txt, "User Multiplexing in RTP payload between IP Telephony Gateways," ago. 1998.

[SSS06] S. Sacker, M. Santaiti, C. Spence, "The Business Case for Enterprise VoIP," Intel Corporation, 2006.

[TA02] A. Trad, H. Afifi, "Adaptive Multiplexing Scheme for Voice Flow Transmission Across Best-Effort IP Networks," INRIA Research Report 4929, sep. 2003.

[TKW05] B. Thompson, T. Koren, D. Wing. RFC 4170, "Tunneling Muliplexed Compressed RTP (TCRTP)," Nov. 2005.

[Tur06] J. S. Turner, "A proposed architecture for the GENI backbone platform," Proc. Arch. for Network and Comm. Systems, 2006.

[TVR+99] W. Townsley, A. Valencia, A. Rubens, G. Pall, G. Zorn, B. Palter, RFC 2661: "Layer Two Tunneling Protocol: l2tp," Ago. 1999.

[Ubi05] Ubicom White Paper, "OPScore, or Online Playability Score: A Metric for Playability of Online Games with Network Impairments", 2005, http://www.kevingee.biz/wp-content/uploads/2011/04/IP3K-DWP-OPSCORE-10.pdf, última visita 7/9/2011.

[VS94] C. Villamizar, C. Song, "High performance TCP in ANSNET," ACM Computer Communication Review, oct. 1994.

[VS08] A. Vishwanath, V. Sivaraman, "Routers with Very Small Buffers: Anomalous Loss Performance for Mixed Real-Time and TCP Traffic," IEEE IWQoS, Netherlands, 2008.

[VSR09] A. Vishwanath, V. Sivaraman, G. N. Rouskas, "Considerations for Sizing Buffers in Optical Packet Switched Networks," IEEE INFOCOM, Brazil, 2009.

[VST09] A. Vishwanath, V. Sivaraman, M. Thottan, "Perspectives on router buffer sizing: recent results and open problems,"

SIGCOMM Comput. Commun. Rev. 39, 2, Mar. 2009, pp 34-39.

[VV08] K. V. Vishwanath, A. Vahdat "Evaluating distributed systems: Does background traffic matter?," Proc. USENIX Annu. Tech. Conf., p.227, 2008.

[Wan06] S. Y. Wang, "Using the innovative NCTUns 3.0 network simulator and emulator to facilitate network researches," Testbeds and Research Infrastructures for the Development of Networks and Communities, TRIDENTCOM, pp.4-8, Barcelona, 2006.

[WCL07] D. X. Wei, P. Cao, S. H. Low, "Packet Loss Burstiness: Measurements and Implications for Distributed Applications," Parallel and Distributed Processing Symposium, International, p. 222, 2007.

[WKKG06] H. Wei, K. Kim, A. Kashyap, S. Ganguly, "On Admission of VoIP Calls Over Wireless Mesh Network," IEEE International Conference on Communications, 2006. ICC '06. vol.5, no., pp.1990-1995, jun. 2006.

[WKVA06] A. F. Wattimena, R. E. Kooij, J. M. van Vugt, O. K. Ahmed, "Predicting the perceived quality of a first person shooter: the Quake IV G-model," Proc. 5th ACM SIGCOMM workshop Network and system support for games (NetGames '06), ACM, New York, NY, USA, 2006.

[WLSR+02] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, Ch. Barb, A. Joglekar, "An integrated experimental environment for distributed systems and networks," Proc. 5th symposium on Operating systems design and implementations, Boston, 2002.

[WMXZ06] S. Wang, Z. Mai, D. Xuan, W. Zhao, "Design and implementation of QoS-provisioning system for voice over IP," IEEE Transactions on Parallel and Distributed Systems, vol.17, no.3, pp. 276-288, mar. 2006.

[WSKW07] S. Wanke, M. Scharf, S. Kiesel, S. Wahl, "Measurement of the SIP Parsing Performance in the SIP Express Router," Proc. 13th Open Eur. Summer School and IFIP TC6.6Workshop (EUNICE 07) Enschede, Netherlands, 2007.

[YA07] J. Yu, I. Al-Ajarmeh. "Call Admission Control and Traffic Engineering of VoIP," Proc. Second Intenational Conference on Digital Telecommunications, IEEE ICDT 2007.

[ZA04] S. Zander, G. Armitage, "Empirically Measuring the QoS Sensitivity of Interactive Online Game Players," Australian

Telecommunications Networks & Applications Conference 2004 (ATNAC2004), Sydney, Australia, dic. 2004.

[ZA05] S. Zander, G. Armitage, "A traffic model for the Xbox game Halo 2," Proc. Int. Workshop on Network and OS support for digital audio and video (NOSSDAV '05), ACM, New York, NY, USA, pp 13-18, 2005.

[Zav08] P. Zave, "Understanding SIP through Model-Checking," Principles, Systems and Applications of IP Telecommunications. Services and Security for Next Generation Networks: Second International Conference, IPTComm 2008, Heidelberg, Germany, jul. 2008. pp 256 – 279.

[ZJB06] J. Zhou, Z. Ji, R. Bagrodia, "TWINE: A Hybrid Emulation Testbed for Wireless Networks and Applications," Proc. IEEE INFOCOM 2006.

[ZN02] P. Zheng, L. M. Ni, "EMWin: emulating a mobile wireless network using a wired network," Proc. 5th ACM international Workshop on Wireless Mobile Multimedia, Atlanta, 2002.

3GPP	3rd Generation Partnership Project
AAA	Authentication, Authorization and Accounting
ADSL	Asymmetric Digital Subscriber Line
ATM	Asynchronous Transfer Mode
CAC	Call Admission Control
CAIA	Centre for Advanced Internet Architectures
CPU	Central Processing Unit
CRTP	Compressed RTP
DiffServ	Differentiated Services
D-ITG	Distributed Internet Traffic Generator
ECRTP	Enhanced Compressed RTP
FIFO	First Input First Output
FIRE	Future Internet Research and Experimentation
FPS	First Person Shooter
FTP	File Transfer Protocol
GENI	Global Environment for Network Innovations
GeRM	Generic RTP Multiplexing
GNU-GPL	GNU General Public License
GPS	Global Positioning System
HTTP	HyperText Transfer Protocol
IAX	Inter-Asterisk eXchange protocol
ICE	Interactive Connectivity Establishment
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IMS	IP Multimedia Subsystem

IntServ	Integrated Services
IP	Internet Protocol
IPDV	Instantaneous Packet Delay Variation
IPHC	IP Header Compression
ITU	International Telecommunication Union
JTG	Jugi's Traffic Generator
L2TP	Layer 2 Tunneling Protocol
MAC	Medium Access Control
MBAC	Measurement Based Admission Control
MGCP	Media Gateway Control Protocol
MMORPG	Massive Multiplayer Online Role Playing Game
MOS	Mean Opinion Score
MTU	Maximum Transfer Unit
NAT	Network Address Translation
NCP	Network Control Program
NCTUns	National Chiao Tung University Network Simulator
NTP	Network Time Protocol
OWD	One Way Delay
P2P	Peer to Peer
PBX	Private Branch eXchange
PHB	Per-Hop Behaviour
PPP	Point-to-point Protocol
PPPMux	PPP Multiplexing
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
RFC	Request For Comments
ROHCv2	RObust Header Compression version 2

RSVP	Resource ReSerVation Protocol
RTC	Red Telefónica Conmutada
RTCP	Real-Time Transport Control Protocol
RTP	Real-Time Transport Protocol
RTS	Real Time Strategy
RTT	Round Trip Time
SATA	Serial Advanced Technology Attachment
SCTP	Stream Control Transmission Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SMTP	Simple Mail Transfer Protocol
SRT	System Response Time
STUN	Session Traversal Utilities for NAT
TCM	Tunelado, Compresión, Multiplexión
TCP	Transmission Control Protocol
TCRTP	Tunneled multiplexed Compressed RTP
TISPAN	Telecommunications and Internet converged Services and Protocols for Advanced Networking
TLS	Transport Layer Security
TOS	Type of Service
UDP	User Datagram Protocol
UML	User Mode Linux
vBET	Virtual Machine-based emulation testbed
VJHC	Van Jacobson Header Compression
VMM	Virtual Machine Monitor
VoIP	Voice over Internet Protocol
WiMAX	Worldwide Interoperability for Microwave Access

PUBLICACIONES REALIZADAS DURANTE LA TESIS

Se citan, por orden de aparición en el texto, los principales artículos publicados durante el desarrollo de esta tesis.

José Mª Saldaña, Eduardo Viruete, Julián Fernández-Navajas, José Ruiz-Mas, José I. Aznar, **"Hybrid Testbed for Network Scenarios,"** en **SIMUTools 2010**, the Third International Conference on Simulation Tools and Techniques. Torremolinos, Málaga (Spain). Mar. 2010.

José Mª Saldaña, Jenifer Murillo, Julián Fernández-Navajas, José Ruiz-Mas, Eduardo Viruete y José I. Aznar, **"Distributed IP Telephony System with Call Admission Control,"** Proc. Conference on ENTERprise Information Systems **CENTERIS 2010**. Viana do Castelo, Portugal. Oct. 2010.

José Mª Saldaña, Jenifer Murillo, Julián Fernández-Navajas, José Ruiz-Mas y Eduardo Viruete, **"Implementación de un CAC basado en medidas de QoS para sistemas de telefonía IP,"** Actas de las VIII Jornadas de Ingeniería Telemática (JITEL 2009).

José Mª Saldaña, José I. Aznar, Eduardo Viruete, Julián Fernández-Navajas y José Ruiz-Mas, **"QoS Measurement-Based CAC for an IP Telephony system," Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering** 22, Springer 2009.

José Mª Saldaña, Jenifer Murillo, Julián Fernández-Navajas, José Ruiz-Mas, Eduardo Viruete y José I. Aznar, **"QoS and Admission Probability Study for a SIP-based Central Managed IP Telephony System,"** Proc. 5th International Conference on New Technologies, Mobility and Security, **NTMS 2011**, Paris. Feb. 2011.

José Mª Saldaña, Jenifer Murillo, Julián Fernández-Navajas, José Ruiz-Mas, Eduardo Viruete y José I. Aznar, **"Evaluation of Multiplexing and Buffer Policies Influence on VoIP Conversation Quality,"** Proc. **CCNC 2011** - 3rd IEEE International Workshop on Digital Entertainment, Networked Virtual Environments, and Creative Technology, pp 1147-1151, Las Vegas. Ene. 2011.

José Mª Saldaña, Jenifer Murillo, Julián Fernández-Navajas, José Ruiz-Mas, Eduardo Viruete y José I. Aznar, **"Influence of the Distribution of TCRTP Multiplexed Flows on VoIP Conversation Quality,"** Proc. **CCNC 2011**. The 8th Annual IEEE Consumer Communications and Networking Conference - Work in Progress papers, pp 711-712, Las Vegas. Ene. 2011.

Jenifer Murillo, José Mª Saldaña, Julián Fernández-Navajas, José Ruiz-Mas, Eduardo Viruete y José I. Aznar, **"Improving Quality in a Distributed IP Telephony System by the use of Multiplexing Techniques,"** International Symposium on Performance Evaluation of Computer and Telecommunication Systems **SPECTS 2011**, La Haya, Holanda, Jun. 2011.

José Mª Saldaña, Julián Fernández-Navajas, José Ruiz-Mas, José I. Aznar, Eduardo Viruete y Luis Casadesus, **"First Person Shooters: Can a Smarter Network Save Bandwidth without Annoying the Players?," IEEE Communications Magazine**, Consumer Communications and Networking Series, vol. 49, no. 11, pp. 190-198, Nov. 2011.

José Mª Saldaña, Julián Fernández-Navajas, José Ruiz-Mas, José I. Aznar, Luis Casadesus y Eduardo Viruete, **"Comparative of Multiplexing Policies for Online Gaming in terms of QoS Parameters," IEEE Communications Letters**, vol. 15, no. 10, pp. 1132-1135, Oct. 2011.

José Mª Saldaña, Jenifer Murillo, Julián Fernández-Navajas, José Ruiz-Mas, José I. Aznar y Eduardo Viruete, **"Bandwidth Efficiency Improvement for Online Games by the use of Tunneling, Compressing and Multiplexing Techniques,"** Proc. International Symposium on Performance Evaluation of Computer and Telecommunication Systems **SPECTS 2011**, La Haya, Holanda, Jun. 2011.

José Mª Saldaña, Julián Fernández-Navajas, José Ruiz-Mas, José I. Aznar, Eduardo Viruete y Luis Casadesus, **"Influence of the Router Buffer on Online Games Traffic Multiplexing,"** Proc. International Symposium on Performance Evaluation of Computer and Telecommunication Systems **SPECTS 2011**, La Haya, Holanda, Jun. 2011.