Ahorro de Ancho de Banda en Juegos Online Mediante el uso de Técnicas de Tunelado, Compresión y Multiplexión

José Mª Saldaña, Julián Fernández-Navajas, José Ruiz-Mas, José I. Aznar,
Eduardo Viruete Navarro, Luis Casadesus
Grupo de Tecnologías de las Comunicaciones – Instituto de Investigación en Ingeniería de Aragón
Dpt. IEC. Centro Politécnico Superior Universidad de Zaragoza
Edif. Ada Byron, 50018, Zaragoza
{jsaldana, navajas, jruiz, jiaznar, eviruete, luis.casadesus}@unizar.es

***Resumen*- Las empresas desarrolladoras de juegos *online* necesitan recursos de *hardware* y ancho de banda para dar un buen servicio a los usuarios. Estos juegos producen altas tasas de paquetes UDP de pequeño tamaño desde el cliente al servidor, teniendo una baja eficiencia. Las acciones de los jugadores se tienen que propagar al servidor y al resto de jugadores en muy poco tiempo, por lo que los retardos de red son muy críticos. Este trabajo presenta un método que ahorra ancho de banda mediante un agente local que comprime las cabeceras y utiliza un túnel para enviar varios paquetes dentro de uno multiplexado. Se ha estudiado el comportamiento del sistema para IPv4 e IPv6, mostrando que el ahorro de ancho de banda es significativo. Como contrapartida, se añade un retardo que tiene una cota superior modificable. Si el número de jugadores es suficiente, este retardo no empeora la experiencia del usuario.**

***Palabras Clave* - juegos online, retardo, multiplexión, compresión, First Person Shooter**

I. INTRODUCCIÓN

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 título: necesitan recursos *hardware* y ancho de banda para evitar que su infraestructura se sature. Dado que el éxito de un nuevo título no es muy predecible, en ocasiones deben sobredimensionar los recursos para dar un buen servicio a los usuarios. En [1] 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.

Dos de los géneros más populares son los MMORPG (Juegos Masivos de Rol Multijugador Online, *Massive Multiplayer Online Role Playing Game*) y los FPS (Tirador en Primera Persona, *First Person Shooter*). En [2] 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.

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. Por eso el *overhead* causado por las cabeceras IP y UDP es significativo. Los paquetes del servidor al cliente son habitualmente más grandes.

Existen escenarios en los que muchos jugadores comparten la ruta desde la red de acceso hasta el servidor del juego: por ejemplo, los *cibercafés* (Fig. 1a y 1b), muy populares en algunos países, disponen frecuentemente de ordenadores capaces de ejecutar juegos *online* y grupos de jugadores suelen acudir a estos establecimientos. El tráfico entre los servidores de un mismo juego es otro escenario donde se comparte la misma ruta (Fig. 1c).

El tráfico de estos grupos 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*. Mediante el uso de un agente local 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.

Se han presentado varias propuestas [3], [4] para ahorrar carga de trabajo al servidor central del juego, incluyendo algunos elementos *(proxies)* cercanos a la red de acceso. En contraste, el agente local que se propone en este trabajo se podría distribuir con el juego, igual que la aplicación servidor se distribuye con algunos títulos. El túnel desde los clientes hasta el servidor se podría crear bien desde una máquina dedicada (Fig. 1a), o bien desde la máquina de uno de los jugadores (Fig. 1b), evitando así el coste de un nuevo equipo.

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, como veremos [5]). Esto no conlleva 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 paquetes por segundo sería beneficioso para el servidor.

Si el número de jugadores es lo suficientemente grande, se puede esperar que, añadiendo pequeños retardos, un gran

número de paquetes se pueda multiplexar en otro más grande. El ahorro de ancho de banda no sólo afectará al tráfico del juego, sino que también podrá ser beneficioso para el tráfico que comparte el acceso con él. Es más, la multiplexión tiene la ventaja de reducir el número de paquetes por segundo que el *router* debe gestionar.

Existen otras aplicaciones y escenarios en los que varios flujos de información en tiempo real comparten la misma ruta, como por ejemplo el denominado *trunking* usado en aplicaciones de voz sobre IP (VoIP), donde las técnicas de multiplexado y compresión llevan tiempo usándose e incluso se han estandarizado [6]. Muchos juegos presentan patrones de tráfico similares, generando altas tasas de paquetes pequeños y presentando por eso un gran *overhead*. La novedad del presente trabajo es la aplicación de las técnicas usadas para VoIP al caso de juegos *online*, donde también se pueden conseguir ahorros significativos de ancho de banda sin perder calidad.

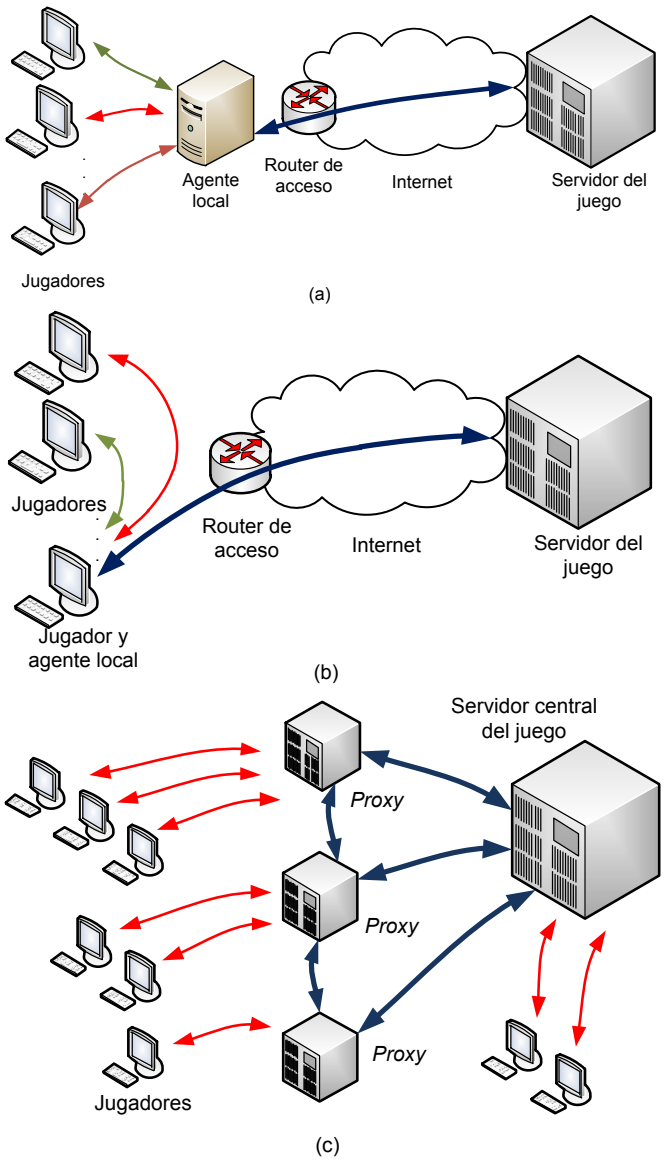


Fig. 1. Escenarios donde varios jugadores comparten la misma ruta: a) jugadores que comparten la red de acceso, usando un agente local. b) jugadores usando la máquina de otro para crear el túnel. c) Tráfico entre *proxies* del mismo juego. Las líneas gruesas representan el tráfico de varios jugadores.

En este trabajo nos centraremos en los escenarios que se presentan en las Fig. 1a y 1b, pero muchas de las conclusiones se pueden aplicar también al escenario de la Fig. 1c. Trataremos de optimizar el tráfico del cliente al servidor, ya que es donde se puede conseguir un mayor ahorro y, en muchos escenarios, como en un acceso ADSL, usa el enlace más restrictivo. Tendremos que medir también 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.

Aunque esta técnica también se puede aplicar a otros géneros, en este trabajo 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 [7]. 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.

El resto del trabajo se ha organizado de la siguiente manera: la siguiente sección presenta los trabajos relacionados. La sección III explica el método de tunelado, compresión y multiplexión propuesto. Los resultados se detallan en la sección IV y, finalmente se exponen las conclusiones.

II. Trabajos Relacionados

A. Tráfico de los juegos online

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 es el generado 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 [8] se presentó un método para extrapolar el tráfico del servidor al cliente, obtenido a partir de medidas empíricas. Obtuvieron las distribuciones para un juego de N jugadores a partir del de 2 o 3. En ese trabajo también se decía que el tráfico del cliente al servidor es independiente del número de jugadores.

En [9] 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. También en [10] se analizaron otros juegos en términos de tamaño de paquete y tiempo entre paquetes. En [11] se presentó un resumen de diferentes modelos de tráfico que existen en la literatura para 17 juegos muy populares. Todos estos estudios muestran que estos juegos generan altas tasas de paquetes pequeños. Esto produce una eficiencia muy pobre, por lo que se pueden conseguir ahorros de ancho de banda si se comprimen las cabeceras y se multiplexan paquetes.

En [9] 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.

B. Infraestructura para soporte de juegos online

El problema de la infraestructura necesaria para dar soporte a estos juegos se ha tratado también en diversos trabajos. Desde el punto de vista del usuario, en [12] 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.

Desde el punto de vista del servidor, existen dos arquitecturas para soportar 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 [13] no se necesita un servidor central, ya que los jugadores se intercambian directamente la información. Pero esta arquitectura no suele usarse en juegos comerciales.

El problema de la escalabilidad de la infraestructura para soportar estos juegos ha sido estudiado por Mauve y otros [3], y propusieron el uso de *proxies* para conseguir control de congestión, robustez, reducción de los retardos y evitar las trampas de algunos jugadores. Algunos *proxies* podrían situarse cerca de los jugadores, evitando trabajo al servidor central. Así, en la Ref. [4] también se propuso el uso de *booster-boxes,* 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. Como se ha dicho en la introducción, la solución propuesta en el presente trabajo podría incluso correr en la máquina de uno de los jugadores.

C. Algoritmos de compresión

Existen algunos estándares del IETF para compresión de cabeceras, que se desarrollaron hace varios años: en primer lugar, VJHC [14] presentó un método para comprimir las cabeceras IP/TCP. Algunos años después se presentó IPHC [5], capaz de comprimir también las cabeceras UDP e IPv6. En ese mismo momento se publicó CRTP para comprimir las cabeceras IP/UDP/RTP. Varios años después se mejoró y se denominó ECRTP. Pero estos dos protocolos no son adecuados para comprimir el tráfico de juegos, ya que éste no es RTP. También hubo una propuesta interesante [15] de usar un protocolo similar a RTP para juegos *online*. Una ventaja de esta propuesta es 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.

Estos algoritmos comprimen la cabecera sólo de nodo a nodo, utilizando la redundancia de los campos de las cabeceras IP, UDP y TCP para evitar enviar algunos de ellos. 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 en las primeras cabeceras. Los *aleatorios* no se comprimen. Los clasificados como *delta* 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.

ROHCv2 [16] es un estándar más reciente, que puede comprimir las cabeceras IP/UDP/RTP, y también las IP/UDP. 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 [17]. El uso de estas técnicas avanzadas hace más difícil la implementación [18], añadiendo más retardo de procesado.

III. MÉTODO DE COMPRESIÓN Y MULTIPLEXADO

En esta sección explicaremos el método propuesto para comprimir y multiplexar. Se presentará también un estudio de la eficiencia en ancho de banda que se puede conseguir.

A. Algoritmo de Tunelado, Compresión y Multiplexión

En el RFC 4170 [6], el IETF aprobó TCRTP (*Tunneled Compressed* RTP) 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.

En este trabajo se ha usado un esquema similar, pero en nuestro caso el tráfico no es RTP, por lo que sólo podemos comprimir las cabeceras IP/UDP usando IPHC o ROHCv2. Denominaremos a este método TCM (Tunelado, Compresión, Multiplexión). La Fig. 2 muestra la pila de protocolos y la estructura de un paquete TCM. Se puede dividir en las siguientes partes:

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

B. 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 1500 bytes, cosa cierta para los tráficos usados.

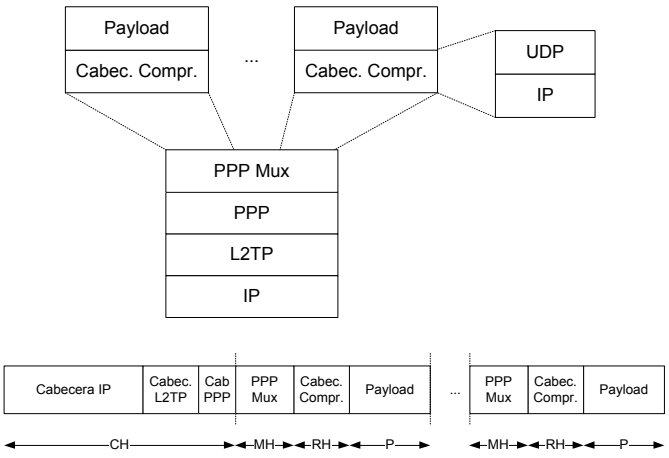


Fig. 2. Pila de protocolos de TCM y esquema de un paquete multiplexado

Como se ha dicho en la introducción, el retardo tiene una gran importancia en este servicio. Por eso, hemos usado una política de multiplexión que mantiene el retardo por debajo de una cota superior, denominada *T,* de manera que un paquete multiplexado se envía al final de cada periodo, incluyendo a todos los que ya hayan llegado (Fig. 3). Hay dos excepciones: si no ha llegado ningún paquete, no se envía nada; y si hay un solo paquete, se enviará en su forma nativa, ya que el túnel lo haría más grande.

Nos referiremos a los paquetes generados por la aplicación como *nativos*, en contraste con los multiplexados (*mux*). Con esta política, el retardo añadido será en media *T/2*, y su cota superior será *T*.

Un parámetro interesante es la relación de anchos de banda (*Bandwidth Relationship, BWR)*, entre las situaciones de usar o no multiplexión. Denominaremos *k* al número de paquetes que llegan en un periodo. *NH* se refiere al tamaño de una cabecera normal IP/UDP. Para obtener *BWR*, calcularemos primero el número de bytes enviado en un periodo cuando se usa multiplexión, distinguiendo los casos de uno o varios paquetes:

$$bytes_{mux} = Pr\,(k=1)(\,NH + E\,[P]) + \\ + Pr(\,k>1)\,[CH+E[k|k>1](MH+E[RH]+E[P])] \quad (1)$$

Y en el caso de los paquetes nativos será:

$$bytes_{nativo} = E[k]\ (NH + E\,[P]) \quad (2)$$

Dividiendo (1) entre (2) obtenemos *BWR*:

$$BWR = \frac{\Pr(k=1)}{E[k]} + \Pr(k>1)\frac{CH}{E[k](NH+E[P])} + \\ + \Pr(k>1)\frac{E[k \mid k>1]}{E[k]}\frac{MH+E[RH]+E[P]}{NH+E[P]} \quad (3)$$

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.

Por tanto, si tenemos un gran número de usuarios y un periodo grande, el número de paquetes multiplexados será también grande, y 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|k>1]/E[k]*. De esta manera, podemos obtener una expresión para la asíntota de *BWR:*

$$BWR_a = \frac{MH+E[RH]+E[P]}{NH+E[P]} \quad (4)$$

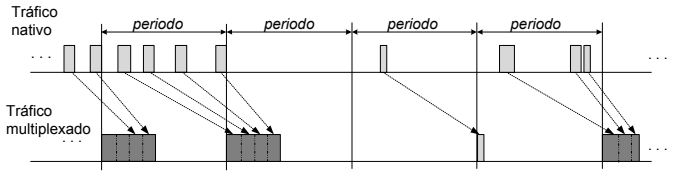


Fig. 3. Comportamiento de la política de multiplexión

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 hacen 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 retardos menores. El incremento de *T* será beneficioso para *BWR*, pero no lo podemos aumentar indefinidamente, puesto que los jugadores son muy sensibles al retardo.

Por claridad, no se va a incluir aquí el cálculo de *E[k|k>1], Pr(k=1)* y *Pr(k>1)*, que se puede encontrar en el Apéndice. Como se verá, estos valores varían según el comportamiento de cada juego.

Para poder obtener resultados numéricos y gráficos, utilizaremos parámetros reales de algunos juegos comerciales, y también los de los protocolos usados:

- *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 por segundo generados por los *N* jugadores.
- *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, 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 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. Se han obtenido para IPv4 e IPv6. Hemos seleccionado algunos juegos populares, y los valores concretos se han obtenido de [10] y [11]. Los valores para Halo2 se refieren a una consola con un solo usuario [19]. El valor de λ (paquetes por segundo) para Quake 3 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.

Para hacernos una mejor idea de los beneficios del sistema, presentaremos ahora algunas gráficas que ilustren el comportamiento del *BWR* no sólo en la asíntota, sino con diferentes valores de número de usuarios y periodo. Dado que tenemos que representar las gráficas para un juego concreto, hemos seleccionado *Half Life Counter Strike 1* a causa de su popularidad y de la disponibilidad de muchos estudios de su comportamiento [9], [20]. La Fig. 4a

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	GoldSrc	41.09	24.65	68%	53%
Halo 2	Halo2	43.2	25	69%	54%

Tabla 1. Valores de la asíntota de *BWR* para diferentes juegos

representa el *BWR* para IPv4 en función del número de jugadores y el periodo. Si dejamos fijo el número de jugadores, obtenemos la Fig. 4b y fijando el periodo, la Fig. 4c. 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. 4b, 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 *T*. 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 *T*. Si la red es rápida, podremos permitirnos añadir un retardo mayor para así mejorar el ahorro de ancho de banda.

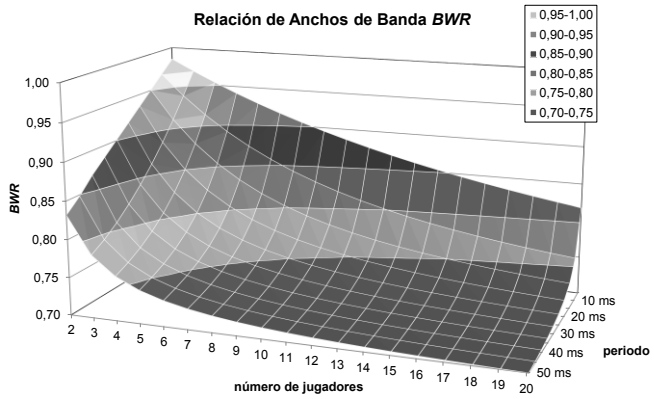
Podemos hacer una última observación: como se ha dicho en las secciones previas, una limitación de los *router* comerciales es el número de paquetes por segundo que pueden gestionar. El método presentado también reduce esta cantidad por un factor de *E[k]*.

C. *Retardos en el sistema*

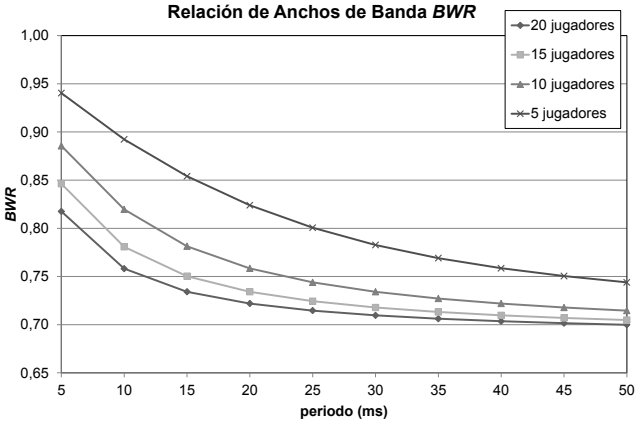
En este apartado se estudia el impacto del método propuesto en el *SRT*. La Fig. 5 muestra un esquema del sistema, con los retardos que se suman para obtener el valor de *SRT*.

- $T_{retención}$ es el tiempo que un paquete pasa en la cola del multiplexor.
- $T_{proceso}$ representa el tiempo que el paquete pasa en el multiplexor y demultiplexor. En [21] se implementó un multiplexor para tráfico RTP, y el tiempo de procesado era menor de 1 ms.
- $T_{encolado}$ es el tiempo de espera en la cola del *router* de acceso. El método presentado no lo modifica directamente, aunque al multiplexar cambiará el tráfico ofrecido a la cola.
- T_{red} es el retardo de red, que tampoco se ve afectado.

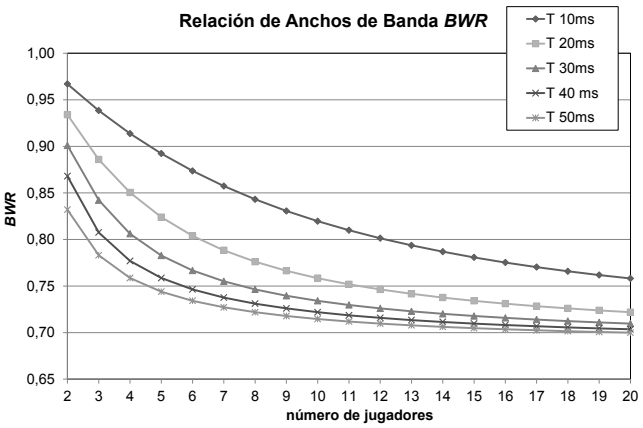
Por tanto, el único retardo significativo que se añade es $T_{retención}$, que es *T/2* en media. En [7] se concluye que el retardo tolerado para Quake 3 está entre 150 y 180 ms, mientras que para Counter Strike está por encima de 200 ms. Por tanto, el tiempo añadido al multiplexar puede ser asumido sin problemas para el usuario.



(a)



(b)



(c)

Fig. 4 *BWR* en función de a) número de jugadores y periodo. b) periodo. c) número de jugadores.

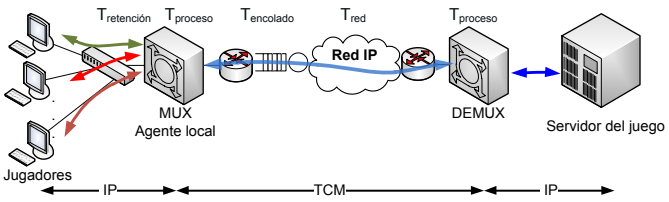


Fig. 5. Retardos del sistema

En este trabajo no se ha considerado la posibilidad de modificar la aplicación pero, si se pudiese hacer, se podría incluir una primera fase de sincronización para hacer que todas las máquinas de la misma partida generasen los paquetes en el mismo instante. De esta manera, este retardo se podría reducir significativamente para los juegos que usan un tiempo entre paquetes fijo.

IV. Pruebas y Resultados

En esta sección presentamos algunos resultados de simulación, obtenidos con trazas reales de un juego FPS. En primer lugar describiremos el método usado para generar los tráficos usados en las pruebas.

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 [22]). 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 propiedad del tráfico del cliente al servidor, cuya distribución es independiente del número de jugadores [8], [20]: 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. 6. 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 periodo T se obtienen los tamaños e instantes de los paquetes multiplexados.

El juego estudiado tiene tres diferentes comportamientos posibles, dependiendo del método de *render* [11], [20]. En nuestro caso, las trazas se han obtenido con OpenGL, que es el método más usado, y tiene dos posibles valores para el tiempo entre paquetes: 33 y 50 ms, cada uno con una probabilidad del 50%. Esto hace que el valor de λ sea 24 paquetes por segundo. El análisis de las probabilidades para este método se ha incluido en el Apartado B del Apéndice.

La Fig. 7 compara los valores teóricos de *BWR* con los obtenidos en las simulaciones. Se puede comprobar que los valores son muy similares, excepto por pequeñas diferencias para pocos jugadores y valores pequeños del periodo. La causa de estas diferencias es que el tiempo entre paquetes no es exactamente el esperado, sino que hay pequeñas variaciones en torno a 33 y 50 ms, como se puede ver en el histograma (Fig. 8).

La Fig. 9 presenta el número de paquetes por segundo que el *router* tiene que gestionar. Como se ha explicado en la sección II.A, la reducción de este parámetro también resulta interesante. Se observa que a mayor valor del periodo, más se reduce la cantidad de paquetes por segundo, que tiende a ser el inverso del periodo, independientemente del número de jugadores. La Fig. 10 presenta el tamaño medio de los paquetes multiplexados, que crece linealmente con el periodo.

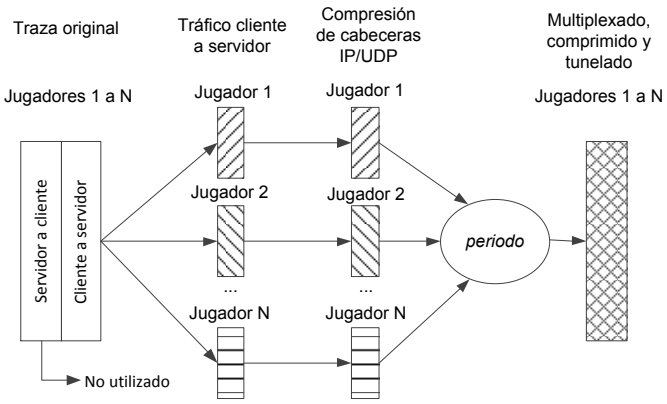


Fig. 6. Método utilizado para construir las trazas

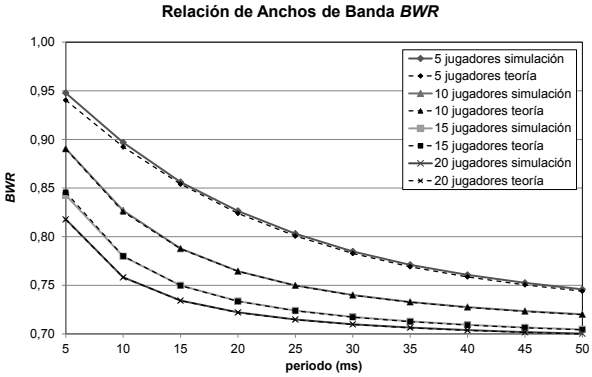


Fig. 7. Comparación de los resultados teóricos y de simulación para *BWR*

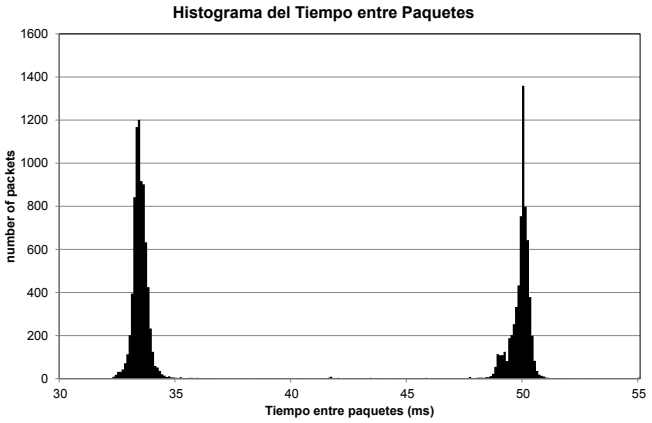


Fig. 8. Histograma del tiempo entre paquetes en ms

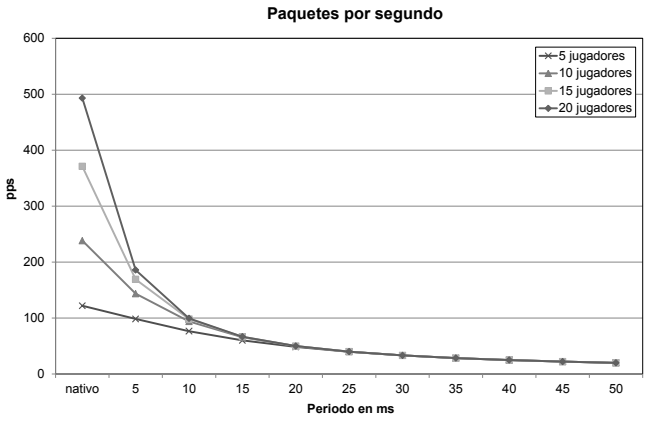


Fig. 9. Paquetes por segundo gestionados por el *router*

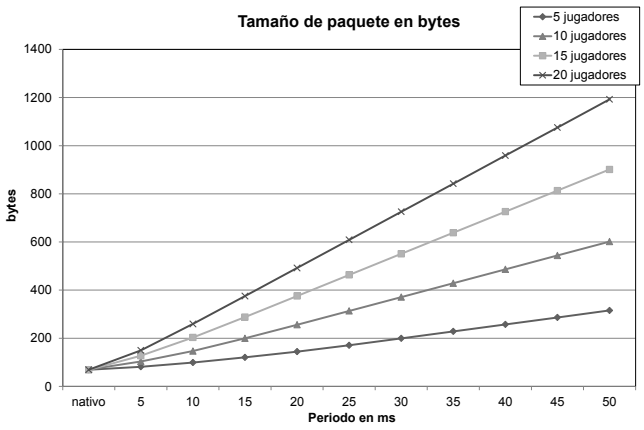


Fig. 10. Tamaño de los paquetes multiplexados

V. CONCLUSIONES

En este trabajo se ha presentado una técnica de tunelado, compresión y multiplexión, que se puede usar para ahorrar ancho de banda mediante la compresión de cabeceras y la unión de paquetes pequeños en otros más grandes. Puede reducir el *overhead* del tráfico de juegos *online,* ya que estas aplicaciones generan altas tasas de paquetes pequeños. El método utiliza un protocolo de compresión de las cabeceras IP/UDP, multiplexión con PPPMux y un túnel L2TP para poder trabajar extremo a extremo.

Las empresas que desarrollan juegos podrían estar interesadas en reducir el ancho de banda utilizado, y también el número de paquetes por segundo que tienen que gestionar. El ahorro de ancho de banda también puede proporcionar un mejor comportamiento en redes de acceso con ancho de banda limitado.

El método se ha probado con tráfico de juegos FPS, ya que estos programas tienen unos requerimientos temporales muy estrictos, pues los jugadores demandan una gran interactividad. Se han realizado simulaciones para estudiar el ahorro de ancho de banda, y los resultados muestran que se puede reducir un 38% para IPv4, y más de un 50% para IPv6. Los retardos añadidos se pueden mantener en niveles bajos si hay un número suficiente de jugadores que compartan la misma ruta. Como línea futura se considera el desarrollo de un algoritmo que ajuste dinámicamente los parámetros de la multiplexión según el número de jugadores y las estadísticas de la red, como el retardo y las pérdidas.

APÉNDICE

Se presentan aquí los cálculos necesarios para obtener una expresión analítica de *E[k|k>1],* que se requiere para construir las gráficas de *BWR* presentadas en la sección III. En primer lugar, podemos calcular *E[k]* como:

$$E[k] = Pr(k=0)\ E[k|k=0] + Pr(k=1)\ E[k|k=1] + \\ + Pr(k>1)\ E[k|k>1] \quad (5)$$

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

$$E[k \mid k>1] = \frac{E[k] - \Pr(k=1)}{\Pr(k>1)} \quad (6)$$

Por lo tanto, necesitamos obtener las expresiones de $Pr(k=0)$, $Pr(k=1)$ y $Pr(k>1)$. En los análisis previos hemos definido k como el número total de paquetes llegados al multiplexor en un periodo, es decir, la suma de los paquetes de cada jugador. Ahora definiremos l como el número de paquetes de un solo jugador. Si consideramos que las llegadas de paquetes de los N jugadores son independientes, tenemos $E[k]=N\,E[l]=N\,\lambda T$.

A. Tiempo entre paquetes constante

Las probabilidades que tenemos que calcular dependen de la distribución estadística del tiempo entre paquetes de cada juego. Consideraremos en primer lugar un tiempo entre paquetes constante, como ocurre en muchos juegos [11]. Denominaremos t al tiempo entre paquetes. Consideramos que se cumple $T < 2t$, para evitar grandes retardos. Por tanto, el valor máximo de l será 2, y entonces:

$$E[l] = \lambda T = Pr(l=1) + 2\,Pr(l=2) \quad (7)$$

En el caso de $T \le t$, tenemos que $Pr(l=2)=0$, por lo tanto si usamos (7) obtenemos:

$$Pr(l=1) = E[l] = \lambda T \quad (8)$$

$$Pr(l=0) = 1 - Pr(l=1) = 1 - \lambda T \quad (9)$$

Y en el caso de $T > t$, tenemos que $Pr(l=0)=0$, así que teniendo en cuenta que la suma de las probabilidades es la unidad y usando (7) de nuevo, obtenemos:

$$Pr(l=1) = 2 - E[k] = 2 - \lambda T \quad (10)$$

$$Pr(l=2) = 1 - Pr(l=1) = E[k] - 1 = \lambda T - 1 \quad (11)$$

Por tanto, podemos hallar $Pr(k=0)$:

$$Pr(k=0) = [Pr(l=0)]^N \quad (12)$$

Pero $Pr(k=1)$ es nula en el caso de tener más de un jugador y $T > t$, ya que cada jugador habrá enviado al menos un paquete en el periodo. Por tanto, la expresión para $Pr(k=1)$ con $T \le t$ será:

$$Pr(k=1)|_{T \le t} = \binom{N}{1} Pr(l=1)[Pr(l=0)]^{N-1} \quad (13)$$

B. *Dos posibles valores para el tiempo entre paquetes*

En este apartado se considera el caso de un juego que genera paquetes usando dos diferentes valores para el tiempo entre paquetes. Denominaremos t_1 al menor y t_2 al mayor, siendo p_1 y p_2 lar probabilidades respectivas de tener t_1 y t_2. Consideraremos, como ocurre en el juego considerado en este trabajo, que $T < 2t_1$, y $t_2 < 2t_1$. En este caso podemos obtener el valor de λ como:

$$\lambda = \frac{1}{p_1 t_1 + p_2 t_2} \tag{14}$$

Ahora distinguiremos dos casos diferentes. En primer lugar, si tenemos $T < t_1$, se trata del mismo caso que el visto en (8) y (9), ya que $Pr(l=2)=0$.

Y en el caso de $t_1 \leq T < t_2$, deberemos encontrar $Pr(l=0)$, es decir, la probabilidad de que no hayan llegado paquetes en el periodo T, que se corresponde con la probabilidad de que el periodo comience en los primeros t_2-T segundos de un tiempo entre paquetes de duración t_2. Por tanto, considerando que los tiempos entre paquetes consecutivos son independientes:

$$\Pr(l = 0) = p_2 \frac{t_2 - T}{p_1 t_1 + p_2 t_2} = p_2 \lambda (t_2 - T) \tag{15}$$

Y ahora, usando (7) y sabiendo que la suma de las probabilidades debe ser la unidad, podemos obtener:

$$Pr(l=1) = \lambda[T - 2p_1(T-t_1)] \tag{16}$$

$$Pr(l=2) = p_1 \lambda (T-t_1) \tag{17}$$

Finalmente, podemos usar los resultados de (12) y (13) para obtener las probabilidades de los distintos valores de k.

AGRADECIMIENTOS

Este trabajo ha sido financiado parcialmente por el Proyecto CPUFLIPI (MICINN TIN2010-17298), por el Proyecto MBACToIP, de la Agencia I+D del Gobierno de Aragón e Ibercaja Obra Social, y por el Proyecto NDCIPI-QQoE de la Cátedra Telefónica, de la Univ. de Zaragoza.

REFERENCIAS

[1] C. Chambers, W. Feng, S. Sahu, D. Saha: Measurement-based Characterization of a Collection of On-line Games. In Proceedings of the 5th ACM SIGCOM conference on Internet Measurement (IMC'05). USENIX Association, Berkeley (2005)

[2] K. Chen, P. Huang, C. Lei: Game traffic analysis: An MMORPG perspective. In Proceedings of the international workshop on Network and operating systems support for digital audio and video (NOSSDAV'05), pp. 19-24. ACM, New York (2005)

[3] M. Mauve, S. Fischer, J. Widmer: A Generic Proxy System for Networked Computer Games. In Proceedings of the 1st workshop on Network and system support for games (NetGames'02), pp. 25--28. ACM, New York (2002)

[4] D. Bauer, S. Rooney, P. Scotton: Network Infrastructure for Massively Distributed Games. In Proceedings of the 1st workshop on Network and system support for games (NetGames'02), pp. 36-43. ACM, New York (2002)

[5] M. Degermark, B. Nordgren, D. Pink: RFC 2507: IP Header Compression (1999)

[6] B. Thompson, T. Koren, D. Wing. RFC 4170: Tunneling Multiplexed Compressed RTP (TCRTP), Nov. 2005.

[7] 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.

[8] P. Branch, G. Armitage: Extrapolating Server To Client IP traffic From Empirical Measurements of First Person Shooter games. In Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games (NetGames '06). ACM, NY, USA.

[9] 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)

[10] W. Feng, F. Chang, W. Feng, J. Walpole: A Traffic Characterization of Popular On-Line Games. IEEE/ACM Trans. Netw., pp. 488-500 (2005)

[11] S. Ratti, B. Hariri, S. Shirmohammadi, A Survey of First-Person Shooter Gaming Traffic on the Internet, IEEE Internet Computing, pp. 60-69, Sept./Oct. (2010)

[12] K. Lee, B. Ko, S. Calo: Adaptive Server Selection for Large Scale Interactive Online Games. In Proc. 14th International Workshop on Network and operating systems support for digital audio and video (NOSSDAV'04), pp. 152--157. ACM, New York (2004)

[13] L. Gautier, C. Diot: Design and Evaluation of MiMaze, a Multi-player Game on the Internet. In Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMS'98), IEEE Computer Society, pp. 233. Washington (1998)

[14] V. Jacobson: RFC 1144: Compressing TCP/IP Headers for Low-Speed Serial Links (1990)

[15] M. Mauve, V. Hilt, C. Kuhmünch, W. Effelsberg: RTP/I-Toward a Common Application Level Protocol for Distributed Interactive Media. In Proceedings of IEEE Transactions on Multimedia, pp. 152-161 (2001)

[16] G. Pelletier, K. Sandlund, RFC 5225: RObust Header Compression Version 2 (ROHCv2) (2008)

[17] 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)

[18] 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)

[19] S. Zander, G. Armitage, A traffic model for the Xbox game Halo 2. In Proc. International Workshop on Network and operating systems support for digital audio and video (NOSSDAV'05). ACM, New York, NY, USA, pp 13-18 (2005)

[20] T. Lang, G. Armitage, P. Branch, H. Choo: A Synthetic Traffic Model for Half-Life. In Australian Telecom, Networks and Applications Conference (ATNAC) Melbourne (2003)

[21] H. Sze, C. Liew, J. Lee, D. Yip: A Multiplexing Scheme for H.323 Voice-Over-IP Applications. IEEE J. Select. Areas Commun. Vol. 20, pp. 1360-1368 (2002)

[22] 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