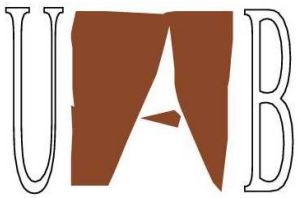


Universidad Autónoma de Barcelona

Escuela Técnica Superior de Ingeniería
Departamento de Arquitectura de Computadores y Sistemas Operativos

Un sistema de vídeo bajo demanda a gran escala tolerante a fallos de red

Memoria presentada por **Javier A. Balladini** para optar al grado de **Doctor** por la Universidad Autónoma de Barcelona.

Barcelona (España), Mayo de 2008

Un sistema de vídeo bajo demanda a gran escala tolerante a fallos de red

Memoria presentada por Javier A. Balladini para optar al grado de Doctor por la Universidad Autónoma de Barcelona. Trabajo realizado en el Departamento de Arquitectura de Computadores y Sistemas Operativos (DACSO) de la Escuela Técnica Superior de Ingeniería de la Universidad Autónoma de Barcelona, dentro del programa de Doctorado en Informática, opción A: "Arquitectura de Computadores y Procesamiento Paralelo", bajo la dirección del Dr. Remo Suppi Boldrito.

Barcelona, Mayo de 2008

Director:

Dr. Remo Suppi Boldrito

En verdad, todo problema después de resuelto parece muy simple. La gran victoria —que hoy parece fácil— fue el resultado de una serie de pequeñas victorias que pasaron desapercibidas.

Paulo Coelho

Deseo expresar mi agradecimiento al Dr. Remo Suppi Boldrito por su dirección, apoyo y confianza que me ha transmitido para llevar adelante este trabajo. A los miembros del grupo de investigación de VoD, principalmente al Dr. Porfidio Hernández Budé, por su predisposición y colaboración.
A todos mis compañeros de doctorado con quienes compartimos un mismo camino.
Por último, muy especialmente a mi familia que con su cariño y a la distancia, me han acompañado durante los cuatro años que he dedicado a este trabajo, lejos de mi gente y mi lugar.

A mi familia

Prólogo

Un sistema de vídeo bajo demanda a gran escala (LVoD, *Large-Scale Video-on-Demand*) brinda un servicio de visualización de vídeos a una gran cantidad de usuarios dispersos geográficamente. El tipo de servicio de vídeo bajo demanda (VoD, *Video-on-Demand*) más completo permite al usuario solicitar su vídeo preferido y reproducirlo casi instantáneamente, con la posibilidad de utilizar comandos interactivos (congelado de imagen, retroceso y avance lento/rápido, etc.) tal como si estuviese visualizando el vídeo con un reproductor de VHS o DVD. A este tipo de servicio se lo conoce como vídeo bajo demanda verdadero (T-VoD, *True Video-on-Demand*), e implica un diseño y desarrollo de elevada complejidad.

La mayoría de los sistemas de VoD fueron diseñados para trabajar en redes dedicadas o que permiten hacer reserva de recursos. Sin embargo, las arquitecturas de estos sistemas no son aplicables a entornos LVoD, fundamentalmente debido a los nuevos requerimientos de escalabilidad, costo del sistema y tolerancia a fallos. Cuando el entorno de red pasa de ser de una red local a una red de área amplia (como Internet), aumenta la probabilidad de fallos, disminuye el ancho de banda, y la calidad y clasificación de servicios es suplantada por un modelo de servicio de "mejor esfuerzo".

En el presente trabajo se propone una arquitectura de un sistema de LVoD distribuido, que permite ofrecer un servicio de T-VoD, con comunicaciones unitransmisión (*unicast*) sobre una red sin calidad de servicio como Internet. La nueva arquitectura, denominada VoD-NFR (*Video-on-Demand with Network Fault Recovery*) tiene como fin garantizar, ante fallos de la red y caídas de servidores, la entrega del contenido multimedia a los clientes sin disminuir la calidad de los mismos y sin sufrir interrupciones durante su visualización.

La presente memoria se organiza en los siguientes capítulos:

- Capítulo 1: Se enmarcan los problemas, retos, y requerimientos que involucran los sistemas de LVoD, describiendo el entorno donde se despliegan, y

las tareas internas que deben llevar a cabo cada uno de los componentes del sistema. A partir de este conocimiento básico, se justifica la necesidad de la nueva propuesta del sistema VoD-NFR y se describen los objetivos y alcance de este trabajo.

- Capítulo 2: En base al marco teórico dispuesto en el capítulo anterior, en este apartado se presenta la nueva propuesta, el sistema VoD-NFR. Se comienza analizando las características y arquitectura del sistema, para luego realizar una descripción funcional que permite comprender los detalles de su diseño con respecto a, entre otras cosas, el tratamiento de comandos interactivos, el proceso de entrega de los vídeos, y la tolerancia a fallos.

- Capítulo 3: El sistema VoD-NFR está sostenido por un componente que efectúa una delicada gestión de las comunicaciones al nivel de la capa de red, el cual es objeto de estudio de este capítulo. Para este sistema se han diseñado dos protocolos, uno de transporte de *streams* y otro de control de congestión, del cual se presenta una implementación para un entorno real destinada a un sistema operativo de propósito general.

- Capítulo 4: Dado que el sistema ha sido implementado tanto para entorno real como de simulación, este capítulo explica el diseño de cada uno de los módulos que lo componen, mediante descripciones estáticas (clases) y dinámicas, y se presenta la especificación del protocolo de comunicación utilizado entre los clientes y servidores.

- Capítulo 5: En este apartado se analiza la efectividad de las técnicas propuestas en el sistema VoD-NFR, mediante experimentos llevados a cabo en entornos reales y de simulación.

- Capítulo 6: Finalmente, se exponen las conclusiones del presente trabajo, haciendo hincapié en las aportaciones realizadas, y se presentan las líneas abiertas y futuros trabajos a realizar.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción a los sistemas de VoD

En este capítulo se enmarcan los problemas, retos y requerimientos que involucran los sistemas de LVoD, describiendo el entorno donde se despliegan, y las tareas internas que deben llevar a cabo cada uno de los componentes del sistema. A partir de este conocimiento básico, se justifica la necesidad de la nueva propuesta del sistema VoD-NFR y se describen los objetivos y alcance de este trabajo.

Un sistema de vídeo bajo demanda (VoD, *Video-on-Demand*) ofrece grandes posibilidades para la distribución de contenido audiovisual. El servicio de VoD más completo permite a los usuarios seleccionar un vídeo, reproducirlo casi instantáneamente y, además, permite un control completo sobre la visualización del contenido; es posible hacer una pausa, reanudar, posicionarse en un nuevo punto del vídeo, avanzar y retroceder rápidamente, etc., tal como si se tratara de un reproductor de VHS o DVD. El servicio de VoD tiene múltiples aplicaciones, entre las cuales pueden citarse: aprendizaje a distancia, televisión, bibliotecas digitales, servicios informativos que permitan al usuario visualizar únicamente las noticias que le interesan, o alquiler de vídeos. Esta última aplicación ha generado un gran interés por parte de las empresas que ofrecen servicios de televisión, ya que desean agregar un mayor grado de interactividad a sus contenidos y, a su vez, brindar un servicio instantáneo de alquiler de vídeos.

El diseño de una arquitectura de VoD es altamente complejo e involucra a una gran variedad de áreas de conocimiento científico: la compresión y descompresión de datos, la seguridad de los datos, el diseño de los componentes del servidor y los clientes, el diseño de sistemas operativos de tiempo real, el desarrollo de protocolos de comunicación, tecnologías de redes y dispositivos de interconexión, sistemas de procesamiento paralelos o distribuidos, como así también, estudios psicológicos para determinar el comportamiento de los usuarios. El auge de los sistemas de VoD se debe al gran avance en la compresión de vídeo y audio, y al aumento de las conexiones de Internet de banda ancha en los hogares, donde se concentra la principal masa de usuarios.

Un sistema de VoD puede clasificarse principalmente según el entorno al cual está orientado. Existen varios sistemas que trabajan en entornos cerrados, esta cla-

se de entornos son totalmente controlados, tienen redes de alta velocidad, en las cuales puede realizarse reserva de recursos. Por lo tanto, estos sistemas, tienen una complejidad relativamente baja. En este tipo de entornos se pueden encontrar: sistemas para hoteles, aviones (donde cada pasajero dispone de una pantalla en la que puede visualizar contenidos de VoD) [1], y grandes empresas. A su vez hay sistemas que trabajan en entornos donde la red es compartida y hay tráfico ajeno, donde los contenidos deben viajar por redes de terceros (perdiendo el control de los recursos de la red), y además, donde existe una gran cantidad de usuarios geográficamente distribuidos. Los usuarios pueden estar dispersos al nivel de redes de área metropolitana (MAN, *Metropolitan Area Network*), o a niveles aún mayores como son las redes de área amplia (WAN, *Wide Area Network*). Estos sistemas de VoD son denominados sistemas abiertos y a gran escala, o LVoD (*Large-Scale Video-on-Demand*).

El desarrollo de los sistemas de LVoD implica un gran esfuerzo debido a la alta complejidad de los mismos, convirtiéndolos en una de las áreas más activas de la investigación en informática, impulsada además por el creciente interés de la industria por comercializar el servicio de LVoD. El presente trabajo está dedicado a este tipo de sistemas, que se desarrolla detalladamente en este capítulo. Finalmente se presentan los problemas, antecedentes y objetivos de esta tesis.

1.1. Componentes de un sistema de VoD

Un sistema de VoD está compuesto de tres componentes que son esenciales para ofrecer cualquier tipo de servicio de VoD: el servidor, el cliente, y la red de comunicación. A continuación, se describe la funcionalidad de cada uno de estos componentes:

1. **Servidor**: Es responsable de almacenar y gestionar los contenidos multimedia, de recibir las peticiones de los clientes y transmitir los contenidos solicitados. Para llevar a cabo estas funciones, debe administrar cuidadosamente los recursos del servidor mismo y de la red para garantizar la calidad de servicio (QoS, *Quality of Service*) contratada por el cliente. A grandes rasgos, las tareas de un servidor pueden ser clasificadas en: gestión de alto nivel de aplicación, gestión de memoria secundaria, y gestión del nivel de red. La gestión de alto nivel de aplicación debe controlar que no se admitan más clientes que los soportados, debe implementar un modelo de servicio para organizar y distribuir los contenidos de los vídeos a los clientes, y debe decidir qué vídeos se almacenarán localmente y cuales no. La gestión de memoria secundaria es responsable de planificar la recuperación de los datos de

los vídeos desde la memoria secundaria para ser transmitidos a los clientes. Finalmente, la gestión del nivel de red se ocupa de proporcionar un transporte de los datos a los clientes solucionando los inconvenientes que pueda ofrecer la red de comunicación.

2. **Cliente**: Es el encargado de recibir, procesar, y visualizar en una pantalla la señal del vídeo generada por el servidor. Es capaz de recoger y procesar los eventos generados por el usuario y por el servidor para definir y acondicionar la apariencia de lo que el espectador observa. La unidad receptora de vídeo puede ser un computador hogareño con una interfaz web o aplicación específica, o un *Set-Top-Box* (STB). El STB es un dispositivo dedicado exclusivamente a la recepción del vídeo y su decodificación que se conecta directamente a la pantalla de visualización. Es importante destacar que el cliente, en algunos sistemas de VoD, además de cumplir con la función de recepción del contenido multimedia, también actúa como servidor de los vecinos cercanos.

3. **Red de comunicación**: Es el medio de comunicación que une los servidores con los clientes, y por el cual se transmiten los contenidos multimedia. Una red de área local (LAN, *Local Area Network*) presenta grandes facilidades para la implementación de un servicio de VoD ya que, además de ser redes rápidas, son privadas y pueden ser configuradas a conveniencia. El desafío se encuentra principalmente en redes más grandes, MAN o WAN, como la red Internet. El gran crecimiento que han experimentado las tecnologías de redes, han permitido a las compañías de telecomunicaciones ofrecer anchos de banda cada vez mayores a los usuarios hogareños, permitiendo el uso de servicios de VoD. La tecnología más utilizada es la Línea de Abonado Digital (DSL, *Digital Subscriber Line*), que convierte las líneas de telefonía fija en conexiones digitales que integran voz y datos.

1.2. Tipos de servicios de VoD

Los sistemas de VoD pueden ser clasificados de acuerdo al tipo de servicio que ofrecen a los usuarios. La funcionalidad básica, que ofrece un sistema de VoD, es la posibilidad de visualizar un vídeo en forma remota. Las características adicionales que ofrezcan estos sistemas, con respecto a la funcionalidad básica, dan origen a distintos tipos de sistemas de VoD, los cuales pueden clasificarse según [2, 3] en:

No-VoD: Es el sistema más simple de VoD que consiste en el envío de un único vídeo mediante comunicación por difusión (*broadcast*). Estos sistemas son llamados No-VoD porque no permiten comandos interactivos de vídeo.

PPV: El sistema de "pagar para ver" (*Pay-Per-View*) es una modalidad en la que el abonado paga por los eventos individuales que desea ver y, por lo general, la señal se transmite de forma simultánea para todos los clientes suscritos. Algunos ejemplos pueden ser eventos deportivos, películas recién estrenadas, conciertos musicales, etc.

Q-VoD: En el servicio "Casi Vídeo bajo Demanda" (*Quasi Video-on-Demand*), el funcionamiento es similar a un sistema de colas. Los usuarios eligen el vídeo que quieren ver, pero el servidor espera hasta conseguir un número mínimo de usuarios para iniciar una nueva transmisión. Los usuarios se agrupan según políticas de optimización en función de la utilización de recursos. No implementa funcionalidades interactivas, aunque los usuarios pueden pasarse de un grupo a otro que está visualizando el mismo vídeo en un tiempo de reproducción diferente.

N-VoD: En el "Vídeo bajo Demanda Limitado" (*Near Video-on-Demand*), el proveedor comienza a transmitir un vídeo cada cierto intervalo de tiempo. Entonces, las funciones como adelantar o retroceder son simuladas por transiciones en intervalos de tiempo discretos, por ejemplo, si el intervalo de tiempo entre transmisiones de un mismo vídeo es de 10 segundos, el usuario puede esperar hasta 10 segundos para comenzar a visualizarlo. Aunque no sea posible utilizar la función de pausar/continuar, este espectador podría retroceder el vídeo cambiándose a otro canal que esté mostrando la misma película pero 10 segundos atrás. Para adelantar el vídeo, se utiliza el mismo mecanismo que para el retroceso.

T-VoD: En el "Vídeo bajo Demanda Verdadero" (*True Video-on-Demand*), el usuario puede solicitar un contenido multimedia en cualquier momento, sin estar sujeto a programaciones preestablecidas por el operador, y a su vez tener un control total sobre la reproducción. El usuario escoge el contenido entre una lista y comienza a visualizar rápidamente el vídeo seleccionado, teniendo las capacidades de interactividad de los reproductores de VHS y DVD, además de poder incluir:

1. Inicio/Reanudar (*play*/*resume*): Comienza a visualizar un vídeo desde el comienzo o continúa desde otro punto donde se había detenido.
2. Parada (*stop*): Detiene de manera temporal o permanente la reproducción del vídeo.

3. Pausa (*pause*): Congela la imagen.
4. Salto hacia adelante/atrás (*jump fordward/backward*): Salta a un punto particular del vídeo, en una dirección hacia adelante o hacia atrás, sin mostrar el vídeo intermedio.
5. Avance/Retroceso rápido/lento (*fast/slow forward/backward*): Visualiza las escenas en cámara rápida o lenta, en un sentido hacia adelante o hacia atrás.
6. Otras funciones interactivas como por ejemplo: repetición de secuencias, marcado de escenas interesantes, etc.

Algunos autores, presentan una clasificación más de los VoD llamada "VoD Interactivo" (I-VoD, *Interactive Video-On-Demand*). Algunos consideran al I-VoD como un servicio equivalente al T-VoD, mientras que otros lo definen como un servicio que incluye las funcionalidades interactivas del T-VoD, pero sin las restricciones de tiempo desde que el usuario hace la solicitud del vídeo hasta que es atendido. Por ejemplo, en esta última definición, se incluiría a un sistema que descarga todo el vídeo en la unidad receptora antes de comenzar su visualización.

La selección de uno de estos tipos implica más o menos interactividad del usuario con el sistema; cuanta más interactividad se ofrezca, más complejidad tendrá el desarrollo del sistema de VoD. El sistema de VoD que se propone en esta tesis, se basa en el servicio T-VoD, que presenta la más alta complejidad debido a que requiere una administración de recursos extremadamente cuidadosa para poder garantizar la calidad del servicio ante la ejecución de comandos interactivos.

1.3. Características del contenido multimedia

Al diseñar un sistema de VoD se deben tomar decisiones que derivan de las características particulares de la información gestionada por estos sistemas: los contenidos multimedia.

Los contenidos multimedia están compuestos por vídeo, audio, subtítulos, marcas de escenas, meta-datos e información de sincronización. Todos estos componentes son almacenados en archivos denominados "contenedores multimedia". Algunos de los contenedores multimedia más conocidos son: AVI (*Audio Video Interleave*), MPG (*MPEG*), QT (*QuickTime Movie*), WMV (*Windows Media Video*), y Ogg. Por ejemplo, en un mismo contenedor multimedia, es posible almacenar un vídeo con 6 canales de audio utilizando sonido AC-3 (*Dolby Digital 5.1*) o DTS (*Digital Theater System*).

Cada vídeo consiste en una sucesión de imágenes estáticas, llamadas cuadros o marcos (*frames*). Las pequeñas diferencias que existen entre las imágenes, sumado al alto número de imágenes mostradas por unidad de tiempo, produce a la vista la sensación de movimiento. Una de las principales medidas de la calidad de la reproducción es la frecuencia de representación de las imágenes (*play rate*), que se mide en cuadros por segundos (fps, *frames per second*). Una frecuencia próxima a 30 fps produce una buena calidad de imagen (el estándar PAL utiliza una frecuencia de 25 fps y NTSC utiliza 29,97 fps), aunque para vídeos de alta definición se requieren frecuencias mayores (HDTV utiliza hasta 60 fps).

Los archivos de vídeo necesitan almacenar una gran cantidad de datos, y su tamaño depende de la duración, calidad y formato de los mismos; para tener una idea del tamaño que ocuparán estos vídeos que será necesario almacenar y transmitir, la captura de una pantalla completa de 640×480 píxeles, con 24 bits de color y a 30 fps consumiría:

$$640 \times 480 \times 3(colores) \times 30(imágenes) \times 60(seg) = 1.658.880.000\ bytes/min$$

es decir, se requieren 1,54 GB para un minuto de vídeo sin incluir sonido. Hay diversas maneras de reducir el tamaño de este tipo de archivo, las cuales son:

- Reducir el tamaño de la ventana de reproducción
- Disminuir el número de colores
- Reducir el número de fps
- Comprimir el archivo

Las tres primeras técnicas producen una notable pérdida de calidad, sin embargo las técnicas de compresión (*codecs* de vídeo) buscan un compromiso entre la calidad de la media, que tiene una estrecha relación con los niveles de percepción del ser humano, y la cantidad de almacenamiento requerido. Una señal de vídeo puede ser comprimida en función de las semejanzas o redundancias que generalmente presentan. Por ejemplo, un escena de cine, normalmente contiene los mismos objetos con algunos desplazamientos de estos objetos, por lo tanto, los cuadros consecutivos de esta secuencia de vídeo tienen una redundancia temporal que puede ser explotada. Otro tipo de redundancia que puede ser explotada, pero en un cuadro en particular es la espacial, dado que las amplitudes de los píxeles cercanos generalmente están correlacionadas.

Los estándares de compresión de vídeo MPEG definen tres tipos básicos de cuadros codificados:

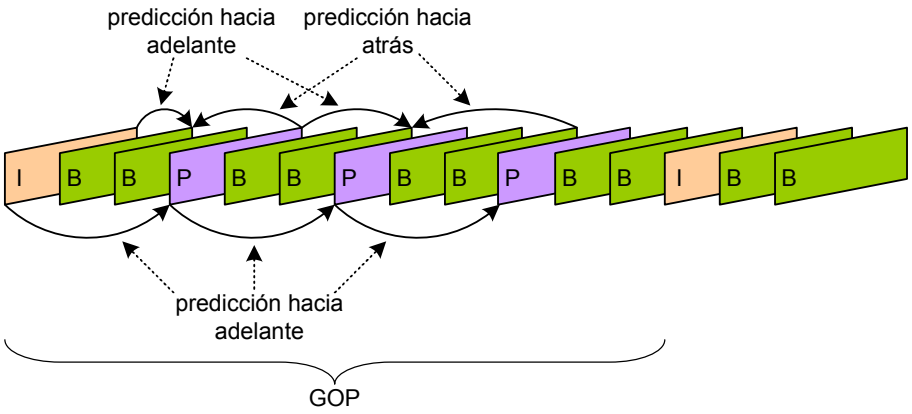


Figura 1.1: Ejemplo sobre las dependencias de predicción entre cuadros

1. Cuadros intra codificados o *cuadros-I*: Estos cuadros están codificados independientemente de los otros.
2. Cuadros codificados en forma predictiva o *cuadros-P*: Estos cuadros se codifican basándose en un cuadro anterior.
3. Cuadros codificados en forma predictiva bidireccional o *cuadros-B*: Estos cuadros están codificados basándose en cuadros anteriores y posteriores.

La figura 1.1 muestra los diferentes cuadros codificados y las dependencias predictivas para un ejemplo de grupo de imágenes de MPEG.

Entre todos los estándares de compresión de vídeo, existen dos grandes familias desarrolladas por la ITU-T (*Telecommunication Standardization Sector*) y la ISO (*International Organization for Standardization*). Los formatos de vídeo del ITU-T e ISO se denominan H.26x y MPEG-x, respectivamente. El nombre de los formatos MPEG-x es debido al grupo de desarrollo del ISO, llamado MPEG (*Moving Picture Experts Group*). Los formatos de compresión de vídeo más usuales son: el H.261, H.263, MPEG-1, MPEG-2, MPEG-4, WMV, RealVideo, FLV (*Flash Video*) y MOV (*QuickTime Movie*). En la tabla 1.1 se muestran los requerimientos de ancho de banda para el formato MPEG-2.

En cuanto al tratamiento del audio, la situación es similar a la del vídeo. El audio se caracteriza por el número de canales, la frecuencia de muestreo, y el número de bits por muestra. Con audio DTS (6 canales), una frecuencia de muestreo de 44,1 KHz (calidad de CD) y 16 bits por muestra, un minuto de audio consumiría:

$$60(seg) \times 6(canales) \times 44.100(muestras) \times 16(bits/muestra)/8 = 30{,}28\,MB$$

Calidad	Tasa de bits requerida
vídeo por teléfono	16 kb/s
videoconferencia	128–384 kb/s
VCD	1,25 Mb/s
DVD	5 Mb/s
HDTV	15 Mb/s
HD DVD	36 Mb/s
Blu-ray Disc	54 Mb/s

Tabla 1.1: Requerimientos del vídeo MPEG-2

Calidad	Tasa de bits requerida
radio AM	32 kb/s
radio FM	96 kb/s
CD	224–320 kb/s

Tabla 1.2: Requerimientos del audio MP3

Sin ser tan significativo como el tamaño de un archivo de vídeo sin comprimir, el tamaño de un archivo de audio es muy grande, y por lo tanto es necesario aplicar métodos de compresión (*codecs* de audio). Algunas técnicas, llamadas "con pérdida", eliminan frecuencias de la señal original inaudibles para el ser humano, mientras que las "sin pérdida" comprimen el audio sin perder información. Los formatos de compresión de audio típicos con pérdida son: AAC (*Advanced Audio Coding*), MP3 (*MPEG-1 Audio Layer 3*), Vorbis, WMA (*Windows Media Audio*), y RealAudio; y sin pérdida se encuentran: AIFF-C (*Audio Interchange File Format Compressed*), FLAC (*Free Lossless Audio Codec*), y WAV (*Windows Audio Waveform*, aunque normalmente se usa sin compresión). En la tabla 1.2 se muestran los requerimientos de ancho de banda para el formato de compresión de audio MP3.

1.4. Sistemas de VoD a gran escala

Un sistema de VoD a gran escala (LVoD, *Large-Scale Video-on-Demand*) presenta características que complica aún más las tareas normales de los sistemas de VoD. Un LVoD presta servicio a un elevado número de usuarios, distribuidos geográficamente al nivel de redes MAN y WAN. Esto implica que la red de comunicación tenga un ancho de banda mucho menor. Además, en una LAN, es posible hacer una reserva de recursos para poder garantizar la entrega del contenido a los

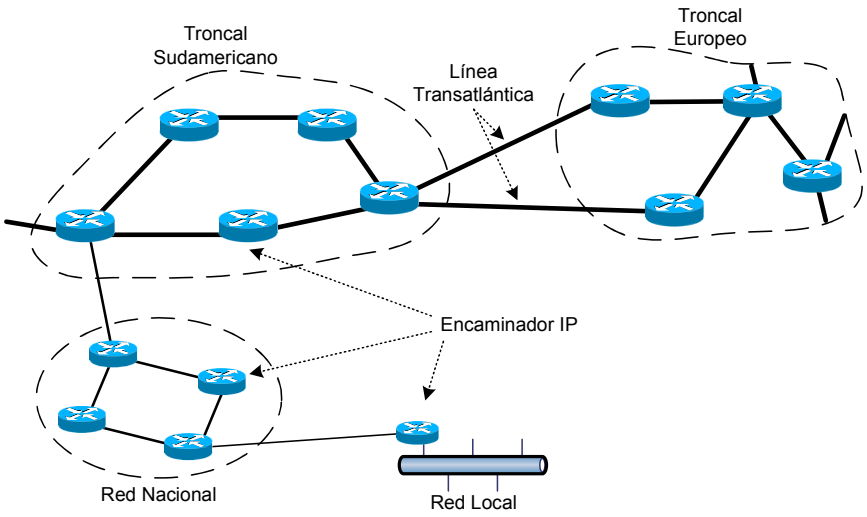


Figura 1.2: La red Internet es un conjunto de subredes interconectadas

clientes. Sin embargo, hoy en día, esto no es posible hacerlo en una red MAN o WAN.

A continuación se presentan las características de la red Internet, la red típica para un sistema de LVoD; más adelante, se exponen los requerimientos de los sistemas de LVoD, y las diferentes arquitecturas de alto nivel de estos sistemas.

1.4.1. Caracterización del entorno: Internet

La red Internet puede definirse como un conjunto de subredes, o sistemas autónomos (AS, *Autonomous System*) interconectados. No hay una estructura real, pero existen varios enlaces principales o troncales (*backbones*). A estos troncales, se conectan redes regionales (MAN), y a estas redes regionales, se conectan redes locales (LAN) de universidades, compañías y proveedores de servicio de Internet. La figura 1.2 muestra un ejemplo de la organización de la red Internet.

En esta red todos los paquetes son tratados de la misma manera, sin discriminación o garantías explícitas de entrega, conocido como "modelo de servicio de mejor esfuerzo" (*Best-Effort Service Model*) [4]. Es decir, los paquetes no son clasificados, no se marcan, no pueden procesarse de manera diferente y la calidad de servicio como los recursos no pueden garantizarse. Como la red es compartida, el ancho de banda disponible varía dependiendo de la cantidad de tráfico que esté circulando por la red en cada momento, a su vez el retardo también varía dependiendo, entre otras cosas, del camino que recorra la información por la red.

Además, las grandes dimensiones de la red aumenta considerablemente la probabilidad de fallos por razones físicas (generalmente, fallas de los dispositivos activos de comunicación).

La comunicación en Internet funciona de la siguiente manera: los datos se dividen en datagramas, en teoría, los datagramas pueden ser de hasta 64 kB cada uno, pero en la práctica por lo general son de aproximadamente 1.500 bytes, y luego se transmiten a través de Internet, posiblemente fragmentándose en unidades más pequeñas en el camino. Cuando todas las piezas llegan a la máquina destino se ensamblan y, finalmente, se entrega el datagrama original. Las unidades de transmisión son conocidas como paquetes IP (*Internet Protocol*), los cuales son encaminados independientemente a través de la red por los encaminadores (*routers*). Los encaminadores mantienen información sobre el estado de la red en tablas de encaminamiento, utilizada para calcular la ruta adecuada y, además, intercambian información para conocer los cambios en la topología y el estado de la red.

Congestión

Cuando hay demasiados paquetes en la red hay una degradación de las prestaciones; a esta situación se la denomina congestión. La congestión toma lugar cuando la red está sobrecargada por la demanda de recursos de red, y esta demanda está cerca o excede la capacidad total de la red [4]. La figura 1.3 muestra este problema de disminución del rendimiento de la red. Cuando la transmisión es efectuada dentro de la capacidad de la red, los paquetes llegan a su destino (excepto unos pocos posiblemente afectados por ruido) y el número de paquetes recibidos es proporcional al número de paquetes enviados. Sin embargo, cuando aumenta el tráfico, la red ya no puede manejarlo, y comienza a perder paquetes dando como resultado un estado de congestión, el cual continúa aumentando y empeora. Cuando existe una pérdida de un paquete (por ejemplo si un encaminador lo ha descartado), el temporizador de retransmisión podría expirar y así el emisor retransmitiría el paquete. La presencia de un número mayor de paquetes en la red empeora la situación, y como la capacidad de entrega de la red continúa siendo la misma, la relación de paquetes recibidos contra enviados desciende aún más [4, 5].

Para dar solución a este problema, se utilizan algoritmos de control de congestión que disminuyen o aumentan la tasa de transmisión en función de si se detecta o no algún síntoma de congestión. En la Internet, debido al gran uso de la fibra óptica en los enlaces, la mayoría de las expiraciones de paquetes se deben a descartes y no a errores de transmisión; por lo tanto, los algoritmos de control de congestión asumen las expiraciones de los paquetes como síntomas de congestión de la red.

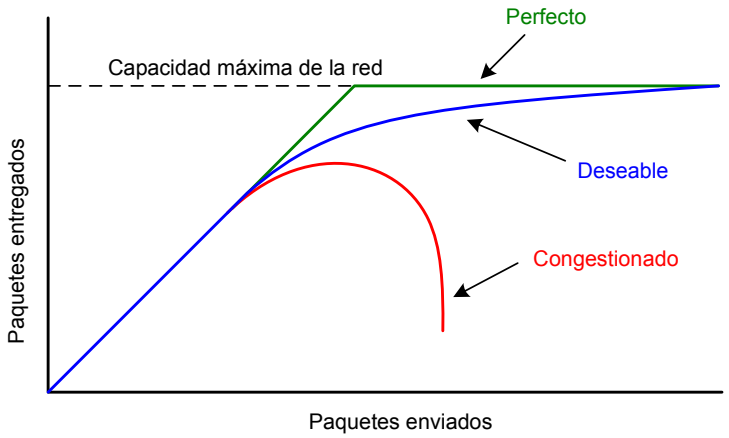


Figura 1.3: Degradación del desempeño por congestión de la red

Envío por multitransmisión y MBone

La comunicación IP normal se realiza entre un transmisor y un receptor, sin embargo, en algunas aplicaciones como videoconferencia y VoD es útil que un proceso pueda transmitir el mismo material a una gran cantidad de receptores simultáneamente. Esta comunicación es conocida como multitransmisión (*multicast*), donde la red es responsable de duplicar los bloques de datos para entregar a los distintos clientes. En una red convencional, el emisor debería transmitir el mismo conjunto de datos a cada cliente, así el envío por multitransmisión disminuye la carga de trabajo del nodo transmisor y el requerimiento de ancho de banda de la red en el camino hacia los clientes.

El encaminamiento multitransmisión de paquetes IP es implementado en la Internet por encaminadores multitransmisión. La idea básica es ilustrada en la figura 1.4. La figura muestra un emisor transmitiendo un *stream* a múltiples receptores. Los encaminadores multitransmisión que se observan están configurados como un árbol multitransmisión que permite una eficiente transmisión de datos . La red entre dos encaminadores multitransmisión no necesita tener soporte de multitransmisión, de esta manera, para atravesar redes no multitransmisión, los encaminadores encapsulan los mensajes en mensajes con direcciones del tipo unitransmisión (*unicast*) al siguiente encaminador multitransmisión.

El MBone [6] es un conjunto de encaminadores multitransmisión que proveen un troncal multitransmisión (*multicast backbone*) para la Internet. Históricamente MBone surgió en 1.992 tras una de las reuniones de coordinación del IETF (*Internet Engineering Task Force*), con el fin de experimentar sobre el concepto de

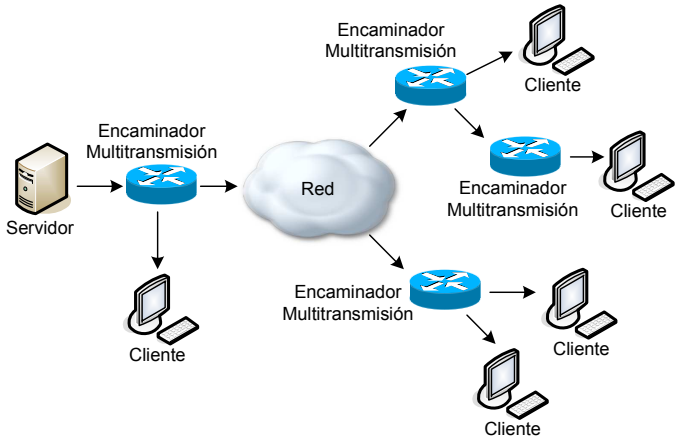


Figura 1.4: Multitransmisión IP

direcciones multitransmisión propuesto por Steve Deering (Universidad de Stanford) en su tesis doctoral. Van Jacobson, de los laboratorios Lawrence Berkeley (LBL), Steve Casner, del ISI (*Information Science Institute*) y varios ingenieros más, se unieron al proyecto propuesto por Steve Deering dando lugar a un grupo de trabajo que se encargaría de sentar las bases de lo que ahora es MBone.

En 1.995, la red MBone estaba comprendida por 1.700 redes, distribuidas aproximadamente entre 20 países. El tamaño de la red MBone, comparado a toda la red Internet, es relativamente pequeño. En febrero de 1.995, la Internet tenía 48.500 subredes, por lo que MBone comprendía sólo el 3,5 % de toda la Internet. Estudios más recientes como [7] confirman que, aunque la puesta en funcionamiento de las comunicaciones multitransmisión han continuado en aumento, este crecimiento es relativamente lento. Además, sólo un pequeño porcentaje de grupos multitransmisión tienen niveles de actividad significativos, por lo que el aprovechamiento de esta tecnología es muy bajo.

La capa de transporte

La capa de transporte está diseñada para permitir que los nodos de origen y destino lleven a cabo una comunicación, la cuál se efectúa entre los extremos, sin implicación de los nodos intermedios. En esta capa se definen dos protocolos: el protocolo de control de transmisión (TCP, *Transmission Control Protocol*) y el protocolo de datagramas de usuario (UDP, *User Datagram Protocol*). TCP es un protocolo confiable orientado a la conexión en el cual el emisor y el receptor establecen una conexión antes de iniciar la transmisión de mensajes. Para regular la

tasa de transmisión, TCP provee dos mecanismos: el control de flujo, que asegura la sincronización entre emisores rápidos y receptores lentos, y el control de congestión que permite disminuir el envío de información a la red cuando se detecta un estado de congestión de la red. El protocolo TCP provee una entrega confiable a través de la retransmisión de mensajes perdidos; por otro lado, el protocolo UDP provee una funcionalidad mínima de comunicación entre extremos, sin garantizar la entrega de los paquetes ni el orden de los mismos.

Existe un protocolo muy conocido denominado Protocolo de Transporte de Tiempo Real [8] (RTP, *Real-Time Transport Protocol*) destinado a la entrega de datos multimedia como *streams* de vídeo y audio. Existen discrepancias sobre si RTP es un protocolo de nivel de aplicación o de transporte. El RTP debe funcionar sobre UDP o TCP, en general se coloca por encima de UDP y provee funcionalidad de marcas temporales, numeración en secuencia de los mensajes, identificación de contenidos, y seguimiento de entrega [9].

1.4.2. Requerimientos de los sistemas de LVoD

Un sistema de LVoD presenta grandes requerimientos en función de las características de la información que gestionan, es decir, del contenido multimedia, y del deficiente medio de transmisión disponible. La problemática de estos sistemas está dada por el gran volumen de datos que debe transmitirse a cada uno de los clientes, para que puedan visualizar el contenido multimedia, manteniendo la QoS. Mantener la calidad de servicio implica que el vídeo deba ser entregado con restricciones de tiempo altas.

La arquitectura de un sistema de LVoD, con un servicio del tipo T-VoD, debe ser cuidadosamente diseñada, teniendo en cuenta los siguientes requerimientos que caracterizan a este tipo de sistemas [10]:

Gran capacidad de almacenamiento: tal como se ha expuesto en la sección 1.3, los archivos de vídeo presentan un gran tamaño y, sumado a la enorme cantidad de vídeos que estos sistemas manejan, es necesario que el servidor de VoD esté preparado para poder administrar esta cantidad de información. Un repositorio de vídeos puede requerir decenas de TB (Terabytes); por ejemplo, un vídeo con formato HDTV de 2 horas de duración y compresión MPEG-2 requiere aproximadamente 14 GB, por lo tanto, un repositorio de 500 vídeos requiere cerca de 7 TB.

Elevado ancho de banda: el hecho de que los contenidos multimedia se caracterizan por hacer uso de un elevado ancho de banda (ver tabla 1.1), unido a la

necesidad de dar servicio a un elevado número de usuarios, hace necesario el máximo aprovechamiento posible del ancho de banda de red disponible. Manejar un gran volumen de información por unidad de tiempo, no solo involucra a la red, sino que también atañe al sistema de almacenamiento, uso de CPU y memoria.

Restricciones de tiempo y QoS: cuando un usuario solicite un vídeo, el sistema debe ser capaz de prestarle servicio en un tiempo razonable y predecible. Si no se garantiza un límite de tiempo de estas características, el usuario estará disconforme y terminará abandonando el servicio. Asimismo, es de esperar un servicio de entrega de contenidos libre de interrupciones.

Soporte de gran cantidad de usuarios: el sistema debe ser capaz de soportar y atender a una elevada cantidad de peticiones, que puedan ser realizadas al mismo tiempo. Este factor es determinante debido a los requerimientos de recursos (fundamentalmente de memoria) que supone la administración de tantos clientes conectados al sistema.

Tolerancia a fallos: los fallos en estos sistemas tan grandes son muy frecuentes por el alto número de componentes que poseen. Es necesario que el sistema continúe prestando servicio, incluso si uno o más componentes de la arquitectura fallan. Por lo tanto, se requiere poseer mecanismos, tanto en hardware como en software, para reaccionar ante posibles fallos permitiendo, por ejemplo, cambios de código en ejecución, aislamiento del fallo, y proveer caminos o servidores alternativos para no interrumpir el servicio.

Escalabilidad: el sistema debe ser capaz de prestar servicio a una determinada cantidad de usuarios en un determinado contexto, y de crecer, mediante un incremento de los recursos, para dar soporte a una mayor cantidad de usuarios.

Uso eficiente de recursos y balanceo de carga: debido a las grandes exigencias de recursos que demandan los servicios de VoD, es muy importante extremar el cuidado del consumo de recursos de cualquiera de los tres componentes básicos de los sistemas de VoD (servidor, cliente, y red). Una forma de ahorro de recursos es la compartición de recursos, posiblemente aplicables a la red mediante el uso de transmisión por difusión o multitransmisión, o mediante el uso de los recursos libres de los clientes. También es de suma importancia distribuir equitativamente la carga del sistema para no saturar estos componentes. Una correcta distribución de la carga en el sistema, reduce la probabilidad de rechazo de servicio a los usuarios [11].

1.4.3. Arquitecturas

Actualmente, un sistema de LVoD requiere del uso de varios servidores para llevar a cabo sus tareas. Las arquitecturas varían de acuerdo a la disposición de estos servidores y de los clientes en la red, presentando cada una de ellas sus propias ventajas y desventajas. Es posible encontrar arquitecturas centralizadas, de servidores independientes, arquitecturas jerárquicas, o arquitecturas distribuidas. A continuación se presentan las características de cada uno de estos sistemas.

Arquitectura centralizada

A grandes rasgos, esta arquitectura puede ser vista como un único servidor de VoD que atiende a todos los clientes del sistema; sin embargo, el "servidor" puede ser en realidad un grupo (*cluster*) de servidores que trabajan coordinadamente. El problema que presenta es que se requiere una conexión a la red extremadamente elevada y, además, el sistema es muy poco tolerante a fallos. Por ejemplo, un corte de energía o de red, que afecte al servidor, dejaría a todos los clientes del sistema sin servicio. La escalabilidad no es muy buena ya que el crecimiento está limitado por la conexión a la red. Otra dificultad que presenta es que sólo podría prestarse servicio a clientes con cierta localidad, ya que clientes muy distantes podrían sufrir muchos problemas de red. No obstante, la elevada comunicación del grupo de servidores permite compartir fácilmente los recursos y balancear la carga de trabajo.

Arquitectura de servidores independientes

En esta arquitectura cada servidor atiende a una cantidad determinada de clientes y, cuando se desea ampliar la capacidad del sistema, se instala un nuevo servidor para que dé soporte a más clientes. No existe ninguna comunicación entre los servidores, por lo tanto, todos los vídeos deben ser almacenados en cada uno de los nodos. La figura 1.5 muestra este tipo de esquema de arquitectura de servidores independientes.

La ventaja de esta organización es su escalabilidad ya que no está limitada en cuanto a la capacidad de servicio, la cual aumenta de forma directamente proporcional al número de servidores. No requiere servidores tan complejos ni grandes capacidades de conexión a red como la arquitectura centralizada. Sin embargo, no dispone de balanceo de carga y, al tener replicación de vídeos, no es escalable respecto a la capacidad de almacenamiento. Un importante problema que presenta es la tolerancia a fallos, ya que no se dispone de caminos alternativos de servicio a los

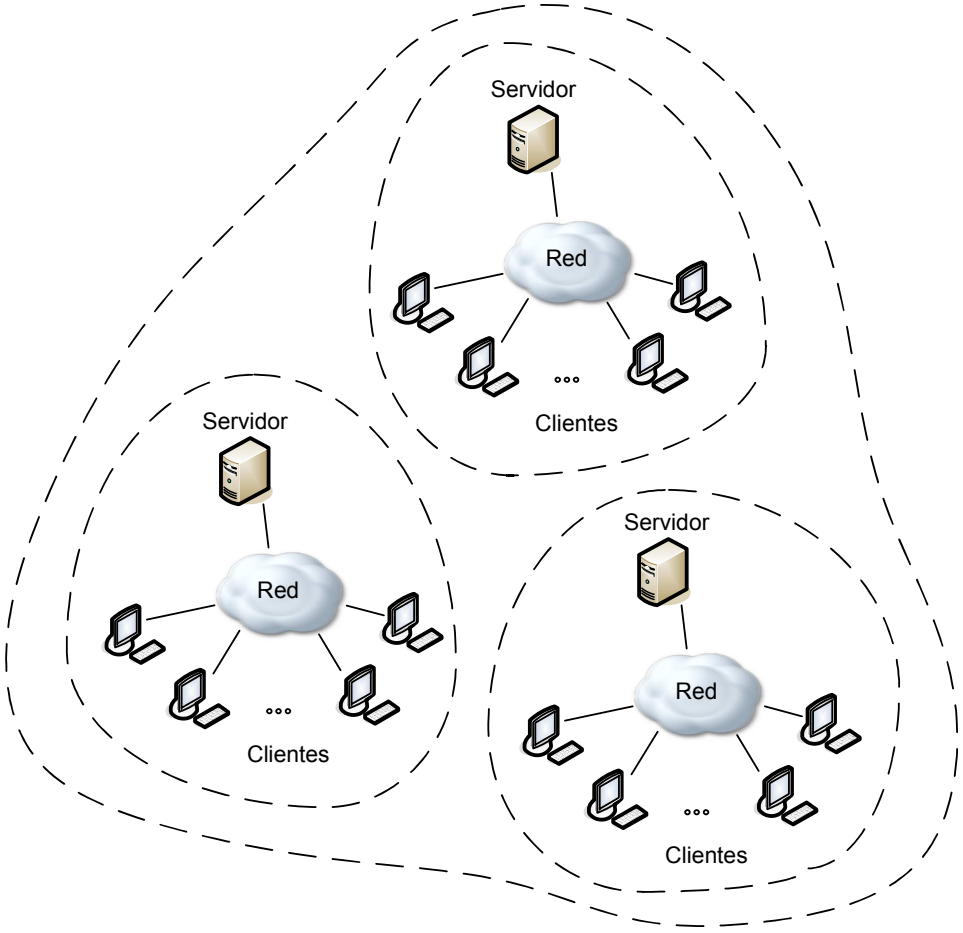


Figura 1.5: Arquitectura de servidores independientes

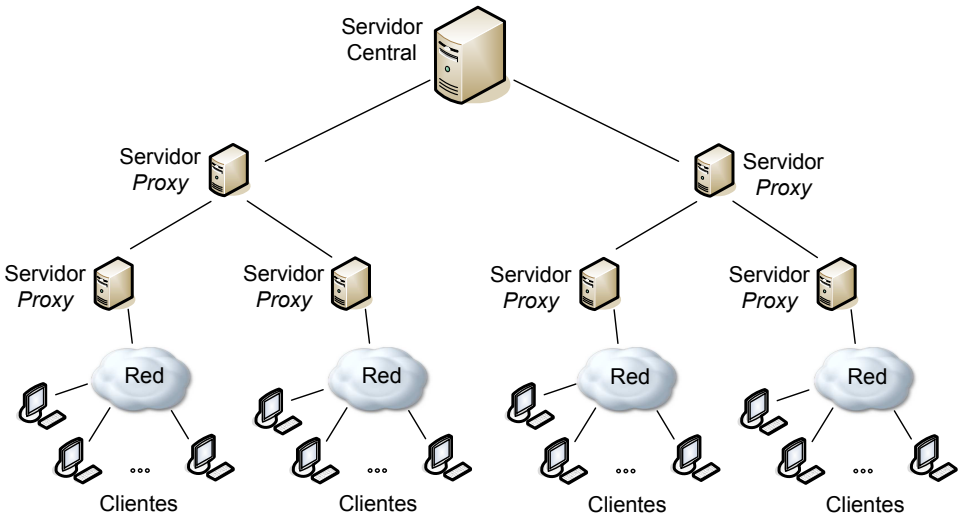


Figura 1.6: Arquitectura jerárquica

clientes, y una caída de uno de los servidores implica la denegación de servicio a todos los clientes conectados al mismo.

Arquitectura jerárquica de servidores

Con el objetivo de reducir el alto grado de replicación de vídeos de la arquitectura de servidores independientes, se propone una arquitectura jerárquica. Este esquema evita la replicación de contenidos pero conlleva un problema propio de las arquitecturas centralizadas: un único punto de falla por tener un servidor central. El servidor central almacena la totalidad de los vídeos. Otros servidores, llamados servidores *proxy* funcionan de manera similar a los servidores *proxy* de páginas web, almacenando vídeos completos o trozos de ellos en la *cache*; las peticiones son realizadas a servidores *proxy* [12, 13, 14] del nivel más bajo, y si el servidor no dispone del contenido requerido en la *cache*, la petición se redirige al servidor correspondiente siguiendo la jerarquía y, en último caso, es el servidor central (de mayor nivel dentro de la jerarquía) quien atiende la petición. En la figura 1.6 se muestra la arquitectura jerárquica de servidores.

La presencia del servidor central se convierte en un cuello de botella a medida que se incrementa el número de servidores *proxy*, lo que limita enormemente la escalabilidad del sistema.

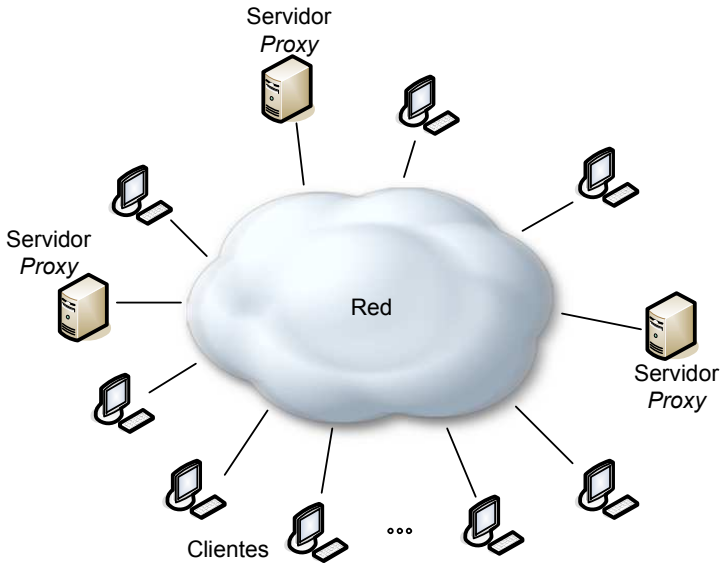


Figura 1.7: Arquitectura distribuida

Arquitectura distribuida

En este tipo de arquitecturas, el catálogo de contenidos multimedia no está centralizado, sino que se encuentra distribuido entre los diferentes servidores. Además, hay cierta replicación de contenidos, y uso de esquemas de *cache*. Esta arquitectura tiene un funcionamiento dinámico, en la cual los usuario no tienen asociado un servidor fijo, sino que un usuario puede ser atendido por cualquier servidor o varios de ellos. La figura 1.7 presenta el esquema de una arquitectura totalmente distribuida.

Un usuario realiza una petición a uno de los servidores *proxy*, y si el vídeo no se encuentra en él, la petición es redirigida a otro servidor *proxy* que disponga del vídeo. Los servidores *proxy* se reparten los usuarios para balancear la carga. Este sistema presenta una alta tolerancia a fallos, porque, si un servidor falla, el resto de los servidores *proxy* se repartirán los usuarios afectados para prestarles servicio. Además, presenta una buena escalabilidad ya que el agregado de nuevos servidores a la estructura es muy simple.

Dentro de una arquitectura distribuida, entran los esquemas conocidos como *Peer-To-Peer* (P2P) donde los clientes también actúan como servidores, colaborando en la entrega de contenidos a otros clientes. La idea de estos esquemas es que los clientes contribuyan en las tareas del sistema de VoD, cediendo poder computacional y ancho de banda de red. Se han llevado a cabo muchas investigaciones en

sistemas P2P para proveer contenido multimedia en el entorno Internet, entre las cuales pueden nombrarse a [15, 16, 17, 18, 19, 20].

1.5. Planificación en servidores de VoD

En esta sección se presenta una taxonomía de los diferentes tipos de planificaciones que se realizan en un servidor de VoD. En la literatura disponible es muy difícil identificar los distintos planificadores y sus funciones debido a diversas causas: no hay un consenso en cuanto a la nomenclatura y fundamentalmente porque no existe una clara distinción y separación de aspectos de los diferentes planificadores. Sin embargo, una clara división podría hacer los sistemas más fáciles de comprender, modificar, adaptar, y evolucionar. Por estas razones, en esta sección se intenta clasificar las tareas efectuadas por un servidor de VoD en los diferentes tipos de planificadores, y se explica cómo ellos interactúan para llevar a cabo la tarea de transmitir los flujos (*streams*) de vídeo a los clientes.

Cuando una petición llega a un servidor, un módulo de control de admisión será el responsable de aceptar o rechazar la solicitud de acuerdo a si es factible o no, en función de los recursos disponibles, poder atender esa nueva petición manteniendo las garantías de QoS. En sistemas de VoD distribuidos, cuando hay más de un candidato para atender la petición, es necesario utilizar algún algoritmo para decidir cual de todos los servidores atenderá la petición [21]. De esta manera, nos encontramos con el primer algoritmo de planificación que participa en un servidor de VoD, el balanceador de carga. Este algoritmo intentará seleccionar un servidor que pueda atender correctamente la petición, balanceando la carga del sistema, distribuyendo equitativamente las peticiones entre los distintos servidores. A partir de que la petición ha llegado a un servidor, numerosos algoritmos de planificación entran en juego para cumplir con el objetivo del sistema de VoD.

En la figura 1.8 se presenta una taxonomía de los principales planificadores utilizados en sistemas de VoD: el balanceador de carga, el planificador de peticiones, los modelos de servicio, el planificador de canales lógicos, y el planificador del tráfico de red. No siempre se encuentran claramente identificados o implementados de forma independiente, pero es importante hacer una separación conceptual para comprender todas las funciones que se realizan.

En la figura 1.9 se muestra la interacción entre los distintos planificadores. El Balanceador de Carga se ocupa de distribuir el trabajo entre los componentes del sistema, el Planificador de Peticiones es el encargado de aceptar/rechazar y priorizar las peticiones, y los Modelos de Servicio asignan las peticiones a canales lógicos. El Planificador de Canales Lógicos se utiliza para decidir qué canal lógico

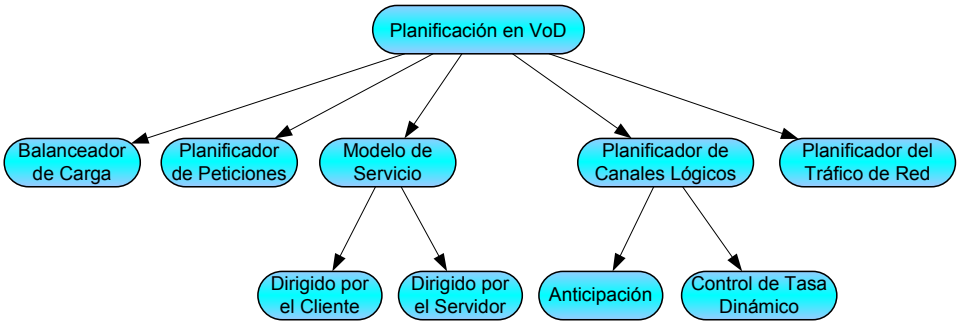


Figura 1.8: Taxonomía de los principales planificadores de un sistema de VoD

será servido en cada momento, y finalmente el Planificador del Tráfico de Red se encarga de administrar el nivel de red, dando solución a problemas (si existiesen) de congestión y pérdidas de paquetes.

A continuación, se describe el balanceador de carga, el planificador de peticiones, los modelos de servicio, el planificador de canales lógicos, y el planificador del tráfico de red, en las secciones 1.5.1, 1.5.2, 1.5.3, 1.5.4 y 1.5.5, respectivamente. Para cada uno de ellos, se detallan los principales algoritmos, técnicas, arquitecturas, o mecanismos que utilizan.

1.5.1. Balanceador de carga

El balance de carga se mantiene gracias a un algoritmo que divide el trabajo de la manera más equitativa posible, para evitar sobrecargar a ciertos servidores. Se han propuesto diversos algoritmos para balancear la carga; en los trabajos [22, 23] se presenta una revisión de algunos algoritmos de balanceo de carga y sus clasificaciones. El balance de carga no es una tarea tan simple como hacer asignaciones estáticas, entre peticiones y servidores, en el momento del arribo de las peticiones al sistema de VoD. Los clientes al salir del sistema producen desbalanceo, además de que ciertas peticiones pueden generar una redistribución de la carga para que sean aceptadas. Por ejemplo, supongamos dos servidores *S1* y *S2*, donde *S1* tiene las películas α y β, y *S2* tiene la película β. Si tenemos un cliente *C1* que está viendo la película β del servidor *S1*, y a continuación llega un cliente *C2* que quiere ver la película α, si la capacidad de *S1* está colmada, será necesario migrar la petición del cliente *C1* desde el servidor *S1* a *S2* (ya que la película β se encuentra en ambos servidores), y así liberar recursos para atender a *C2*. En este caso, la redistribución dinámica de la carga ha impedido denegar el servicio al cliente *C2*.

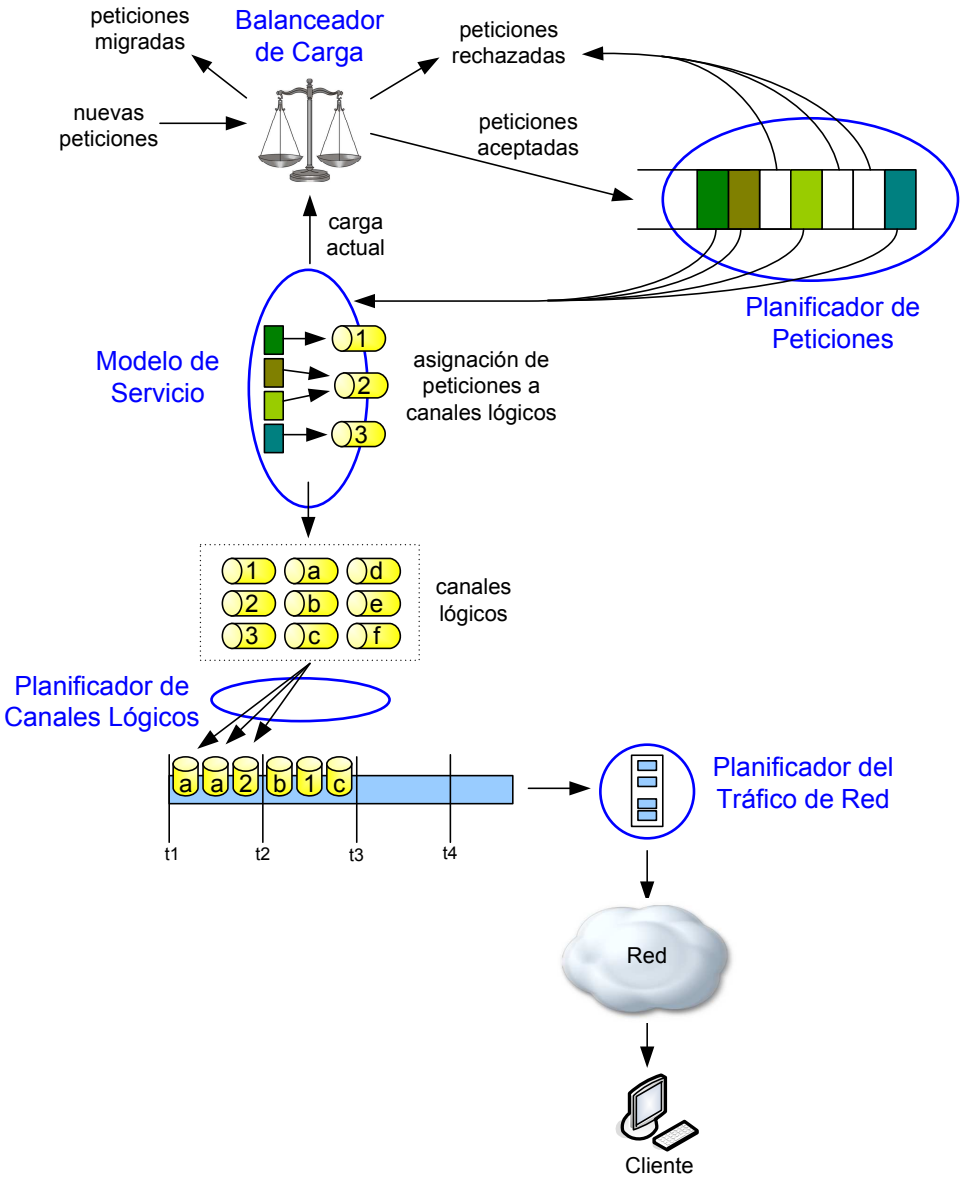


Figura 1.9: Esquema de funcionamiento y colaboración de los diferentes planificadores de un servidor de VoD

Un balanceador de carga puede tomar sus decisiones de manera centralizada o distribuida. Tomar las decisiones de manera distribuida significa que cada servidor deberá decidir si acepta o rechaza las peticiones de acuerdo a información local, desconociendo la carga de los demás servidores del sistema. Normalmente se utiliza la opción centralizada cuando la arquitectura del sistema es centralizada, y distribuida en otros casos, aunque también es posible implantar alguna solución híbrida.

1.5.2. Planificador de Peticiones

Si la utilización del servidor es baja, todas las peticiones pueden ser satisfechas inmediatamente. Sin embargo, si no hay suficientes recursos disponibles, estas peticiones son introducidas en una cola de espera hasta que son seleccionadas para ser atendidas. Las política de selección de peticiones conforman el Planificador de Peticiones (*Request Scheduler*).

Es necesario determinar cuándo y qué petición será servida para asegurar la satisfacción de los clientes y el cumplimiento de ciertas pautas preestablecidas como pueden ser: justicia, diferenciación por prioridades, diferenciación por clases de servicio, etc. [24]. Los principales objetivos que suelen requerirse son [9]:

1. Minimizar la probabilidad de rechazo a largo plazo.
2. Minimizar el tiempo de espera promedio. Este tiempo de espera, denominado *startup delay*, es el tiempo que el cliente puede tolerar entre que envía la petición y recibe los datos del vídeo.
3. Ser justo, es decir, la probabilidad de rechazo para cada una de las peticiones debe ser la misma.
4. La respuesta del sistema a las operaciones interactivas debe estar acotada en el tiempo.

1.5.3. Modelos de Servicio

El modelo de servicio especifica la manera en que los datos del vídeo son planificados y entregados al cliente [25], es decir, es quien determina la forma en que las peticiones de los clientes son atendidas por medio de los *streams*; por ejemplo, es posible realizar una agregación de clientes para ser servidos por un único *stream* de datos utilizando la multitransmisión.

Los modelos de servicio se clasifican en dos clases [26]: dirigido por el cliente (*Client Pull*) y dirigido por el servidor (*Server Push*). En la técnica dirigida por el cliente, el *stream* es entregado al cliente cuando este lo solicita (o un pequeño instante después). La técnica dirigida por el servidor, se vale del hecho de que normalmente los clientes pueden tolerar cierto retardo entre que envían la petición y reciben los datos del vídeo. De esta manera, el servidor puede enviar, cada cierto tiempo predeterminado, los *streams* sin que existan peticiones de clientes.

Es posible optar por un esquema híbrido en donde se combinen ambas técnicas, por ejemplo, podría utilizarse un esquema dirigido por el cliente para los vídeos no muy solicitados y un esquema dirigido por el servidor para los vídeos más solicitados.

A continuación se describen los modelos de servicio dirigidos por el cliente y luego los dirigidos por el servidor.

Modelos de servicio dirigidos por el cliente

En esta técnica, el envío de *streams* de vídeo es dirigido por el cliente, es decir, el servidor no envía *streams* de vídeo sin que ellos hayan sido solicitados. Dentro de estos modelos de servicio es posible citar a:

- *Batching:* En esta técnica descripta por Dan et al. [27], las peticiones que arriban en un período de tiempo corto son agrupadas y servidas usando un solo canal de comunicación multitransmisión; de esta manera se provee un servicio N-VoD. Básicamente hay cuatro políticas de *batching*: FCFS, MQL, FCFS-n, y MFQ.

- *Virtual Batching:* En la técnica de *batching* es necesario que los clientes sean demorados para agruparlos en un solo canal. Sheu et al. en [28] propone la técnica de *virtual batching* que evita esta demora de la siguiente manera: cuando un cliente hace una petición, este es atendido inmediatamente y, cuando llega otra petición del mismo vídeo, el cliente anterior podría reenviarle la primera parte de la película que tiene almacenada en memoria, siempre y cuando la distancia de reproducción entre los clientes sea menor que el tiempo de reproducción de los datos almacenados en memoria por el cliente anterior. El problema aquí es que un cliente no debería tener la obligación de servir a otro cliente. Este es un caso de colaboración entre clientes, o esquema P2P.

- *Patching:* Hua et al. [29] propone la técnica denominada *patching* que provee un servicio de T-VoD. Este esquema trabaja de manera muy similar a *Virtual*

Batching. La diferencia radica en que la parte inicial del vídeo que un cliente necesita, no es entregada por otro cliente, sino que es el servidor el encargado de enviarle esos datos por un canal unitransmisión. De esta manera, no hace falta implicar a ningún cliente en transmisiones de vídeo a otro cliente. Al igual que *Virtual Batching*, se requiere que el espacio de almacenamiento del cliente pueda contener la diferencia de reproducción de los datos del vídeo. Hay algunas variantes de implementación que se pueden aplicar a esta técnica [30]:

- *Greedy Patching*: si el espacio de almacenamiento del cliente no es suficiente para almacenar la diferencia de reproducción, el resto del canal de multitransmisión se descarta y ese trozo de vídeo descartado se recibe por un canal unitransmisión.
- *Grace Patching*: si el espacio de almacenamiento del cliente no es suficiente para almacenar la diferencia de reproducción, entonces se transmite un nuevo *stream* por multitransmisión.

- *Controlled Multicast:* La técnica *Controlled Multicast* propuesta por Gao et al. [31] es similar a *Greedy Patching*, excepto que para poder unirse a un canal de multitransmisión, los clientes deben arribar dentro de un intervalo de tiempo T, una vez comenzada la transmisión. Si un cliente no puede unirse al canal de multitransmisión, entonces se crea un canal unitransmisión para enviarle la totalidad del vídeo. El intervalo T es determinado para minimizar el ancho de banda necesario, en término del promedio de canales usados.

Modelos de servicio dirigidos por el servidor

En esta técnica, el envío de *streams* de vídeo es dirigido por el servidor, el cual envía los vídeos cada cierto tiempo fijo, sin que los clientes los hayan solicitado. Algunos modelos de servicio de este tipo son:

- *Staggered Broadcasting:* Esta técnica, descripta en [32], es la primera que apareció dentro de esta clase de modelos de servicio. Consiste en la retransmisión del mismo vídeo por canales diferentes, en tiempos de inicio igualmente espaciados. La transmisión escalonada es simple de implementar, pero requiere un número relativamente alto de canales por cada vídeo, para poder alcanzar un tiempo de espera razonable. Considérese, por ejemplo, el caso de un vídeo que dura dos horas, que es el tiempo de duración promedio de una película. Para garantizar que ningún cliente tenga que esperar más de 5

minutos, habría que transmitir 24 copias diferentes del vídeo (24 canales de transmisión), comenzando cada 5 minutos.

- *Pyramid Broadcasting:* Viswanathan e Imielinski [33] propusieron en el año 1995, una solución mejor a la *Staggered Broadcasting*, denominada *Pyramid Broadcasting*, que asume que los clientes pueden recibir y guardar algunos segmentos de un vídeo mientras ven otros segmentos. Su transmisión piramidal transmite segmentos cada vez más largos de los vídeo en canales diferentes. Es más eficiente que el *Staggered Broadcasting* ya que la espera del cliente disminuye exponencialmente cuando el ancho de banda del servidor se incrementa. Este esquema requiere que el cliente disponga de un *buffer* (espacio de almacenamiento) de aproximadamente un 70 % de la longitud del vídeo y también requiere un gran ancho de banda del servidor.

- *Otros esquemas de Broadcasting*: El protocolo *Pyramid Broadcasting* ha sido seguido por numerosas propuestas, entre las cuales se pueden encontrar:

  - *Permutation-based pyramid broadcasting protocol* [34] de Aggarwal, Wolf y Yu,
  - *Skyscraper Broadcasting* [35] de Hua et al.,
  - *Fast Broadcasting* [36] de Li-Shen Juhn y Li-Ming Tseng,
  - *Pagoda broadcasting* [37] de Pâris, Carter y Long,
  - *Harmonic broadcasting* [38] de Li-Shen Juhn y Li-Ming Tseng; y sus variantes.
  - *Optimal broadcasting* [39] de Hung et al, un esquema reciente para sistemas de VoD móviles.

1.5.4. Planificador de Canales Lógicos

El planificador de canales lógicos (LCS, *Logical Channels Scheduler*), o también denominado planificador de *streams* (*Streams Scheduler*), es el encargado de decidir qué canal es servido en cada momento para garantizar la QoS. Es decir, el ancho de banda de salida de un servidor debe distribuirse entre los distintos *streams* de tal manera que los clientes (o receptores de esos *streams*) reciban la media con la calidad adecuada y en el tiempo correcto.

Los objetivos requeridos por un LCS de un servidor T-VoD son [40]:

- Optimización del aprovechamiento del ancho de banda disponible de conexión del servidor

- Distribución del ancho de banda de manera justa entre los *streams*
- Priorizar el servicio de las Unidades de Datos de Aplicación (ADU, *Application Data Unit*) de menor tiempo de expiración [41]. Una ADU es una unidad que contiene bloques lógicos de información que conforman el *stream*. Los bits correspondientes a un cuadro de vídeo forman una ADU, las cuales pueden ser asignadas a una colección de unidades de datos de red (tal como celdas ATM —Asynchronous Transfer Mode— o paquetes IP).

Es necesario un algoritmo que adapte su envío de datos a un determinado ancho de banda. El algoritmo debe contar con información del ancho de banda disponible para cada cliente (o grupo de clientes si se usan canales de multitransmisión), y del ancho de banda de salida del servidor.

En muchos algoritmos, la asignación de recursos de red es estática y determinada al inicio de la sesión, sin considerar variaciones del estado de la red que comunica con los clientes. Si se dispone de una red donde no se puedan reservar recursos, puede haber congestión con lo que el ancho de banda podría variar de un momento a otro. Para lograr adaptar el flujo al ancho de banda disponible, es posible utilizar dos tipos de estrategias: Anticipación y Control de Tasa Dinámico.

Anticipación

Las soluciones de anticipación son aplicables a vídeos prealmacenados (en el servidor) o que el cliente recibe con cierto retraso desde el momento de la generación de la señal de vídeo. De esta manera, es posible tomar ventaja de períodos en que el ancho de banda disponible de la red es elevado, para enviar por adelantado la media que el cliente consumirá más tarde, y así poder tolerar momentos en que la red o el servidor estén sobrecargados [41].

Las soluciones de anticipación adoptan políticas basadas en tiempo límite (*deadline based policy*). El propósito de estas políticas es que los *streams* compitan justamente, asegurando que ningún *stream* pueda monopolizar el ancho de banda disponible. Las prioridades de los *streams* dependen de cuan cercano esté el tiempo límite, es decir, tendrá prioridad aquel *stream* cuyo destinatario (o destinatarios si se utiliza multitransmisión) esté más próximo a quedarse sin vídeo para reproducir. Estas políticas también suelen denominarse de “Tiempo Límite Basado en Crédito” (DCB, *Deadline-Credit-Based*).

Control de Tasa Dinámico

Los mecanismos de control de tasa dinámico (*Dynamic Rate Control*) ajustan dinámicamente la tasa de transmisión de acuerdo a las condiciones de la red [42]. Lo que se hace es bajar la calidad del vídeo que se está transmitiendo para así tener que enviar menos datos al cliente.

Hay varios tipos de filtros que pueden ser utilizados para adaptar el flujo [43]:

- *Shaping Filters*: manipulan el vídeo codificado explotando la composición estructural de los flujos para adaptarse a las capacidades de QoS de la red o los clientes. Estos filtros son generalmente situados en el emisor del flujo, y requieren un poder de cómputo considerable. Ejemplos son el filtro DRS (*Dymanic Rate-Shaping*) y el filtro SBR (*Source Bit Rate*). Por lo general, se trabaja eliminando coeficientes DCT (*Discrete Cosine Transform*) o realizando nuevas cuantificaciones.
- *Selection Filters*: son usados para seleccionar sub-señales y descartar media. Por ejemplo, hay una técnica que codifica el vídeo en varias capas, donde la capa base ofrece la mínima QoS necesaria, y cuantas más capas se agregan más QoS se obtendrá. Si se quiere bajar la tasa de transferencia, entonces simplemente se descartan capas. Otra posible técnica sería la de distinguir los distintos tipos de cuadros de un vídeo, y realizar descartes basándose en la importancia de cada uno. Además, en el caso de que un paquete se pierda, podría decidirse si necesita ser retransmitido o puede ser descartado.

Las diferentes arquitecturas que pueden adoptarse para efectuar un control de tasa dinámico son [42]:

- Manejadas por el emisor (*sender driven*): Estos esquemas requieren que el emisor se encargue de resolver las fluctuaciones en el servicio que ofrece la red (ancho de banda y retardo), y ajustar sus transmisiones en consecuencia. En estos casos, es bastante común utilizar *Shaping Filter* para reducir la tasa de envío de tráfico. Este esquema tiene la dificultad adicional de determinar una única tasa de transmisión óptima en respuesta a la información enviada por clientes heterogéneos.
- Manejadas por el receptor (*receiver-driven*): Estos métodos especifican un mecanismo para cada receptor de manera tal que les permita seleccionar la transmisión de una calidad particular de acuerdo al servicio que esté recibiendo de la red. Por ejemplo, si se utiliza la técnica *Selection Filter*, codificando

el vídeo en varias capas, donde la acumulación de capas incrementa la calidad del vídeo, entonces, el servidor puede hacer envíos por multitransmisión de diferentes capas por canales diferentes. Cuando el cliente no nota congestión, entonces se suscribe a más canales para recibir más capas y, de esta manera, aumentar la calidad del vídeo visualizado; caso contrario, el cliente deja de recibir capas con el objeto de disminuir el tráfico en la red y así colaborar para solucionar la congestión.

- Orientadas a la red (*network-oriented* o *transcoder-based*): Esta estrategia coloca pasarelas (*gateways*) en lugares apropiados para entregar diferentes niveles de calidad a regiones de redes con diferentes tipos de conectividad o diferentes niveles de congestión.

1.5.5. Planificador del Tráfico de Red

La red puede perder paquetes, ya sea por congestión o por razones físicas (por ejemplo: ruido que afectan la transmisión de las señales, caída de enlaces, etc). Por esta razón, es necesario disponer de funciones, de más bajo nivel que el Planificador de Canales Lógicos, para administrar las comunicaciones al nivel de red de tal manera que pueda garantizarse la QoS. Este componente, el Planificador del Tráfico de Red (NTS, *Network Traffic Scheduler*), realimenta al Planificador de Canales Lógicos (LCS), informándole del estado de la comunicación entre el servidor y cada uno de los clientes. Si se detecta congestión en el camino de comunicación con un cierto cliente, el NTS le comunica al LCS que debe bajar la tasa de transferencia de ese canal lógico. Es necesario que el NTS y el LCS trabajen coordinada y cooperativamente para optimizar la solución.

Si se dispone de una red en la cual se puede reservar recursos, el trabajo del NTS sería mucho más simple, y hasta podría desaparecer.

1.6. Problemas, antecedentes y objetivo de la Tesis

Este trabajo está enmarcado en los sistemas de LVoD, con una prestación de servicio de T-VoD, en una red sin calidad de servicio como la red Internet. De esta manera, es necesario proveer al sistema de un mecanismo tolerante a fallos del servidor y de la red.

A continuación se plantea el problema que se necesita resolver, para luego presentar los antecedentes y el objetivo de la tesis.

1.6.1. Presentación y descripción del problema

El principal problema que presentan los sistemas de LVoD se debe a la restricción que presenta la comunicación en la red Internet. Como se ha discutido en 1.4.1, la red tiene un servicio sin garantías de entrega, donde el tráfico presenta fluctuaciones originadas por estados de congestión. La congestión puede producirse ya sea por un aumento del tráfico o por una falla física como la caída de enlaces o encaminadores. Cuando un enlace se cae, los algoritmos de encaminamiento derivan el tráfico hacia otros enlaces, pudiendo producir congestión en los enlaces que han recibido la derivación del tráfico. Un interesante trabajo [44], muestra el impacto de la falla de los enlaces en sistemas de voz sobre IP (VoIP). En él se encontró que, a pesar de la protección del encaminamiento IP, las caídas de enlaces y encaminadores fueron seguidas por largos períodos de inestabilidad en el encaminamiento de paquetes, produciendo múltiples descartes de los mismos a causa de reenvíos por caminos incorrectos. Por lo tanto, no solo hay fluctuaciones en el ancho de banda de las comunicaciones, sino que también es posible que por problemas de encaminamiento, un destino sea inaccesible.

Dadas las deficiencias que presenta la red de comunicaciones analizada, un sistema de LVoD, en su objetivo de prestar un servicio de alta calidad, deberá ser capaz de:

1. Detectar estados de congestión y ajustar la tasa de transmisión para darles solución.
2. Determinar el ancho de banda de comunicación con los clientes.
3. Planificar la entrega del contenido multimedia de acuerdo al ancho de banda disponible.
4. Detectar inconvenientes en la comunicación entre servidores y clientes, y con la debida antelación, tomar medidas que permitan continuar prestando un servicio de calidad.

Estas cuatro responsabilidades que deben cubrir los sistemas de LVoD, pueden ser enmarcadas en tres componentes. Las dos primeras responsabilidades son propias de un *planificador del tráfico de red*, mientras que la tercera y cuarta responsabilidad son tareas de un *planificador de canales lógicos* y un *módulo de garantía de la calidad del servicio*, respectivamente.

Otra restricción, que se encuentra al desarrollar un sistema que funcione en la red Internet, está relacionada con el tipo de transmisión disponible. Un sistema de

VoD que hace uso de comunicaciones multitransmisión, permite compartir recursos entre clientes que están reproduciendo el mismo vídeo. Así, puede reducirse la carga de los servidores y el uso del ancho de banda de la red. Sin embargo, como se ha expuesto en 1.4.1, el principal inconveniente es que la multitransmisión solo está disponible en una reducida porción de Internet. Para solucionar este problema, se ha optado por utilizar la unitransmisión, por ser el único tipo de comunicación soportado por la red entera.

1.6.2. Antecedentes

A continuación se presentan los antecedentes, encontrados en la literatura, respecto a los componentes, en un servidor de VoD, que resultan más involucrados debido a la ausencia de la calidad de servicio de la red de comunicación utilizada.

Planificador del tráfico de red

El control de congestión para comunicaciones unitransmisión ha sido estudiado por muchos años, no obstante, los protocolos existentes no son muchos si nos restringimos a protocolos TCP-*Friendly*[1] para transmisión multimedia en la red Internet. Jacob et al. [46] presenta un algoritmo de control de congestión similar a TCP excepto que este no hace retransmisiones. Cent et al. [47] propone el protocolo *Streaming Control Protocol* (SCP) para transmisión continua en tiempo real de *streams* multimedia por la red Internet. Dorgham Sisalem et al. describe el protocolo *Loss-Delay Adjustment Algorithm* (LDA) en [48] y su variante LDA+ en [49]. Sally Floyd et al. expone el protocolo *TCP-Friendly Rate Control* (TFRC) en [50], y más tarde, M. Handley et al. analiza este protocolo en el RFC 3448 [51]. Reza Rejaie et al. presenta en [52, 53] el protocolo *Rate Adaptation Protocol* (RAP), y un mecanismo de adaptación de calidad por capas para *streaming* de vídeo en Internet, y en un contexto de unitransmisión, en [54].

Sin embargo, ninguno de estos protocolos son diseñados para trabajar con un alto número de conexiones y una alta capacidad de transmisión implicando bajo consumo de recursos. TFRC es el único que presenta una implementación real, aunque es experimental y su rendimiento es inaceptable debido a la imprecisión del mecanismo de temporizadores que utiliza. Otros trabajos muestran resultados de

[1] Para ser utilizado en Internet, un algoritmo de control de congestión debe tener la propiedad de ser TCP-*Friendly*. Es decir, el uso de ancho de banda (en un estado estable) no debe ser mayor que el requerido por TCP bajo circunstancias similares [45]. En el capítulo 3 se cubre más ampliamente este concepto.

implementaciones pero realmente no describen la implementación y sus problemas derivados, tal como [55] y [56].

Debido a las características de un planificador de tráfico de red, el principal problema de su implementación es obtener un disparo preciso de los temporizadores con una granularidad fina. Este problema también afecta al software generador de tráfico (usado para generar patrones de tráfico). Generadores de tráfico como *TCPivo* [57], *iperf* [58], y *tcpreplay* [59], no tienen buenas soluciones para temporizaciones de granularidad fina en sistemas operativos de propósito general. Los problemas de *TCPivo* y *iperf* se deben a que ellos utilizan la espera ocupada (con alto consumo de CPU) para implementar los temporizadores, y *tcpreplay* necesita modificar el núcleo del sistema operativo.

Planificador de canales lógicos

Se han presentado varias propuestas para aplicaciones que envían media prealmacenada, y en especial para media continua prealmacenada "*semisoft*". Las aplicaciones "*semisoft*" tienen una baja tolerancia temporal inicial, y por lo tanto, solo una pequeña cantidad de información puede ser enviada por adelantado. Dentro de estos esquemas pueden citarse a [41], [60], y [61]. Sin embargo, no son apropiados para nuestros objetivos porque están diseñados para redes específicas donde los dispositivos activos de comunicación incluyen capacidades adicionales y particulares para manejar el tráfico. Otros esquemas tienen el problema de que la asignación de recursos de red es estática y es determinada al inicio de la sesión, tal es el caso de [62].

Nuestro grupo de investigación ha propuesto en [63, 64] el algoritmo CB_MDA, sin embargo está ideado para un entorno de red con reserva de recursos, y no permite trabajar con vídeos del tipo VBR (*Variable Bit Rate*). Otras propuestas se han diseñado para adaptarse a las condiciones variables de la red Internet, pero disminuyen la calidad del vídeo ante estados de congestión, tal es el caso de [54].

Módulo de garantía de la calidad del servicio:

Las soluciones actuales de sistemas de LVoD se centran fundamentalmente en el alto nivel de gestión, es decir, en los *modelos de servicio*, y prácticamente sin considerar que el modelo de servicio no funcionará de manera adecuada si no tiene el soporte de un buen *planificador de canales lógicos y de red*. Es decir, pocos son los trabajos orientados a la tolerancia a fallos de la red. No obstante, hay mucha investigación con respecto a la tolerancia a fallos de servidores, entre los que se puede mencionar a [65].

Algunos dan soluciones adecuadas para *streaming* pero inadecuadas para VoD, tal es el caso de [66, 18, 67, 68]; además, la mayoría de estos trabajos no describe el mecanismo de detección de fallos de la red. En [69] se presenta un mecanismo de detección de fallos de red que aclama ser para VoD. En realidad esta orientado a *streaming*, debido a que, en VoD, la media no necesariamente es regulada a la velocidad de su reproducción, por lo tanto el cliente no dispone de suficiente información para determinar cuándo hay un fallo de red o cuándo el servidor ha decidido suspender momentáneamente su servicio mientras atiende a otros clientes de mayor urgencia. Algo similar ocurre con [70], quien utiliza un mecanismo de *heartbeat* para proveer tolerancia a fallos de la red en un sistema de VoD. Mediante este esquema, se puede detectar la pérdida total de comunicación y el incremento del retraso de los paquetes pero, sin embargo, no permite obtener suficiente información de la red, imposibilitando tomar medidas apropiadas.

También se ha propuesto un mecanismo de continuación de servicio para conexiones TCP [71], de suma importancia en comercio electrónico, pero con requerimientos totalmente diferentes a los de un sistema de VoD.

1.6.3. Objetivos de la Tesis

Las carencias detectadas en los trabajos encontrados, referentes a los tres componentes que desempeñan un rol fundamental en el éxito de este tipo de sistemas de VoD motivan el desarrollo de esta tesis, cuyo objetivo principal es la propuesta de:

> **Una arquitectura de un sistema de LVoD distribuido, que permite ofrecer un servicio de T-VoD con comunicaciones unitransmisión sobre la red Internet. Esta nueva arquitectura, denominada VoD-NFR *(Video-on-Demand with Network Fault Recovery)* tiene como fin garantizar, ante fallos de la red o caídas de servidores, la entrega del contenido multimedia a los clientes sin disminuir la calidad de los mismos y sin sufrir interrupciones durante su visualización.**

El sistema VoD-NFR no tiene la finalidad de presentar un nuevo modelo de servicio, sino contribuir con los sistemas de LVoD en el aumento de la capacidad de adaptación a las variaciones de los anchos de banda de las comunicaciones, y la detección y tratamiento de los fallos de la red. Por lo tanto, sin pérdida de generalidad, se asume para este sistema un modelo de servicio simple que permite concentrar

los esfuerzos en los componentes de interés. Asimismo, VoD-NFR intenta presentar un modelo de las relaciones generales de componentes en un sistema de LVoD que pueda ser implantado en otras arquitecturas y modelos de servicio, desde esquemas centralizados, hasta distribuidos y P2P.

Para alcanzar las metas propuestas, fue necesario desglosar el objetivo general en subobjetivos que permiten abordar el problema más eficazmente y de una manera incremental. Son estos subobjetivos los que se detallan a continuación.

Desarrollo del planificador del tráfico red

La base de un sistema de LVoD es el planificador de red, sin este componente todos los esfuerzos para planificar la entrega de los vídeos a un alto nivel se vuelven ineficaces e ineficientes. Debido a que es el último componente que entra en acción en el proceso de envío de la información de los vídeos desde el servidor hasta el cliente, el primer objetivo de esta tesis consiste en diseñar y desarrollar el planificador del tráfico de red. Este planificador debe ser capaz de: 1) adaptarse a los estados de congestión de la red de una manera TCP-*Friendly*, 2) generar información del estado de las comunicaciones con los clientes, y 3) extremar el ahorro de recursos para soportar una elevada carga de trabajo.

Desarrollo del planificador de canales lógicos

Luego de haber desarrollado el planificador del tráfico de red, el segundo objetivo de esta tesis se basa en diseñar y desarrollar el planificador de canales lógicos que distribuya los contenidos a los clientes teniendo en cuenta el estado de las comunicaciones entre los servidores y los clientes. Específicamente, este planificador debe ser capaz de: 1) utilizar vídeos VBR, 2) adaptarse dinámicamente al ancho de banda disponible de las comunicaciones, 3) no degradar la calidad del vídeo para adaptarse a un ancho de banda menor al requerido por el vídeo, y 4) distribuir equitativamente el vídeo a los clientes priorizando a aquellos con mayor necesidad de media.

Desarrollo del módulo de garantía de la calidad de servicio

Una vez desarrollados el planificador del tráfico de red y el planificador de canales lógicos, el tercer y último objetivo de esta tesis consiste en crear un módulo de garantía de la calidad de servicio, capaz de: 1) detectar inconvenientes en

la comunicación entre servidores y clientes, y 2) con la debida antelación, tomar medidas que permitan continuar prestando un servicio sin pérdida de calidad.

Mantener el servicio ante problemas de comunicaciones obliga a buscar soluciones mediante la colaboración del resto de los servidores del sistema que permitan tomar vías de comunicación alternativas para enviar los vídeos a los clientes. Ante una situación de estas características, la cantidad de fallos recuperables no solo depende de cuan bueno sea el mecanismo de detección de fallos y el proceso de recuperación, sino que también depende de los siguientes cuatro factores: la infraestructura de red, la distribución de los servidores dentro de la topología de la red, la distribución de los contenidos multimedia en los servidores, y la política de selección de servidores alternativos que es aplicada cuando el cliente debe abandonar el servicio prestado por su servidor principal. No obstante, ninguno de estos últimos cuatro factores se consideran objeto de estudio de la presente tesis.

Capítulo 2

El sistema VoD-NFR

En base al marco teórico dispuesto en el capítulo anterior sobre los sistemas LVoD, en este capítulo se presenta la nueva propuesta, el sistema VoD-NFR. Se comienza analizando las características y arquitectura del sistema, para luego realizar una descripción funcional que permite comprender los detalles de su diseño con respecto a, entre otras cosas, el tratamiento de comandos interactivos, el proceso de entrega de los vídeos, y la tolerancia a fallos.

Es significativo que ninguno de los autores consultados haya destacado la importancia de un problema que consideramos decisivo: el tratamiento de los fallos de la red para garantizar la QoS de un sistema de VoD. Por esta razón, hemos desarrollado el sistema VoD-NFR (*Video-on-Demand with Network Fault Recovery*), un sistema de vídeo bajo demanda que tiene la habilidad de recuperarse ante fallos de red y caídas de servidores.

El sistema VoD-NFR dispone de un esquema innovador e integral en relación al tratamiento de fallos producidos en la red. Está diseñado especialmente para proteger al cliente de posibles interrupciones durante la visualización del contenido multimedia. Además, se persiguen otros objetivos, no menos importantes pero sí más frecuentes en la literatura, como son la alta escalabilidad, la tolerancia a fallos de servidores, el alto rendimiento, y el bajo coste. Soporta, además, un servicio T-VoD, con un repertorio completo de funciones interactivas, y un tiempo de respuesta pequeño, proporcionando al usuario de una agradable experiencia en la visualización de contenidos multimedia.

Este sistema, diseñado para funcionar en la red Internet, utiliza la unitransmisión debido a los inconvenientes de utilizar la tecnología multitransmisión en esta red (ver detalles en sección 1.4.1). VoD-NFR es totalmente distribuido y no hace uso de esquemas P2P, no obstante, podría extenderse para soportar este tipo de arquitectura.

Se supone una distribución previa de contenidos entre los servidores, con cierta replicación para evitar cuellos de botella y acercar los contenidos a los clientes. Es también factible la implantación de esquemas *proxies*, sin embargo está fuera del alcance de este trabajo.

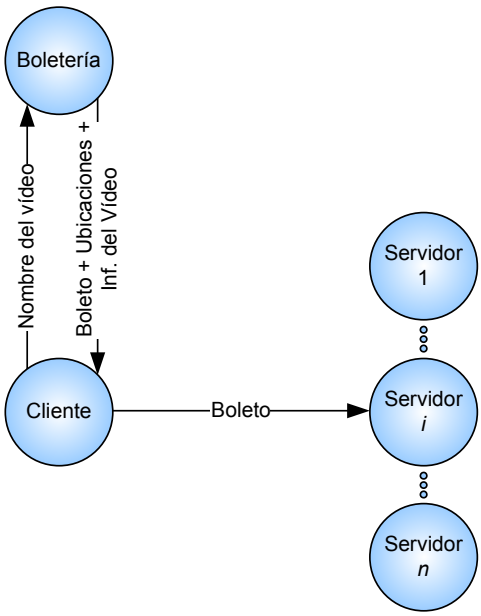


Figura 2.1: Arquitectura de alto nivel del sistema VoD-NFR

En la sección 2.1 se presenta la arquitectura de alto nivel del sistema VoD-NFR, la arquitectura de los servidores y clientes, mientras que en las secciones sucesivas se describen las características funcionales del sistema.

2.1. Arquitectura

La arquitectura del sistema VoD-NFR es mostrada en la figura 2.1. Hay tres componentes que intervienen en una transacción completa que involucra el alquiler y visualización de un contenido multimedia. Ellos son: la Boletería, el Cliente, y los Servidores. Básicamente, el cliente hace una petición de un vídeo al componente boletería. Luego de efectuar el pago, la boletería le entrega el boleto (que especifica, entre otras cosas, el tiempo de expiración del alquiler), y con este boleto el cliente puede dirigirse a un servidor para solicitar la visualización del vídeo.

La comunicación entre el cliente y el servidor se efectúa a través de dos canales de comunicación: un canal de control del *stream* (SCC, *Stream Control Channel*), y un canal dedicado a la transmisión pura del contenido multimedia. El SCC utiliza la comunicación TCP, mientras que el contenido multimedia es transmitido a tra-

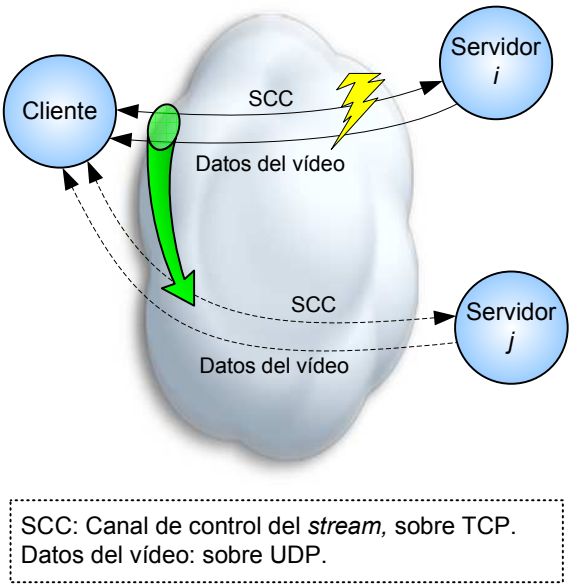


Figura 2.2: Esquema de migración de servicio efectuado desde el servidor *i* al *j*

vés del protocolo UDP. Estos canales de comunicación tienen definidos protocolos propios que se sitúan sobre los protocolos TCP y UDP mencionados.

Si durante la sesión de visualización de un vídeo, se encuentran problemas de comunicación entre ese cliente y ese servidor, el cliente automáticamente contactará a otro servidor que pueda hacerse cargo del servicio, todo esto de forma transparente para el usuario. La migración del servicio, de un servidor a otro, puede ser dirigida por el servidor o por el cliente afectado. En el primer caso, el servidor, cuando detecta inconvenientes en la comunicación, ordena al cliente la migración. En el segundo caso, cuando la comunicación entre el cliente y el servidor está totalmente interrumpida, la orden del servidor no puede llegar, por lo tanto el cliente es el encargado de detectar el problema y migrar. En ningún caso existe una comunicación directa entre servidores, siempre se efectúa a través del cliente. En la figura 2.2 se muestra un ejemplo donde se representa el esquema de migración; en este caso, el servicio brindado al cliente es migrado desde el servidor *i* al *j*.

El problema que ahora se encuentra está en determinar a qué servidores contactará el cliente para ser servido y cuál servidor será el más adecuado. Para resolver esto, el componente boletería tiene una tabla que especifica, para cada vídeo, los servidores donde se encuentra disponible y la tasa de bits máxima y promedio requeridos para visualizarlo. Además, cada cliente tiene una tabla que almacena los anchos de banda de comunicación con los servidores más próximos. De esta mane-

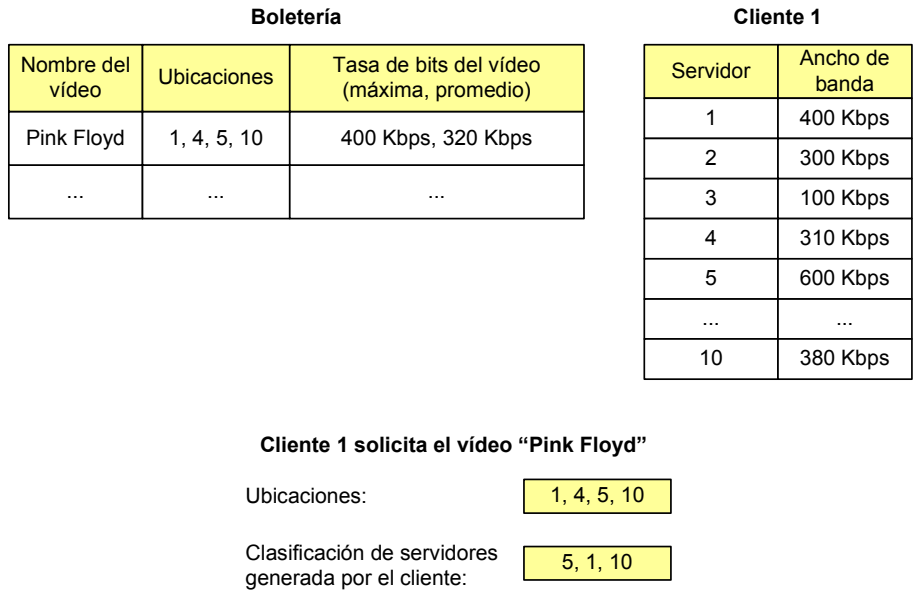


Figura 2.3: Ejemplo de solicitud de un vídeo efectuada por un cliente a la boletería

ra, cuando un cliente solicita un vídeo, la boletería le retorna el boleto, los posibles servidores que puede contactar, y la tasa de bits máxima y promedio del contenido. El cliente, entonces, utilizando la información que almacena en su tabla, establece una clasificación de servidores a los que puede contactar, ordenados de mayor a menor ancho de banda de conexión. En la figura 2.3 se muestra un ejemplo de cómo el cliente 1 determina la clasificación de servidores que pueden prestarle servicio. El cliente 1 quiere visualizar el vídeo “Pink Floyd”, disponible en los servidores 1, 4, 5 y 10. Dados los requerimientos de este vídeo, con una tasa máxima y promedio de bits de 400 y 320 kb/s respectivamente, el cliente 1 determina la clasificación de servidores: 5, 1 y 10. Como se observa en la tabla del cliente, el servidor 5 es el de mejor comunicación con este cliente, y el servidor 10 el de peor comunicación. El ancho de banda de estos tres servidores supera a la tasa promedio requerida para el vídeo, razón por la cual fue descartado el servidor 4. Note que el ancho de banda del servidor 10 (380 kb/s) no soporta la tasa máxima de bits del vídeo (400 kb/s), lo que en algunos casos podría perjudicar la correcta visualización del contenido.

La implementación del componente boletería no se describe en este documento, puesto que no presenta mayores dificultades. Cabe mencionar, que este componente podría estar distribuido, con lo cual se utilizaría una base de datos distribuida, evitando que este componente se vuelva un cuello de botella del sistema.

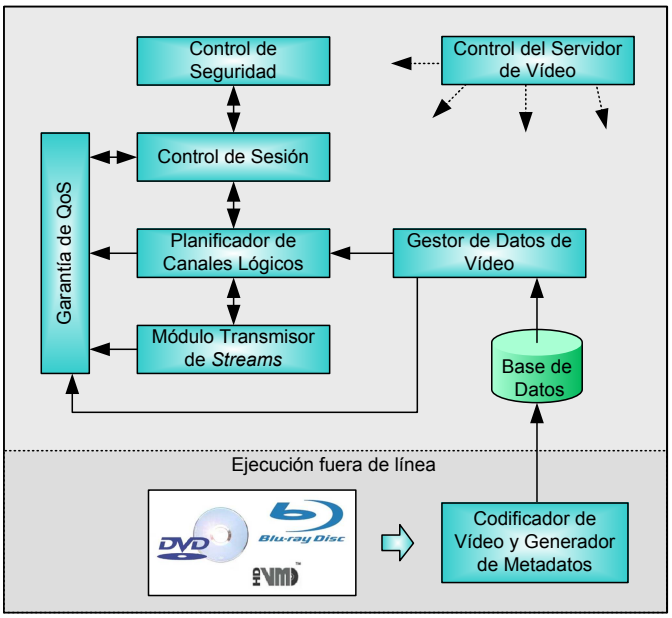


Figura 2.4: Arquitectura del servidor de vídeo

2.1.1. Arquitectura de los servidores

En todo sistema de VoD, el servidor cumple un rol fundamental en el éxito de proveer un servicio de VoD de alta calidad. Por esta razón, un gran esfuerzo se dedica al diseño de cada uno de los componentes internos del servidor. En la figura 2.4 pueden observarse, a nivel funcional, cuáles son los bloques que componen el sistema y cómo interactúan.

Control del servidor de vídeo

Este componente, de reducidas dimensiones, tiene la responsabilidad de coordinar la puesta en ejecución del servidor de vídeo, y proveer una interfaz de comunicación centralizada para el administrador del sistema.

Desde él se puede configurar todos los aspectos del servidor de vídeo, que abarcan a los demás componentes del servidor. Cuando llega una orden de iniciar el servidor, este componente pone en marcha cada uno de los módulos, controlando que la ejecución de ellos sea correcta.

Codificador de Vídeo y Generador de Metadatos

La funcionalidad de este módulo es llevada a cabo totalmente fuera de línea. Su responsabilidad es la de codificar el vídeo para su transmisión, y la generación de metadatos, es decir, información que acompaña a cada uno de los vídeos que es de suma utilidad para que los demás módulos del sistema lleven a cabo sus funciones. La generación de metadatos, que caracterizan a cada uno de los vídeos, involucra tareas como el análisis de las marcas de tiempo de los cuadros de los vídeos y su ubicación dentro de los archivos de vídeo.

Gestor de Datos de Vídeo

La responsabilidad del gestor de datos de vídeo (VDM, *Video Data Manager*) es la de administrar el catálogo de vídeos del servidor, proporcionando una interfaz de acceso a los datos de los vídeos, ya codificados, para los demás módulos del sistema. Además, provee información relativa a los vídeos (como longitud, tasa de bits, fps, etc.), y una correspondencia entre posiciones (número de byte) dentro de los ficheros de vídeo y valores de tiempo de reproducción. Una de las utilidades de esta "correspondencia" puede observarse con el siguiente ejemplo: si durante la visualización de un vídeo, se desea hacer un salto al tiempo x, habrá que buscar el primer byte del cuadro de vídeo cuya marca temporal sea x, y comenzar la transmisión de datos del vídeo a partir de ese byte. Toda esta información adicional al contenido multimedia específico es extraída de los metadatos generados fuera de línea por el componente "Codificador de Vídeo y Generador de Metadatos".

Planificador de Canales Lógicos

El Planificador de Canales Lógicos (LCS, *Logical Channels Scheduler*) es el encargado de decidir qué canal lógico es servido en cada momento para garantizar la QoS. Es decir, es el responsable de distribuir el ancho de banda del servidor entre los distintos canales, respetando las necesidades de media de cada uno de ellos. El LCS utilizado en este sistema es del tipo Anticipación (ver sección 1.5.4), el cual permite adaptarse a las fluctuaciones del ancho de banda de comunicación, y mantener así la calidad del vídeo entregado al cliente a lo largo de toda su reproducción. Como el modelo de servicio utilizado es de comunicaciones unitransmisión, cada petición de un vídeo es asignada a un único canal lógico, y cada canal lógico es utilizado para servir una única petición. Es decir, hay una correspondencia de uno a uno entre peticiones y canales lógicos.

El LCS define un mecanismo que prioriza la atención de clientes, basándose tanto en la necesidad de media como en el estado de reproducción. De esta manera, es posible diferenciar a un cliente, que se encuentra en un estado de reproducción normal, con un cliente que se encuentra en una reproducción rápida o, más aún, distinguir entre un cliente que está almacenando media en el *buffer* que comenzará a reproducir unos instantes después, y un cliente que ya está visualizando la media. Para esto, es necesario que el LCS mantenga continuamente el estado de reproducción de cada cliente.

Módulo Transmisor de *Streams*

El módulo transmisor de *streams* (STM, *Stream Transmission Module*) es el módulo más complejo y delicado que permite al resto de los componentes del servidor hacer su trabajo correctamente. El STM, componente base del sistema VoD-NFR, tiene como objetivo proporcionar un transporte de los datos de los vídeos del servidor a los clientes, con soporte de servicio T-VoD, y también de proveer información del estado de las comunicaciones.

El STM incluye un Planificador del Tráfico de Red (NTS, *Network Traffic Scheduler*) para cumplir con parte de sus responsabilidades. El NTS se encarga del envío de paquetes por la red, utilizando un protocolo específico para controlar la congestión, y de generar datos sobre el estado de las comunicaciones con cada uno de los clientes. El STM aumenta la funcionalidad del NTS, mediante un protocolo de capa superior, que se ocupa de mantener la pista del origen y sección de media enviada, para que el cliente pueda clasificar los datos recibidos.

Un servicio T-VoD requiere de delicadas acciones del STM para poder responder a las peticiones de comandos interactivos en tiempos muy cortos. Para soportar este requerimiento, el STM fue dotado de un mecanismo especial que mantiene un control total de los paquetes encolados en el sistema para ser enviados a los clientes. Estos paquetes encolados, son paquetes que han sido entregados al STM para ser transmitidos. Sin embargo, cuando se solicita un comando interactivo, es posible que ellos ya no sirvan, con lo cual el STM los descarta, provee un informe de ello, y transmite el nuevo contenido solicitado.

En el caso de requerirse una migración de una sesión de un cliente, el STM también es capaz, gracias a su refinado control de paquetes, de informar las secciones de los vídeos que no han sido entregados al cliente, para que sean reenviados por el nuevo servidor.

Control de Seguridad

El componente de Control de Seguridad (SecC, *Security Control*) se encarga de autenticar y autorizar al cliente en la visualización de un determinado vídeo. El componente SecC puede autorizar, denegar, o cancelar la visualización del contenido multimedia a un cliente. El proceso de autenticación tiene que ver con comprobar si el cliente es quien dice ser, y el proceso de autorización tiene que ver con verificar que el cliente está dentro del tiempo permitido para visualizar el contenido multimedia, o si el tiempo ya ha expirado.

Control de Sesión

El componente Control de Sesión (SC, *Session Control*) es responsable de llevar a cabo las comunicaciones a través del SCC con los clientes. Estas comunicaciones involucran la atención de: peticiones de visualización de vídeos, migración de vídeos, y solicitud de comandos interactivos. Además, se encarga de coordinar las actividades de los demás componentes del servidor para resolver los requerimientos que surgen de las comunicaciones con los clientes.

Cuando el SC recibe una petición de visualización de vídeo, consulta al módulo Garantía de QoS que le indica si es posible aceptar este nuevo cliente o si debe ser rechazado, y también consulta al módulo de Control de Seguridad para autenticar y autorizar la reproducción. Si estos dos componentes aceptan la reproducción, entonces se crea una nueva sesión, y se le asigna un canal lógico para entregarle la media al cliente. Si hay un rechazo de parte del módulo Garantía de QoS, podría utilizarse un mecanismo para negociar con el cliente una menor QoS de la media para que la petición pueda finalmente ser atendida. La recepción de una solicitud de comandos interactivos es comunicada al LCS para que envíe la nueva media solicitada. Cuando la sesión termina, este componente coordina la liberación de recursos que cada uno de los módulos del servidor tenía dedicados a la atención de este cliente.

Este componente, es el encargado de coordinar todas las acciones pertinentes para efectuar la recuperación de un fallo en el sistema. Además, también detecta ciertos fallos básicos de la red, por medio de un análisis efectuado en el canal de comunicación SCC.

Garantía de QoS

El componente Garantía de QoS (QoSA, *Quality of Service Assurance*) tiene tres responsabilidades fundamentales: aceptar o rechazar solicitudes de vídeos de

acuerdo a la disponibilidad de recursos del servidor, balancear la carga del sistema, y detectar los fallos de comunicación con el cliente para luego tomar las medidas apropiadas que permitan garantizar la QoS de la entrega de los vídeos.

La aceptación de una petición de un nuevo cliente, significa que ha sido aceptado por el balanceador de carga y también que el servidor tiene recursos disponibles para prestarle servicio. Este componente garantiza que la aceptación de la petición no compromete el buen funcionamiento del servidor, lo que significa que la QoS de la media entregada a los clientes que ya habían sido aceptados para ser servidos, no será afectada en lo más mínimo.

El módulo QoSA recibe constantemente información del estado de las comunicaciones por parte del Módulo Transmisor de *Streams*, así como también el estado de reproducción del cliente que lo provee el Planificador de Canales Lógicos. Con esta información, el componente QoSA, decide si una determinada comunicación con un cliente es suficiente o no, para continuar con la prestación del servicio. En caso de no serlo, se efectúa la migración del servicio de un servidor a otro. Sin embargo, la comunicación es siempre a través del cliente, y no entre servidores. El esquema de migración cuenta con un mecanismo de preaviso, que efectúa una reserva de recursos en otro servidor, para que, en caso de que se haga efectiva la migración, el cliente no tenga que perder tiempo en buscar otro servidor que pueda atenderlo.

2.1.2. Arquitectura de los clientes

En la figura 2.5 se muestra un diagrama de bloques de la estructura del cliente, y sus interacciones. Su arquitectura se basa en el patrón, desarrollado por Trygve Reenskaug, denominado Modelo-Vista-Controlador (MVC, *Model-View-Controller*) [72]. Este patrón separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos: el modelo, la vista, y el controlador. En este caso, el "modelo" incluye al componente Receptor del *Stream*, al Gestor de Datos del Vídeo y al Control de Sesión, la "vista" es el módulo Interfaz, y el "controlador" es el módulo Control de la Interfaz.

Receptor del *Stream*

El componente Receptor del *Stream* es el responsable de recibir la media enviada por el Módulo Transmisor de *Streams* del servidor. En este componente se encuentran implementados los mismos protocolos de comunicaciones que utiliza su par en el servidor. Cada paquete que recibe es entregado al componente Control del Vídeo.

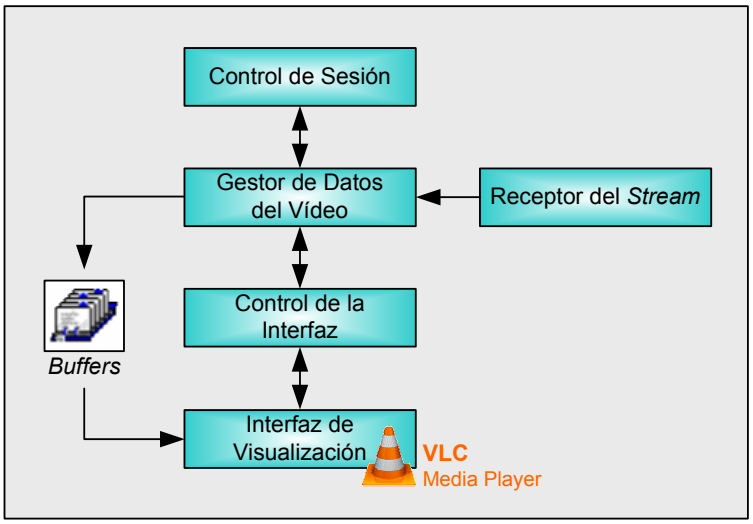


Figura 2.5: Arquitectura del cliente de vídeo

Control de Sesión

Este componente se ocupa tanto de la administración del canal de control SCC como de la ejecución de las migraciones (recuperación de fallos). La conexión con un servidor determina el establecimiento de un nuevo canal SCC. Por este, se transmiten nuevas peticiones de visualización de vídeos, solicitudes de comandos interactivos, y todo lo referente al proceso de las migraciones.

Gestor de Datos del Vídeo

La función principal de este componente es la de gestionar el almacenamiento del contenido multimedia que se ha recibido desde el servidor. Para esto es necesario almacenar en *buffers* las secciones de media recibidas haciéndolas, de esta manera, disponibles para el reproductor multimedia que se encarga de su visualización. El Gestor de Datos del Vídeo, además de guardar los datos en *buffers*, indica el momento en que el cliente puede comenzar a reproducirse la media porque ya se encuentra almacenada la cantidad mínima estipulada con el servidor.

La gestión de datos del vídeo involucra también la atención de solicitudes de comandos interactivos. Cuando surge una solicitud, inmediatamente se transmite al servidor utilizando como intermediario al componente Control de Sesión. De esta manera, el servidor dejará de transmitir la media actual y enviará la nueva. Cuando

se disponga de suficiente media se indicará, a la interfaz, iniciar la visualización del contenido.

Control de la Interfaz

El módulo Control de la Interfaz es el encargado de controlar todos los aspectos genéricos de una interfaz que visualice el contenido multimedia. Así, se independiza la lógica de control con respecto a una interfaz particular de visualización. Este componente recibe las instrucciones del usuario desde la interfaz de visualización, y las envía al Gestor de Datos del Vídeo. No existe comunicación alguna entre la Interfaz de Visualización y el Gestor de Datos del Vídeo, toda esa comunicación se realiza a través de este componente de control de la interfaz. Este módulo podría, por ejemplo, inhibir una orden de avance rápido cuando un vídeo no lo soporta, o, cuando se está en medio de un proceso de migración, dar la orden a la interfaz gráfica de deshabilitar todas las funciones interactivas.

Interfaz de Visualización

Este componente se encarga de visualizar el vídeo, y de proporcionar un método para que el usuario pueda invocar la ejecución de comandos interactivos. Para este sistema, se ha optado por realizar una interfaz gráfica que, haciendo uso del *plugin* (componente incrustado) del reproductor VLC [73], se muestra el vídeo y un conjunto de botones que permiten ejecutar todas las acciones interactivas como son: iniciar, parar, pausa, reanudar, saltos hacia adelante y atrás, reproducción rápida hacia adelante y hacia atrás con soporte de múltiples velocidades.

2.2. Aspectos de seguridad del sistema y de los datos

Cuando el sistema genera un boleto, este se encripta y entrega al cliente. De esta manera, nadie que se apodere del boleto puede modificar su contenido. La autenticación se lleva a cabo a través del almacenamiento, en el boleto, de la dirección IP autorizada para recibir el vídeo. Si el boleto fuese compartido o capturado por otras personas, las mismas no podrán acceder ya que su IP será diferente a la del boleto. Con fines de autorización, en el boleto se almacena el identificador del vídeo alquilado y el tiempo de expiración. El identificador del vídeo garantiza que un boleto solo sirva para la película adquirida en la boletería. El tiempo de expiración determina el instante de tiempo en que el boleto caducará.

Los controles de autorización no solo se llevan a cabo cuando el cliente hace una solicitud de visualización, sino que también se realizan durante todo el transcurso del tiempo. A intervalos determinados, se verifica que los clientes que ya están siendo atendidos por el servidor, aún estén autorizados.

2.3. Los metadatos

Se denomina "metadatos" a los datos que acompañan y describen el contenido de los vídeos. Es necesario disponer de esta información, no solo para que el LCS planifique correctamente el envío de los datos de los vídeos, sino también para calcular los recursos del sistema que consumirán los *streams* y así poder tomar la decisión de aceptar o rechazar las nuevas peticiones.

Existen dos métodos de compresión de datos de vídeo, que se diferencian en la variabilidad de la tasa de bits. La tasa de bits es la relación de bits por segundo que se lee del archivo de vídeo (o audio) a efectos de su reproducción. Uno de los métodos se basa en una tasa de bits constante (CBR, *Constant Bit Rate*) y el otro método en una tasa de bits variable (VBR, *Variable Bit Rate*).

CBR: El método CBR se trata de un método de compresión de datos utilizado en la codificación de audio y vídeo que conserva constante la tasa de bits en todo el archivo.
Uno de los principales métodos de compresión (MPEG) se basa en, además de comprimir la imagen fija, guardar los cambios entre un cuadro (o cuadros) y el siguiente (o siguientes). Aunque la tasa de bits sea escasa, no habrá problemas de calidad en escenas con poco movimiento y pocos cambios de imagen entre cuadro y cuadro. El problema llega con escenas de acción en las que la cámara se mueve con rapidez y un cuadro es muy diferente (o totalmente diferente) del anterior o del siguiente. En ese caso, el ancho de banda necesario para guardar los cambios entre cuadro y cuadro crece considerablemente y queda menos espacio para comprimir la imagen, deteriorándola notablemente, tanto más cuanto menor sea la tasa de bits.
Su principal ventaja es la predicción del tamaño final del archivo en función de la duración del mismo. Su principal inconveniente es la poca eficiencia que presenta. CBR otorga la misma capacidad de información para los fragmentos complejos que para los no complejos (silencios, imágenes estáticas, etc.), por lo tanto, se desaprovecha capacidad y se obtiene un archivo de mayor tamaño que el necesario.

VBR: El método VBR es un método de compresión de datos que prioriza la calidad del vídeo, por lo tanto la cantidad de datos a asignar depende de la complejidad de la secuencia de audio o vídeo que se está codificando. Lo que se hace es otorgar una cantidad de bits que se corresponda con lo que realmente haga falta.

En los vídeos, hay escenas donde la cámara está fija, hay poca luz, y poco movimiento. Hay otras escenas donde la cámara se mueve, hay movimientos bruscos de los objetos de la escena, etc. Lo que hará el VBR, en este caso, es ahorrar bits en la representación de la escena lenta para aplicarlos después a la escena rápida y conseguir así que esta última se vea lo mejor posible. En el audio sucede lo mismo, es distinto comprimir la información acústica de un pasaje en silencio o de uno con muchos instrumentos sonando simultáneamente.

Así, el VBR otorga la tasa de bits necesaria a cada parte del archivo, ya sea de audio o de vídeo, consiguiendo una calidad mayor en archivos de un tamaño reducido.

El LCS necesita saber cuántos bytes deben ser enviados por cada segundo de un determinado vídeo. Esto es trivial si se considera el método de compresión CBR, pero no lo es para los métodos VBR donde los segundos son codificados con una cantidad variable de bytes. Por lo tanto, es necesario generar, fuera de línea, un archivo de información para cada vídeo, que tenga una tabla donde se indique el primer byte en el archivo del vídeo que corresponde a la reproducción de cada uno de los segundos del vídeo.

Veamos el siguiente fragmento de un archivo que presenta una tabla de información de un vídeo:

```
1 155844 155844
2 495148 339304
3 869312 374164
4 1034316 165004
5 1355028 320712
6 1671092 316064
```

La primer columna representa el número de segundo de reproducción, la segunda columna es la posición en el archivo de vídeo, del último byte que corresponde al número de segundo que indica la primer columna, y la tercer columna indica la cantidad de bytes que corresponden a la reproducción de ese segundo.

Para generar esta información, se ha utilizado una librería llamada libmpeg2 [74] que decodifica vídeo MPEG, y que es utilizada por muchos de los reproductores más conocidos como Xine, VideoLan, o MPlayer.

Veamos el siguiente ejemplo. Para reproducir el segundo número 3 hacen falta 374.164 bytes, comprendidos del byte número 495.149 al 869.312 del archivo de vídeo. Si se necesita saber cuántos bytes se requiere para reproducir desde el segundo 3 al segundo 3,5, se puede suponer que dentro de cada segundo hay una tasa constante de bits (CBR). De esta manera, la reproducción será desde el byte número 495.149 al 682.230 ($495.149 - 1 + 374.164/2$). Esta suposición, que evita el almacenamiento de la información de cada cuadro del vídeo, ahorra espacio en memoria cometiendo un error despreciable.

A continuación, se presenta un método para simplificar o reducir el tamaño de estos archivos de información de los vídeos y, más adelante, se definen las funciones de traducción *sec2Byte* y *byte2sec*, que obtienen sus datos de estos archivos de información.

2.3.1. Simplificación de los archivos de información de los vídeos

Si consideramos que un vídeo normal tiene aproximadamente 100 minutos, necesitaríamos en memoria una estructura de 6.000 entradas (1 entrada por segundo) donde cada entrada es la segunda columna del archivo de información. Sin embargo, es posible aplicar algunas técnicas para ahorrar recursos de memoria.

Supongamos que se representa la tabla del archivo de información del vídeo mediante una curva, donde el eje x es la primer columna, y el eje y es la segunda columna, y donde la curva define la función $y = f(x)$. Para ilustrar el comportamiento de una curva de un vídeo VBR, en la figura 2.6 se presenta una curva no real de un supuesto vídeo.

Una de las técnicas que suele utilizarse para este tipo de problemas, es la de obtener una *función aproximada con polinomios* que represente la curva de información del vídeo. Sin embargo, esta opción no es buena cuando la curva no es lo suficientemente simple. Esta estrategia, llevada a cabo utilizando el método de mínimos cuadrados, genera como resultado unos desvíos enormemente altos con respecto a la curva original, aún utilizando grados de polinomios muy grandes (en el orden de los 40). Por lo tanto, esta técnica resulta inaceptable para resolver el problema planteado.

Otra forma de resolver el problema, es realizar una *selección de puntos fundamentales de la curva*, y trazar rectas entre esos puntos. La idea es minimizar el error de ajuste a la curva original, con un margen de error predeterminado y haciendo uso de la menor cantidad de puntos. La figura 2.7 ilustra este concepto, donde se resaltan los puntos fundamentales y las rectas que los unen.

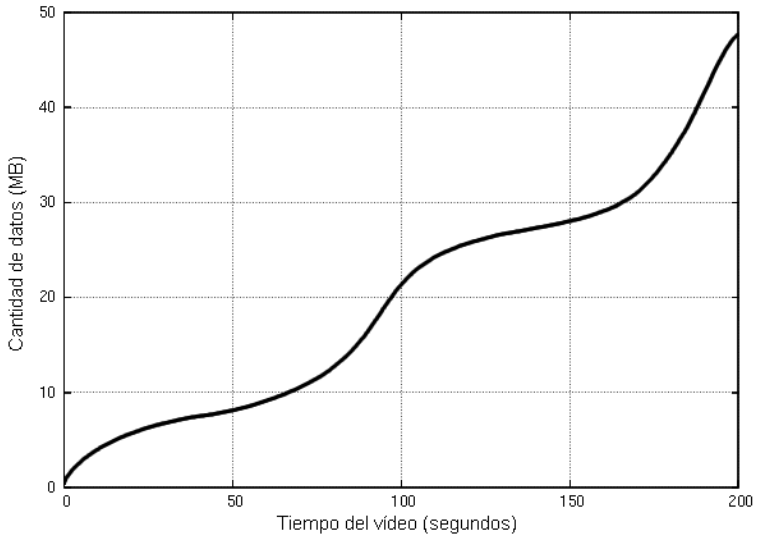


Figura 2.6: Curva de información de un vídeo VBR

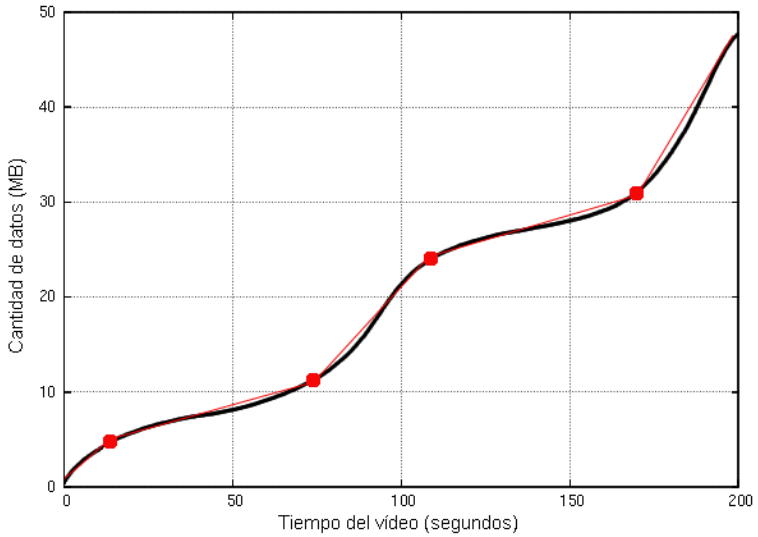


Figura 2.7: Selección de puntos fundamentales de la curva

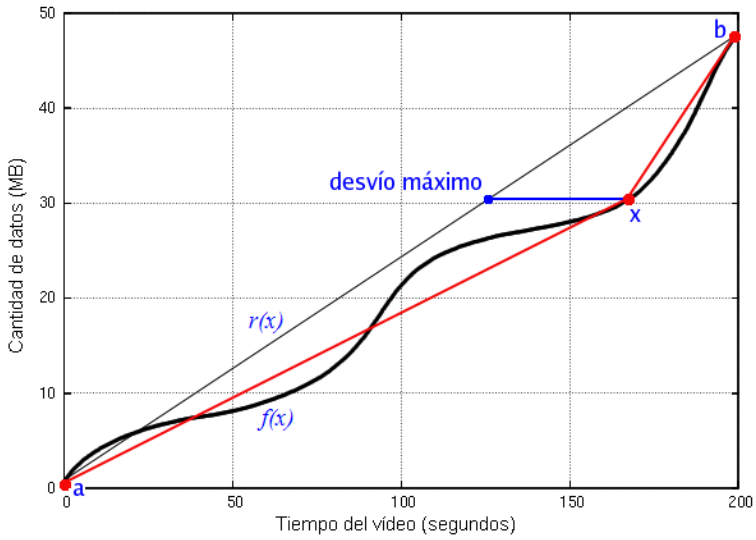


Figura 2.8: Búsqueda del desvío máximo

Con este objetivo, se ha desarrollado una aplicación que encuentra los puntos de la curva, que unidos por líneas rectas, se aproximan a la curva original con un desvío máximo especificado. El método utilizado en esta aplicación, trabaja de la siguiente manera. Sea la función $y = f(x)$ que define la curva, y un intervalo $[a,b]$ perteneciente a la curva, se define la función $y = r(x)$ que es la recta que contiene los puntos P_a y P_b, donde $P_a = (a, f(a))$ y $P_b = (b, f(b))$. Luego, se busca un valor x que satisfaga la siguiente ecuación:

$$\exists x \in [a,b] \,/\, \forall z \in [a,b] \,:\, |f(x) - r(x)| \geq |f(z) - r(z)| \tag{2.1}$$

Este valor x, que define el punto $P_x = (x, f(x))$, presenta el desvío máximo de segundos de la curva original $f(x)$ con respecto a la recta $r(x)$. Luego, se definen dos nuevas rectas, una que pasa por los puntos P_a y P_x, y otra que pasa por los puntos P_x y P_b. Este proceso puede verse en la figura 2.8.

Ahora se tienen dos nuevos intervalos, el intervalo $[a,x]$ y el intervalo $[x,b]$, por lo que el proceso se repite para cada uno de estos intervalos, y así sucesivamente, hasta que se encuentre un x tal que $|f(x) - r(x)| \leq maxError$, donde $maxError$ determina el máximo error de segundos admitido.

El método aquí utilizado, para seleccionar los puntos fundamentales de la curva, claramente no es el óptimo, pero es suficientemente bueno para obtener la ganancia esperada. Para comprobar las prestaciones del método se han realizado ex-

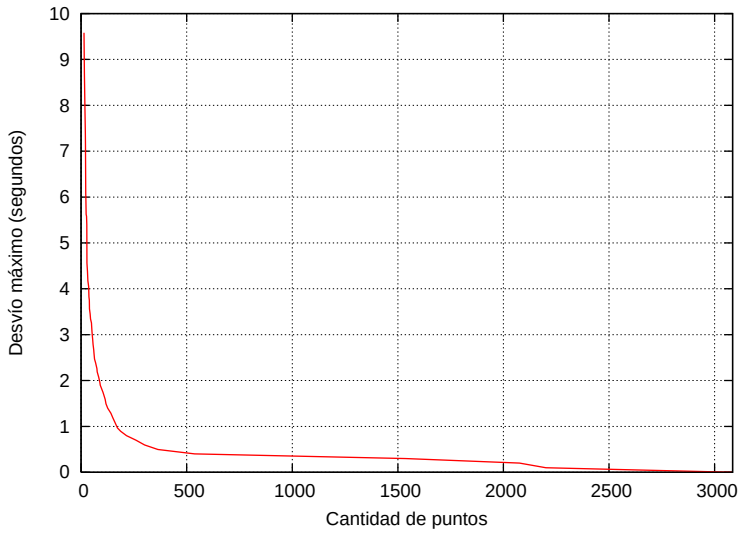


Figura 2.9: Precisión de la curva aproximada, utilizando la técnica de selección de puntos fundamentales

perimentos con un vídeo codificado con el método VBR, con una duración de 3.165 segundos y 784 MB de almacenamiento. En este experimento, se calculó la cantidad de puntos requeridos para obtener desvíos máximos desde 0 hasta 10 segundos con un paso de 0,1 segundos, cuyos resultados pueden observarse en la figura 2.9. Como puede apreciarse, con 500 puntos es posible lograr, para este vídeo, un desvío máximo de 0,5 segundos. De esta manera, la cantidad de información necesaria, para almacenar la curva del archivo de información de vídeo, ha pasado de ser de 3.165 entradas a 500 entradas, lo que es equivalente a una reducción del 84,2 % del almacenamiento requerido.

2.3.2. Definición de funciones de traducción *sec2byte* y *byte2sec*

El LCS necesita que el Gestor de Datos de Vídeo (VDM) le proporcione dos funciones llamadas $sec2byte$ y $byte2sec$. La función $sec2byte(s)$, dado un determinado tiempo de reproducción s, retorna el número de byte (idealmente es el primer byte del cuadro de vídeo) asociado al tiempo de reproducción s. La función $byte2sec(b)$, dada una determinada posición b (número de byte) dentro de un archivo de vídeo, retorna el tiempo de reproducción asociado a esa posición.

El cálculo de ambas funciones se realiza con los datos proporcionados por los archivos de información de los vídeos. Inicialmente, a partir de un archivo de in-

formación de vídeo, se define una lista denominada *pointList*, comprendida por puntos fundamentales *point*, donde $point_x$ es el valor del tiempo de reproducción expresado en segundos, y $point_y$ es el número de byte o posición dentro del archivo.

A continuación se presenta el cálculo de la función $sec2byte(s)$. Primero, se buscan los puntos a y b, que satisfagan las ecuaciones (2.2) y (2.3), respectivamente.

$$\begin{array}{l} \exists a \in pointList \,/\, \forall p \in pointList_{minor} : a_x \geq p_x \\ donde \\ pointList_{minor} = \{w/w \in pointList \wedge w_x \leq s\} \end{array} \tag{2.2}$$

$$\begin{array}{l} \exists b \in pointList \,/\, \forall p \in pointList_{greater} : b_x \leq p_x \\ donde \\ pointList_{greater} = \{w/w \in pointList \wedge w_x \geq s\} \end{array} \tag{2.3}$$

Finalmente, se calcula el valor de $sec2byte(s)$ basándose en la siguiente ecuación:

$$sec2byte(s) = \frac{a_y + (s - a_x) \times (b_y - a_y)}{b_x - a_x} \tag{2.4}$$

La función $byte2sec$ es la inversa de la función $sec2byte$. Es decir:

$$byte2sec(b) = sec2byte^{-1}(b) \tag{2.5}$$

por lo tanto:

$$byte2sec(sec2byte(s)) = s \ \wedge \ sec2byte(byte2sec(b)) = b$$

2.4. Representación del estado de reproducción del cliente

Cuando el cliente recibe los datos de un vídeo desde el servidor, este los almacena en un *buffer*. El *buffer* es un archivo, almacenado en el disco del cliente, que tiene asignado un espacio de almacenamiento igual al tamaño del vídeo original ubicado en el servidor. Inicialmente, los datos almacenados en él son inválidos. Cuando el cliente recibe una sección del vídeo, esta se almacena en la parte correspondiente del archivo. Al momento de reproducir la media, el cliente debe conocer qué partes del *buffer* tiene datos del vídeo y cuales no. El cliente no sólo necesita

conocer el estado del *buffer*, para que el planificador de canales lógicos (LCS) pueda llevar a cabo sus tareas, también es necesario que el servidor conozca el punto de reproducción y el estado de los *buffers* de los clientes.

De acuerdo a estas necesidades, se define una estructura denominada "mapa del *buffer* del vídeo" (VBM, *Video Buffer Map*), que representa las secciones de vídeo almacenadas por el cliente en un *buffer* determinado. Un *buffer* de un vídeo en un cliente es representado por dos VBM, uno mantenido por cliente y el otro por el servidor. El VBM del servidor es actualizado cada vez que se transmite media al cliente. El VBM del cliente es actualizado cada vez que se recibe media desde el servidor.

El servidor, para conocer el punto de reproducción del cliente, podría consultar al cliente cada vez que necesita esta información. Sin embargo, el retardo de la comunicación haría que esta información no llegue en el momento adecuado. Por lo tanto, el servidor debe calcular el punto del vídeo que el cliente está reproduciendo. El tiempo de reproducción actual del cliente es representado con dos datos, uno es el *último punto de reproducción* registrado, y el otro es el *tiempo de inicio* de la reproducción. Suponer que un cliente tiene como *último punto de reproducción* el tiempo 34' 35" y un *tiempo de inicio* con registro del día 1 de enero de 2008 a las 18:15 hs. A las 19 hs de ese mismo día (01/01/2008), el servidor desea conocer el tiempo de reproducción en que se encuentra el cliente, entonces, resta a la hora actual la hora del *tiempo de inicio* y la suma al tiempo del *último punto de reproducción*, es decir, el tiempo de reproducción es igual a 79' 35" (19hs − 18:15 hs + 34' 35"). Cada vez que el cliente ejecuta algún comando interactivo, puede establecerse un nuevo punto de reproducción, que queda registrado mediante la actualización del *último punto de reproducción* y un *tiempo de inicio* nulo. El *tiempo de inicio* será actualizado en el momento en que el *buffer* del cliente disponga de la cantidad de datos de vídeo preestablecidos para comenzar con su visualización.

En la figura 2.10 se muestra un VBM donde se señalan los puntos importantes de la estructura, que son:

- Punto de reproducción: corresponde al último punto de reproducción establecido por el cliente.
- Siguiente bloque vacío: es el primer bloque vacío que está adelante del punto de reproducción. Note que el siguiente bloque vacío puede no existir.
- Punto de *deadline*: es el número del primer byte del *siguiente bloque vacío*.

A partir de los datos del VBM, se calcula el tiempo de *deadline*, basándose en la siguiente ecuación:

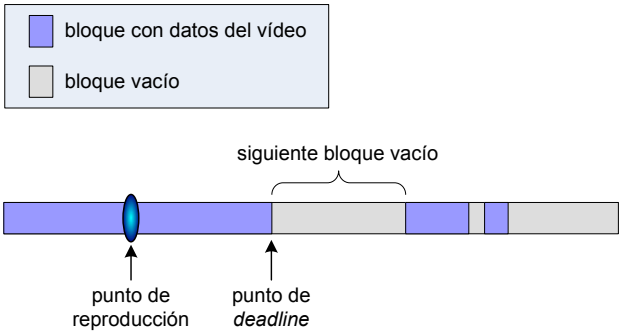


Figura 2.10: Mapa del *buffer* del vídeo (VBM)

$$deadline = byte2sec(punto\ de\ deadline) - tiempo\ de\ recuperación \qquad (2.6)$$

donde *byte2sec(b)* es la ecuación (2.5) (definida en la sección 2.3) la cual transforma el dato *b*, expresado en una medida de bytes, a una medida de segundos. Es decir, es la correspondencia del byte *b* del vídeo con el tiempo de reproducción que tiene asociado. La variable *tiempo de reproducción* es el tiempo de reproducción actual del cliente, calculado en base al *último punto de reproducción* y al *tiempo de inicio* de la reproducción. El valor del *deadline* nunca puede ser negativo, ya que la reproducción no puede continuar si no hay media en *buffer*. Cuando el *siguiente bloque vacío* no existe, se asigna, al *deadline*, un valor negativo para indicar su inexistencia.

2.5. Algoritmo planificador de canales lógicos

El servidor reserva ancho de banda de red para atender a todas las peticiones aceptadas. Sin embargo, es importante determinar el orden de atención de los canales lógicos, para asegurar que la media llegue en el tiempo adecuado a cada cliente, conforme la vayan necesitando. Esta tarea es la que realiza el algoritmo planificador de canales lógicos.

Dadas las características del módulo LCS, descriptas en la sección 1.1, en esta sección se describe el funcionamiento de su algoritmo de planificación. Este nuevo algoritmo, toma algunos conceptos básicos del algoritmo CB_MDA [63, 64], desarrollado por nuestro grupo de investigación, para un entorno de red con reserva de recursos y vídeos CBR. El algoritmo aquí presentado, entre otras mejoras, soporta

vídeos VBR, trabaja en un entorno sin reserva de recursos, e incluye un esquema de selección de canales mucho más refinado.

La planificación se lleva a cabo a intervalos regulares de tiempo, y su alcance es hasta la siguiente planificación. Cada intervalo de planificación se denomina *slot*. En una planificación se toman dos decisiones: la selección de los canales que serán servidos, y la cantidad de datos que se enviarán por cada uno de ellos.

Para que un canal sea seleccionado por el proceso de selección de canales, primero debe cumplir ciertos prerrequisitos. El primero de ellos es que el cliente necesite recibir nuevos datos, y el segundo es que, al momento de realizar la planificación, el canal disponga de capacidad para recibir nuevos datos. Puede ser que el cliente no necesite recibir nuevos datos porque ya dispone, en *buffer*, del contenido multimedia necesario para terminar de ver el vídeo; por lo menos, mientras permanezca visualizando el contenido sin hacer saltos hacia atrás o cambie la velocidad de reproducción actual. El canal puede no tener capacidad disponible para recibir nuevos datos, cuando una disminución del ancho de banda de comunicación ha impedido la transmisión de los datos del *slot* anterior. Dado un canal y el tamaño del *slot*, el módulo STM es el encargado de calcular la capacidad de recepción de ese canal para el siguiente *slot*.

Una vez que se tiene el conjunto de canales que cumplen con los prerrequisitos, se determinan sus prioridades. El esquema de prioridad se basa en el *deadline* (ver sección 1.5.4) y en el estado de reproducción que se encuentra el cliente de cada canal. Los estados en que puede encontrarse cada cliente son:

Reproducción normal: Es el estado de reproducción a velocidad normal ($1x$).

Reproducción rápida: Es el estado de reproducción a velocidad rápida (hacia adelante o hacia atrás).

Prefetch: Para soportar más fácilmente los momentos de fluctuaciones del ancho de banda de red, además de proporcionar una mayor flexibilidad para planificar la entrega de los vídeos, el cliente almacena una cantidad determinada de media antes de comenzar a reproducirla. Este estado de almacenamiento sin reproducción se denomina *Prefetch*.

Reproducción pausada: El cliente se encuentra haciendo una pausa en el vídeo que está visualizando.

Los tres primeros estados son excluyentes unos de otros. Sin embargo, el estado de reproducción pausada no puede existir sin que el cliente también esté en alguno de los otros tres estados.

La prioridad de los canales es asignada por las siguientes reglas, las que son presentas en función del grado de prioridad que poseen:

1. *Reproducción normal* con *deadline* menor a 10 segundos
2. *Reproducción rápida* con *deadline* menor a 5 segundos
3. *Prefetch*
4. *Reproducción normal no pausada, rápida* con *deadline* menor a 10 segundos, y *normal pausada* con *deadline* menor a 60 segundos
5. *Resto de canales*

Los valores de *deadline*, utilizados en las reglas para discriminar clientes con mayor o menor necesidad de media, son sólo de carácter experimental.

Finalmente, se selecciona para ser servido el canal de mayor prioridad y menor *deadline*. Este mecanismo de prioridades hace una diferenciación según las necesidades de datos de vídeo de los clientes: clientes con necesidad urgente (regla 1 y 2), clientes con necesidad alta (regla 3), clientes con necesidad media y baja (regla 4), y clientes con necesidad muy baja (regla 5).

A pesar de que los recursos están reservados, se intenta proteger aún más a los usuarios que están visualizando el contenido multimedia a una velocidad de reproducción normal. Por eso es que la regla 2, siendo de carácter urgente, se ubica en segundo lugar en la escala de prioridades. Asimismo, uno se puede preguntar: ¿por qué la regla de reproducción rápida es con *deadline* menor de 5 segundos y no de 10?. Normalmente los usuarios ejecutan estas operaciones por períodos cortos de tiempo, y no tiene sentido priorizar a estos canales porque tal vez la media que se les envíe, luego no sea utilizada. Es mejor, entonces, priorizar a un cliente que está esperando a recibir media para visualizarla lo antes posible (estado de *prefetch*). De hecho, un cliente, en estado de reproducción rápida con *deadline* mayor o igual a 10 segundos, tiene la menor prioridad de todas.

Este proceso, de selección de canales lógicos, es libre de inanición. Esto se verifica porque, dado un cliente, cuanto más tiempo pasa sin recibir media, más prioridad tendrá. Aunque un cliente, por ejemplo, en estado de *prefetch*, no pueda aumentar más su prioridad (siempre estará incluido en la regla 3), su atención no corre peligro. Luego de destinar más ancho de banda para aumentar el *deadline* de los clientes de las reglas 1 y 2, será suficiente destinar el ancho de banda reservado a esos clientes para que mantengan su *deadline*, y el resto del ancho de banda se podrá utilizar para servir a los clientes de la regla 3 (en estado de *prefetch*).

El siguiente paso es determinar la cantidad de datos que se van a transmitir por ese canal, valor que se obtiene aplicando la siguiente ecuación:

$$sent = \text{mín}(emptyBlockSize, remainingCV, remainingSCV, MaxDataBlock) \quad (2.7)$$

donde

emptyBlockSize es el tamaño en bytes, del siguiente bloque de datos vacío del *buffer* del cliente, que se encuentra inmediatamente después del punto de reproducción.

remainingCV es la cantidad restantes de bytes que el LCS puede entregar al STM para ser transmitidos a un cliente particular. Este valor depende de la capacidad de recepción del cliente y de los datos ya enviados durante el *slot* actual. Esto implica que, una vez conocida la cantidad de bytes a enviar, esta debe ser descontada del *remainingCV*.

remainingSCV indica la cantidad de bytes restantes que el LCS puede entregar al STM, durante el presente *slot*, para ser transmitidos a los clientes. Depende de la capacidad de la conexión de red del servidor, y de los datos ya enviados durante el *Slot* actual. De esta manera, luego de seleccionar el tamaño de los datos a enviar por el canal dado, se debe restar ese valor al *remainingSCV*.

MaxDataBlock determina la cantidad máxima de bytes que se permiten enviar por el canal, cada vez que es seleccionado para ser servido. Este parámetro determina la granularidad del algoritmo de planificación. Es decir, cuanto más pequeño sea el *MaxDataBlock*, más precisión tendrá la planificación pero más cálculos requerirá.

Una vez que se ha determinado la cantidad de bytes a enviar por el canal seleccionado, se actualizan los valores *remainingCV* y *remainingSCV*. La planificación continúa hasta que se verifica la siguiente proposición:

$$(remainingSCV = 0) \vee (\forall_{canal} remainingCV = 0 \vee emptyBlockSize = 0)$$

Es decir, no hay más canales que cumplan con los prerrequisitos, o ya se ha planificado todo el ancho de banda disponible del servidor.

2.5.1. Caso de estudio

Aquí se presenta un ejemplo de una planificación realizada por el LCS que ayudará a comprender su algoritmo de planificación. Lo siguiente es un fragmento de una planificación realizada por el LCS:

```
Slot length: 0.1 seconds
Available Server Credit Value by Slot: 125000 bytes (10 Mbits)
MaxDataBlock: 58560 bytes
------------- SLOT 263 ------- (Remaining SCV = 120697) -------------
 Stream | Remaining CV (B) | State   | Playback (sec) | Deadline (sec)
    6:0 |           102756 | normal |         26.200 |         12.502
    7:0 |           110089 | fast   |         26.200 |          4.882
    8:0 |            14203 | normal |         26.200 |          3.060
# Stream selected: 8:0
# Data sent: 14203
    6:0 |           102756 | normal |         26.200 |         12.502
    7:0 |           110089 | fast   |         26.200 |          4.882
# Stream selected: 7:0
# Data sent: 58560
    6:0 |           102756 | normal |         26.200 |         12.502
    7:0 |            51529 | fast   |         26.200 |          5.100
# Stream selected: 7:0
# Data sent: 47934
```

Este fragmento presenta los siguientes campos:

- *Stream:* es el identificador del canal, el cual se compone de dos datos: <dirección IP : puerto>. En este ejemplo, las direcciones IP y puertos son representados por números enteros.
- *Remaining CV:* representa el valor de la variable *RemainingCV* que registra la cantidad restantes de bytes que el LCS puede entregar al STM durante este *slot*, para ser transmitidos al cliente.
- *State*: es el estado del cliente que se encuentra conectado al presente canal.
- *Playback:* es el punto del vídeo, expresado en segundos, que el cliente está reproduciendo.
- *Deadline:* expresa la cantidad de segundos que el cliente tiene en *buffer* y aún no ha reproducido.

Todos los canales tienen asociado el mismo vídeo, que tiene una duración de 53 minutos y un tamaño de aproximadamente 800 MB. El servidor tiene una conexión de red de 100 Mb/s. El LCS realiza 10 planificaciones por segundo, por lo tanto tendrá disponible cerca de 10 Mbit cada 0,1 segundo. El bloque más grande que puede transmitirse cada vez que se selecciona un canal (parámetro *MaxDataBlock*) es de 58.560. El cliente del canal 6:0 y 8:0 se encuentran en un estado de reproducción a velocidad normal, mientras que el cliente del canal 7:0 se encuentra reproduciendo el vídeo a una velocidad rápida de $2x$. Los clientes han comenzado a reproducir el vídeo de manera continua desde el inicio, y en ningún momento han efectuado un salto hacia atrás o una reproducción rápida hacia atrás. Por lo tanto, el *siguiente bloque vacío*, en el caso de los clientes del canal 6:0 y 8:0, tiene un valor cercano a 800 MB, mientras que para el cliente 7:0 tiene un valor cercano a los 300 MB (tamaño aproximado de la versión del vídeo a una velocidad de $2x$).

A continuación se analiza cómo el LCS hace la planificación. Según la traza, en el *slot* 263 el servidor tiene 120.697 bytes disponibles para transmitir. Cuando se realiza el primer proceso de selección de canales, se determina la prioridad de cada uno; el canal 6:0 tiene una prioridad equivalente a la regla 4, mientras que el canal 7:0 corresponde a la regla 2, y el canal 8:0 a la regla 1. Por lo tanto, el canal 8:0 es seleccionado para ser atendido. A continuación, se calcula la cantidad de datos que el LCS enviará al cliente, determinada por la ecuación (2.7). Siendo el *emptyBlockSize*≈800 Mb, *remainingCV*=14.203, *remainingSCV*=120.697, y *MaxDataBlock*=58.560, la cantidad de datos enviados por el canal 8:0 es de 14.203 bytes.

Ahora, el LCS tiene 106.494 bytes (tenía 120.797 bytes y envió 14.203 bytes al canal 8:0) disponibles para transmitir. Como se aprecia, el canal 8:0 ahora no figura entre los posibles canales a ser seleccionados, porque ya se ha agotado su capacidad de recepción en este *slot* (es decir, *RemainingCV* es igual a 0). El canal 6:0 es de regla 4 y el canal 7:0 es de regla 2. Así, se selecciona el canal 7:0 por tener mayor prioridad, y luego se calcula la cantidad de datos a enviar (según la ecuación (2.7)). Siendo el *emptyBlockSize*≈300 Mb, *remainingCV*=110.089, *remainingSCV*=106.494, y *MaxDataBlock*=58.560, la cantidad de datos enviados por el canal 7:0 es de 58.560 bytes.

La tercera planificación ocurre de la siguiente manera. El servidor tiene 47.934 bytes disponibles para transmitir (en la planificación anterior tenía 106.494 bytes y se le envió 58.560 bytes). El canal 7:0 (a diferencia del *stream* 8:0) aún aparece en la lista con posibilidades de ser servido, porque su *remainingCV* es mayor que cero. Esto se debe a que, en la planificación anterior, se le envió una cantidad de bytes menor a la que tenía disponible para recibir. Ahora se tiene a ambos canales, el canal 6:0 y el 7:0, con regla 4, por lo tanto, se selecciona el canal 7:0 por

tener un *deadline* menor. Luego se calcula la cantidad de datos a enviar. Siendo el *emptyBlockSize*≈300 Mb, *remainingCV*=51.529, *remainingSCV*=47.934, y *MaxDataBlock*=58.560, la cantidad de datos enviados por el canal 7:0 es de 47.934 bytes.

Ninguna nueva planificación es realizada hasta el siguiente *slot* de tiempo porque el LCS ha agotado su capacidad de envío, es decir, ha distribuido, entre los diversos canales, todo el ancho de banda que tenía disponible.

2.6. Soporte de comandos interactivos

La principal diferencia entre un sistema de VoD y uno de *streaming* es la capacidad de los sistemas de VoD de ofrecer interactividad a través de las operaciones tipo DVD. Existe una gran variedad de operaciones que pueden ser implementadas en estos sistemas. En esta sección, se explican las operaciones más típicas: pausa, saltos hacia adelante y atrás, y avance y retroceso rápido.

En la operación de pausa, se congela la imagen que el cliente está visualizando, durante un período indefinido de tiempo. Es la única de las operaciones interactivas que favorece al sistema, ya que al haber clientes sin requerir media de manera inmediata, se puede atender a usuarios con mayor urgencia de datos. Los saltos hacia adelante y hacia atrás, simplemente modifican el punto de reproducción. Luego el cliente puede entrar en un estado de *prefetch*, o iniciar la reproducción si hay suficiente media a partir del nuevo punto de reproducción. Las operaciones interactivas que son conflictivas son las de avance y retroceso a velocidad rápida, las cuales se discuten en la sección 2.6.1, junto con las técnicas más usuales que se utilizan para implementarlas. Finalmente, en la sección 2.6.2, se describe el esquema adoptado por VoD-NFR para avance y retroceso rápido.

2.6.1. Revisión de las técnicas de implementación del avance y retroceso rápido

Las operaciones interactivas de avance y retroceso rápido pueden ser fácilmente implementadas en vídeo sin comprimir, ya que: el tamaño de los cuadros es fijo, la posición de cada marco en la secuencia de bits es conocido, y los cuadros pueden ser accedidos y mostrados independientemente de los otros. Para vídeo comprimido, la implementación de estos controles son realmente un desafío; los cuadros comprimidos de un vídeo VBR tienen un tamaño variable, y su posición en la secuencia de bits puede no conocerse. Además, si se utiliza codificación predictiva

(como MPEG), un cuadro dado puede no ser decodificado independientemente de los otros cuadros [75].

Existen varias formas de producir el efecto de avance y retroceso rápido. El método de *fuerza-bruta* consiste en decodificar y mostrar un vídeo más rápido que su tasa normal; si un vídeo está codificado a 30 fps, pero se decodifica a 60 fps, el vídeo se verá al doble de la velocidad normal. Reproducir un vídeo a más de la velocidad normalmente utilizada de 30 fps, tiene la desventaja de requerir mayor procesamiento y por lo tanto se necesita un procesador más potente, fundamentalmente para imágenes de alta resolución. No obstante, el mayor problema es que el consumo de bits por segundo aumenta proporcionalmente a la velocidad de reproducción; si un vídeo tiene una tasa de bits de 1 Mb/s, a una velocidad de $4x$ tendrá una tasa de 4 Mb/s. De esta manera, el servidor tendrá problemas para hacer la reserva de recursos y, además, la red de transmisión puede no soportar tasas de envío tan altas.

Una posible alternativa al esquema simple de *fuerza-bruta* es utilizar una técnica de salto de cuadros, que consiste en enviar solamente cuadros intercalados. Sin embargo, el problema que afecta a esta técnica deriva de las dependencias entre los cuadros del vídeo codificado; una solución es transmitir solo los cuadros de tipo *I*, los cuales no tienen dependencias, pero el inconveniente es que generalmente no se encuentran más de dos cuadros *I* cada 30 cuadros de vídeo. Si se requiere ver un vídeo de 30 fps a una velocidad de $2x$, un segundo de vídeo solo estará compuesto de 4 cuadros, lo cual es una frecuencia demasiada baja para obtener una visualización agradable. Codificando el vídeo con más cuadros *I*, aumentará considerablemente la tasa de bits, por lo tanto esta técnica no es una buena solución.

Una solución propuesta en [76] presenta un sistema escalable con soporte de comandos interactivos, que reduce la transmisión del servidor a través de la segmentación de los vídeos y el uso de servidores *proxy*. Sin embargo, esta técnica no resuelve el problema del aumento de la tasa de bits que recibe el cliente.

Otros tipos de esquemas se valen de la transcodificación. La transcodificación de vídeos realiza una o más operaciones, tal como transformaciones de tasa de bits y conversiones de formato, para transformar un vídeo comprimido en otro [77]. En [78] y [79] se proponen soluciones que consisten en transmitir la media con una calidad inferior para reducir la tasa de bits del vídeo pero, aunque la degradación de la calidad del vídeo sea tolerable, se estaría malgastando ancho de banda para transmitir cuadros que no se requieren para la visualización rápida.

A partir de la revisión que se ha efectuado sobre las diversas posibilidades para implementar el avance y retroceso rápido en sistemas de VoD, a continuación se presenta la técnica adoptada para el sistema VoD-NFR.

2.6.2. Técnica de avance y retroceso rápido de VoD-NFR

En nuestro sistema se ha implementado una solución que permite implementar el avance y retroceso rápido sin malgastar el ancho de banda de red. Esta técnica consiste en recodificar los vídeos a menos fps, generando una nueva versión por cada velocidad de reproducción que se quiera soportar. Si un vídeo de 30 fps es recodificado a 15 fps, pero su decodificación y visualización se hace a 30 fps, el usuario entonces verá el vídeo al doble de velocidad; además, el tamaño del nuevo vídeo será de aproximadamente la mitad del vídeo original. Similarmente, para visualizarlo a una velocidad de $3x$, el vídeo se recodificará a 10 fps, y así sucesivamente. Estas nuevas versiones son realizadas fuera de línea, y no incluyen sonido. El problema que se tiene es el aumento del espacio de almacenamiento, pero la tasa de bits del vídeo es casi igual a la del vídeo original. Así, el uso de las operaciones de avance o retroceso rápido no requiere un mayor ancho de banda de red, y tampoco requieren una reserva especial de recursos en el servidor. El rápido avance en las tecnologías de almacenamiento hace que este recurso sea cada vez más económico, con lo cual es altamente conveniente sacrificar almacenamiento secundario por ancho de banda de red.

La característica de reproducción rápida permite que el espectador no pueda observar tantos detalle de las imágenes como en la reproducción a velocidad normal. De esta forma, es posible perder calidad de la media sin que el usuario lo note, y así reducir un poco el tamaño de los vídeos. No obstante, en caso de que se desee reducir aún más el almacenamiento, y dada la inviabilidad de llevar a cabo la recodificación completa del vídeo en línea (para reducir los fps) por el alto costo de procesamiento, es posible utilizar la técnica propuesta en [80]. Esta solución almacena, fuera de línea, solamente los vectores de movimiento de las versiones de los vídeos con reducidos fps. Cuando se necesite enviar un vídeo con menos fps, se recodifica en línea, leyendo los vectores de movimiento de los archivos en vez de ser computados. Esto permite aumentar considerablemente la velocidad de recodificación y reducir el consumo de CPU.

Detalles de implementación

El esquema utilizado para implementar el avance y retroceso rápido en VoD-NFR, permite que el sistema pueda soportar múltiples velocidades de reproducción, donde su única limitación se encuentra en el aumento del espacio de almacenamiento. El cliente mantiene un VBM para cada una de las versiones de vídeo disponibles. El servidor mantiene un VBM por cada VBM de los clientes. El tra-

tamiento de la operación de avance y retroceso rápido, comprende las siguientes acciones:

1. Notificar al servidor de la ejecución del comando interactivo, indicando la velocidad de reproducción y el nuevo punto de reproducción relativo a la versión del vídeo que solicita observarse. El cliente es el encargado de hacer la conversión del tiempo de reproducción de la versión actual al nuevo tiempo de reproducción. El tiempo de reproducción, relativo a la versión del vídeo que desea visualizarse, se calcula como:

$$\frac{velocidad1 \times tiempo1}{velocidad2} \tag{2.8}$$

   donde *velocidad1* y *tiempo1* son la velocidad y el tiempo de reproducción de la versión del vídeo que se estaba observando, y *velocidad2* es la nueva velocidad de reproducción deseada. Para ejemplificar, suponga que un cliente está visualizando un vídeo, a velocidad normal, en el tiempo 10' 20" y quiere hacer un avance rápido a velocidad $2x$. El nuevo tiempo de reproducción para la versión $2x$ será de 5' 10" ($1 \times 10'\ 20"/2$).
   Cuando el servidor recibe la notificación de la ejecución del comando interactivo, inmediatamente cancela el envío de todos los paquetes encolados cuyo destino sea este cliente. A continuación, establece como la versión de vídeo actual a la versión del vídeo solicitada, y comienza a transmitir los datos correspondientes.

2. Redirigir el visualizador al nuevo *buffer* donde se encuentra el vídeo de la velocidad requerida. Dadas las características del *plugin* de VLC [81], se decidió por utilizar un *plugin* de visualización por cada *buffer*. En caso de requerir un cambio de velocidad, simplemente se muestra el nuevo *plugin*, y se oculta el otro. El problema de tener un solo *plugin* es que el cambio de origen de la media produce varios efectos no deseados.

2.7. Tolerancia a fallos

Un sistema de VoD a gran escala es propenso a sufrir caídas de servidores y fallos de red. Los servidores pueden dejar de estar operativos por diferentes causas, ya sea por hardware o software. La red falla cuando se caen o congestionan los enlaces, y la detección de fallos de red es relativa al contenido multimedia que por ella se transmite. Esto quiere decir, que un fallo de red para un contenido multimedia

puede no serlo para otro. Por ejemplo, si se tiene un ancho de banda de comunicación de 3 Mb/s, y a causa de un estado de congestión, el ancho de banda promedio baja a 1,5 Mb/s, este puede ser suficiente para un vídeo de 1 Mb/s pero insuficiente para un vídeo de 2 Mb/s. Así, si el cliente estuviese visualizando el primer vídeo, no habría ningún fallo de red, pero sí que lo habría si el vídeo visualizado es el segundo. Un corte total de la comunicación sería un fallo de red para ambos casos.

El sistema VoD-NFR, como ya se ha mencionado, es tolerante a fallos del servidor y de la red. El mecanismo de tolerancia a fallos aquí presentado fue desarrollado especialmente para este sistema. El sistema VoD-NFR está preparado para detectar y solucionar problemas como puede ser la caída de un servidor de VoD, o la congestión o caída de los enlaces de red. Esta tolerancia es posible gracias a que el sistema presenta servidores distribuidos geográficamente, los cuales colaboran entre sí para recuperarse de los posibles fallos.

Un proceso de gestión de fallos consiste de dos etapas, una etapa de detección del fallo y otra etapa de recuperación. VoD-NFR solapa la etapa de detección con la de recuperación, produciendo una ganancia de tiempo determinante para poder mantener la QoS de la media entregada a los clientes. Este solapamiento es fruto de un mecanismo de preaviso, que permite seleccionar y reservar recursos en otro servidor, anticipándose a una posible migración del cliente.

La detección de fallos de red se realiza fundamentalmente en el servidor y se comunica al cliente para que este contacte a otro servidor que lo pueda atender. El cliente tiene un mecanismo de protección que actúa ante la ausencia del aviso del fallo (que emite el servidor) debido a una caída del servidor, o a una pérdida total (o casi total) de la comunicación. De esta manera, la detección de fallos del sistema VoD-NFR utiliza dos mecanismos de detección que colaboran para brindar tolerancia ante los fallos de servidores y de red. Un mecanismo se encarga de detectar cualquier tipo de fallos de red, y está ubicado en el servidor. El mecanismo de protección, ubicado en el cliente, reporta un fallo total cuando la comunicación cliente-servidor, a través del canal SCC, se ha perdido.

2.7.1. Mecanismo de protección efectuado por el cliente

El canal SCC es implementado utilizando el protocolo de transporte de datos confiable de Internet, denominado TCP. Si el canal de comunicación se ha roto, por ejemplo, por problemas de la red, el servidor no podrá notificar la migración al cliente. De esta manera, el cliente necesita saber cuándo el canal se ha roto, para automáticamente iniciar la migración. La inactividad en el canal de comunicación TCP puede producir desconexiones inesperadas. Es normal que si la conexión se

realiza a través de un *proxy* NAT o un *firewall*, la conexión sea terminada sin razón alguna. Todo esto puede ser resuelto utilizando el mecanismo "*keepalive*" del protocolo TCP [82], el cual está desactivado por defecto. Este mecanismo envía, cada cierto tiempo, mensajes (*heartbeat*) al otro extremo, preguntando si se encuentra "vivo". De esta manera, no solo puede saberse si la conexión se ha cerrado, sino que también se evita la inactividad en la comunicación y la consiguiente posibilidad de los cierres inesperados de la misma. No obstante, se ha decidido utilizar un mecanismo similar al recientemente descripto, pero implementado al nivel de aplicación. Así, se evitan inconvenientes de variaciones en las implementaciones o disponibilidad del mecanismo en los distintos sistemas operativos.

Este mecanismo de protección es implementado por el módulo de Control de Sesión del servidor y por el módulo que recibe el mismo nombre del cliente. Ambos módulos son los encargados de gestionar el SCC y de la recuperación de los fallos.

2.7.2. Mecanismo de detección de fallos efectuado por el servidor

Para efectuar la detección del fallo de red, es necesario monitorizar el ancho de banda de comunicación con el cliente. Las muestras de ancho de banda, suministradas por el STM, son utilizadas para estimar el ancho de banda de la comunicación. Cuando este ancho de banda baja de cierto umbral, determinado por la tasa de bits del vídeo, se emite un preaviso para que el cliente busque un servidor alternativo que tenga capacidad para atenderlo. El servidor alternativo efectuará una reserva de recursos para el cliente, por si el estado de comunicación entre el cliente y el servidor actual no mejora, y se requiere hacer una migración. Una vez pasado cierto tiempo, si la comunicación no ha mejorado, el servidor actual envía al cliente la orden de migrar. En caso contrario, notifica al cliente la cancelación de la migración y, a su vez, el cliente notifica al servidor alternativo la cancelación de la reserva de recursos.

La selección del servidor alternativo se basa en la clasificación de servidores efectuada inmediatamente luego de que el cliente ha adquirido el boleto (ver sección 2.1). La lista de servidores de la clasificación es recorrida en forma circular, es decir, inicialmente se seleccionará, como servidor alternativo, el servidor de la lista que está a continuación del servidor actual. Si ese servidor no puede prestar servicio, se le solicitará al siguiente, y así sucesivamente.

A continuación se explica cómo se efectúa la estimación del ancho de banda, luego se describe el procedimiento de detección de fallos, y finalmente se presenta el diagrama funcional del mecanismo. Este diagrama permite observar las interacciones de los componentes del servidor de VoD involucrados en la tarea realizada por el mecanismo aquí descripto.

Estimación del ancho de banda

Las muestras de ancho de banda se toman cada vez que el algoritmo de control de congestión del STM modifica la distancia de envío entre paquetes (IPG, *Inter Packet Gap*). El algoritmo modifica el IPG como mínimo una vez por cada ida y vuelta de un paquete desde el servidor al cliente (RTT, *Round Trip Time*). Si suponemos un RTT acotado entre 80 y 210 ms, según pruebas realizadas[1], se tienen aproximadamente entre 4 y 12 muestras por segundo. Con el objeto de unificar la frecuencia de muestreo, se promedian las muestras tomadas dentro de intervalos de 1 segundo; estas muestras promediadas, a menos que se indique explícitamente, se denominarán en lo sucesivo simplemente como "muestras".

Para estimar el ancho de banda, es necesario dar un peso mayor a las muestras más recientes, ya que es más probable que reflejen el comportamiento futuro. Una técnica habitual de predicción de valores futuros, a partir de una serie de valores pasados, es utilizar un promedio exponencial [83]:

$$S_{n+1} = \alpha M_n + (1-\alpha) S_n \tag{2.9}$$

donde

- M_n es la n-ésima muestra del ancho de banda de la comunicación desde el servidor al cliente.
- Sn es el valor predicho para el caso n-ésimo.
- α es un factor constante de ponderación acotado ($0 < \alpha < 1$) que determina el peso relativo dado a las observaciones más y menos recientes.

Independientemente del número de observaciones pasadas, se llega a una situación en la que se tienen en cuenta todos los valores pasados, pero los más distantes reciben un peso menor. Para verlo con más claridad, considérese el siguiente desarrollo de la ecuación (2.9):

[1] Las pruebas de RTT se efectuaron el día 28 de marzo de 2008 a las 12:00hs de España. El RTT de 80 ms fue registrado entre un computador con conexión doméstica (IP: 84.77.107.86) ubicado en Cerdanyola del Vallès (Barcelona, España) y un servidor web con la URL http://www.marca.es (IP: 84.53.134.51), correspondiente a un servidor en España de la red de distribución de contenidos Akamai. El RTT de 210 ms fue obtenido desde el mismo computador doméstico a un servidor web con la URL http://www.rionegro.com.ar (IP: 74.86.198.2), ubicado en General Roca, Río Negro, Argentina.

$$S_{n+1} = \alpha M_n + (1-\alpha)\alpha M_{n-1} + \cdots + (1-\alpha)^i \alpha M_{n-1} + \cdots + (1-\alpha)^n S_1 \quad (2.10)$$

Como α y $(1-\alpha)$ son menores que uno, cada uno de los sucesivos términos de la ecuación es más pequeño, por lo tanto, cuanto más antigua es la muestra, menos peso tiene en la predicción. Esto puede observarse en el desarrollo de la ecuación con $\alpha = 0{,}2$:

$$S_{n+1} = 0{,}2M_n + 0{,}16M_{n-1} + 0{,}128M_{n-2} + 0{,}1024M_{n-3} + \cdots$$

Veamos qué ocurre al aumentar el valor del coeficiente, utilizando $\alpha = 0{,}8$:

$$S_{n+1} = 0{,}8M_n + 0{,}16M_{n-1} + 0{,}032M_{n-2} + 0{,}0064M_{n-3} + \cdots$$

Como se puede apreciar, cuanto mayor es el valor del coeficiente α, mayor es el peso de las muestras más recientes. Emplear un valor α próximo a 1 hace que la predicción refleje rápidamente las variaciones de las muestras. Sin embargo, si las variaciones de las muestras son muy bruscas, la predicción oscilará demasiado y por tanto dejará de ser útil.

A continuación se analiza un caso particular, con el objeto de estudiar el comportamiento de las curvas suavizadas al variar el valor del coeficiente α. Un servidor envía datos de un vídeo a un cliente por un camino de comunicación de 10 Mb/s y un RTT promedio de 80 ms. El servidor toma las muestras de ancho de banda, con una frecuencia aproximada de 12 por segundo. En la figura 2.11 se observa la curva de observación (obtenida a partir de las muestras individuales), la curva de muestras promediadas (en intervalos de 1 segundo), y una curva suavizada exponencialmente. La curva suavizada exponencialmente tiene un coeficiente α de 0,5, y un valor base de la predicción de 10 Mb/s ($S_1 = 10$). Se puede observar la relación y comportamiento de estas tres curvas, donde la segunda se construye en base a la primera, y la tercera en base a la segunda. En la figura 2.12 se muestra la curva de observación contrastada a tres curvas suavizadas exponencialmente con valores de α de 0,2, 0,5 y 0,8, y un valor base de predicción de 10 Mb/s. Note que la curva con $\alpha = 0{,}2$ se comporta muy bien hasta el segundo 25. En ese momento, la red experimenta una fuerte congestión que hace disminuir drásticamente el ancho de banda disponible para el cliente, de 10 a 1 Mb/s. La curva tarda aproximadamente 15 segundos en converger al nuevo ancho de banda disponible. En cambio, las curvas con $\alpha = 0{,}5$ y $\alpha = 0{,}8$, convergen más rápidamente pero también presentan una fluctuación mayor.

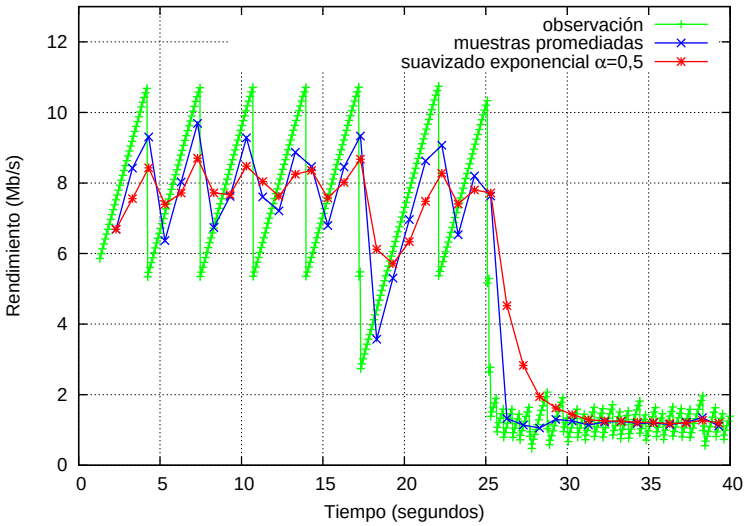


Figura 2.11: Unificación de la frecuencia de muestreo y suavización exponencial

Se ha determinado empíricamente que un valor de $\alpha = 0{,}5$ provee un comportamiento adecuado para la mayoría de los casos, con un balance correcto entre suavidad de la curva y reacción a los cambios del ancho de banda de las comunicaciones.

Procedimiento de detección de fallos

Una vez estimado el ancho de banda, el siguiente paso es utilizar esta información para determinar si la comunicación es adecuada para servir el vídeo al cliente. La comunicación será adecuada mientras soporte una tasa de transferencia igual o mayor a la tasa media del vídeo.

Cuando el ancho de banda (predicho) es menor a la tasa media del vídeo, el servidor envía un preaviso o alerta de migración al cliente, y se entra en un período de análisis para determinar si la migración se efectuará o no; a este período se lo conoce como *changüí*. El *changüí* es una ventaja u oportunidad que se da a la comunicación para que mejore. Si durante este tiempo se determina que la comunicación no es suficiente para servir al cliente con la QoS adecuada, se ordena la migración al cliente. Caso contrario, se notifica al cliente la cancelación de la migración.

Cuando termina el tiempo de *changüí*, se obtiene el promedio de todas las muestras tomadas durante ese intervalo. Si el promedio de las muestras es menor a

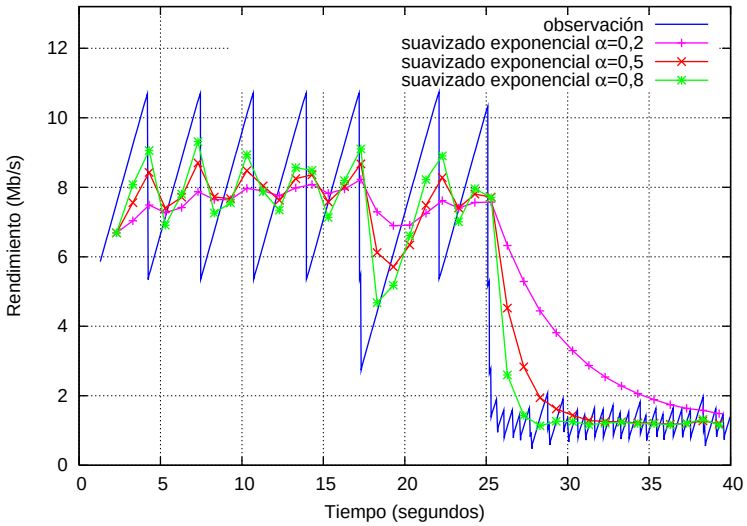


Figura 2.12: Variación del coeficiente de suavizado exponencial

la tasa media del vídeo, se determina que la comunicación es insuficiente. En otro caso, la comunicación se considera adecuada para el transporte del vídeo. Promediar las muestras dentro del intervalo evita llevar a cabo migraciones innecesarias cuando el ancho de banda presenta pronunciadas fluctuaciones.

El *changüí* se calcula en base a la siguiente ecuación:

$$changüí = \text{mín}(changüí\ máximo, deadline - tiempo\ de\ recuperación) \quad (2.11)$$

donde *changüí máximo* es el tiempo máximo estipulado de duración del *changüí*, y *tiempo de recuperación* es el tiempo estimado de duración del proceso de recuperación del fallo, una vez que el mismo ha sido detectado.

En la figura 2.13 se presenta un gráfico que visualiza las diferentes variables involucradas en el cálculo del tiempo de *changüí*. En este ejemplo, el vídeo tiene una tasa promedio de 5 Mb/s y el ancho de banda predicho de la comunicación baja de este límite en el segundo 49,88; en ese momento se envía una alerta de migración al cliente y se calcula el *changüí*. Dado un *changüí* máximo de 11 segundos, un tiempo de recuperación estimado de 2 segundos, y un *deadline* de 12 segundos, el *changüí* toma un valor de 10 segundos. En el segundo 59,88 se promedian las muestras tomadas durante el intervalo de análisis, obteniendo un resultado de 4,98 Mb/s y, como el promedio de las muestras es menor a 5 Mb/s, se decide por indicar al cliente que debe migrar.

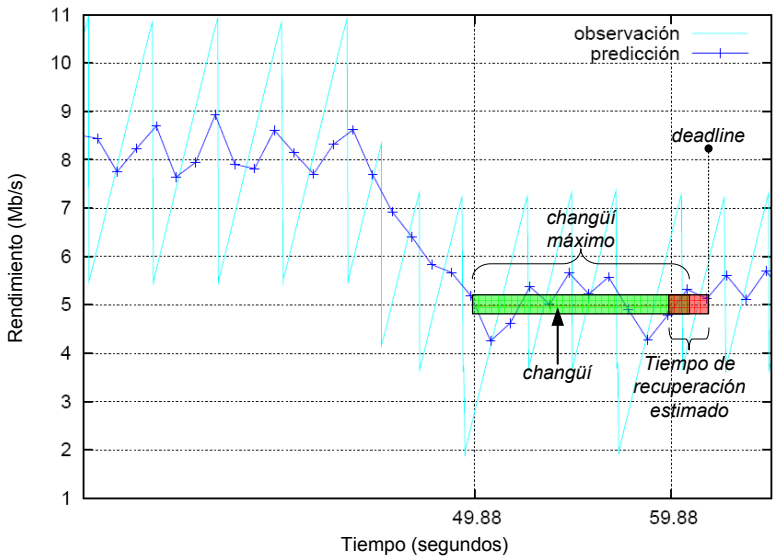


Figura 2.13: Tiempo de *changüí*

Diagrama funcional del mecanismo de detección de fallos

El esquema general de funcionamiento del mecanismo de detección de fallos efectuado por el servidor es mostrado en la figura 2.14. Se puede apreciar la relación de los múltiples componentes del servidor que colaboran con el módulo Garantía de QoS (QoSA): el Módulo de Transmisión de *Streams* (STM), el Planificador de Canales Lógicos (LCS), y el Control de Sesión (SC). El módulo QoSA requiere, para efectuar la detección de fallos, información que suministra el STM y LCS. Todas las decisiones que tome el QoSA son comunicadas al SC para que las lleve a cabo. Estas decisiones involucran: enviar una alerta de migración al cliente, e indicar al cliente que cancele la reserva de recursos efectuada en el servidor alternativo, o, en caso contrario, que inicie la migración.

2.7.3. Recuperación de fallos

Cuando se detecta un fallo, se deben tomar acciones para corregir el problema, evitando la pérdida de la QoS. En el sistema VoD-NFR, el proceso de recuperación de fallos es dirigido íntegramente por el cliente. El cliente es quien tiene toda la información necesaria para poder restablecer su conexión al sistema.

Cuando se efectúa una recuperación de un fallo, el cliente puede disponer o no de una reserva previa de recursos efectuada en un servidor alternativo. Es posi-

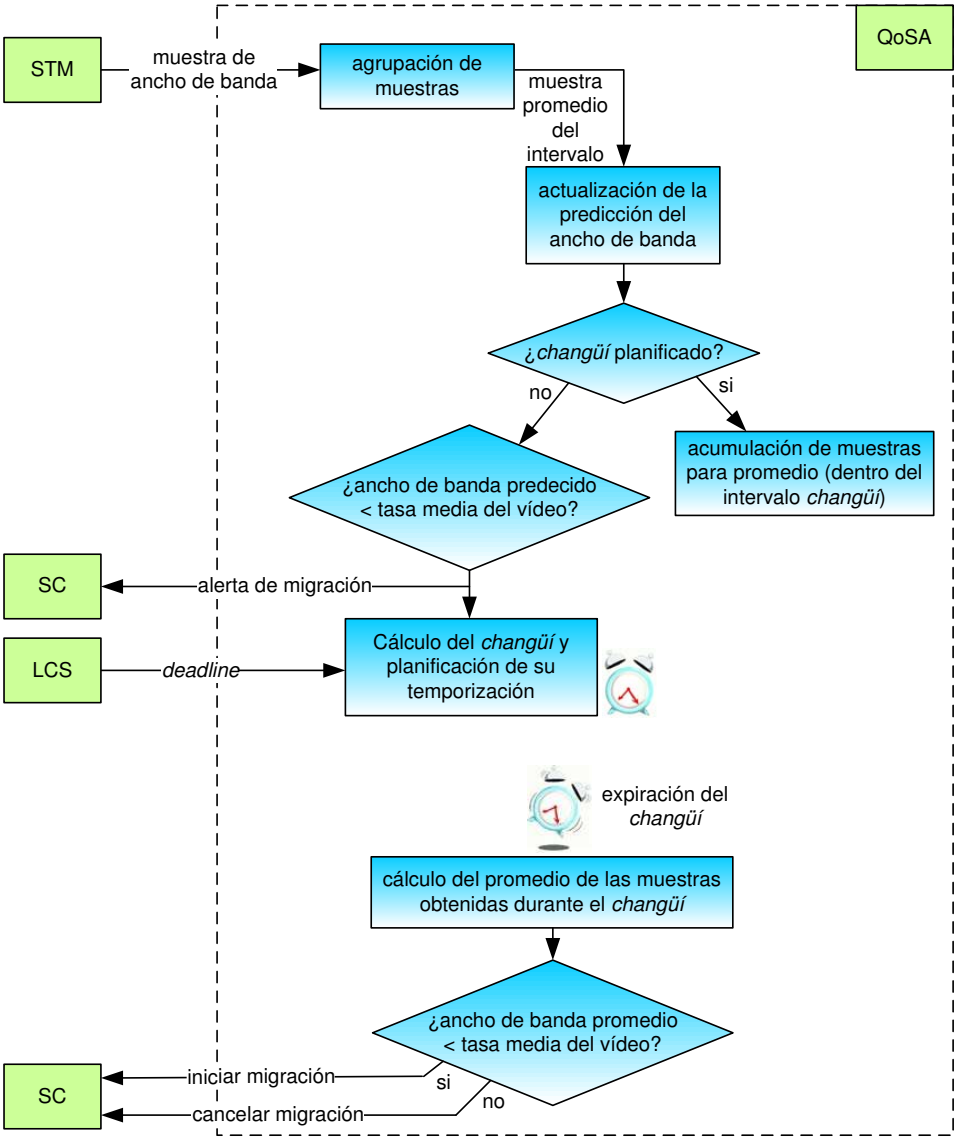


Figura 2.14: Diagrama funcional de la detección de fallos de red

ble que el tiempo de *changüí* no haya sido suficiente para encontrar a un servidor alternativo, o que la detección del fallo fue determinada por el mecanismo de protección del cliente. Si se dispone de esta reserva, el cliente enviará una aceptación de migración al servidor alternativo. Caso contrario, el cliente deberá encontrar un servidor alternativo que acepte prestarle servicio; proceso que será mucho más costoso en tiempo cuantos más servidores denieguen la solicitud.

Independientemente de haber tenido o no recursos reservados en un servidor alternativo, el cliente debe comunicar al servidor alternativo su estado actual de reproducción. El cliente, entonces, transmite los VBMs para que el nuevo servidor conozca el estado de sus *buffers*, y finalmente envía el comando interactivo correspondiente para continuar con la visualización desde el punto y modo actual de reproducción.

2.8. Control de admisión y balanceo de carga

Antes de admitir un nuevo cliente a un servidor de VoD, este debe asegurarse que tiene suficientes recursos para garantizar la QoS contratada con los clientes existentes. Por lo tanto, se ha considerado para el sistema VoD-NFR un conjunto de políticas que garantizan la reserva de recursos para las peticiones aceptadas durante el tiempo que dure cada sesión. Sin embargo, que el control de admisión acepte una petición no es suficiente para que la petición entre al sistema. También debe ser aceptado por el balanceador de carga, para el cual se ha asumido el esquema presentado en [21, 84]. Este esquema es denominado *Random Early Migration* (REM). Cuando arriba una nueva petición, REM asigna una probabilidad de rechazo a la petición, que es más grande cuando mayor es la carga de servicio del servidor. Este balanceo de carga es totalmente distribuido ya que las decisiones son tomadas independientemente por cada servidor, sin requerimientos de información del estado de los otros servidores.

El control de admisión debe verificar la existencia de cuatro recursos para prestar servicio a un nuevo cliente: CPU, memoria principal, almacenamiento secundario, y red. Cada uno de estos recursos es controlado por un módulo independiente, que incluye su propia política de control y reserva de recursos.

Desafortunadamente, las características de los vídeos VBR presentan dificultades para determinar la cantidad de recursos necesarios para transmitirlos a los clientes. Esto implica que el control de admisión tendrá problemas para determinar cómo garantizar y reservar los recursos requeridos. Para solucionar este problema, pueden aplicarse dos tipos de soluciones: una optimista y la otra pesimista. La solución optimista reserva recursos sólo para la tasa promedio de bits, esperando que

cuando se transmite un vídeo en una sección de tasa de bits alta, existan otros que están siendo transmitidos en una sección de tasa de bits menor a la de su promedio. La solución pesimista, reserva recursos para la tasa más alta de bits, produciendo una baja utilización de los recursos.

El recurso de red y almacenamiento secundario tiene una relación directa con la tasa de transferencia de los vídeos VBR. Por el contrario, los recursos de CPU y memoria principal están ligados al número de canales servidos y no a las características de los vídeos ya que el servidor no efectúa, en línea, ninguna codificación ni decodificación de los vídeos. Suponiendo transmisiones a la máxima capacidad de conexión de red del servidor, el consumo de CPU y memoria principal estará supeditado exclusivamente al número de canales servidos. Por cada canal servido, el servidor necesita controlar su estado; además, los algoritmos planificadores del sistema de VoD tendrán más conexiones que planificar. Estas tareas involucran tanto cómputo como espacio en memoria donde almacenar los datos de control.

En el sistema VoD-NFR, se utiliza un tipo de solución para el recurso de red y el almacenamiento secundario, y otra solución para el recurso de memoria principal y CPU.

2.8.1. Recurso de red y memoria secundaria

Se ha optado por utilizar una solución con estilo pesimista para los módulos que controlan el recurso de almacenamiento secundario y el de red. Si los recursos fueran reservados de acuerdo a la tasa promedio de bits de los vídeos, no podrían enviarse datos de los vídeos por adelantado para soportar las fluctuaciones del ancho de banda de las comunicaciones. Es decir que, para que el sistema funcione adecuadamente en el entorno de red para el cual fue diseñado, es necesario que los servidores, además de reservar el ancho de banda indispensable para servir a los vídeos a su frecuencia natural de bits, destinen una sección extra del ancho de banda para enviar media por adelantado.

De esta manera, para asegurar la existencia de ambos recursos (memoria secundaria y red) y aceptar una nueva solicitud, se debe satisfacer la siguiente ecuación:

$$\sum avg_i \leq maxBW - extra \tag{2.12}$$

donde

avg_i es la tasa de bits promedio del i-ésimo canal lógico (o *stream*), incluyendo a los canales existentes y al nuevo. Dado que se utilizan diferentes versiones de los vídeos para soportar las diversas velocidades

de reproducción, el valor de avg_i es la mayor de todas las tasas de bits promedio de las distintas versiones.

maxBW es el ancho de banda máximo del recurso, ya sea de memoria secundaria o conexión de red.

extra es la sección del ancho de banda que se reserva para soportar las fluctuaciones de las comunicaciones. Su valor es ajustado persiguiendo una búsqueda racional de equilibrio entre la adaptabilidad a la red y el número de canales soportados.

2.8.2. Recurso de CPU y memoria principal

Es posible llevar a cabo el control de estos recursos de dos formas diferentes. Una es determinar analíticamente el uso de memoria y CPU para atender a un cliente, y extrapolar este valor a los recursos disponibles, para determinar la cantidad máxima de clientes admitidos. Sin embargo, resulta realmente muy complejo debido a las grandes dimensiones del sistema.

La otra opción, la cual hemos adoptado, es realizar una prueba de estrés en cada servidor, que permite determinar el número máximo de peticiones admitidas. Así, durante la prueba, se van ingresando nuevas peticiones de clientes mientras se efectúa un seguimiento de los recursos. Cuando se alcanza el límite máximo de consumo de estos recursos (por ejemplo: 90 % de uso de CPU), se registra la cantidad de clientes que admite el servidor. Para que esta prueba sea válida, es necesario asegurarse de que el servidor utiliza todo el ancho de banda disponible de red para servir a los diferentes canales. Cuando arriba una nueva petición al servidor, simplemente se verifica que no se excede de la cantidad máxima de clientes admitidos, calculada en la prueba de estrés.

Capítulo 3

Transmisión de *streams* en VoD-NFR

En el capítulo anterior se ha expuesto la arquitectura del nuevo sistema VoD-NFR, junto con una descripción funcional referente a los aspectos más relevantes que deben ser tenidos en cuenta en un sistema de LVoD. Sin embargo, para que el sistema VoD-NFR pueda llevar adelante todas sus funciones, necesita de un componente que gestione las comunicaciones al nivel de la capa de red. Por esto, a continuación se presenta el módulo de transmisión de streams, para el cual se han diseñado dos nuevos protocolos, uno destinado al transporte de streams y otro cuya función es la de controlar la congestión de la red. Asimismo, se describe una implementación del presente módulo para un sistema operativo de propósito general.

El módulo transmisor de *streams* (STM) del sistema VoD-NFR cumple con el objetivo de proporcionar, al módulo LCS, un soporte total al servicio T-VoD y un transporte de los vídeos con QoS. Pero sus tareas no terminan ahí, ya que además, recolecta información del estado de las comunicaciones, que es de suma importancia para que los módulos LCS y QoSA puedan operar de manera efectiva.

Para el STM hemos desarrollado un Protocolo Transmisor de *Streams* (STP, *Stream Transmission Protocol*) y un componente Planificador del Tráfico de Red (NTS, *Network Traffic Scheduler*). El NTS incluye un nuevo protocolo de control de congestión, al que hemos denominado Protocolo de Adaptación de Tasa Mejorado (ERAP, *Enhanced Rate Adaptation Protocol*), utilizado para transmitir los datos de los vídeos y para generar información sobre el estado de las comunicaciones de cada canal. El STP es un protocolo que se ubica por encima del ERAP, el cual se ocupa de mantener la pista del origen y sección de media enviada, para que el cliente pueda dirigir los datos a los *buffers* correspondientes.

Las características de un servicio de T-VoD requiere de respuestas rápidas por parte de un módulo de transmisión. Cuando se solicita un comando interactivo, nuevos datos de vídeo deben ser enviados al cliente para atender la solicitud. Por ejemplo, si un cliente que visualiza un vídeo a velocidad normal requiere un avance rápido a $2x$, el sistema debe transferir inmediatamente los datos de la versión del vídeo de velocidad $2x$, y dejar de transmitir los datos del vídeo de velocidad normal. Sin embargo, esto no es posible hacerlo utilizando los servicios de transmisión de los sistemas operativos de propósito general (a través de los *sockets*). Usualmente, la función de envío (*send*) para UDP y TCP, copia los datos salientes en áreas del núcleo del sistema operativo y permite que la aplicación continúe la ejecución mientras los datos se transmiten por la red. De esta manera, la aplicación pierde el

control de estos paquetes encolados por el sistema operativo, imposibilitando que ellos sean descartados (antes de salir a la red) para enviar inmediatamente los datos de la versión del vídeo requerida.

Para soportar este requerimiento, el STM fue dotado de un mecanismo especial que mantiene un control total de los paquetes encolados en el sistema. Así, el STM descarta los paquetes inútiles, provee un informe al LCS sobre los paquetes no entregados, y transmite el nuevo contenido solicitado. Gracias a este control de paquetes, en el caso de requerirse una migración de una sesión de un cliente, el STM puede informar al LCS sobre las secciones de los vídeos que no han sido entregados al cliente, para que sean reenviados por el nuevo servidor.

A continuación, se presentan las generalidades del componente NTS, luego se describen los protocolos ERAP y STP, y finalmente se describe la implementación del STM.

3.1. NTS

El NTS es el corazón del STM, y el encargado de gestionar las comunicaciones al nivel de red. Es necesario que el NTS soporte una gran cantidad de conexiones de clientes y sea capaz de transmitir los datos a la máxima capacidad de la conexión de red del servidor. El NTS informa al LCS sobre el estado de las comunicaciones con cada cliente. De esta manera, el LCS puede llevar a cabo sus tareas con información real y actualizada de la red. Si el NTS encuentra que la comunicación de cierta conexión ha mejorado, el LCS podrá incrementar la tasa de transmisión de ese canal. En caso de que la comunicación empeore, el LCS reducirá la tasa de transmisión inmediatamente.

El NTS es equipado con un algoritmo de control de congestión para cumplir con sus responsabilidades. Todo algoritmo de control de congestión que se utilice en la red Internet debe tener la propiedad de ser TCP-*Friendly*. Es decir, el uso de ancho de banda, en un estado estable, no debe ser mayor que el requerido por TCP bajo circunstancias similares [45]. Si usuarios maliciosos o irresponsables capturan más ancho de banda del que les corresponde, degradarían el servicio de entrega de los usuarios que cooperan para el buen funcionamiento de la red. Además, amenazarían la estabilidad y operatividad del sistema entero [85]. Si alguien diseña una aplicación que envía una gran cantidad de paquetes UDP, sin disminuir la tasa de transmisión cuando la red está congestionada, esta aplicación se apropiaría de casi todo el ancho de banda del camino de comunicación. El resto de los flujos disminuirían la tasa de sus transmisiones al detectar la congestión, pero la aplicación sin control continuaría transmitiendo a la misma frecuencia. Es claro que

una aplicación de esta característica no es TCP-*Friendly* y, por lo tanto, debería ser sancionada.

Existen diferentes estrategias para implementar el control de congestión, pero el NTS usa una estrategia de tipo caja-negra[1] dado que no hay una retroalimentación (*feedback*) de los dispositivos de interconexión (encaminadores). La única retroalimentación al NTS es provista por los receptores. Si el NTS varía la tasa de transmisión con los clientes de forma abrupta, la información de ancho de banda que el NTS provea al LCS será válida sólo para un pequeño período de tiempo. Por el contrario, el LCS requiere que el intervalo de validez de la información sea lo más largo posible, idealmente del tamaño del intervalo de tiempo de planificación del LCS (el *slot*). En consecuencia, es importante que el tráfico fluya de forma continua, y con pocas variaciones de la tasa de envío a lo largo del tiempo. Esta propiedad es conocida como "tasa de envío suavizada" (*Smooth Sending Rate*).

Se ha optado por el uso de un protocolo *basado en tasa* (en vez de basado en ventana) porque ellos tienen la ventaja de no transmitir paquetes en ráfagas. Si un emisor tiene la capacidad de transmisión de b paquetes por segundos, es mejor enviar un paquete cada $1/b$ segundos y no una secuencia de b paquetes cada vez que ha transcurrido un segundo. Una secuencia de b paquetes podría ser inaceptable para un dispositivo de interconexión que no tiene la cantidad suficiente de memoria para almacenarlo temporalmente. Por otro lado, si la memoria fuese suficiente, las largas colas en los encaminadores incrementarían el retardo de extremo a extremo. Esto puede provocar retransmisiones (al expirar el tiempo de los paquetes), las cuales producirán un incremento del estado de congestión.

Existen en la literatura varios protocolos de control de congestión basados en tasa, por ejemplo LDA, RAP, TFRC, y SCP (ver sección 1.6.2). Muchos de los protocolos existentes son propuestos por investigadores, y en muchos casos no existe una implementación real, o ellos no se adaptan a los requerimientos del NTS. Por lo tanto, se ha desarrollado el protocolo ERAP, especialmente diseñado y optimizado para cumplir con los requerimientos del NTS. La política de control de congestión de ERAP se basa en la política del RAP debido a que este protocolo tiene todas las propiedades esperadas: basado en tasa, TCP-*Friendly*, caja-negra, y su tasa de transmisión es suavizada. Además, está incluido en el simulador más conocido en el ámbito de investigación y educación, que es el Network Simulator - NS2 [86, 87]. El principal problema que tiene RAP es que, oficialmente, no tiene una implementación real.

[1] En esta estrategia, también conocida como de "extremo a extremo" *(end-to-end)*, la red es representada como una caja negra o cerrada. Esto quiere decir que el protocolo opera entre las entidades de los extremos de la comunicación, sin intervención alguna de los elementos intermedios de la red.

En la siguiente sección, se presenta el ERAP, y se explica su política de control de congestión y sus características funcionales.

3.2. ERAP

ERAP es un protocolo de control de congestión TCP-*Friendly*, de caja-negra y basado en tasa la cual produce una transmisión suavizada. Este protocolo provee un envío de datagramas, en el cual la confiabilidad es opcional (se utiliza activada en el sistema VoD-NFR), y el orden de los paquetes y la protección contra duplicados no están garantizados (VoD-NFR utiliza otro protocolo de capa superior para obtener estas propiedades). ERAP provee, a los programas de aplicación, un procedimiento para enviar mensajes a otros programas con un mínimo uso de los recursos de los nodos (memoria y CPU) y la red.

En ERAP, un paquete es encapsulado en un paquete UDP para su transmisión por la red IP. El protocolo tiene un mecanismo de retransmisiones debido a que el sistema VoD-NFR requiere de comunicaciones confiables. Sin embargo, este mecanismo puede ser desactivado si algún otro programa de aplicación necesita usar ERAP y no desea las retransmisiones. La sobrecarga del protocolo en la red es muy reducido debido a que usa encabezados de paquetes muy cortos. Para minimizar el consumo de recursos del sistema, ERAP incluye una gestión de *buffers* que reduce el uso de CPU y memoria. El gestor de eventos soporta períodos de inactividad de las conexiones con el menor consumo de recursos, permitiendo incrementar el número de conexiones concurrentes. La arquitectura de ERAP fue diseñada con un envío y recepción de paquetes centralizado, el cual permite: (1) ahorrar recursos al compartirlos entre diferentes conexiones, (2) tener un control total de los paquetes en los *buffers* hasta que ellos salen a la red, (3) mejorar el comportamiento del protocolo distribuyendo homogéneamente los paquetes a lo largo del tiempo, respetando la velocidad de transmisión de la red.

La política de control de congestión se describe en la sección 3.2.1, y las características funcionales son explicadas con detalle en la sección 3.2.2.

3.2.1. Política de control de congestión

Una política de control de congestión especifica cuándo, cómo, y con qué frecuencia se incrementa o decrementa la tasa de transmisión. La política de control de congestión de ERAP deriva del protocolo RAP, y utiliza un algoritmo AIMD (*Additive Increase Multiplicative Decrease*) para adaptar la tasa de transmisión de una forma TCP-*Friendly*.

El protocolo ERAP es implementado principalmente en el emisor. Un emisor ERAP envía paquetes de datos con un número de secuencia, y un receptor ERAP envía un reconocimiento (ACK, *acknowledgement*) por cada paquete. El tiempo transcurrido desde la operación de envío hasta la recepción del ACK es utilizado para obtener el valor del RTT (*Round-Trip-Time*). Este protocolo considera las pérdidas de paquetes como síntoma de congestión. Para detectar pérdidas se utiliza la información de los ACK (por medio de distancias en el espacio de secuencia) y tiempos de espera límite (*timeouts*). A diferencia de TCP, un emisor ERAP puede enviar varios paquetes antes de recibir un nuevo ACK.

Un paquete de ACK incluye el número de secuencia del paquete de datos entregado y, para recuperarse de pérdidas simples de ACKs, se agrega la siguiente información de redundancia:

- *lastRecv:* es el número de secuencia del último paquete recibido.
- *lastMiss:* es el número de secuencia del último paquete faltante anterior a *lastRecv,* o 0 si no falta ningún paquete.
- *prevRecv:* es el número de secuencia del paquete recibido anterior a *lastMiss*, o 0 si *lastRecv* es el primer paquete de la secuencia.

Por ejemplo, suponer que el servidor ha recibido reconocimientos de los paquetes con números de secuencia “1, 4, 7”. En este caso, los valores de los parámetros son: *lastRecv* = 7, *lastMiss* = 6, y *prevRecv* = 4.

Un paquete con número de secuencia Seq_i será considerado recibido si se cumple la siguiente proposición:

$$((lastRecv \geq Seq_i) \wedge (Seq_i > lastMiss)) \vee (Seq_i = prevRecv)$$

En otro caso, el paquete será considerado como perdido si hay una distancia en el espacio de secuencia mayor o igual a tres (es decir, Seq_i es menor o igual al último numero de secuencia menos 3), o si el tiempo transcurrido desde el envío del paquete es mayor que el tiempo de espera límite denominado *Timeout*. El valor de *Timeout*, que define el tiempo de vida o expiración de los paquetes, es actualizado basándose en la última muestra de RTT usando el algoritmo de Jacobson/Karels [88]. Una lista llamada *transmissionHistory* almacena los paquetes enviados pero no reconocidos. Todos los paquetes en esta lista son puestos en el estado *ignorar* cuando se detecta la pérdida de un paquete. Así, el algoritmo actúa solo una vez ante una ráfaga de pérdidas de paquetes, correspondientes al mismo caso de congestión.

El algoritmo usa un esquema de adaptación AIMD que, si no detecta congestión, incrementa periódicamente y de manera aditiva la tasa de transmisión. En caso de detectar congestión, la tasa de transmisión se decrementa inmediatamente y de forma multiplicativa. La tasa de transmisión es incrementada o decrementada, decrementando o incrementando respectivamente la distancia entre paquetes (IPG, *Inter Packet Gap*). El *IPG* es actualizado una vez por cada *SRTT* segundos. El *SRTT* es una estimación del RTT y, al igual que *Timeout*, es actualizado basándose en la última muestra del RTT usando el algoritmo de Jacobson/Karels.

Cuando no se detectan pérdidas, la tasa de transmisión S_i se aumenta en un valor determinado α, actualizando el valor del *IPG* de acuerdo a la ecuación:

$$S_i = \frac{PacketSize}{IPG_i} \qquad IPG_{i+1} = \frac{IPG_i \times C}{IPG_i + C} \tag{3.1}$$

donde C es medida en unidades de tiempo y determina el valor de α. El valor de *SRTT* es asignado a C para emular el mecanismo de ajuste de ventana de TCP en un estado estable. De esta manera, α representa el incremento de un paquete en cada punto de ajuste. Al detectar pérdidas de paquetes, la tasa de transmisión es decrementada multiplicativamente por un cierto valor β, actualizando el valor del *IPG* de acuerdo a la siguiente ecuación:

$$IPG_{i+1} = IPG_i / \beta \tag{3.2}$$

Se considera un valor de $\beta = 0{,}5$ para seguir el comportamiento adoptado por TCP.

La funcionalidad esperada sería de un incremento progresivo de la tasa de transmisión en ausencia de congestión, y un decremento abrupto de esta ante congestión (diente de sierra), tal como puede observarse en la figura 3.1.

3.2.2. Características funcionales

Las características funcionales del protocolo ERAP son descriptas a continuación:

Confiabilidad

La confiabilidad en ERAP es opcional debido a que algunos sistemas no requieren retransmitir todos los paquetes perdidos. Por ejemplo, muchos sistemas de VoD retransmiten solo los paquetes que incluyen cuadros de alta relevancia, sin embargo, el sistema VoD-NFR no descarta ningún paquete para no degradar la calidad de

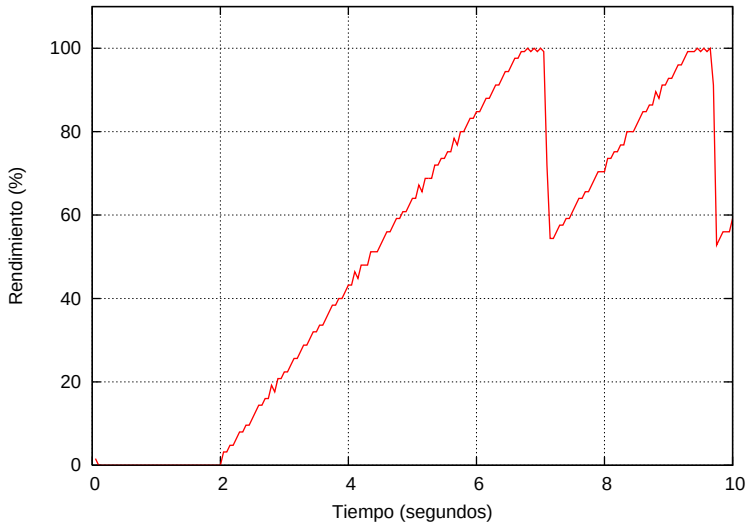
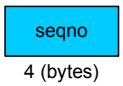


Figura 3.1: Comportamiento característico de ERAP

la imagen. En ERAP, los paquetes son retransmitidos con un nuevo número de secuencia y tienen la prioridad más alta de acceso a la red. Estos paquetes no pueden ser ordenados y no hay garantías de protección de duplicados porque los mismos datos tienen distintos números de secuencia. Esto no representa ningún problema porque el STM coloca el protocolo STP por encima del ERAP para identificar las secciones de vídeo transmitidas.

Formato de encabezados

La figura 3.2 (a) y (b) muestra los encabezados de los paquetes de datos y reconocimiento (ACK) del protocolo ERAP, respectivamente. El encabezado de los paquetes de datos tiene solo un campo, denominado *seqno*, que es el número de secuencia que identifica el paquete. El encabezado de ACK se compone de los siguientes campos: *port* es el puerto destino del paquete de datos que está siendo reconocido; *seqno* es el número de secuencia del paquete que está siendo reconocido; *lastRecv*, *lastMiss*, y *prevRecv* constituyen la información redundante de reconocimiento (ver sección 3.2.1).



(a) Encabezado de paquetes de datos

(b) Encabezado de paquetes de ACK

Figura 3.2: Formato de los encabezados de ERAP

Arquitectura del protocolo

El objetivo del diseño de la arquitectura del protocolo es reducir el consumo de recursos y maximizar el rendimiento, especialmente con un alto número de conexiones de clientes. ERAP usa un esquema centralizado para el envío de paquetes y recepción de ACKs desde múltiples destinos, en contraposición a los esquemas típicos que son descentralizados.

La recepción de ACKs descentralizada no funciona bien cuando es utilizada en un servidor con muchas sesiones activas. En este esquema, cada emisor debe gestionar su propia recepción de paquetes. Así, en una implementación real, cada emisor tendría un *thread* encargado de la recepción de paquetes (un *thread* por conexión). ERAP usa un esquema centralizado de recepción de ACKs desde múltiples destinos. De esta manera, solo un simple *thread* es necesario para controlar todos los paquetes de ACK. Cuando un paquete llega al servidor, este es administrado por un módulo central, llamado gestor ERAP, y distribuido a los correspondientes emisores ERAP.

Veamos el escenario que se plantea en la figura 3.3 (a), con el envío independiente efectuado por cada emisor. Se tiene un servidor con una conexión de red de 10 Mb/s, cada emisor (E) tiene una comunicación de 5 Mb/s con su correspondiente receptor. Si cada emisor debe enviar 1 Mbit, en 1 segundo se habrá completado la transferencia de todos los paquetes. Sin embargo, como cada emisor envía los paquetes de manera independiente a los demás, todos los emisores intentarán enviar en el segundo 0 y finalizar en el segundo 0,2. Esto trae aparejado algunos problemas, que difieren según si la operación de envío (*send*) de paquetes UDP es bloqueante o no, en determinado sistema operativo.

A continuación se presentan los problemas relacionados a cada uno de los tipos de envío de paquetes UDP:

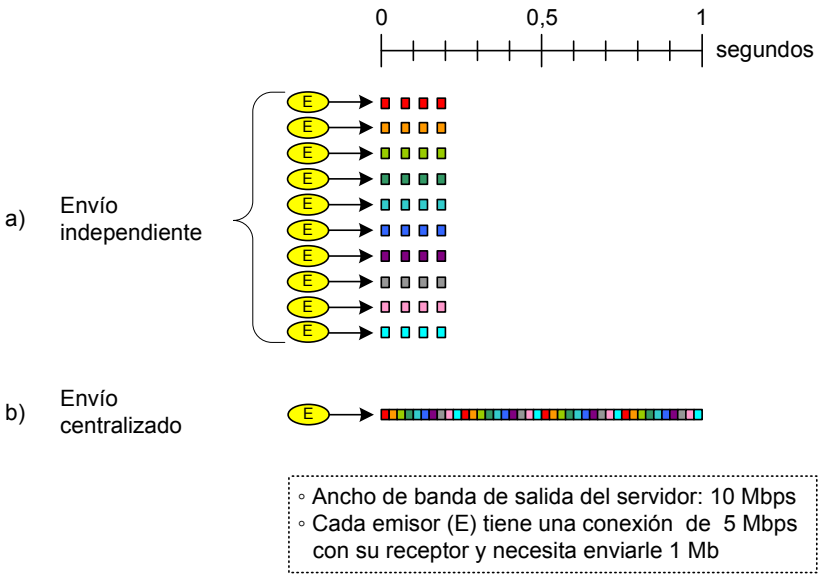


Figura 3.3: Envío independiente frente a envío centralizado

- *Envío bloqueante*. Con un *send* bloqueante, el problema aparece cuando varios emisores intentan enviar paquetes a la red simultáneamente. Entonces, el sistema operativo bloqueará el envío de paquetes hasta que la tarjeta de interfaz de red (NIC, *Network Interface Card*) los acepte. Durante este tiempo, los *threads* que ejecutan el *send*, permanecerán bloqueados reduciendo el rendimiento del sistema. En el problema anterior, planteado por la figura 3.3 (a), algunos *threads* permanecerán bloqueados hasta 0,8 segundos, desde el segundo 0,2 hasta el segundo 1 (para el *thread* asociado al último paquete que sale del servidor). Debido a la gran cantidad de conexiones que gestiona un servidor de VoD es posible que, con este esquema, muchos *threads* permanezcan en ejecución sin ninguna utilidad. Un *threads* bloqueado consume memoria y produce que el núcleo del sistema operativo tenga más recursos que administrar. Además, los sistemas operativos generalmente limitan la cantidad de *threads* que un proceso puede tener asociado, lo que implica que estos *threads* bloqueados produzcan un marcado desaprovechamiento de recursos.

- *Envío no bloqueante*. En el caso del *send* no bloqueante, si la longitud de la cola de transmisión en el emisor es muy pequeña, muchos paquetes serán descartados. Entonces, la longitud de la cola es incrementada para tolerar las fluctuaciones del tráfico y reducir el número de descartes. Sin embargo, ahora

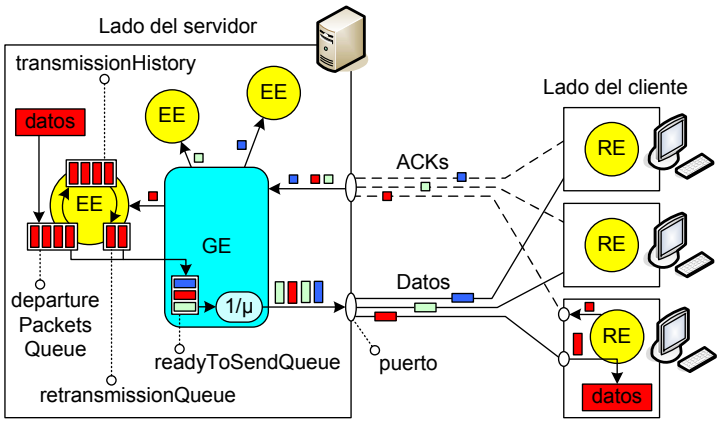
los paquetes podrían tener una marca de tiempo de envío incorrecta a causa de la gran distancia existente entre el tiempo de la llamada a la operación *send* y el tiempo en el cual el paquete fue enviado. Por lo tanto, en base a información falsa, el algoritmo de control de congestión tomará decisiones equivocadas y que no se corresponden con la realidad. Por ejemplo, si un paquete permanece mucho tiempo en la cola del sistema operativo, su tiempo límite podría expirar y entonces el paquete sería retransmitido.

ERAP resuelve todos estos problemas centralizando el envío de paquetes de todas las conexiones, entregando los paquetes al sistema operativo de una manera regulada. Esta regulación implica una distribución de los paquetes en el tiempo de tal manera que una vez que salen del control del ERAP, puedan ser inmediatamente despachados a la red. La figura 3.3 (b) muestra la regulación efectuada por un solo emisor, encargado de enviar los paquetes de todas las conexiones.

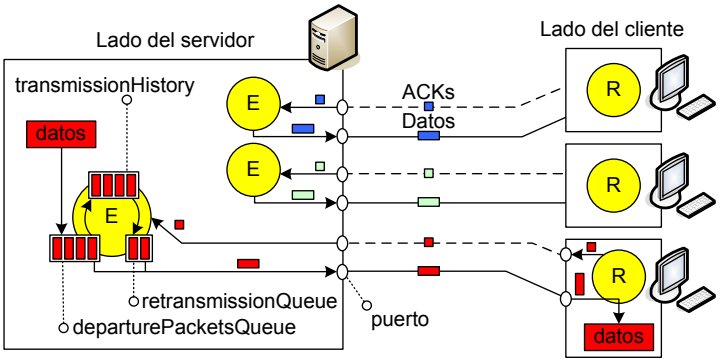
Para soportar el esquema centralizado de envío de paquetes, ERAP incluye una gestión de *buffers* que funciona de la siguiente manera. Una cola global, denominada *readyToSendQueue*, es utilizada para centralizar todos los paquetes listos para enviar de cada emisor ERAP. El gestor ERAP usa esta cola para enviar los paquetes a la red a una tasa regulada. De esta forma, ERAP nunca intenta transmitir más datos que los aceptados por la red. Estos cambios resuelven el problema del *send* bloqueante. En el caso del *send* no bloqueante, el problema queda solucionado porque la cola de la NIC siempre estará prácticamente vacía.

La arquitectura del ERAP se muestra en la figura 3.4 (a), donde *EE* representa un emisor ERAP, y *RE* son los receptores ERAP. Cada emisor ERAP recibe paquetes de datos de la aplicación, y los almacena en la cola *departuraPacketsQueue*. Los paquetes perdidos son almacenados en una cola, llamada *retransmissionQueue*, para ser retransmitidos. Cuando el gestor ERAP (GE en la figura) envía un paquete a la red, este es registrado en la lista *transmissionHistory*. En el lado del cliente, los paquetes de datos son recibidos por los receptores ERAP, y un paquete de ACK es enviado al servidor. Cuando el ACK llega al servidor, el gestor ERAP lo entrega al emisor ERAP correspondiente. Durante el ciclo de vida del paquete en el NTS, sólo una copia del paquete es mantenida en memoria, evitando el alto costo de las operaciones de copia de memoria.

En la figura 3.4 (b) se aprecia una arquitectura genérica de un protocolo de transporte, tal es el caso de la mayoría de los protocolos, como por ejemplo RAP y TCP. Cada conexión es totalmente independiente de las demás y no existe ningún tipo de optimización global.



(a) Arquitectura del ERAP, con recepción de ACKs y envío centralizado



(b) Arquitectura genérica de un protocolo de transporte

Figura 3.4: Arquitectura del ERAP frente a una arquitectura genérica

Diagrama de flujo del algoritmo

El diagrama de la figura 3.5 describe las operaciones llevadas a cabo por ERAP, en el lado del servidor, para manejar cada conexión. Las flechas gruesas representan los eventos que dirigen el algoritmo, las cajas representan los procedimientos principales, y las flechas finas representan llamadas a procedimientos.

El algoritmo comienza cuando se entregan, al NTS, nuevos paquetes de datos para enviar al cliente. Entonces, se ejecuta el procedimiento *Send*, el cual pone los paquetes en la cola *departurePacketsQueue*. A continuación, se activan los temporizadores *RttTimer* e *IpgTimer* si ellos se encuentran desactivados. El *RttTimer* es disparado una vez por cada *SRTT* segundos, produciendo la ejecución del procedimiento *RttTimeout* para decrementar el *IPG*. El *IpgTimer* es disparado una vez cada *IPG* segundos, para dar ejecución al procedimiento *IpgTimeout*. En este procedimiento, si se detectan pérdidas de paquetes a causa de que se ha superado el tiempo límite de espera (tarea efectuada por el procedimiento *TimerBasedLossDetection*) entonces se llama al procedimiento *LossHandler* para incrementar el *IPG*. Si no se detectan pérdidas entonces se selecciona un paquete de datos para enviarse al cliente. Este paquete se toma de la cola *retransmissionQueue* o *departurePacketsQueue*, donde los paquetes de la primer cola tienen mayor prioridad que los de la segunda. Después, se agrega un número de secuencia al paquete y se coloca en la cola *readyToSendQueue* para ser enviado a la red. Cuando el paquete ha sido enviado, se ejecuta el procedimiento *PacketExitFromReadyToSendQueue*. Este procedimiento almacena una copia del paquete en la lista *transmissionHistory*. Además, se registra el tiempo de envío del paquete para poder calcular el RTT una vez recibido el ACK.

Cuando un paquete de datos arriba al cliente, el receptor ERAP inmediatamente genera un paquete ACK. Luego, el paquete de datos es entregado al programa de aplicación.

En el momento en que el servidor recibe el paquete de ACK, se ejecuta el procedimiento *RecvAck*. En este procedimiento se actualizan los valores del *SRTT* y del *Timeout* a partir del nuevo valor de RTT obtenido. Si se detecta una pérdida de paquete basándose en la información de reconocimiento (tarea efectuada por el procedimiento *AckBasedLossDetection*), entonces se llama al procedimiento *LossHandler* para incrementar el *IPG*.

Sin importar si la pérdida del paquete se detecta a través de la función de detección basada en información de reconocimiento (*AckBasedLossDetection*) o basada en el tiempo de vida de los paquetes (*TimerBasedLossDetection*), el paquete es puesto en la cola *retransmissionQueue* para ser retransmitido. Todos los paquetes, ya sean recibidos o perdidos, son eliminados de la lista *transmissionHistory*.

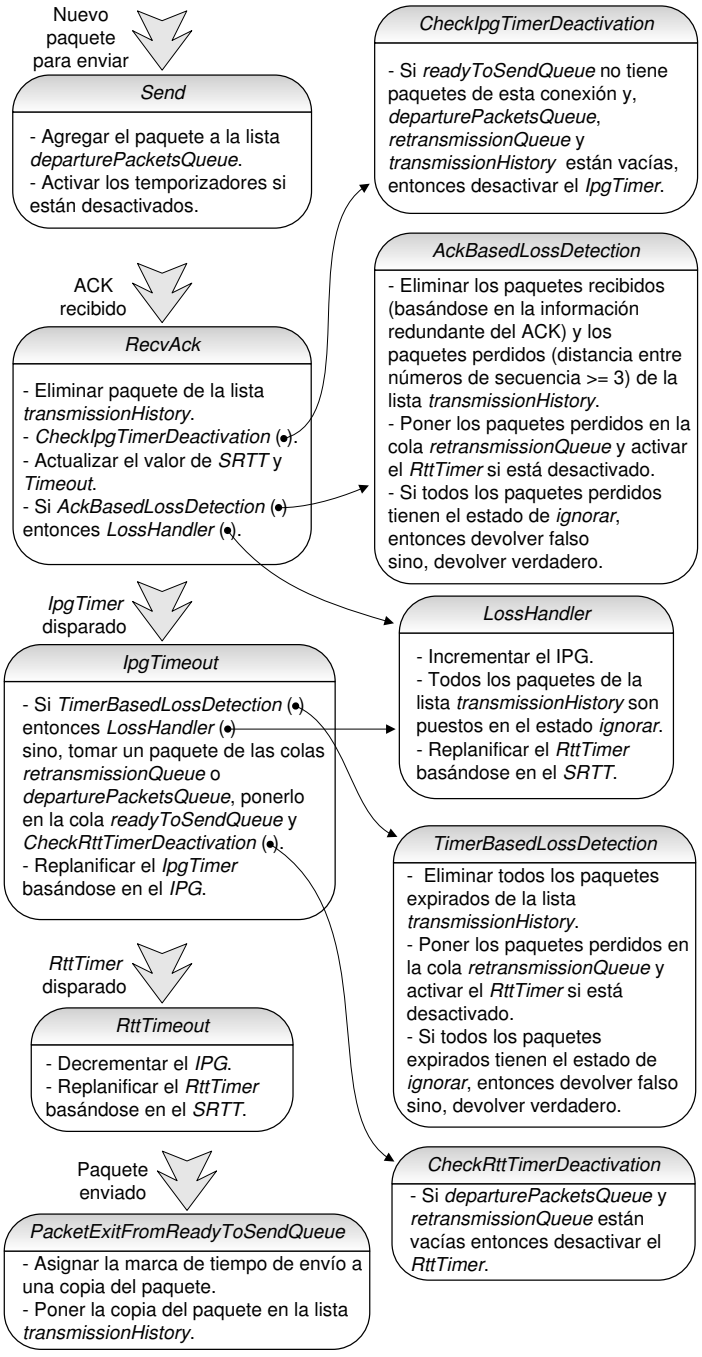


Figura 3.5: Algoritmo de ERAP

Cuando en una sesión se suspende temporalmente la transmisión de paquetes, y luego es reiniciada, el protocolo debería continuar la transmisión suponiendo que el estado de la red no ha cambiado. De otra forma, se desperdiciaría la información del estado de la red, recopilada durante el transcurso previo de uso de la conexión. En este momento de inactividad (es decir, cuando no hay más paquetes para enviar) no tiene sentido continuar disparando eventos del protocolo. Por lo tanto, ERAP detecta los periodos de inactividad y desactiva los temporizadores (procedimientos *CheckIpgTimerDeactivation* y *CheckRttTimerDeactivation*). ERAP mantiene el estado de la sesión tal que, cuando se reanuda la transmisión, se reactivan los temporizadores y la tasa de transmisión continúa desde el punto previo.

Generación de información requerida por los módulos LCS y QoSA

El LCS necesita información del STM para planificar la entrega de la media de un determinado intervalo, y el módulo QoSA precisa muestras del ancho de banda para detectar los fallos de red. Esta información es generada por el algoritmo de control de congestión del NTS, el ERAP, y comprende:

- Capacidad de transmisión disponible por conexión: Para planificar un *slot* de tiempo, el LCS necesita conocer la capacidad máxima c_i, expresada en Mbit, que el STM puede transmitir al cliente i durante el siguiente *slot*. El valor de c_i es calculado de la siguiente manera:

$$c_i = \text{máx}\,(slot \times b_i - e_i, 0) \tag{3.3}$$

  donde *slot* es medida en segundos y representa la longitud del *slot*; b_i es la tasa de transmisión promedia supuesta (medida en Mb/s) soportada por la red en el camino de comunicación con el cliente durante el siguiente *slot*; y e_i es el número de Mbit encolados por el NTS que están pendientes de ser transmitidos desde algún *slot* previo. La tasa de transmisión media b_i se predice de una manera optimista, y se calcula como:

$$b_i = MaxDataSize \times \left(\frac{1}{IPG_i} + \frac{slot}{2 \times SRTT} \right) \tag{3.4}$$

  donde *MaxDataSize* es el número máximo de bits de carga útil de un paquete de datos ERAP, *slot* es la longitud medida en segundos, e *IPG* y *SRTT* son las variables previamente definidas (ver sección 3.2.1). Se supone que, durante el tiempo del *slot*, la tasa de transmisión es incrementada, pero nunca decrementada. De esta forma, se evita la inactividad del NTS mientras el

LCS tenga datos disponibles para ser transmitidos.
El número de Mbit encolados e_i es calculado de la siguiente manera:

$$e_i = dpq_i + rq_i \tag{3.5}$$

donde dpq_i y rq_i son el tamaño en Mbit de las colas *departurePacketsQueue* y *retransmissionQueue*, respectivamente.

- Muestras de ancho de banda por conexión: Para que el módulo QoSA sea capaz de predecir el ancho de banda de red, y así detectar los fallos de la misma, es necesario tomar muestras del ancho de banda $sampleBW_i$, expresada en Mb/s, de una conexión con un cliente *i* dado. Las muestras del ancho de banda se toman cada vez que ERAP modifica el *IPG*, y se calculan según la siguiente ecuación:

$$sampleBW_i = \begin{cases} 0 & npkts_i \geq 1/IPG_i \\ b_i & en\ otro\ caso \end{cases} \tag{3.6}$$

donde b_i se define en la ecuación (3.4), y $npkts_i$ es el número de paquetes enviados después de recibir el último paquete de ACK; por lo tanto, se supone una muestra de 0 cuando se han transmitido la misma cantidad, o más, de paquetes que pueden transmitirse de manera continua durante un segundo, sin recibir ningún ACK.

- Datos pendientes para ser transmitidos por el STM: Para planificar un *slot* de tiempo, el LCS necesita conocer el número de datos disponibles que pueden ser transmitidos durante ese intervalo. Este número es obtenido restando los Mbit encolados actualmente (por el NTS) a la capacidad de conexión de red. Los Mbit encolados, *eGlobal*, es calculado basándose en la ecuación siguiente:

$$eGlobal = rsq + \sum_{i=1}^{count} min\left(slot \times b_i, e_i\right) \tag{3.7}$$

donde *rsq* es el tamaño en Mbit de la cola *readyToSendQueue*, *count* es el número total de emisores ERAP, *slot* es la longitud del *slot* medida en segundos, y b_i y e_i son definidos por las ecuaciones (3.4) y (3.5), respectivamente.

3.3. Protocolo de transmisión de *streams* (STP)

El protocolo de transmisión de *streams* (STP, que son las siglas de *Stream Transmission Protocol*), es un protocolo diseñado para el transporte de información multimedia. Fue desarrollado de acuerdo a las necesidades del transporte de contenido multimedia del sistema VoD-NFR.

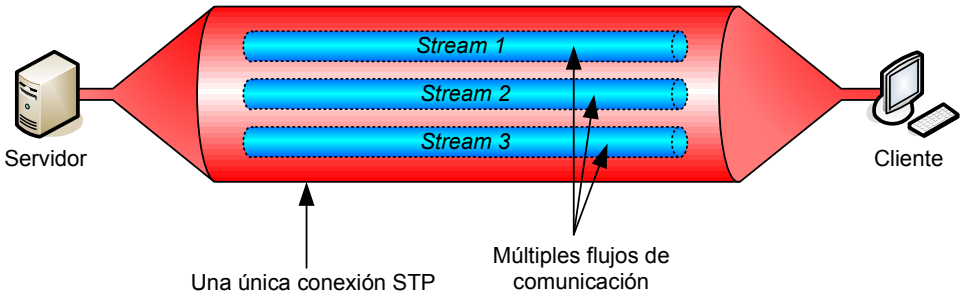


Figura 3.6: Propiedad *multi-streaming* de STP

STP ofrece la capacidad de *multi-streaming*, es decir, soporta múltiples *streams* de datos. En VoD-NFR, cuando un usuario solicita la ejecución de un comando interactivo, el servidor comienza a enviar datos que corresponden a un archivo de vídeo diferente al que se estaba transmitiendo antes de solicitar tal comando interactivo. No obstante, muchos paquetes pueden estar en viaje por la red, los cuales corresponden a la versión anterior del vídeo. Entonces, el cliente debe ser capaz de diferenciar los distintos contenidos multimedia transmitidos por un mismo canal de comunicación. En la figura 3.6 se observan *streams* independientes que son transmitidos por un mismo canal. Para soportar *multi-streaming*, STP asocia a cada *stream* un número identificador que es codificado dentro de los paquetes STP.

STP mantiene la pista de los datos encapsulados en paquetes mediante la asignación de un valor que determina la ubicación de los datos del paquete en el *stream*. De esta forma, el cliente puede determinar la ubicación de los datos recibidos respecto del *stream*. Pero esta no es la única utilidad que se obtiene al mantener la pista de los datos transportados por los paquetes. La ejecución de un comando interactivo implica que el STM vacíe las colas de paquetes del canal en curso para, inmediatamente, comenzar la transmisión de la nueva media solicitada. Este vaciado de colas implica que ciertos datos que el LCS suponía haber entregado al cliente, realmente no lo han sido. Por lo tanto, el STM debe informar al LCS de estos paquetes no entregados. Haciendo uso de los datos del STP, se proporciona al LCS la ubicación de los datos de los paquetes en el *stream* junto con sus longitudes. De esta manera, el LCS puede actualizar los VBMs (VBM es el mapa del *buffer* del vídeo, sección 2.4) correspondientes para reflejar los datos de vídeo que el cliente no ha recibido.

ERAP no garantiza ni el orden de entrega de los paquetes ni detecta duplicados. A simple vista, esto parecería un problema que el STP debería de atacar, pero realmente no es necesario que lo haga. El orden de entrega de los paquetes deja de tener sentido cuando un paquete, por si mismo, tiene asociado la posición de los

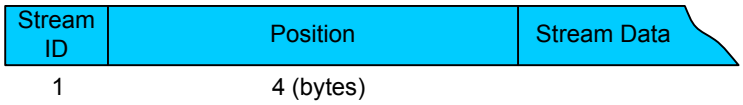


Figura 3.7: Estructura de un paquete STP

datos que transporta dentro del *stream* de datos original. La detección de duplicados agregaría de manera innecesaria una carga extra al sistema, debido a que esta tarea es inherente a los VBMs.

3.3.1. Estructura de un paquete STP

Los segmentos de STP son enviados en paquetes ERAP. La estructura de un paquete STP se muestra en la figura 3.7, y consiste de los siguientes campos:

Stream ID: 8 bits. Es el identificador del *stream* que da soporte al *multi-streaming*. Particularmente, se utiliza una codificación de la velocidad de reproducción como identificador. La codificación es un número con signo en complemento a 2, utilizando signo positivo o negativo para reproducción de avance o retroceso, respectivamente. El identificador codificado es la velocidad de reproducción multiplicada por 10. Por ejemplo, un retroceso rápido a velocidad 2,5x es codificado como -25.

Position: 32 bits. Determina la posición de inicio de los datos transportados por el paquete en el *stream* original. La longitud de los datos es deducida a partir del campo *Total Length* contenido en el encabezado IP.

Stream Data: Son los datos del *stream* transportados por el paquete.

3.4. Implementación

Un protocolo basado en tasa transmite paquetes a intervalos regulares, en vez de hacerlo en respuesta al arribo de paquetes de reconocimiento. Este tipo de protocolos requiere un análisis paquete por paquete. En redes de alta velocidad, considerando paquetes de 1500 bytes, los intervalos requeridos entre paquetes llegan a 12 µs en una red de 1 Gb/s y 1,2 µs en una red de 10 Gb/s. En una entrega de paquetes a la red a gran velocidad, una gran cantidad de paquetes deben procesarse por unidad de tiempo produciendo un alto consumo de CPU. Es normal encontrar

en este tipo de sistemas que el cuello de botella esté en los nodos (u ordenadores) y no en la capacidad de la red [4]. Por ejemplo, se ha reportado que una conexión de red de 1 Gigabit Ethernet (GbE) satura a un Pentium IV de 2,4 GHz. Es decir, solamente la comunicación requiere el uso de toda la capacidad de la CPU, impidiendo la ejecución de las aplicaciones. Para solucionar estos problemas, se han propuesto soluciones a nivel de procesadores y a nivel de las NIC. Intel ha implementado la tecnología "Intel I/OAT", que hace disponible en las nuevas plataformas basadas en procesadores Intel Xeon Dual-Core y Quad-Core. La industria de redes y el IETF (*Internet Engineering Task Force*) han definido el protocolo RDMA (*Remote Direct Memory Access*), que permite a un ordenador colocar directamente información en la memoria de otro ordenador, con una demanda mínima de ancho de banda del bus de memoria y costo de procesamiento de CPU. Sin embargo, este protocolo está destinado a comunicaciones principalmente de *clusters*. También han aparecido los dispositivos *TCP Offload Engine* (TOE), que son tarjetas de red especializadas que desplazan una buena parte del costo de procesamiento del protocolo TCP/IP al adaptador de red. Es decir, en vez de implementar el protocolo TCP/IP en software se hace directamente en hardware, liberando a la CPU de gran parte del procesamiento de las comunicaciones.

Como se puede apreciar en todos estos casos, los protocolos de control de congestión para una red de alta velocidad tienden a ser implementados directamente en hardware. La propuesta de implementación del protocolo ERAP, aquí presentada, se lleva a cabo completamente en software. El protocolo es implementado absolutamente en espacio de usuario en un sistema operativo de propósito general. Claramente, una implementación en software del protocolo ERAP (al igual que TCP) restringe mucho la velocidad de la conexión de red que puede soportar el sistema. No obstante, ha sido evaluada hasta una velocidad de 100 Mb/s con óptimos resultados, algo para nada trivial si se considera que es un protocolo basado en tasa (y no en ventana como TCP).

Transmitir paquetes de 1500 bytes a 100 Mb/s requiere transmitir un paquete cada 120 µs. No es fácil generar una entrega de paquetes precisa con intervalos tan pequeños. Sin embargo, esto es muy importante porque si el mecanismo del tratamiento de las expiraciones de tiempo no es preciso, el sistema puede desperdiciar ancho de banda de red (a causa de temporizaciones retrasadas) y, además, provocar ráfagas de paquetes. El tráfico en ráfagas produce pérdidas significativas de paquetes en las colas de los encaminadores, incrementando la congestión de la red. Los problemas que devienen a un mecanismo de temporización de granularidad gruesa en la QoS han sido descriptos en los trabajos previos [89, 90].

Como se ha explicado, también es necesario decrementar el alto consumo de CPU producido por el procesamiento de una gran cantidad de paquetes. Reducir

el número de paquetes no solo ahorra ancho de banda de red (debido a la menor sobrecarga del protocolo en encabezados) si no que también reduce la carga de trabajo. De esta forma, se disminuye considerablemente la cantidad de interrupciones y cambios de contexto, producidas cuando un paquete es enviado o recibido, y la carga extra de procesamiento de encabezados. Otra forma de reducir el consumo de CPU es disminuir al mínimo posible el número de copias de paquetes en memoria dado que estas operaciones son muy costosas en tiempo.

Considerando todos estos aspectos, se ha diseñado e implementado el NTS, enfocándose en el componente más crítico de esta implementación, el Planificador de Eventos de Tiempo (TES, *Time Event Scheduler*). El éxito de la implementación del NTS depende de este componente. El TES se encarga de gestionar todos los temporizadores, involucrando las etapas de creación, planificación y ejecución del evento asociado en el tiempo de expiración. El objetivo de desarrollar una solución para un sistema operativo de propósito general, en vez de un sistema operativo de tiempo real (tal como KURT [91]), tiene como fin evitar las dependencias a un sistema operativo específico y mejorar la portabilidad del software. Sin embargo, hoy en día, las facilidades convencionales disponibles para una planificación de eventos de gránulo fino en un sistema operativo de propósito general no son buenas.

Las funciones de espera como *usleep* (espera en el orden de µs), *nanosleep* (espera en el orden de ns), o los temporizadores del Spec 1170 y POSIX.1b *Real-time Extension*, realmente no proporcionan una granularidad del orden de los µs. Cuando un *thread* espera un tiempo, este es desplanificado de la CPU, y luego replanificado al momento de expirar su tiempo de espera. El problema de precisión se origina normalmente en el proceso de replanificación influenciado por al menos tres factores: la prioridad del *thread* que ejecuta la operación de espera, la latencia del núcleo del sistema operativo, y la capacidad del planificador del núcleo para operar con una granularidad fina [92]. Por ejemplo, la actual implementación de *nanosleep* está basada en el mecanismo normal de temporización del núcleo, el cual tiene una resolución de 1/HZ segundos (es decir, 10 ms en Linux/i386 y 1 ms en Linux/Alpha). Por lo tanto, *nanosleep* hace pausas de al menos el tiempo especificado, pero puede requerirse hasta 10 ms más de lo especificado para que el proceso ejecute otra vez.

Es posible implementar funciones de espera precisas usando un bucle de consulta al reloj (evitando en la mayoría de los casos el mecanismo de desplanificación y replanificación del proceso), pero la espera ocupada no es viable por su alto consumo de CPU. Otro esquema que se ha seleccionado para nuestro sistema, es utilizar el reloj de tiempo real que funciona con la batería (*Real Time Clock*, RTC). Todos los PCs tienen un RTC (actualmente es integrado en el centro controlador de entrada/salida —*I/O Controller Hub*, ICH—). Este es el reloj que mantiene la

pista del tiempo cuando el sistema está apagado pero que no se utiliza mientras el mismo está en ejecución. Sin embargo, puede ser utilizado para generar señales a una frecuencia lenta de 2 Hz hasta una frecuencia relativamente alta de 8.192 Hz, en incrementos de potencias de dos. Si se configura el RTC para operar a 8.192 Hz, se tiene una espera de 122 µs, suficiente para obtener una granularidad de temporización fina que garantiza la QoS del proceso de envío de paquetes.

En la sección 3.4.1 se describe la implementación del TES, y en la sección 3.4.2 se presenta la implementación del NTS haciendo uso del TES.

3.4.1. Planificador de Eventos de Tiempo

En esta sección se explica el diseño e implementación del TES que se encuentra inmerso en el NTS. El TES trabaja en un sistema operativo de propósito general, pero su diseño no es fácil porque estos sistemas operativos se diseñan para maximizar el rendimiento del caso promedio en decaimiento del rendimiento del peor caso, lo que perjudica a las aplicaciones de tiempo real débil. Sin embargo, es posible cambiar la política de planificación de procesos, obteniendo un control mayor para este tipo de aplicaciones.

El planificador de procesos es la parte del núcleo que decide cuál proceso o *thread* será ejecutado a continuación por la CPU. Se ha seleccionado utilizar para el TES la política de planificación Primero en Entrar - Primero en Salir (*First In-First Out* —FIFO—, denominada SCHED_FIFO en POSIX.1b). La política SCHED_FIFO permite un control preciso sobre la forma en la cual los procesos ejecutables son seleccionados para su ejecución. Cada proceso tiene una prioridad fija; cuando múltiples procesos tienen el mismo nivel de prioridad, ellos ejecutan en orden FIFO, es decir, será seleccionado el proceso que ha entrado primero. Un proceso planificado según la política SCHED_FIFO ejecuta hasta que se bloquea (por ejemplo por una petición de entrada/salida), es reemplazado por un proceso de más alta prioridad, o este llama a la operación *sched_yield* para liberar voluntariamente el procesador sin bloquearse. Para procesos bajo la política SCHED_FIFO, POSIX.1b define un rango de valor de prioridad de 1 a 99, donde los procesos con número de prioridad mayor son planificados antes que los procesos con número de prioridad menor.

El TES maneja dos tipos de temporizadores, de alta y baja prioridad. Por un lado, los eventos disparados por los temporizadores de baja prioridad son ejecutados por *threads* con una prioridad de 1. Por otro lado, los eventos disparados por temporizadores de alta prioridad son ejecutados por *threads* que tienen una prioridad mayor a 1. La prioridad de los *threads* es preconfigurada para cada temporizador de

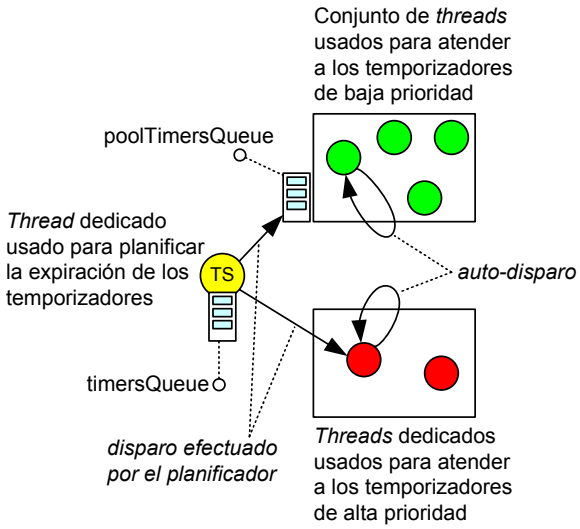


Figura 3.8: Arquitectura del Planificador de Eventos de Tiempo

alta prioridad. Además, todos los temporizadores tienen dos modos de operación, con el modo *yield* activado o desactivado. Si el modo *yield* está activado, después de procesar un evento, el *thread* que ejecuta el evento cede la CPU (utilizando la operación *sched_yield*), produciendo una nueva planificación de procesos.

En la figura 3.8 se muestra la arquitectura de alto nivel del planificador. Un *thread* denominado Planificador de Tiempo (TS, *Time Scheduler*) tiene una prioridad de 3 y se encarga de planificar el disparo de los temporizadores a lo largo del tiempo. Los temporizadores de baja prioridad son disparados por algún *thread* de un conjunto de *threads* disponibles (*thread pool*). El conjunto de *threads* evita gastar tiempo en crear *threads* de forma dinámica. Cada temporizador de alta prioridad tiene un *thread* dedicado y, de esta forma, el costo es menor que con el conjunto de *threads*, porque no es necesario sincronizar ningún conjunto y no se producen retrasos en la ejecución de los eventos cuando no existen *threads* libres en el conjunto. Si el temporizador debe ser disparado inmediatamente, no es necesario que sea planeado por el TS, y por lo tanto es automáticamente disparado (auto-disparo).

Una vez creado el temporizador y especificada su prioridad, el siguiente paso es *replanificar* el temporizador. Esta operación produce una expiración del temporizador luego de transcurridos un cierto número de segundos t. Para llevar a cabo la tarea de replanificación, se calcula el siguiente tiempo de expiración absoluto para el temporizador. El siguiente tiempo de expiración absoluto se obtiene sumando t al último tiempo de expiración absoluto. El algoritmo 1 describe el proceso de

replanificación. En este proceso, cuando el tiempo t es igual a cero ($t = 0$), el temporizador es inmediatamente disparado (línea 17). No obstante, antes de disparar el temporizador, se debe verificar si existe un registro de expiración previo (línea 14), y si no existe, entonces se actualiza el tiempo de expiración al valor del tiempo actual (línea 15). Por el contrario, si el tiempo t no es cero, entonces se debe comprobar si existe o no un registro de expiración previo (línea 2). Si no existe, el tiempo de expiración es actualizado al valor de tiempo actual incrementado en t unidades (línea 10), y luego el temporizador es registrado en el planificador para ser disparado más tarde (línea 11), dando por finalizado el proceso de replanificación. Si existe un registro de expiración previo entonces t es sumado al tiempo de expiración (línea 3) y, luego, si es el momento de disparar el temporizador, se activa una señal (denominada "continúa", presente en la línea 7) indicando que este nuevo evento debe ser inmediatamente procesado a través de un disparo automático (auto-disparo). Si aún no es el momento de disparar el temporizador, entonces este se registra en el TS para ser disparado con posterioridad (línea 5).

Algoritmo 1 Replanificación de un temporizador

Requiere: tiempo t

```
 1: si t >0 entonces
 2:    si expiración está asignado entonces
 3:       expiración ← expiración + t
 4:       si tiempoActual < expiración entonces
 5:          TS.registrarTemporizador(this)
 6:       sino
 7:          continúa ← true
 8:       fin si
 9:    sino
10:       expiración ← tiempoActual + t
11:       TS.registrarTemporizador(this)
12:    fin si
13: sino
14:    si expiración no está asignado entonces
15:       expiración ← tiempoActual
16:    fin si
17:    disparar
18: fin si
```

El TS efectúa la tarea de planificar los eventos a lo largo del tiempo, implementando el algoritmo 2. El TS usa una cola, denominada *timersQueue*, para registrar los temporizadores que deben ser disparados en el futuro. Si *timersQueue*

está vacía, el TS espera el arribo de un nuevo temporizador a la cola *timersQueue* (líneas 1–3 y 15–17). A continuación, se verifica si es el momento de disparar el temporizador del frente de la cola (líneas 5–6). Si no es el momento de disparar el temporizador, el TS espera durante 122 µs usando el RTC (línea 7) y después, verifica nuevamente el tiempo de expiración del temporizador. Cuando sea el momento de dispararlo, el temporizador se elimina de la cola *timersQueue* (línea 9) y, dependiendo de su prioridad, o es puesto en la cola de temporizadores del conjunto de *threads*, denominada *poolTimersQueue*, si el temporizador es de baja prioridad (línea 11), o si es de alta prioridad, el *thread* dedicado correspondiente es desbloqueado (línea 13) para procesar el evento.

Algoritmo 2 Planificador de Tiempo

```
 1: si timersQueue está vacía entonces
 2:     esperar a un nuevo temporizador para planificar
 3: fin si
 4: bucle
 5:     temporizador ← timersQueue.tope
 6:     si temporizador.tiempoExpiración > tiempoActual entonces
 7:         esperar durante 122 ms
 8:     sino
 9:         timersQueue.quitar
10:         si temporizador es de baja prioridad entonces
11:             poolTimersQueue.poner(temporizador)
12:         sino
13:             desbloquear el thread dedicado de este temporizador
14:         fin si
15:         si timersQueue está vacía entonces
16:             esperar a un nuevo temporizador para planificar
17:         fin si
18:     fin si
19: fin bucle
```

Los *threads* dedicados, usados para atender los temporizadores de alta prioridad, tienen un ciclo en el cual se realizan las siguientes tareas. Inicialmente, se verifica el estado de la señal "continúa" que determina si el siguiente evento debe ser procesado de manera inmediata o no. Si la señal está activada, se procesa el siguiente evento (auto-disparo) y, si la señal está desactivada, se bloquea el *thread* para luego ser desbloqueado por el planificador cuando sea el momento de procesar el evento. Finalmente, una vez procesado un evento, si el modo *yield* está activado, es necesario ejecutar la operación *sched_yield* para ceder la CPU y, a continuación,

se ejecuta nuevamente todo el ciclo.

El conjunto de *threads* espera hasta que esté disponible un temporizador, en la cola *poolTimersQueue*, para ser disparado y, a continuación, se elimina el temporizador del tope de la cola y se procesa el evento asociado. Una vez procesado un evento, si la señal "continúa", que determina si el siguiente evento debe ser procesado inmediatamente o no, está activada, entonces se procesa el siguiente evento (auto-disparo) y así sucesivamente. Cuando finaliza el procesamiento de cada evento, si el modo *yield* está activado, es necesario ejecutar la operación *sched_yield* para ceder la CPU. Finalmente, cuando el evento (o eventos en el caso de haber ocurrido auto disparos) fue completamente procesado, se busca en la cola *poolTimersQueue* un nuevo temporizador para disparar.

3.4.2. NTS y su integración con TES

El NTS divide el proceso de entrega de la media en dos tareas principales: la entrega de paquetes desde cada emisor ERAP a la cola *readyToSendQueue*, y la entrega de paquetes desde esta cola a los clientes. La primera parte de la entrega es llevada a cabo por el *RttTimer* y el *IpgTimer*. La segunda parte de la entrega es realizada por una tarea especializada en la regulación del tráfico a la tasa de transmisión de la red. Un temporizador se utiliza para implementar la tarea de regulación de tasa, y otros dos temporizadores, por conexión, son necesarios para el *RttTimer* e *IpgTimer*. En la sección 3.4, se han introducido los requerimientos de precisión de los temporizadores del proceso de entrega de paquetes. Para cumplir con estos requerimientos, ambas partes del proceso de entrega están implementadas usando el gestor de temporizaciones provisto por el TES.

Es necesario determinar la prioridad de los temporizadores, y la prioridad de los *threads* si este es un temporizador de alta prioridad. Ambas partes de la entrega son críticas, pero la regulación de la tasa es la tarea más crítica de ellas. A continuación, se discute esta afirmación. En un proceso de entrega de paquetes se presentan dos problemas, la anomalía de las ráfagas de paquetes, y el desperdicio del ancho de banda de red. En el primer caso, cuando los temporizadores se retrasan, es posible que todos ellos sean disparados juntos en un cierto momento, produciendo una ráfaga de paquetes. Una ráfaga de paquetes podría incrementar la congestión de la red saturando las colas de los encaminadores. En el segundo caso, cuando la aplicación necesita todo el ancho de banda, si un paquete no es enviado a la red, debido al retraso del temporizador, el enlace no se utiliza y por lo tanto se desperdicia el ancho de banda de la red. La primera parte de la entrega solo se relaciona con el primer problema, y la segunda parte se relaciona con ambos problemas. Se ha preferido producir pequeñas ráfagas de paquetes, pero evitar el malgasto del ancho de

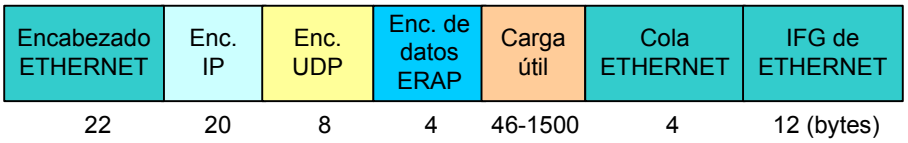


Figura 3.9: Encabezados, colas e intervalo entre marcos, en una red Ethernet de 100 Mb/s

banda de la red. Por lo tanto, se ha priorizado la regulación de la tasa, usando para esta tarea un temporizador de alta prioridad con un *thread* de prioridad 4. El *IpgTimer* y el *RttTimer* usan temporizadores de baja prioridad (es decir, con un *thread* de prioridad 1). Debido al uso de temporizadores de baja prioridad, el *IpgTimer* y el *RttTimer* usan muy pocos recursos porque ellos no tienen *threads* dedicados, ellos usan los *threads* del conjunto solo cuando es necesario. Como estos temporizadores no tienen recursos reservados, una sesión inactiva no consume recursos.

Como el temporizador de la tarea de regulación de tasa tiene la prioridad más alta del sistema (aún más grande que la del propio TS), siempre que un paquete este disponible en la cola *readyToSendQueue* y la red pueda recibirlo, el paquete es enviado a la red. Para conocer cuándo la red puede recibir un paquete, es necesario, por cada paquete enviado, calcular la distancia al siguiente paquete. Se define Gap_i como la distancia (tiempo) que debe existir entre el paquete i e $i+1$, la cual es calculada como:

$$Gap_i = \frac{8 \times PhysicalPacketSize_i}{bandwidth \times 10^6} \tag{3.8}$$

donde *bandwidth* es el ancho de banda saliente de la conexión de red del servidor, expresada en Mb/s. $PhysicalPacketSize_i$ es el tamaño en bytes necesarios para transmitir el paquete i por el enlace físico, considerando el intervalo entre marcos (IFG, *Inter-Frame Gap*), pero sin considerar la codificación de datos (correspondiente a la capa física del modelo de referencia OSI) porque esta no es tenida en cuenta tampoco por la métrica del *bandwidth*. El estándar IEEE 802.3 define el IFG como un periodo mínimo sin uso de la red, que debe existir entre cada par de marcos Ethernet transmitidos. El IFG es el tiempo que se ocupa en transmitir 96 bits de datos en el medio, el cual es de 960 ns para una red Ethernet de 100 Mb/s. La figura 3.9 muestra los bytes utilizados para encabezados (Ethernet, IP, UDP y ERAP), colas (Ethernet), e IFG (Ethernet) en una red Ethernet de 100 Mb/s. Por ejemplo, una transmisión de un paquete de 1.000 bytes de longitud de datos de un vídeo requiere 1.070 bytes, entre los cuales 54 bytes corresponden a encabezados, 1.000 bytes de carga útil, 4 bytes de colas, y 12 bytes (96 tiempos de bit) de IFG.

Capítulo 4

Diseño del sistema VoD-NFR para entorno real y simulado

En los dos capítulos anteriores se ha descripto la arquitectura y funcionalidad del sistema VoD-NFR, y la implementación del componente mas crítico de este sistema. En este capítulo se presenta el diseño del sistema VoD-NFR desarrollado para funcionar en un entorno real y también de simulación. Se explica el diseño de cada uno de los módulos que componen al sistema, mediante descripciones estáticas y dinámicas, y se expone la especificación del protocolo de comunicación, utilizado entre los clientes y servidores.

Para evaluar el sistema VoD-NFR con mayor profundidad, se ha decidido disponer del mismo tanto para un entorno real como de simulación. Con objeto de simular exactamente la misma aplicación que funciona para entorno real, se ha decidido la realización de una única arquitectura de software cuya implementación tenga un único código, compilable para ambos entornos. El diseño de una única arquitectura es bastante más complejo que el de dos arquitecturas independientes, sin embargo, la reducción del tiempo requerido para mantener una sola aplicación, y evitar problemas de versiones con dos códigos que implementan los mismos algoritmos, justifica ampliamente el esfuerzo invertido en la etapa de diseño.

Se ha utilizado el simulador Network Simulator - NS2 [86, 87], versión 2.31 publicada el 10 de marzo de 2007. Este simulador es considerado prácticamente un estándar de facto en la evaluación de protocolos de transporte, y tiene la ventaja de que incluye los protocolos más conocidos, entre ellos: TCP, UDP y RAP. NS-2 es un simulador basado en eventos, escrito en el lenguaje de programación C++ y usa OTcl como una interfaz de comandos y configuración.

Básicamente se pueden encontrar dos grandes diferencias de implementación en una aplicación para un entorno real o de simulación de red. Estas diferencias se encuentran en el gestor de eventos de tiempo y en las comunicaciones. El gestor de eventos de NS-2 no trabaja en tiempo real, cosa que sí debe hacer el gestor de eventos de la aplicación real. En cuanto a las comunicaciones, en el caso real los paquetes son enviados por una red física y, en el caso simulado, los paquetes son entregados al simulador de la red. Teniendo en cuenta estas dos diferencias entre ambos entornos, se ha diseñado un *middleware* de planificación de eventos y un *middleware* de red, presentados en la sección 4.1 y 4.2, respectivamente.

La descripción del diseño del sistema VoD-NFR también incluye: la especi-

ficación del protocolo de comunicación del SCC (sección 4.3), el diseño de los servidores (sección 4.4) y los clientes (sección 4.5), y una interfaz de VoD-NFR en NS2 (sección 4.6).

4.1. *Middleware* de planificación de eventos

El *middleware* de planificación de eventos provee una interfaz de gestión de eventos independiente a su implementación, ya sea en un entorno real o de simulación.

El planificador de eventos de tiempo presenta dos entidades, el planificador en sí mismo y los temporizadores que generan eventos. En la figura 4.1 se muestra el diagrama de clases (notación UML, *Unified Modeling Language*), donde el planificador de eventos es representado por la clase *Scheduler* y los temporizadores por la clase *TimerHandler*. La clase *Scheduler* mantiene una lista de temporizadores ordenados según el tiempo de expiración. La clase abstracta *ImprovedTimerHandler* proporciona características adicionales a los temporizadores como la identificación de estado activado/desactivado y pendiente/no pendiente. Un temporizador está en el estado "pendiente" cuando se encuentra incluido en la lista del planificador de eventos y, caso contrario, está en estado "no pendiente". Un temporizador puede activarse o desactivarse. El planificador de eventos sólo ejecutará el evento asociado al temporizador cuando el mismo se encuentre en el estado "activado". Para ejecutar un evento, *Scheduler* invoca el método *expireITH* de la clase *ImprovedTimerHandler*, el cual es implementado por una subclase concreta de *ImprovedTimerHandler* que define la ejecución del evento.

Hay dos versiones de las clases *TimerHandler* y *Scheduler*, una versión que hemos desarrollado para entorno real y la otra es provista por NS2. La implementación de estas clases para entorno real toma los conceptos de implementación y algoritmos descriptos en la sección 3.4.1. En tiempo de compilación estas clases se enlazan a la versión correspondiente, según el entorno para el cual se está efectuando la compilación (real o simulado).

4.2. *Middleware* de red

El *middleware* de red presenta una interfaz de red común e independiente del medio que hace efectiva la comunicación (la red o el simulador). La interfaz soporta los protocolos de comunicación UDP y TCP, indispensables en VoD-NFR, donde

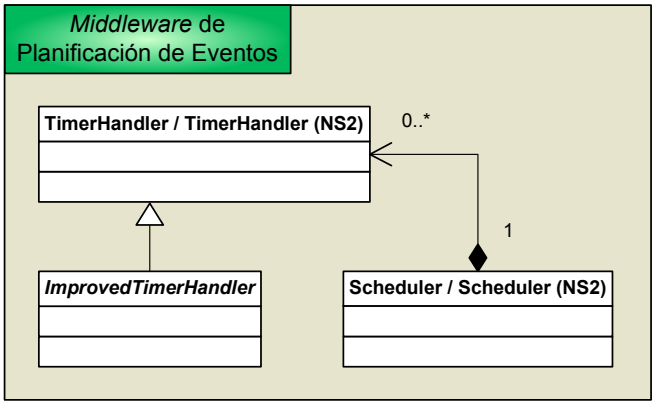


Figura 4.1: Diagrama de clases del *middleware* de planificación de eventos

el primero es necesario para transmitir paquetes ERAP y el segundo es utilizado para la comunicación del SCC.

Una interfaz de red puede implementar la recepción de datos por lo menos de acuerdo a los dos modelos siguientes:

1. Estilo *sockets*: la recepción de mensajes se efectúa mediante la llamada a una función (*recv*) que se bloquea en espera de un mensaje.
2. Estilo *NS2*: cuando un nodo recibe un mensaje, se invoca a una función que se encarga de procesar el mensaje.

Note la diferencia que presentan estos dos modelos. El primero es un modelo secuencial, mientras que el segundo, para mantener la pista de la recepción, es necesario mantener el estado de recepción actual.

A continuación se presenta un ejemplo representado con el primer modelo:

```
recibir(a)
enviar(b)
recibir(c)
enviar(d)
```

Esta representación es equivalente a la siguiente, expresada mediante el segundo modelo:

```
Valor inicial de la variable estado: 1
función recibir(x) {
   si (estado = 1) {
      a ← x
      enviar(b)
      estado ← 2
   }
   sino, si (estado = 2) {
      c ← x
      enviar(d)
   }
}
```

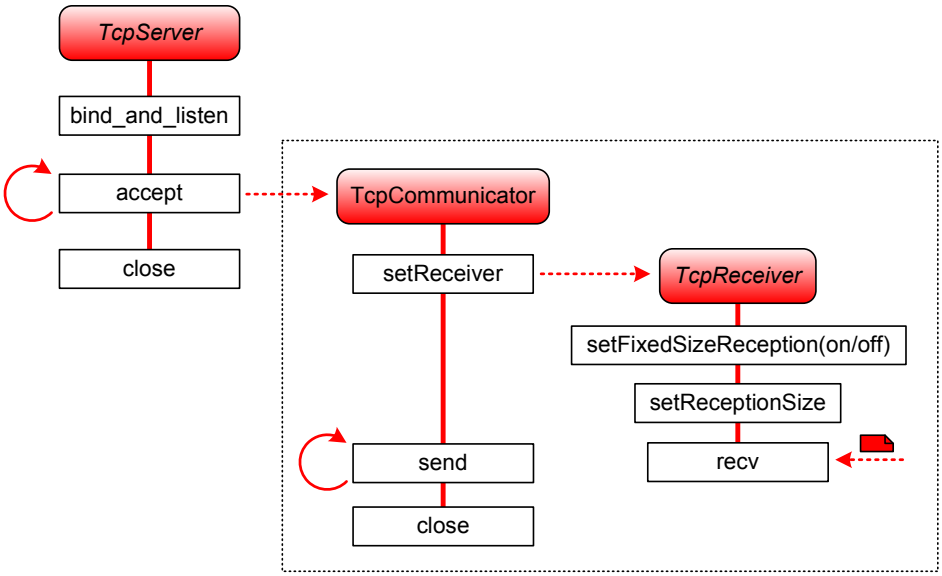
Ambos modelos son igualmente potentes y, a pesar de que el primer modelo sea más simple de utilizar, se ha optado por el segundo modelo. La razón de la elección tiene que ver con que el primer modelo requiere de operaciones de recepción (*recv*) bloqueantes, que implican una compleja implementación debido a la naturaleza secuencial de los simuladores de eventos discretos.

En las figuras 4.2 (a) y (b) se presentan diagramas de secuencia de ejecución de operaciones y entidades participantes de la interfaz de comunicaciones TCP, del lado del servidor y del cliente, respectivamente.

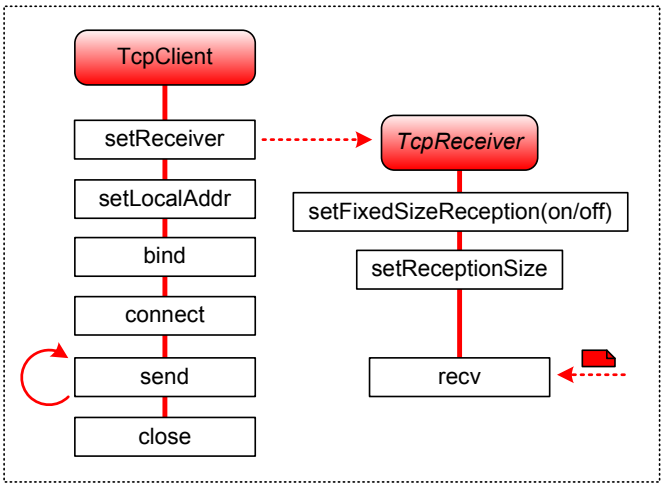
Una subclase de *TcpServer* actúa como servidor TCP. Para su configuración se debe invocar solamente el método *bind_and_listen*, el cual especifica una dirección local de escucha del servidor y la cantidad de peticiones que pueden encolarse antes de ser atendidas. Cuando se establece una nueva conexión se invoca automáticamente el método *accept*, redefinido por la subclase. Al finalizar la recepción de peticiones se debe invocar el método *close* para liberar los recursos ocupados.

La invocación del método *accept* tiene como fin entregar la entidad de administración de la nueva conexión establecida, el *TcpCommunicator*. La clase *TcpCommunicator* incluye el método *setReceiver*, utilizado para definir la clase que se encargará de recibir los datos de la conexión, la cual debe heredar de *TcpReceiver*. La recepción de datos puede ser de tamaño fijo o no, indicado mediante el método *setFixedSizeReception*, y si es de tamaño fijo debe especificarse su tamaño mediante el método *setReceptionSize*. La subclase de *TcpCommunicator* dispone del método *send* para transmitir mensajes, mientras que la subclase de *TcpReceiver* es la encargada de procesar los mensajes transmitidos por el otro extremo de la comunicación. Una vez finalizada la comunicación, esta debe ser liberada mediante el comando *close*.

Una subclase de *TcpClient* representa al cliente TCP. Al igual que un *TcpCommunicator*, un *TcpClient* debe especificar (método *setReceiver*) y configurar



(a) Lado del servidor



(b) Lado del cliente

Figura 4.2: Interfaz de comunicación TCP

(métodos *setFixedSizeReception* y *setReceptionSize*) la entidad que se encargará de procesar los datos recibidos por la conexión TCP. A continuación, es necesario especificar la dirección IP local mediante el método *setLocalAddr*, mientras que en el caso de utilizar NS2, el puerto asignado será reemplazado por otro de asignación automática. La invocación del método *bind* es opcional y asocia la conexión a la dirección local especificada. Una vez efectuada estas configuraciones, se lleva a cabo la conexión mediante la llamada del método *connect*, suministrándole la dirección IP y puerto destino al que se desea conectar. Luego de establecida la conexión, se utiliza el método *send* para transmitir mensajes, mientras los mensajes recibidos son procesados por la subclase de *TcpReceiver*. La finalización de la comunicación se indica mediante el método *close*. que se encarga de liberar los recursos asignados.

En la figura 4.3 se presenta el diagrama de la secuencia de ejecución de las operaciones definidas por la interfaz de comunicaciones UDP. Una subclase de *UdpCommunicator* representa a un extremo de comunicación UDP, el cual puede ser un emisor, receptor, o ambos. Para configurarlo, se debe invocar el método *setLocalAddr*, por medio del cual se especifica la dirección IP local y, si es un receptor, el puerto por el cual se desean recibir los paquetes UDP; asimismo, para el caso del receptor, es necesario invocar el método *prepareForReception* para iniciar el mecanismo de recepción de paquetes. Los paquetes destinados a este extremo de comunicación, serán entregados a través de la invocación de su método *recv*. Cuando se requiere enviar un mensaje, primeramente es necesario especificar su destino por medio del método *setDestination*, para luego ordenar su envío a través del método *send*. Si se desea que los paquete IP enviados tengan un puerto de origen en particular (y no uno cualquiera que simplemente esté disponible), es necesario especificarlo mediante el método *setSourcePortToSend*. Una vez finalizada la recepción de paquetes de una comunicación, se debe invocar el método *closeReception* para liberar los recursos asignados.

En la figura 4.4 se muestra el diagrama estático de todas las clases que componen el *middleware* de red. La características generales de las comunicaciones se especifican en la clase *NetworkTransportInformation*, donde se describe principalmente el ancho de banda de conexión a la red, longitud de paquetes a utilizar en transmisiones UDP, y protocolo de capa de enlace a utilizar (en nuestro caso hemos trabajado con Ethernet). La clase *IpPort* simplemente encapsula una dirección IP y un puerto determinado. La clase *Ns2Info* tiene como única finalidad almacenar una referencia (nombre de la variable en OTcl) a la instancia del simulador NS2 para poder acceder a sus métodos.

Como las comunicaciones TCP y UDP requieren disponer de dos implementaciones, una para una red real y otra para una red simulada por NS2, su diseño se

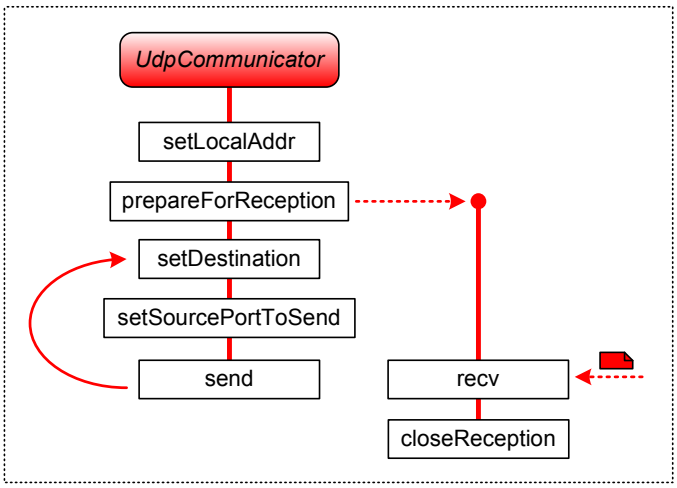


Figura 4.3: Interfaz de comunicación UDP

ha basado en el patrón *Bridge* [93], el cual permite desacoplar abstracciones (en este caso, las comunicaciones TCP y UDP) de su implementación (*sockets* o NS2), de manera que ambas puedan ser modificadas y seleccionadas independientemente. De esta forma se definen las clases *TcpServer* (abstracta), *TcpCommunicator* y *TcpClient*, las cuales cumplen el rol de la clase *Abstraction* del patrón, la clase *TcpImplementor* cumple el rol de la clase *Implementor*, y las clases *UnixTcpSockets* y *Ns2Tcp* cumplen el rol de las clases *ConcreteImplementor*. En una comunicación TCP, en un extremo se tiene un *TcpCommunicator* y del otro un *TcpClient*, los cuales comparten muchas tareas en común, excepto parte de su configuración; entonces, se ha definido la superclase *TcpEnd* que representa genéricamente a un extremo de una comunicación TCP. El mismo patrón aplicado a la implementación de la comunicación TCP, también se ha aplicado a la de UDP, donde la clase *UdpCommunicator* cumple el rol de la clase *Abstraction*, *UdpImplementor* cumple el rol de *Implementor*, y *UnixUdpSockets* y *Ns2Udp* cumplen el rol de las clases *ConcreteImplementor*. En tiempo de compilación, se selecciona la implementación a utilizar, ya sea la de entorno de red real o simulada; en el primer caso, la clase *TcpImplementor* será del tipo *UnixTcpSockets* y la clase *UdpImplementor* del tipo *UnixUdpSockets*, mientras que en el segundo caso, la clase *TcpImplementor* será del tipo *Ns2Tcp* y la clase *UdpImplementor* del tipo *Ns2Udp*.

La implementación de las clases *UnixTcpSockets* y *UnixUdpSockets* son relativamente simples utilizando *sockets* de Unix. Cuando se recibe un mensaje por un *socket*, se invoca a la función de recepción (*recv*) de la clase *TcpReceiver* que se encarga de procesar el mensaje. La clase *Ns2Udp* hereda de una clase propia de NS2

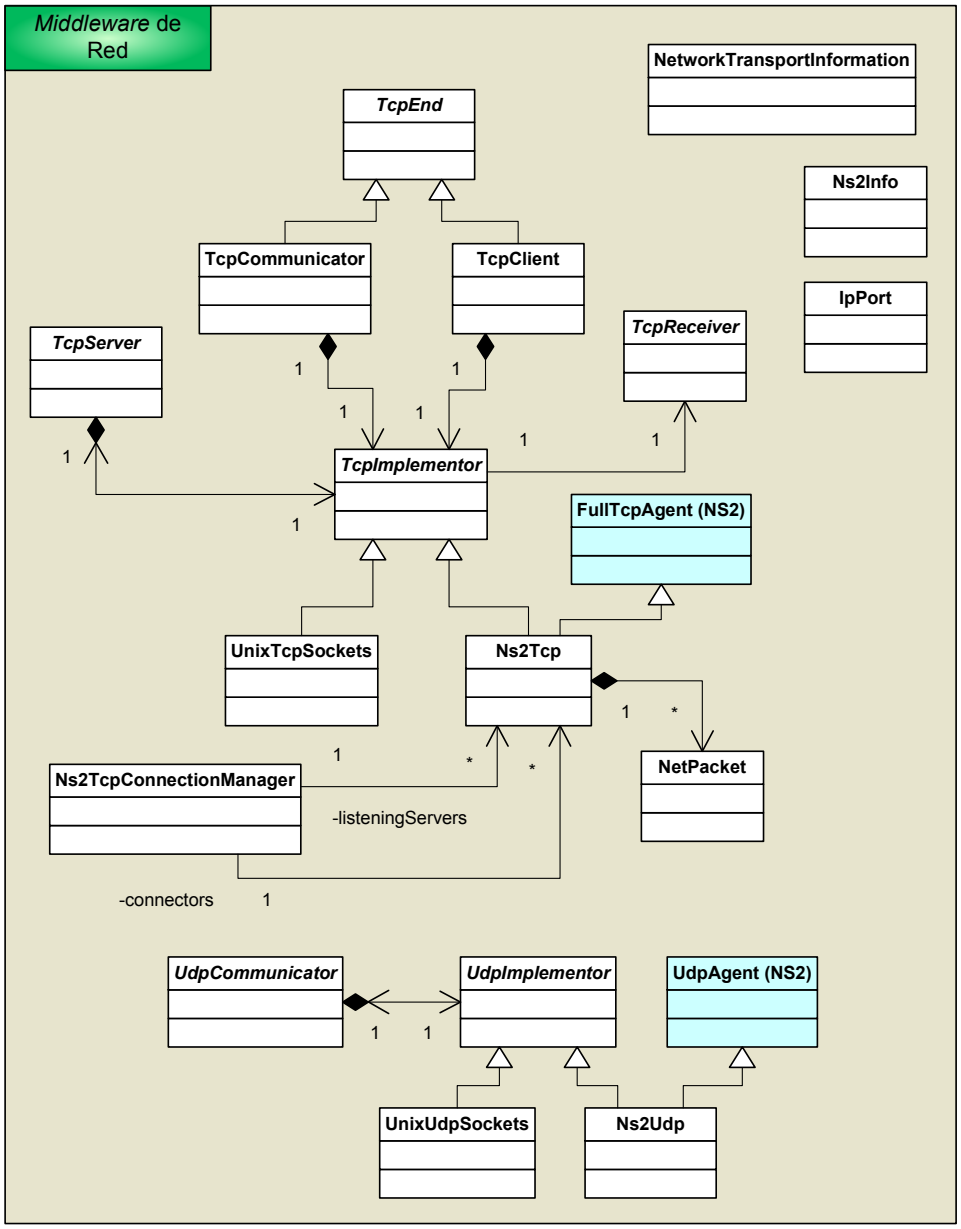


Figura 4.4: Diagrama de clases del *middleware* de red

denominada *UdpAgent*, que representa a un extremo de una conexión UDP en el simulador; *Ns2Udp* se implementa de una forma relativamente directa mediante el uso de los métodos de su superclase. La implementación de la clase *Ns2Tcp* hereda de la clase *FullTcpAgent* de NS2, que representa un extremo de una conexión TCP en el simulador. Uno de los problemas de *FullTcpAgent* es que no transfiere datos de usuario reales, aunque el receptor conoce el tamaño de los datos cuando recibe un paquete TCP; el otro problema es que NS2 no permite generar conexiones TCP dinámicamente. La solución que se ha diseñado para estos dos inconvenientes no modifica en absoluto el código TCP del simulador, y se basa en adicionar nuevas clases que dan soporte a la funcionalidad faltante.

La clase *Ns2TcpConnectionManager* da soporte a conexiones TCP dinámicas, manteniendo un listado de servidores que están esperando por nuevas conexiones. El envío de datos reales se implementa de la siguiente manera: el emisor que desea transmitir un nuevo mensaje solicita a la clase *Ns2TcpConnectionManager* el objeto *Ns2Tcp* que representa al destinatario; a continuación, el emisor crea un objeto *NetPacket* donde almacena los datos reales que se van a transmitir y lo agrega a la bandeja de entrada del destinatario. Cuando el destinatario recibe la indicación del simulador de que ha llegado un paquete, este busca en su bandeja de entrada y retira el paquete con los datos reales.

Este *middleware* de red ha sido diseñado para ser utilizado únicamente desde el ambiente de C++, y no puede ser accedido directamente desde la interfaz de simulación de NS2.

4.3. Protocolo de comunicación del SCC

En esta sección se describe el protocolo de comunicación implementado en el canal de control del *stream*, el SCC. Las comunicaciones del SCC pueden clasificarse según los objetivos que persiguen, los cuales son: solicitud normal de un vídeo, mecanismo de protección del cliente, notificación de migración al cliente, y reserva de recursos y migración de un cliente a un servidor alternativo. Cada una de estas comunicaciones se efectúan siguiendo un protocolo que consiste de una secuencia de intercambio de mensajes. En las secciones 4.3.1, 4.3.2, y 4.3.3, 4.3.4 se describe cada una de las comunicaciones citadas mediante secuencias de mensajes simples. Estos mensajes tienen formatos específicos que se exponen en la sección 4.3.5.

4.3.1. Solicitud normal de un vídeo

Cuando un cliente desea visualizar un vídeo genera una solicitud a un servidor. Si el cliente ha visualizado anteriormente una parte del vídeo y aún mantiene almacenada la media, puede continuar la reproducción aprovechando los datos que ya había recibido. La petición entonces puede tener la característica de ser de "primera visualización" o no.

En la figura 4.5 se presenta la secuencia de mensajes, intercambiados entre el cliente y el servidor, que involucra el proceso completo de solicitud del vídeo hasta la entrega del comando interactivo indicando al servidor que comience a transmitir la media. Luego de recibir el mensaje de "solicitud normal de un vídeo", el servidor puede aceptar o rechazar la solicitud ya sea porque algunos de los módulo SecC o QoSA ha denegado su acceso. Entonces, el servidor envía o un mensaje de "aceptación del requerimiento" o "rechazo del requerimiento". En este último caso, luego de enviar el mensaje, el servidor cierra la conexión del SCC; si el requerimiento es aceptado, el camino a seguir depende de si el tipo de petición es de "primera visualización" o no. Si lo es, entonces el servidor realiza la "transmisión de metadatos" y, en caso contrario, el cliente efectúa la "transmisión de VBMs" para informar del estado de sus *buffers* (no requiere recibir los metadatos porque ya cuenta con ellos). Finalmente, el cliente envía al servidor una "solicitud de comando interactivo" teniendo en cuenta que la reproducción puede comenzar desde el comienzo del vídeo o desde el punto en que el cliente había interrumpido una posible previa visualización.

4.3.2. Mecanismo de protección del cliente (*heartbeat*)

El mecanismo de protección del cliente se lleva a cabo mediante un esquema *heartbeat*. Este esquema se ha implementado mediante el envío de un mensaje (*ping*) desde el cliente al servidor, y la consiguiente respuesta del servidor al cliente (*pong*), tal cual puede apreciarse en la figura 4.6. Si al momento de enviar el siguiente *ping* no se ha recibido el *pong*, entonces el mecanismo de protección determina que debe efectuarse una migración. En caso de que el cliente disponga de una reserva de recursos en un servidor alternativo, migra a él; caso contrario, si no dispone de ningún servidor alternativo, entonces deberá efectuar una solicitud normal de un vídeo a otro servidor indicando que no es "primera visualización".

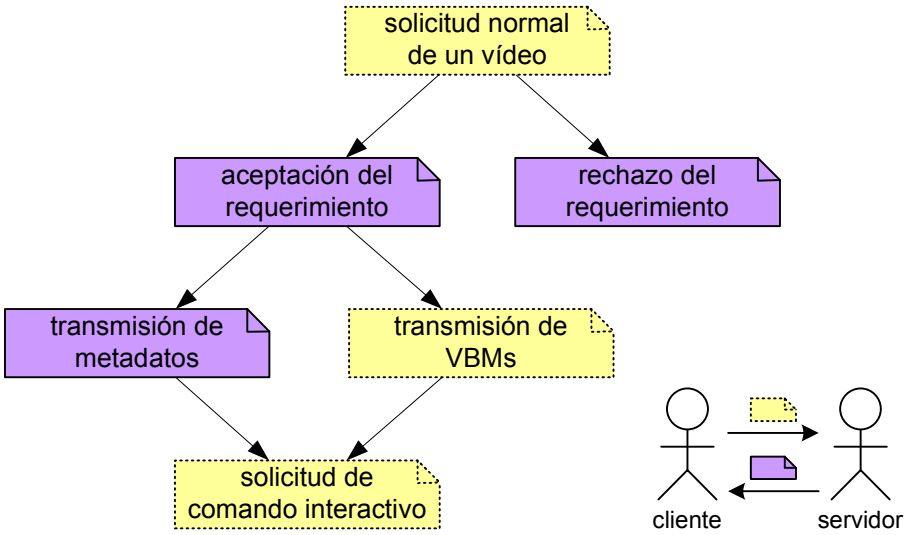


Figura 4.5: Solicitud normal de un vídeo efectuada por un cliente a un servidor

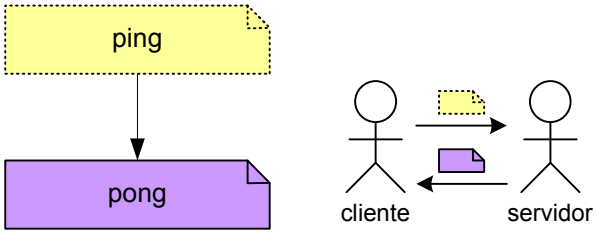


Figura 4.6: Mecanismo de protección del cliente (*heartbeat*)

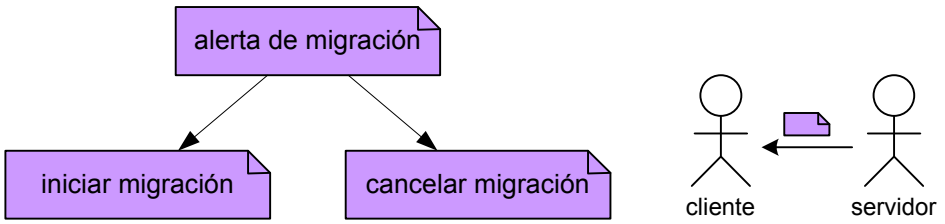


Figura 4.7: Notificación de migración al cliente

4.3.3. Notificación de migración al cliente

Cuando el servidor detecta problemas de comunicación con el cliente, se sigue el protocolo de comunicación presentado en la figura 4.7. El servidor comienza enviando un mensaje de "alerta de migración" de acuerdo con el mecanismo de preaviso de detección de fallos de red. Si, luego de transcurrido el tiempo de *changüí*, el servidor encuentra que la comunicación sigue siendo deficiente, entonces le envía al cliente un mensaje de "iniciar migración" y luego cierra la conexión del SCC. Por el contrario, si la comunicación ha mejorado y cumple con los requisitos para garantizar la QoS, entonces se envía al cliente un mensaje de "cancelar migración".

4.3.4. Reserva de recursos y migración de un cliente a un servidor alternativo

Cuando el cliente recibe un mensaje de "alerta de migración", este debe buscar un servidor alternativo con el fin de reservar recursos que más tarde puede utilizar en caso de requerir de sus servicios. Esta tarea se efectúa siguiendo la secuencia de mensajes definida en la figura 4.8. El cliente envía un mensaje de "solicitud de reserva de recursos" a un servidor, el cual analiza la aceptación de la solicitud mediante sus módulos SecC y QoSA, para luego enviar un mensaje de "aceptación del requerimiento" o "rechazo del requerimiento". En este último caso, luego de enviar el mensaje, el servidor cierra la conexión del SCC, y el cliente deberá buscar otro servidor alternativo. Si el requerimiento es aceptado, el servidor espera hasta que el cliente se contacte nuevamente indicando un "inicio de migración" o "cancelación de migración". Si el cliente cae (falla), ninguno de estos mensaje llegará al servidor y, para evitar que este tenga reservas de recursos inútiles, la misma es mantenida durante un tiempo prudencial, y luego es descartada si no ha recibido contacto desde el cliente.

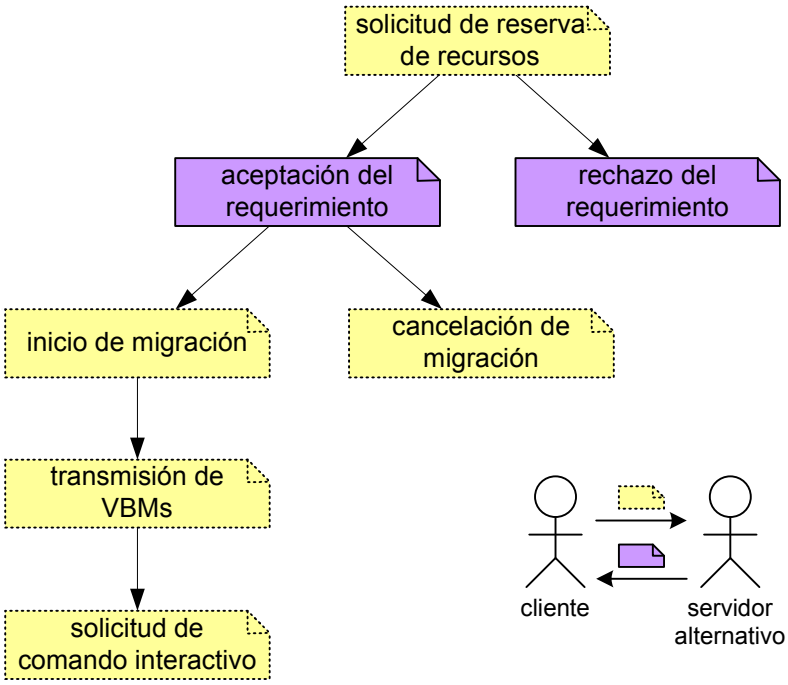


Figura 4.8: Reserva de recursos y migración de un cliente a un servidor alternativo

Si el cliente ha enviado un mensaje de "cancelación de migración", a continuación cierra la conexión del SCC; por el contrario, si había enviado un mensaje de "inicio de migración", el cliente efectúa la "transmisión de VBMs" para informar del estado de sus *buffers*. Finalmente, el cliente envía al servidor una "solicitud de comando interactivo" para indicarle el punto actual de reproducción.

4.3.5. Formatos de mensajes del protocolo

En las figuras 4.9 y 4.10 se muestran los formatos de los mensajes simples que los clientes transmiten a los servidores, mientras que en las figuras 4.11 y 4.12 se muestran los formatos de los mensajes que transmiten los servidores a los clientes. Un formato se describe indicando el nombre de los campos, el tamaño en bits, su representación y, en el caso que corresponda, el valor que debe tener el campo.

En los mensajes que tienen un tamaño total de 8 bits, se agrega un campo denominado *Padding* (también de 8 bits) que se utiliza para evitar un problema que surge en el entorno de simulación de NS2. Este simulador, en una conexión TCP, cuando debe enviar un solo byte, lo retrasa hasta tener un byte más para transmitir. Por lo tanto, cuando sólo se desea transmitir un solo byte y ninguno más a continuación, se agrega este campo a los mensajes para forzar el envío inmediato. El valor de *Padding* es irrelevante ya que el campo solo cumple la función de relleno del mensaje.

A continuación se describe el significado de los campos de cada formato de mensaje.

inicio de migración El único campo útil, *Code*, debe tener el valor fijo de 0 para identificar el inicio de una migración. Este mensaje utiliza el campo de relleno *Padding*.

cancelación de migración La cancelación de la migración se identifica colocando un valor de 1 en el campo *Code*. Este mensaje lleva el campo de relleno *Padding*.

solicitud normal de un vídeo La solicitud normal de un vídeo se identifica poniendo el campo *Type of Service* en 0. *Video ID* es el identificador del vídeo que desea visualizarse y se solicita a través de este mensaje. *Expiration Time* es el tiempo límite que el cliente tiene para visualizar el vídeo. *Client IP Address* es la dirección IP (IPv4) del cliente y *Client Stream Port* es el puerto por el cual el cliente espera recibir la media. Finalmente, *First View* debe tener un valor diferente de 0 cuando es "primera visualización" y, caso contrario, su valor debe ser 0.

inicio de migración

Nombre del campo	Tamaño en bits	Representación	Valor
Code	8	sin signo	0
Padding	8	irrelevante	irrelevante

cancelación de migración

Nombre del campo	Tamaño en bits	Representación	Valor
Code	8	sin signo	1
Padding	8	irrelevante	irrelevante

solicitud normal de un vídeo

Nombre del campo	Tamaño en bits	Representación	Valor
Type of Service	8	sin signo	0
Video ID	32	sin signo	
Expiration Time	32	con signo	
Client IP Address	32	sin signo	
Client Stream Port	16	sin signo	
First View	8	sin signo	falso = 0, verdadero ≠ 0

solicitud de reserva de recursos

Nombre del campo	Tamaño en bits	Representación	Valor
Type of Service	8	sin signo	1
Video ID	32	sin signo	
Expiration Time	32	con signo	
Client IP Address	32	sin signo	
Client Stream Port	16	sin signo	

Figura 4.9: Formato de mensajes enviados por los clientes (1)

transmisión de VBMs

Los VBM se transmiten ordenados de menor a mayor velocidad de reproducción

Nombre del campo	Tamaño en bits	Representación	Valor
VBM Size	32	sin signo	
VBM Data	variable	cadena de bits	
⋮			
VBM Size	32	sin signo	
VBM Data	variable	cadena de bits	

VBM Data =

Nombre del campo	Tamaño en bits	Representación
Interval Start	32	sin signo
Interval End	32	sin signo
⋮		
Interval Start	32	sin signo
Interval End	32	sin signo

ping

Nombre del campo	Tamaño en bits	Representación	Valor
Type of Service	8	sin signo	1
Padding	8	irrelevante	irrelevante

solicitud de comando interactivo

Nombre del campo	Tamaño en bits	Representación	Valor
Type of Service	8	sin signo	0
Code	8	con signo	parar = 0, rep. (vel.) ≠ 0
○ Time Position	64	IEEE 754	

existe solo si (Code ≠ 0)

Figura 4.10: Formato de mensajes enviados por los clientes (2)

aceptación del requerimiento

Nombre del campo	Tamaño en bits	Representación	Valor
Code	8	sin signo	0
Padding	8	irrelevante	irrelevante

rechazo del requerimiento

Nombre del campo	Tamaño en bits	Representación	Valor
Code	8	sin signo	1
Padding	8	irrelevante	irrelevante

transmisión de metadatos

Nombre del campo	Tamaño en bits	Representación	Valor
File Count	8	sin signo	
File Size	32	sin signo	
File Data	variable	cadena de bits	
⋮			
File Size	32	sin signo	
File Data	variable	cadena de bits	

Figura 4.11: Formato de mensajes enviados por los servidores (1)

alerta de migración

Nombre del campo	Tamaño en bits	Representación	Valor
Type of Service	8	sin signo	0
Code	8	sin signo	0

iniciar migración

Nombre del campo	Tamaño en bits	Representación	Valor
Type of Service	8	sin signo	0
Code	8	sin signo	1

cancelar migración

Nombre del campo	Tamaño en bits	Representación	Valor
Type of Service	8	sin signo	0
Code	8	sin signo	0

pong

Nombre del campo	Tamaño en bits	Representación	Valor
Type of Service	8	sin signo	1
Padding	8	irrelevante	irrelevante

Figura 4.12: Formato de mensajes enviados por los servidores (2)

solicitud de reserva de recursos Para identificar este servicio, el campo *Type of Service* debe tener un valor de 1. *Video ID* es el identificador del vídeo que el cliente está visualizando y para el cual quiere reservar recursos. *Expiration Time* es el tiempo límite que el cliente tiene para visualizar el vídeo. *Client IP Address* y *Client Stream Port* es la dirección IP (IPv4) del cliente y el puerto por el cual espera recibir la media.

transmisión de VBMs El cliente transmite los mapa de los *buffers* de los vídeos (VBM) al servidor. El servidor ya conoce la cantidad de VBM que tiene que recibir, que es igual a la cantidad de velocidades de reproducción soportadas por el vídeo en cuestión. Los VBM se transmiten uno a continuación de otro, ordenados de menor a mayor velocidad de reproducción. Para transmitir uno de ellos, primero se transmite su tamaño expresado en el campo *VBM Size* (es una medida en octetos y puede tener un valor de 0), y luego sus datos en el campo *VBM Data*. Este último campo en realidad es una secuencia de intervalos, donde cada intervalo indica la existencia de información en el *buffer*, y se especifican mediante un campo de inicio del intervalo, *Interval Start*, y uno de finalización, *Interval End*.

ping El *ping* es un mensaje que el cliente transmite al servidor para saber si la comunicación aún persiste. Tiene un campo denominado *Type of Service* que debe tener un valor de 1, y también tiene el campo de relleno *Padding*.

solicitud de comando interactivo El mensaje de solicitud de un comando interactivo tiene un campo llamado *Type of Service* que debe tener un valor de 0. También tiene un campo denominado *Code* que identifica al comando interactivo solicitado. El comando "parar" (*stop*) lleva un valor *Code* de 0, mientras que para el resto de los comandos (reanudar, saltar, avanzar o retroceder rápido) *Code* debe contener la codificación de la velocidad de reproducción. La codificación es un número con signo en complemento a 2, utilizando signo positivo o negativo según si la reproducción es en avance o retroceso, respectivamente. El identificador codificado es la velocidad de reproducción multiplicada por 10; por ejemplo, un retroceso rápido a velocidad $1{,}5x$ es codificado como -15, y la velocidad de reproducción normal $1x$ es codificada como 10. Cuando *Code* es diferente de 0, el mensaje también lleva otro campo, denominado *Time Position*, que se utiliza para especificar el tiempo desde donde se debe reproducir la versión del vídeo en cuestión. La combinación de *Code* y *Time Position*, proporcionan una gran flexibilidad permitiendo, por ejemplo, soportar saltos mientras se está en modo de reproducción rápida.

aceptación del requerimiento La aceptación de un requerimiento se identifica colocando un valor de 0 en el campo *Code*. Este mensaje lleva el campo de relleno *Padding*.

rechazo del requerimiento El rechazo de un requerimiento se identifica colocando un valor de 1 en el campo *Code*. Este mensaje lleva el campo de relleno *Padding*.

transmisión de metadatos El servidor transmite los metadatos de los vídeos al cliente. El campo *File Count* se utiliza para especificar la cantidad de archivos de metadatos que se van a transmitir. Ellos se transmiten en secuencia, uno después del otro. Para transmitir un archivo, primero se transmite su tamaño expresado en el campo *File Size* (es una medida en octetos), y luego se transmiten sus datos (campo *File Data*). A continuación se muestran las primeras cinco líneas de un archivo de metadatos donde se observa el formato de los mismos:

```
Playback speed: 1.0
X(seconds) Y(bytes)
0 0
6 496400
10 758222
```

En la primera línea se indica la velocidad de reproducción, en este ejemplo es una velocidad a reproducción normal (1,0x), pudiendo ser un valor negativo para especificar una reproducción en sentido hacia atrás. Desde la tercer línea hacia adelante se especifican los puntos fundamentales de la curva. El primer valor es expresado en segundos y el segundo en octetos. En este ejemplo, el segundo 0 comienza en el octeto 0, el segundo 6 comienza en el 496.400, y así sucesivamente.

alerta de migración Un servidor efectúa una alerta de migración al cliente mediante un mensaje que tiene un campo *Type of Service* y *Code*, ambos con valor 0.

iniciar migración Un servidor da la orden al cliente de iniciar la migración mediante un mensaje que tiene un campo *Type of Service* con valor 0 y un campo *Code* con valor 1.

cancelar migración Un servidor da la orden al cliente de cancelar la migración mediante un mensaje que tiene un campo *Type of Service* y *Code*, ambos con valor 0.

pong El *pong* es un mensaje que el servidor transmite al cliente en respuesta a un *ping*. Tiene un campo denominado *Type of Service* que debe tener un valor de 1, y también tiene el campo de relleno *Padding*.

4.4. Diseño de los servidores

En el presente apartado se describe el diseño de cada uno de los módulos que conforman la arquitectura del servidor: Gestor de Datos de Vídeo (sección 4.4.1), Control de Seguridad (sección 4.4.2), Garantía de QoS (sección 4.4.3), Módulo de Transmisión de *Streams* (sección 4.4.4), Planificador de Canales Lógicos (sección 4.4.5), Control de Sesión (sección 4.4.6), y Control del Servidor de Vídeo (sección 4.4.7).

4.4.1. Gestor de Datos de Vídeo

En la figura 4.13 se presenta el diagrama de clases del Gestor de Datos de Vídeo (VDM). La clase *VideoRepository* representa al repositorio de vídeos, y es el punto de acceso a todos los vídeos del catálogo local del servidor. Un *VideoItem* representa a un vídeo que se localiza mediante un identificador preasignado a cada uno de ellos. Cada vídeo está compuesto de varias versiones codificadas a velocidades diferentes, representadas mediante la clase *VideoAccess*, quien permite el acceso a los datos del archivo del vídeo que tiene asociado. Note que un *VideoItem* no tiene asociado un archivo de vídeo específico como sí lo tiene un *VideoAccess*; por ejemplo, suponiendo que el vídeo con identificador 145 tiene tres versiones del vídeo, codificadas a las velocidades $-2x$, $1x$ y $2x$, se tendrá un *VideoItem* (con identificador 145) y tres *VideoAccess*, uno para cada versión del vídeo codificada a diferente velocidad.

Cada archivo de un vídeo tiene asignado un archivo de metadatos, representado por la clase *MetadataAccess*, que es accedida por medio de la asociación que la clase *VideoAccess* mantiene con ella.

4.4.2. Control de Seguridad

El componente Control de Seguridad (SecC) tiene tres clases que se reparten las tareas del módulo. En la figura 4.14 puede observarse el diagrama de clases del SecC, que incluye la clase *TicketValidator*, el *SessionExpirationControl* y el *Decrypter*, mientras que el resto de las clases son externas al módulo SecC. La clase

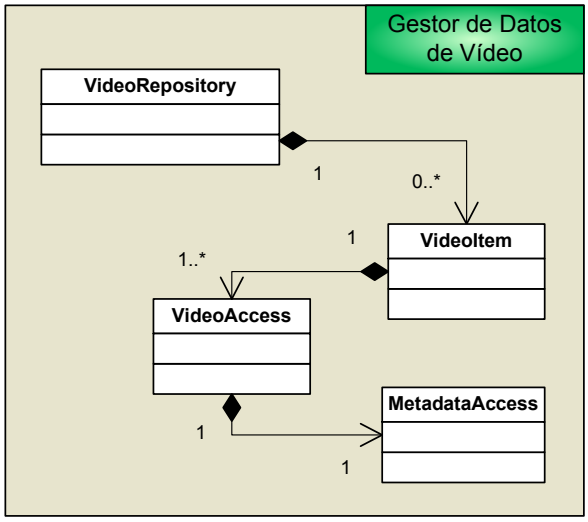


Figura 4.13: Diagrama de clases del gestor de datos de vídeo

TicketValidator tiene como responsabilidad verificar la validez de los boletos, para lo cual necesita tener acceso al catálogo de vídeos (*VideoRepository*). Como los datos del boleto están encriptados, es necesario desencriptarlos; esta tarea ha sido asignada a la clase *Decrypter*, quien se ocupa de desencriptar datos mediante un algoritmo de criptografía asimétrica. La clase *SessionExpirationControl* se encarga de, cada cierto tiempo predefinido, validar cada uno de los boletos de las sesiones que tiene el servidor, verificando que el tiempo de expiración de las sesiones no se haya superado. Las sesiones del servidor son accedidas a través de la clase *SessionsManager*, perteneciente al módulo de Control de Sesión (SC).

4.4.3. Garantía de QoS

El diagrama de clases del módulo de Garantía de QoS (QoSA) se presenta en la figura 4.15. La clase *ResourceAndAdmissionControl* tiene la responsabilidad de aceptar o rechazar nuevas peticiones de vídeos, balanceando la carga del servidor y asegurando que el servidor dispone de los recursos suficientes para prestar un servicio de calidad a los clientes. Para verificar la existencia de recursos, esta clase hace uso de otras cuatro clases, cada una encargada de un recurso diferente. Del recurso de red se encarga la clase *NetworkBroker*, del recurso de memoria secundaria se ocupa la clase *DiskBroker*, y del recurso de memoria y CPU se ocupan las clases *MemoryBroker* y *CpuBroker*, respectivamente.

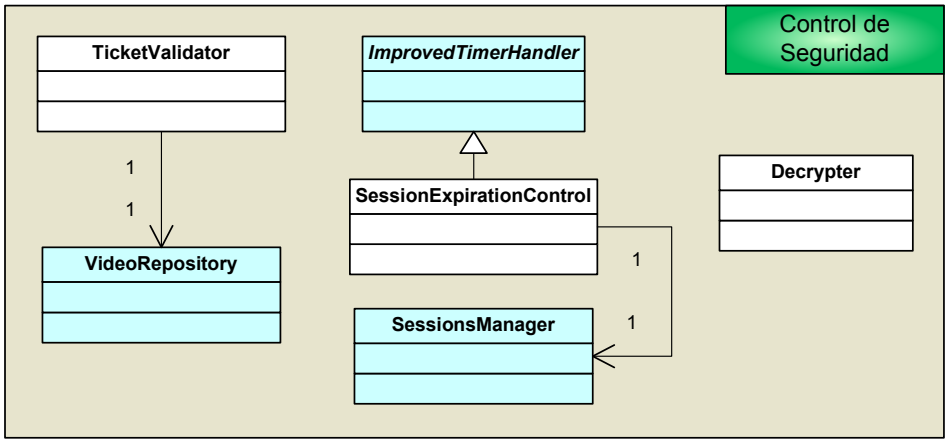


Figura 4.14: Diagrama de clases del control de seguridad

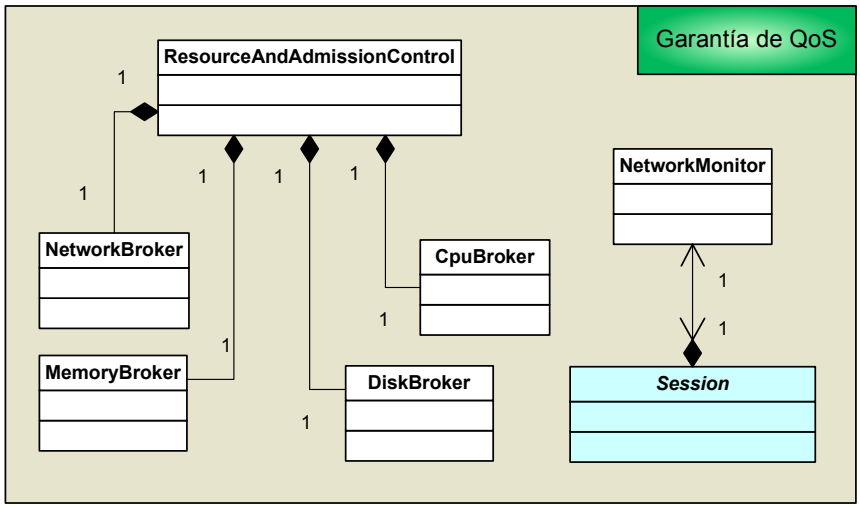


Figura 4.15: Diagrama de clases del módulo que garantiza la QoS

La clase *NetworkMonitor* implementa el mecanismo de detección de fallos de red efectuado por el servidor. La asociación que tiene con la clase *Session* tiene que ver con que esta clase hace de intermediaria del módulo LCS para proveer la información de *deadline*. Además, tanto la alerta de migración como la orden de inicio o cancelación de la migración debe informarse a la clase *Session* para que esta la transmita al cliente en cuestión.

4.4.4. Módulo de Transmisión de *Streams*

En la figura 4.16 se presenta el diagrama de clases del Módulo de Transmisión de *Streams* (STM), en el cual puede apreciarse el componente Planificador del Tráfico de Red (NTS) y el protocolo ERAP; las clases coloreadas no pertenecen al STM.

Se establece una jerarquía de clases de los diferentes paquetes que utiliza el ERAP, que derivan de la clase *NetPacket* del *middleware* de red. Las clases *ErapDataPacket* y *ErapAckPacket* implementan el paquete de datos y el de reconocimiento (ACK) del protocolo ERAP. A su vez, un *ErapDataPacket* puede tener dos estados, el de no enviado y el de enviado, representados por las clases *NotSentPacket* y *SentPacket*, respectivamente.

La clase *ErapManager* implementa el gestor ERAP definido por el protocolo ERAP. Esta clase es una subclase de *UdpCommunicator*, perteneciente al *middleware* de red, mediante la cual transmite los paquetes de la cola *readyToSendQueue*, definida por la asociación que la clase *ErapManager* tiene con la clase *NotSentPacket*. *ErapManager* tiene una asociación a la clase *ErapSender*, denominada *registeredESs*, que permite acceder a todos los emisores ERAP para generar información del estado global de las comunicaciones.

La clase *ErapSender* representa a un emisor ERAP, la cual tiene una referencia a la clase *ErapManager* para entregarle los paquetes que desea transmitir. *ErapSender* mantiene las colas *departurePacketsQueue* y *retransmissionQueue* del protocolo, que en el diagrama se reflejan como asociaciones a la clase *NotSentPacket*, y la lista *transmissionHistory* que se representa como una asociación a la clase *SentPacket*. Cada *ErapSender* provee de muestras de ancho de banda de la comunicación a la clase *NetworkMonitor* correspondiente al módulo de Garantía de QoS.

La clases *IpgTimer_* y *RttTimer_* asociadas a la clase *ErapSender* representan, respectivamente, a los temporizadores *IpgTimer* y *RttTimer* de ERAP. Para darles la capacidad de temporización, el *IpgTimer_* y *RttTimer_* se declaran como subclases de *ImprovedTimerHandler*, e incluyen una referencia al *ErapSender*, que se

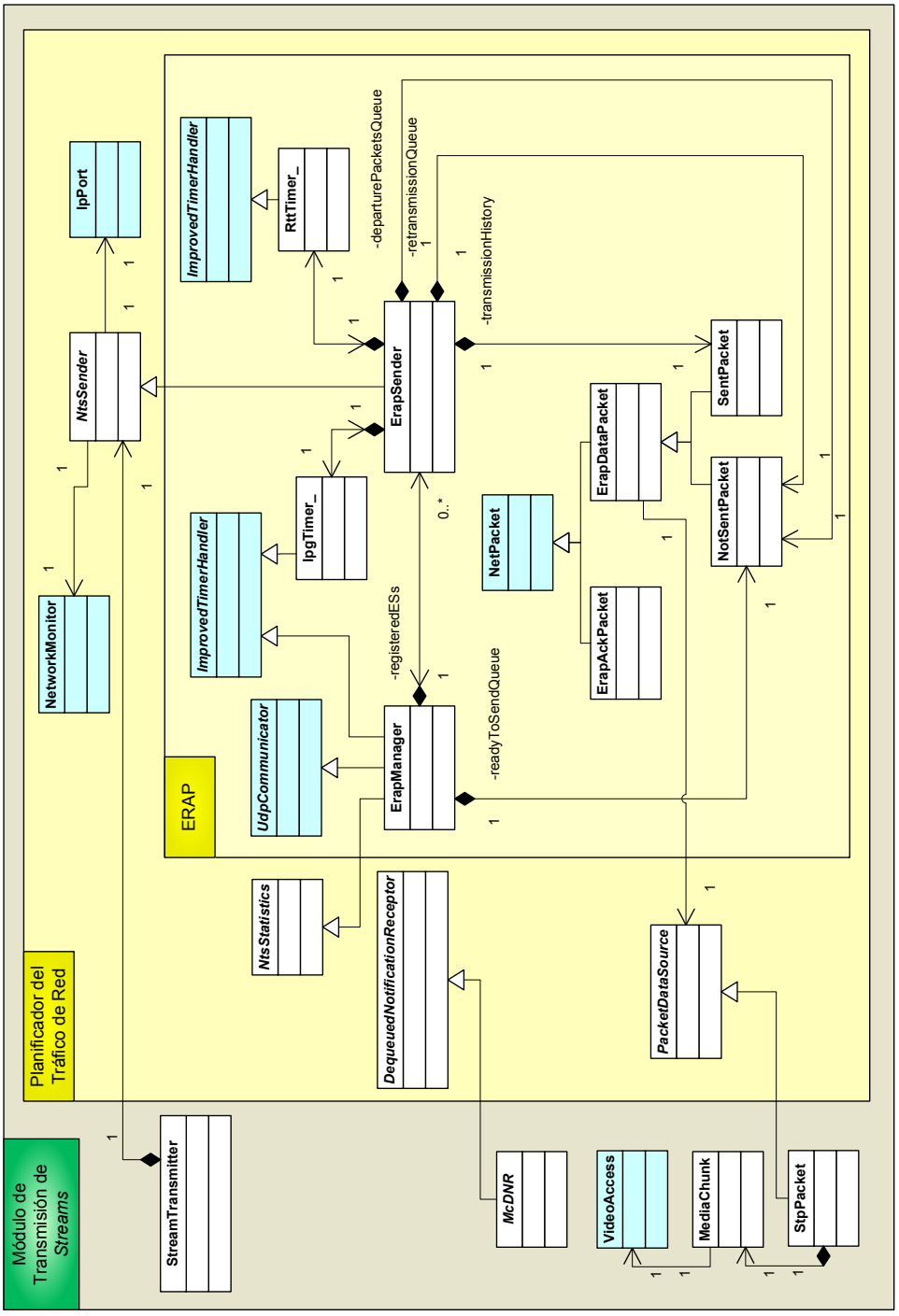


Figura 4.16: Diagrama de clases del módulo de transmisión de *streams*

utiliza para invocar el método correspondiente cuando el tiempo del temporizador ha expirado.

NtsStatistics y *NtsSender* implementan una interfaz clara y sencilla orientada a ofrecer el servicio de transporte de datos del NTS; el único acceso al componente NTS se realiza a través de estas dos clases. *NtsStatistics* ofrece información del estado de la red que involucra al NTS en su conjunto, y *NtsSender* ofrece los servicios particulares a cada conexión establecida entre el servidor y los clientes.

La clase *ErapDataPacket* tiene una asociación con la clase abstracta *PacketDataSource*. Una subclase de *PacketDataSource* proporciona los datos que un *ErapDataPacket* transporta. Dado que VoD-NFR utiliza el protocolo STP por encima de ERAP, se define la clase *StpPacket*, que representa un paquete STP y que hereda de *PacketDataSource*. Un *StpPacket* se compone de un *MediaChunk*, el cual representa una sección de un determinado vídeo, especificado por la clase *VideoAccess*.

Cuando se ordena vaciar las colas de transmisiones de un determinado emisor ERAP, el *NtsSender* entrega los *PacketDataSource*, correspondientes a los paquetes desencolados, a la clase *DequeuedNotificatorReceptor*. La subclase de esta última, denominada *McDNR*, representa a una entidad que es notificada de las secciones de vídeo (*MediaChunk*) que no serán transmitidas y que pertenecían a paquetes que se esperaba transmitir.

Finalmente, un *StreamTransmitter* es la clase que, haciendo uso del NTS, proporciona el servicio de transmisión de *streams* por medio del envío de secciones de vídeo, representadas por objetos de la clase *MediaChunks*. Todos los servicios del STM son accedidos únicamente por las clases *NtsStatistics* y *StreamTransmitter*.

4.4.5. Planificador de Canales Lógicos

En la figura 4.17 se muestra el diagrama de clases del Planificador de Canales Lógicos (LCS). La clase *LCS* implementa el módulo principal de este componente, el cual ejecuta el algoritmo de planificación del LCS. Esta clase hereda de la clase *ImprovedTimerHandler* porque necesita de un temporizador que indique el comienzo de cada *slot* para realizar la planificación correspondiente. Asimismo, para efectuar esta planificación, solicita información al *NtsStatistic* sobre el ancho de banda disponible para el *slot*.

La clase *LCS* mantiene una asociación a múltiples objetos de clase *Stream*, donde cada objeto representa a un canal lógico. Un *Stream* tiene una asociación con la clase *PlaybackPoint* que representa el punto de reproducción actual del cliente, y también tiene asociado un *StreamTransmitter* para llevar a cabo el envío de los

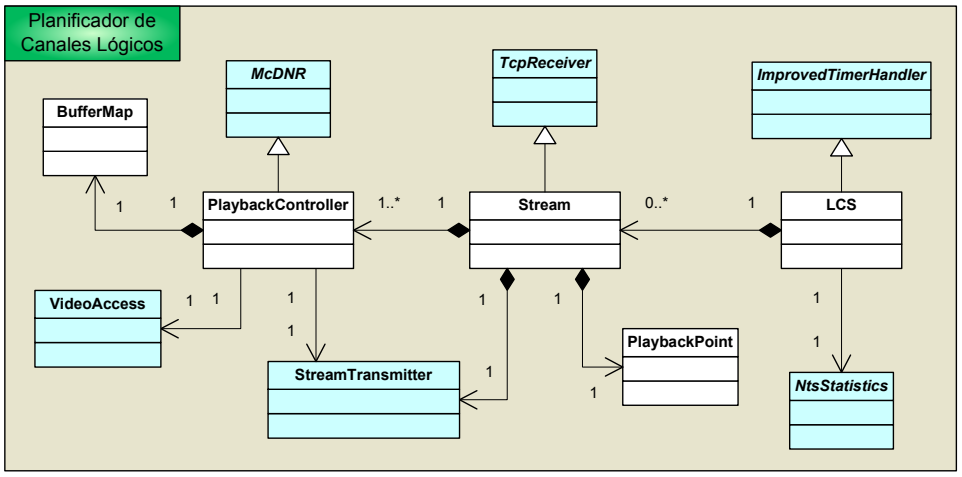


Figura 4.17: Diagrama de clases del planificador de canales lógicos

datos de los vídeos según indicación del algoritmo planificador del *LCS*. Con el fin de recibir los VBM, que transmite un cliente en el marco de una migración, esta clase se declara como subclase de *TcpReceiver*.

Cada objeto de clase *Stream* tiene asociados múltiples objetos de la clase *PlaybackController*, uno por cada versión diferente del vídeo en cuestión (es decir, de velocidades distintas de reproducción). El *PlaybackController* es responsable de encapsular el estado de reproducción de una versión del vídeo, que accede mediante su asociación con la clase *VideoAccess*. Para encapsular el estado de una reproducción, se necesita mantener el mapa de *buffer* del vídeo (VBM), por lo tanto, el *PlaybackController* tiene una asociación con un objeto de la clase *BufferMap* que se ocupa de esto. Cada vez que se transmiten nuevas secciones del vídeo al cliente, estas se registran en el VBM; sin embargo, cuando se solicita un vaciado de colas al STM, es necesario desmarcar las secciones que no han sido recibidas por el cliente. Para poder recibir la notificación de estas secciones, por parte del STM, la clase *PlaybackController* se ha declarado como una subclase de *McDNR*.

4.4.6. Control de Sesión

El diseño del módulo de Control de Sesión se presenta en la figura 4.18 mediante un diagrama de clases. La clase *SessionsManager* se encarga de recibir nuevas conexiones de clientes, capacidad que se obtiene al heredar de la clase *TcpServer*, y generar objetos de la clase *TcpCommunicator*, que representa al extremo de la

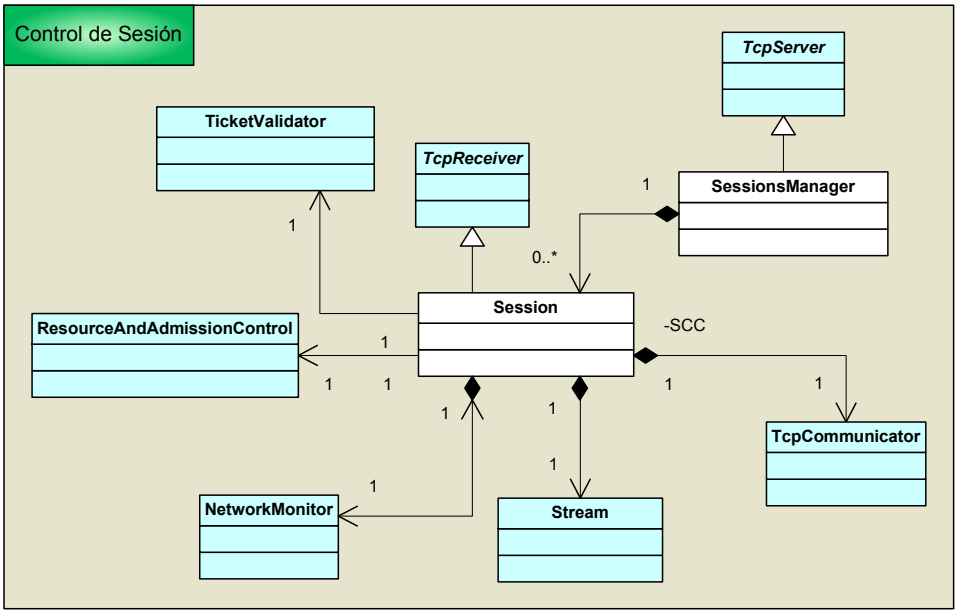


Figura 4.18: Diagrama de clases del control de sesión

conexión del SCC utilizado para comunicarse con ellos. Cuando hay una nueva conexión, se crea un objeto de la clase *Session*, responsable de mantener el estado del nivel de sesión de un cliente, y se le entrega su correspondiente *TcpCommunicator*. Una vez creado uno de estos objetos, se intercambian mensajes con el cliente a través del SCC; la recepción de mensajes se soporta mediante la herencia con la clase *TcpReceiver*. Una vez recibido un boleto, se valida mediante la clase *TicketValidator*, y se solicita al *ResourceAndAdmissionControl* la aceptación de esta conexión. Si ambos controles han sido exitosos, se crea y asocia un objeto *Stream* perteneciente al módulo LCS, y un *NetworkMonitor* perteneciente al módulo QoSA.

La clase *SessionsManager* tiene una asociación con la clase *Session* porque necesita tener acceso a todas las sesiones, por ejemplo, cuando el módulo de Control de Seguridad necesita validar periódicamente los tiempos de expiración de las mismas.

Una sesión puede encontrarse en diferentes estados, cuyo diagrama, en notación UML, se muestra en la figura 4.19. Muchas de las transiciones de estados dependen del contenido de los mensajes que transmite el cliente, los cuales fueron especificados en la sección 4.3.5. A continuación se describe cada uno de los estados y sus transiciones:

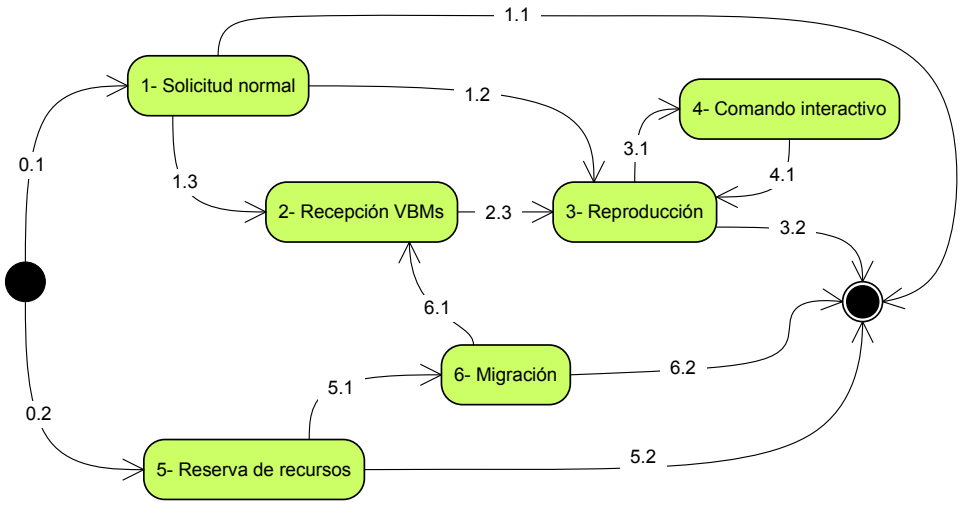


Figura 4.19: Diagrama de estados de una sesión, del lado del servidor

0- Inicio El servidor espera hasta recibir un mensaje de "solicitud normal de un vídeo" o de "solicitud de reserva de recursos", que se distingue comprobando el valor del campo *Type of Service* (primer byte del mensaje); si es 0 se pasa al estado *1-Solicitud normal* (transición 0.1), y si es 1 se pasa al estado *5-Reserva de recursos* (transición 0.2).

1- Solicitud normal El cliente ha entregado un mensaje de "solicitud normal de un vídeo". En este estado se debe validar el boleto y consultar al *ResourceAndAdmissionControl* sobre si es posible aceptar esta conexión. En caso negativo, se pasa al estado final (transición 1.1) y, en caso positivo, se pasa al estado *2-Recepción VBMs* (transición 1.3) o *3-Reproducción* (transición 1.2) dependiendo de si el campo *FirstView* del mensaje es igual o distinto de cero, respectivamente.

2- Recepción VBMs En este estado se indica al objeto *Stream* que debe recibir los datos de los VBMs transmitidos por el cliente. A continuación se pasa al estado *3-Reproducción* (transición 2.3).

3- Reproducción Durante este estado se transmite el contenido del vídeo al cliente. Los mensajes recibidos pueden ser de dos tipos, identificados por el campo *Type of Service*. Si este campo tiene un valor de 1, se envía un mensaje *pong* al cliente y, si es 0, es una solicitud de comando interactivo. En este último caso, si el campo *Code* es 0, el comando solicitado es *parar*, por lo que se pasa al estado final (transición 3.2) y,

si su valor es diferente de 0, se pasa al estado *4-Comando interactivo* (transición 3.1).

4- Comando interactivo En este estado, se recibe el resto del mensaje de solicitud de comando interactivo y se indica al objeto *Stream* que ejecute la operación de interactividad solicitada.

5- Reserva de recursos El cliente ha entregado un mensaje de "solicitud de reserva de recursos". En este estado se debe validar el boleto y consultar al *ResourceAndAdmissionControl* sobre si es posible aceptar esta conexión. En caso negativo, se pasa al estado final (transición 5.2) y, en caso positivo, se pasa al estado *6-Migración* (transición 5.1).

6- Migración Durante este estado, el servidor espera que el cliente transmita un mensaje de inicio o cancelación de migración. Cuando se recibe el mensaje, se determina si es de "inicio de migración" o "cancelación de migración" según si el campo *Code* es 0 o 1, respectivamente. En caso de ser "inicio de migración", se pasa al estado *2-Recepción VBMs* (transición 6.1) y, caso contrario, se pasa al estado final (transición 6.2).

7- Final En este estado se liberan todos los recursos que el servidor a destinado a esta conexión, dando por finalizada la sesión.

4.4.7. Control del Servidor de Vídeo

El diseño del módulo de Control del Servidor de Vídeo se muestra en la figura 4.20 mediante un diagrama de clases. El módulo está conformado solamente por una clase, denominada *VideoServer*, que crea exactamente un objeto de cada una de las clases asociadas: *ErapManager*, *ResourceAndAdmissionControl*, *VideoRepository*, *LCS* y *SessionsManager*. Cada uno de estos objetos creados, que ponen en marcha cada uno de los módulos del servidor, son previamente configurados con los parámetros que rigen el funcionamiento particular de cada componente.

4.5. Diseño de los clientes

En la figura 4.21 se muestra el diagrama de clases de todos los módulos que componen la arquitectura del cliente. Los clientes, al igual que los servidores, se han desarrollado utilizando el *middleware* de red. En las siguientes secciones se discute el diseño de cada uno de los módulos: Receptor del *Stream* (sección 4.5.1),

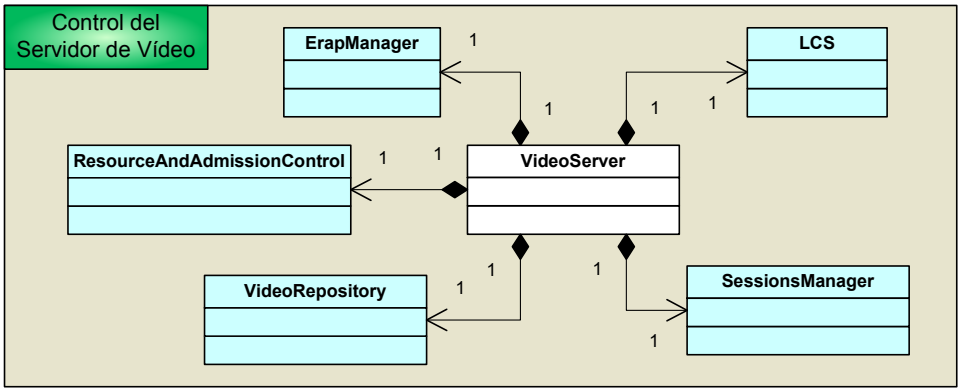


Figura 4.20: Diagrama de clases del control del servidor de vídeo

Control de Sesión (sección 4.5.2), Gestor de Datos del Vídeo (sección 4.5.3), Control de la Interfaz (sección 4.5.4), e Interfaz de Visualización (sección 4.5.5).

4.5.1. Receptor del *Stream*

El módulo Receptor del *Stream* utiliza la jerarquía de clases de paquetes que comprende a la clase *NetPacket*, *ErapAckPacket* y *ErapDataPacket*. La primera clase pertenece al *middleware* de red, mientras que las dos restantes son una copia exacta de las clases definidas en el módulo STM del servidor. La clase *ErapReceiver* representa a un receptor ERAP que, cuando recibe un paquete transmitido por el STM, lo entrega al objeto *VideoControl*.

4.5.2. Control de Sesión

El módulo de Control de Sesión tiene un único objeto de la clase *SessionControl*, responsable de la administración del SCC y de la recuperación de fallos. Al conectarse a un servidor se establece un nuevo SCC, por el cual se transmiten nuevas peticiones de visualización de vídeos, solicitudes de comandos interactivos, y todo lo referente al proceso de las migraciones. Las comunicaciones por el SCC se efectúan mediante el protocolo TCP, razón por la cual *SessionControl* se ha definido como una subclase de *TcpReceiver*, que la habilita para recibir paquetes TCP, y además, tiene una asociación con la clase *TcpClient*, que le permite transmitir paquetes TCP; tanto *TcpReceiver* como *TcpClient* pertenecen al *middleware* de red.

Cuando se recibe una alerta de migración, es necesario contactar a un servidor alternativo que suplante al servidor actual, por lo que el *SessionControl* establece

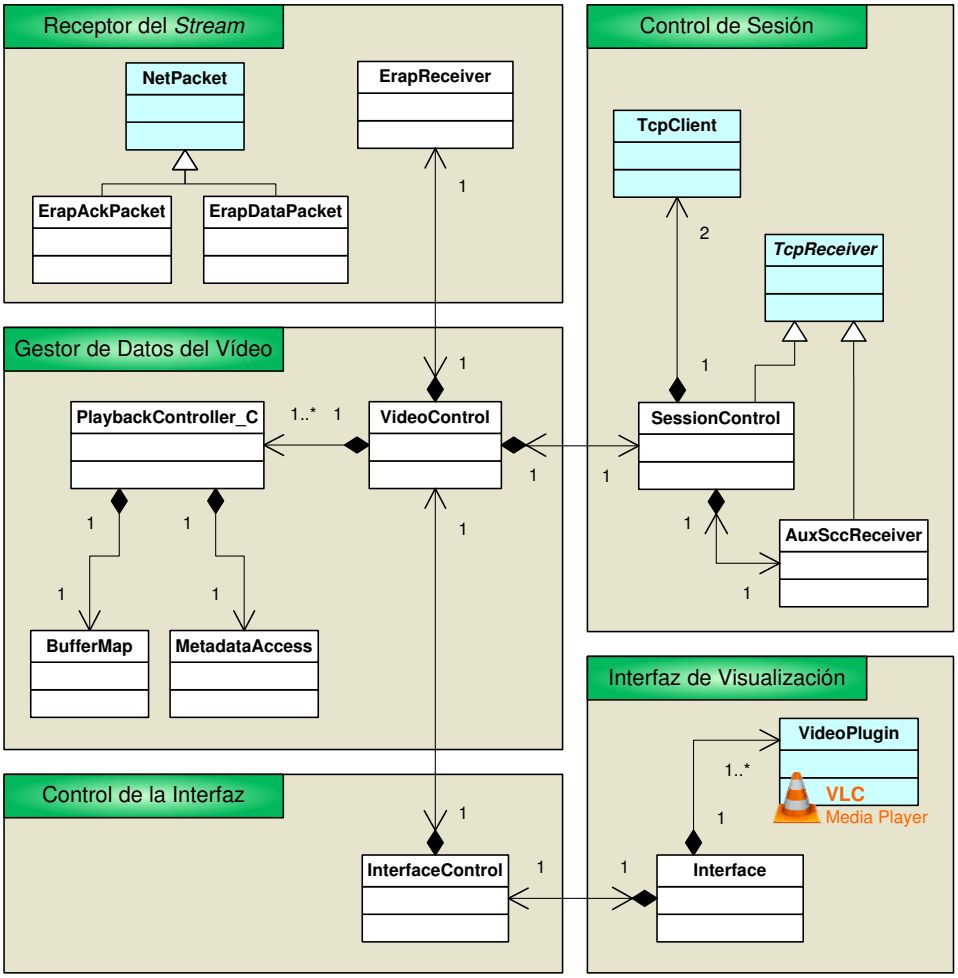


Figura 4.21: Diagrama de clases del cliente

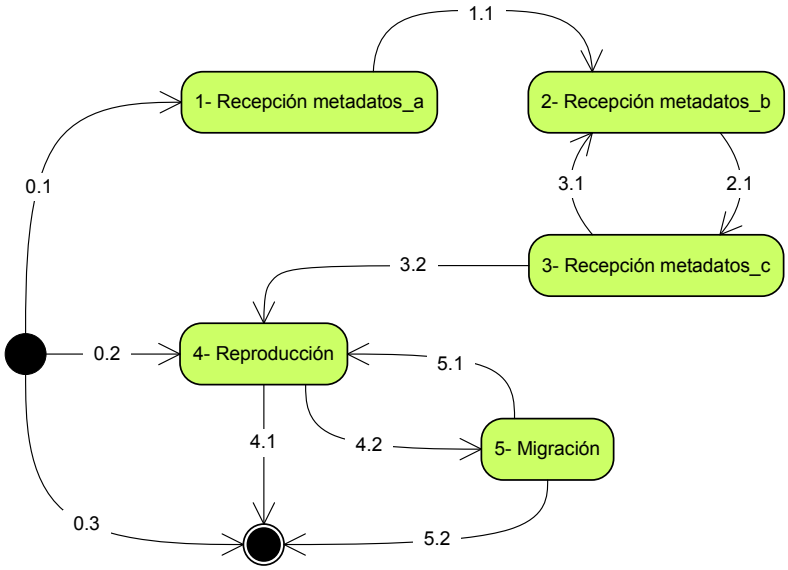


Figura 4.22: Diagrama de estados de una sesión, del lado del cliente

una conexión con un posible servidor alternativo y le transmite la petición de reserva de recursos. Para manejar dos canales SCC al mismo tiempo, uno con el servidor actual y otro con el servidor alternativo, este está asociado con dos *TcpClient*. El encargado de recibir la respuesta del servidor alternativo es un objeto de la clase *AuxSccReceiver*, el cual es una subclase de *TcpReceiver*; si la reserva de recursos es rechazada, este notifica al *SessionControl* para que contacte a un nuevo servidor.

La sesión de un cliente puede pasar por diferentes estados, cuyo diagrama se muestra en la figura 4.22. Muchas de las transiciones de estados dependen del contenido de los mensajes que transmite el servidor, los cuales fueron especificados en la sección 4.3.5. A continuación se describe cada uno de los estados y sus transiciones:

0- Inicio — El cliente transmite al servidor un mensaje de "solicitud normal de un vídeo" y, si el servidor rechaza la solicitud, vuelve a contactar a otro servidor. Si ningún servidor de la lista acepta atender la petición, entonces se pasa al estado final (transición 0.3). Si un servidor acepta la solicitud, el cliente pasa al estado *4-Reproducción* (transición 0.1) o *1-Recepción metadatos_a* (transición 0.2), dependiendo de si el campo *FirstView* del mensaje transmitido al servidor es distinto o igual a cero, respectivamente. Si pasa al estado *4-Reproducción*, previamente trans-

mite el comando interactivo para que el servidor comience a enviar el vídeo.

1- Recepción metadatos_a En este estado, el cliente recibe el campo *File Count* del mensaje "transmisión de metadatos", que identifica la cantidad de archivos de metadatos que deben recibirse. A continuación, se pasa al estado *2-Recepción metadatos_b* (transición 1.1).

2- Recepción metadatos_b En este estado se lee el campo *File Size* del mensaje "transmisión de metadatos", que indica la cantidad de octetos que deben recibirse del archivo de metadatos. Consiguientemente, se pasa al estado *3-Recepción metadatos_c* (transición 2.1).

3- Recepción metadatos_c Durante este estado se reciben los datos del archivo de metadatos y se almacenan en un archivo local. Si aún quedan por recibir archivos de metadatos, se pasa al estado *2-Recepción metadatos_b* (transición 3.1) y, caso contrario, se pasa al estado *4-Reproducción* (transición 3.2), comunicando previamente el comando interactivo para que el servidor inicie la transmisión del vídeo.

4- Reproducción Una vez que se entra en este estado, se transmite el mensaje *ping* al servidor y, luego de transcurrido cierto intervalo de tiempo, se vuelve a transmitir un *ping* solo si se ha recibido un *pong*. Si no se ha recibido el *pong*, se busca un servidor alternativo y se le transmite un mensaje de "solicitud normal de un vídeo" con el campo *First View* con valor 0. Si ningún servidor de la lista acepta atender la petición o si el cliente decide abandonar voluntariamente la sesión, se pasa al estado final (transición 4.1). Si, durante el presente estado, se recibe un mensaje de "alerta de migración", entonces se busca un servidor alternativo, se le envía un mensaje de "solicitud de reserva de recursos", y se pasa al estado *5-Migración* (transición 4.2).

5- Migración Un cliente que se encuentra en este estado está esperando que el servidor envíe un mensaje de "iniciar migración" o "cancelar migración"; cualquiera sea el mensaje recibido, se pasará al estado *4-Reproducción* (transición 5.1). Mientras se está en el presente estado, se continúa con la transmisión del *ping* y la correspondiente recepción del *pong*. Si se encuentra que el tiempo de recepción del *pong* ha superado el tiempo límite, se deberá migrar sin esperar a la recepción del mensaje de "iniciar migración" o "cancelar migración". Para efectuar la migración, se transmite un mensaje de "inicio de migración" al servidor alternativo

si es que ya se dispone de una reserva de recursos, o se busca otro servidor y se le transmite un mensaje de "solicitud normal de un vídeo" con el campo *First View* con valor 0. Si ningún servidor de la lista acepta atender la petición, entonces se pasa al estado final (transición 5.2). Es posible avanzar a este mismo estado final cuando el cliente decide voluntariamente abandonar la sesión.

6- Final En este estado se liberan todos los recursos que el cliente ha destinado a esta conexión, dando por finalizada la sesión.

4.5.3. Gestor de Datos del Vídeo

Al iniciarse el Gestor de Datos del Vídeo, se crea un objeto de la clase *VideoControl*, quien configura y pone en marcha el resto de los módulos del cliente, creando un objeto *SessionControl*, un objeto *ErapManager*, y un objeto *InterfaceControl*. Además, se crea un *PlaybackController_C* por cada versión del vídeo existente para mantener su estado de reproducción.

Cuando el *VideoControl* recibe un paquete de parte del *ErapReceiver*, este extrae el paquete STM del paquete de datos de ERAP. A continuación, los datos del vídeo se entregan al correspondiente *PlaybackController_C*, quien los almacena en un archivo que cumple la función de *buffer*, además de registrar la sección de vídeo recibida, en el mapa de *buffer* del vídeo (VBM) haciendo uso de la clase *BufferMap*.

Para reproducir una versión del vídeo específica, el correspondiente *PlaybackController_C* comprueba la existencia de una cantidad mínima de media en *buffer*. Al estar esta disponible, notifica al *VideoControl* para que, a su vez, este indique al módulo de Control de la Interfaz que ya es posible iniciar la reproducción. Para determinar cuántos segundos representan los datos almacenados en *buffer*, y a partir de ahí saber si es posible o no comenzar la reproducción, es necesario consultar los metadatos, para lo cual se utiliza la clase *MetadataAccess*. Esta clase es una copia exacta de la clase definida en el módulo Gestor de Datos de Vídeo del servidor.

4.5.4. Control de la Interfaz

La clase *InterfaceControl* es responsable de controlar todos los aspectos genéricos de una interfaz de visualización. De esta manera, se efectúa una separación de aspectos entre la lógica de control de la interfaz y la interfaz de visualización propiamente dicha (totalmente dependiente del *plugin* de visualización que se utilice).

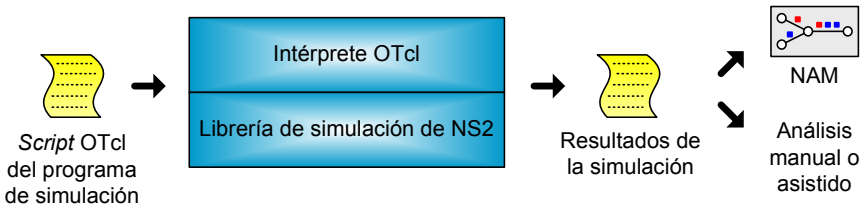


Figura 4.23: Proceso de simulación en NS2

La clase *InterfaceControl* procesa y retransmite información de la clase *VideoControl* a la clase *Interface*, y viceversa.

4.5.5. Interfaz de Visualización

Este módulo se compone de dos clases, la clase *Interface* y *VideoPlugin*. La primera es responsable de presentar en la pantalla de visualización el vídeo correspondiente y, además, de proveer los componentes gráficos adecuados para que el usuario pueda seleccionar la ejecución de comandos interactivos. La *Interface* tiene una asociación con la clase *VideoPlugin*, de la cual existen tantos objetos como versiones diferentes del vídeo se tengan, es simplemente representativa del *plugin* de vídeo que se utilice para la reproducción multimedia. Para este sistema, se utiliza el *plugin* del reproductor VLC.

4.6. Interface de VoD-NFR en NS2

Para usar NS2, se deben programar *scripts* (guiones) en Tcl orientado a objetos (OTcl). Para configurar y ejecutar una simulación de red, un usuario debe escribir un *script* que inicializa un planificador de eventos, configura la topología de red, e indica (por medio del planificador de eventos) cuándo un agente debe iniciar o detener la transmisión de paquetes. Luego de efectuar la simulación, se genera una traza o resultado de la simulación. Esta traza puede ser interpretada por una herramienta de visualización como NAM (*Network Animator*) o analizarse manualmente. En la figura 4.23 se muestra el proceso de simulación, desde que se introduce un programa OTcl al simulador, pasando por la ejecución de la simulación, hasta el análisis de las trazas.

Por razones de eficiencia, una parte del código de NS2 está implementada en C++ y otra parte en OTcl. Con el fin de reducir el tiempo de procesamiento de los paquetes y eventos, el planificador de eventos y los objetos de los componentes

básicos de la red se han escrito y compilado en C++. Estos objetos compilados pueden hacerse disponibles al intérprete OTcl a través de un enlazado, que crea y asocia un objeto OTcl por cada uno de los objetos C++. Asimismo, un objeto totalmente implementado en OTcl también puede ser accedido desde el entorno de C++. Sin embargo, hay que tener en cuenta que los métodos de un objeto OTcl/C++ solo podrán operar con instancias de objetos de su mismo entorno. En cuanto a las jerarquías de clases compiladas (C++) e interpretadas (OTcl), estas funcionan de manera muy similar en ambos lenguajes.

El *middleware* de red se ha definido completamente en C++ y no puede ser accedido desde OTcl. Sin embargo, ha sido necesario crear enlazados, bajo la identificación *Agent/TCP/FullTcp/Ns2Tcp* y *Agent/TCP/FullTcp/Ns2Udp* de la jerarquía de clases interpretadas de OTcl, con el único fin de poder acceder a ciertos métodos que han sido declarados por el simulador para el entorno OTcl y que no están disponibles para su uso con objetos C++.

Se ha definido una interfaz para VoD-NFR en NS2, soportada por el diseño presentado en la figura 4.24. Los enlazados a objetos OTcl se consiguen declarando la clase de C++ (para la cual quiere crearse un enlazado) como una subclase de *TclObject* definida por el propio entorno OTcl. En el presente diseño, la clase *VideoClientTcl*, que representa a un cliente de vídeo, es un enlazado para la clase *VideoControl*, y la clase *VideoServerTcl*, que representa a un servidor de vídeo, es un enlazado para la clase *VideoServer*.

Las clases derivadas de la clase base *TclClass* proveen dos funciones: construir la jerarquía de clases interpretadas para reflejar la jerarquía de clases compiladas, y proveer métodos para instanciar nuevas subclases de *TclObject*. La clase *VideoServerClass* es accedida, en la jerarquía de clases interpretadas, por medio de la clase base "VideoServer", y permite crear nuevas instancias de *VideoServerTcl*. La clase *VideoClientClass* está presente en la jerarquía de clases interpretadas bajo el nombre de "VideoClient", y permite crear nuevas instancias de *VideoClientTcl*.

Para ejemplificar, a continuación se muestra un *script* OTcl de una simulación en NS2 que hace uso de servidores y clientes del sistema VoD-NFR:

```
#Creación de un objeto Simulator
set ns [new Simulator]
#Apertura del archivo de trazas de NAM
set nf [open traza.nam w] $ns namtrace-all $nf
#Definición del procedimiento 'final'
proc final {} {
    global ns nf
    $ns flush-trace
    #Cierre del archivo de trazas de NAM
```

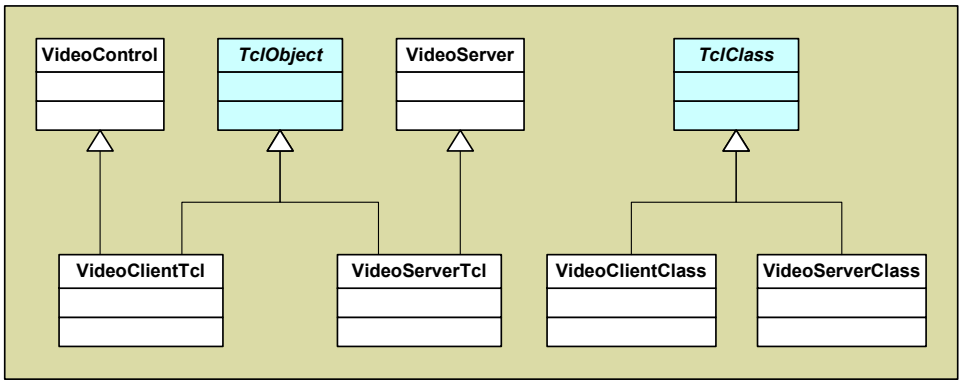


Figura 4.24: Diseño de la interfaz de VoD-NFR con NS2

```
    close $nf
    exit 0
}
#Creación de los nodos
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
#Creación de enlaces entre los nodos
$ns duplex-link $n0 $n1 100Mb 10ms DropTail
$ns duplex-link $n1 $n2 50Mb 10ms DropTail
$ns duplex-link $n2 $n3 10Mb 10ms DropTail
$ns duplex-link $n4 $n2 100Mb 10ms DropTail
#Creación de los servidores de VoD-NFR
set s1 [new VideoServer $n0 $ns]
set s2 [new VideoServer $n4 $ns]
#Creación de un cliente de VoD-NFR
set c1 [new VideoClient $n3 $n4 $n0]
#Planificación de la simulación
$ns at 0 "$c1 start"
$ns at 25 "final"
#Ejecución de la simulación
$ns run
```

En esta simulación, cuya topología es mostrada en la figura 4.25, se ha definido un servidor *s1* ubicado en el nodo *n0*, y un servidor *s2* ubicado en el nodo *n4*. También se ha definido un cliente *c1*, ubicado en el nodo *n3*, que tiene como posibles servidores a los ubicados en los nodos *n0* y *n4*, en ese orden de preferencia. La simulación se ha establecido con una duración de 25 segundos, y en el tiempo 0, el cliente *c1* hace una solicitud de un vídeo. El resto de las configuraciones (de

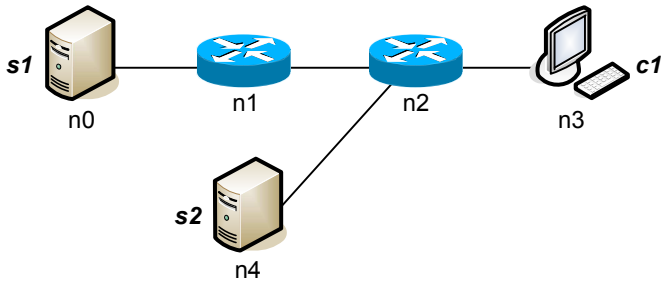


Figura 4.25: Topología especificada por el *script* de ejemplo que utiliza clientes y servidores del sistema VoD-NFR en NS2

topología de red, NAM, etc.) no corresponden a la interfaz VoD-NFR, sino que son propias de NS2 (ver detalles en [94]).

Capítulo 5

Estudio experimental

A partir de las dos aplicaciones desarrolladas del sistema VoD-NFR para entorno real y de simulación, cuyo diseño se ha expuesto en el capítulo anterior, a continuación se procede a analizar la efectividad de las técnicas propuestas en el sistema, mediante experimentos llevados a cabo con ambas aplicaciones.

Con el objeto de presentar los resultados más significativos de las pruebas reales y de simulación, llevadas a cabo para evaluar el comportamiento del nuevo sistema VoD-NFR, el estudio experimental se ha dividido en tres fases incrementales, comenzando por un único componente hasta valorar el sistema completo. En la primera fase, se evalúa el módulo NTS y se validan las simulaciones mediante pruebas reales (sección 5.1), finalizando con una comparación entre los protocolos ERAP y RAP (sección 5.2). En la segunda fase, se analiza el LCS (sección 5.3) y su integración con el NTS que demuestran la adaptabilidad del sistema a las fluctuaciones del ancho de banda de la red. En la última fase, se presentan las pruebas del sistema completo donde se verifica el correcto funcionamiento del mecanismo de detección de fallos (sección 5.4).

Las dos últimas fases, han sido realizadas únicamente mediante el uso del simulador NS2, en ciertos casos, debido a la alta necesidad de recursos para llevar a cabo la experimentación con hardware real. Sin embargo, los resultados de la experimentación en un entorno real deberían ser prácticamente iguales, debido a que la simulación del componente directamente involucrado con las transmisiones de red, y principal consumidor de recursos del servidor, ha sido validada en la primera fase.

Las pruebas reales que involucran al sistema VoD-NFR entero, se han llevado a cabo con un único cliente y un único servidor, con objeto de evaluar, principalmente, la detección de los fallos de red y la ejecución de comandos interactivos. Asimismo, aunque en el presente trabajo no se exponen sus resultados debido a que no aportan información adicional a la ya presentada, han significado de enorme valor para encontrar fallos muy difíciles de descubrir mediante la simulación. Entre estos aportes, tal vez el más significativo ha sido el descubrimiento de la necesidad

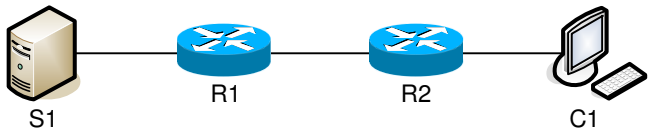


Figura 5.1: Topología de red utilizada para evaluar el NTS

de vaciar la cola del NTS, para mejorar el tiempo de respuesta en la visualización del vídeo, ante un cambio de velocidad de reproducción.

5.1. Evaluación del NTS

En esta sección se presentan los experimentos llevados a cabo para evaluar el NTS. El objetivo es contrastar y validar los resultados obtenidos por simulación, con los experimentos efectuados en entorno real. En la sección 5.1.1 se explica la metodología de evaluación del NTS, y se presenta la topología y configuración de la red utilizada para los experimentos reales y de simulación. En las secciones 5.1.2 y 5.1.3, se evalúan la transmisión de datos continua y discontinua (comportamiento ante tiempos de inactividad), respectivamente. Finalmente, en la sección 5.1.4 se presenta un análisis del rendimiento del NTS a través de una prueba de estrés. Los resultados experimentales del prototipo del NTS en entorno real validan los resultados de la simulación, y demuestran la viabilidad del NTS para trabajar en un entorno real.

5.1.1. Metodología y configuraciones

En esta sección se presenta la metodología utilizada para evaluar el rendimiento del NTS. El NTS es evaluado tanto en simulación como en entorno real, y se utiliza la misma configuración para ambos experimentos de tal manera que puedan ser comparados. En la figura *5.1* se muestra la topología lógica usada para los experimentos, donde *S1* es el servidor de VoD, *R1* y *R2* son los encaminadores, y *C1* es un cliente. El enlace *R1*-*R2* es el cuello de botella y *R1* es el punto del cuello de botella. Los encaminadores utilizan planificación FIFO y una política de colas *DropTail*. En la tabla 5.1 se muestran los parámetros de configuración de la red.

Las simulaciones fueron efectuadas con el simulador NS2. Para experimentos reales, se ha configurado la red mostrada en la figura 5.2, donde un nodo actúa como servidor de VoD (*S1*) y ejecuta el lado del servidor del NTS, otro actúa como un cliente (*C1*) y ejecuta el lado del cliente del NTS, y el restante es un encaminador

Enlace	**S1–R1**	**R1–R2**	**R2–C1**
Ancho de banda	100 Mb/s *full duplex*	10 Mb/s *full duplex*	100 Mb/s *full duplex*
Tamaño de la cola	20 paquetes	20 paquetes	20 paquetes
Retardo de propagación	0.000008 ms	20 ms	0.000008 ms

Cuadro 5.1: Parámetros de configuración de la red

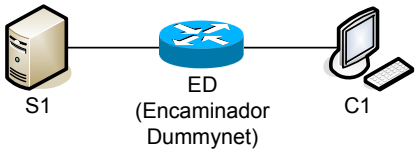


Figura 5.2: Topología de red utilizada para experimentos reales

(*DR*). La primer máquina utiliza el NTS para enviar datos de vídeo a la segunda máquina que ejecuta el lado del cliente del NTS. El tráfico entre las primeras dos máquinas atraviesa la tercer máquina, que es un encaminador que ejecuta el sistema Dummynet [95], una herramienta flexible que moldea o configura el tráfico de la red. Dummynet se utiliza para manipular las variables de la red mientras que es parte de una red real, y se ha configurado para simular el enlace *R1–R2* de la topología de red utilizada para la experimentación (ver figura *5.1*). Dummynet trabaja interceptando los paquetes de la pila de protocolos, y pasándolos a través de objetos (colas y tuberías — *pipes* —) se simulan los efectos de limitación de ancho de banda, retardos de propagación, colas finitas, pérdidas de paquetes, y multicaminos.

El encaminador Dummynet FreeBSD se ha configurado con dos tarjetas Ethernet de 100 Mb/s para conectarlo al cliente y al servidor. Para los enlaces de red se han utilizado cables tipo par trenzado UTP de categoría 5, y todas las tarjetas de red se configuraron para transmisión *full-duplex* a 100 Mb/s. Las especificaciones del hardware y software se listan en la tabla 5.2. El enlace *R1–R2* se configura utilizando Dummynet, con un tamaño de cola de 20 paquetes y 10 ms de retardo de propagación. Note que Dummynet no considera el costo del protocolo Ethernet para la especificación del ancho de banda. Por ejemplo, el ancho de banda configurado en Dummynet para paquetes de 1.000 bytes y un ancho de banda de 10 Mb/s es $10 \times (1 - 38/1000) = 9{,}62$ Mb/s donde 38 es el costo por paquete originado por el protocolo Ethernet. Los otros enlaces, *S1–R1* y *R2–C1*, son los enlaces reales *S1–DR* y *DR–C1*. El retardo de propagación según la especificación del cable de categoría 5 es aproximadamente 548 ns / 100 m y, como se han utilizado cables de

	Servidor	Encaminador	Cliente
CPU	Intel Pentium(R) 4 3.00 GHz	Intel Pentium(R) 4 3.00 GHz	Intel Pentium(R) 4 2.00 GHz
RAM	1 GB	256 MB	768 MB
Tarjeta Ethernet	100 Mb/s	100 Mb/s	100 Mb/s
Sistema Operativo	núcleo de Linux versión 2.6.17	PicoBSD 0.445	núcleo de Linux versión 2.6.17
eth0: IP / máscara	192.168.1.2 / 24	192.168.1.1 / 24	192.168.2.2 / 24
eth1: IP / máscara	N/A	192.168.2.1 / 24	N/A
Gateway defecto	192.168.1.1	N/A	192.168.2.1

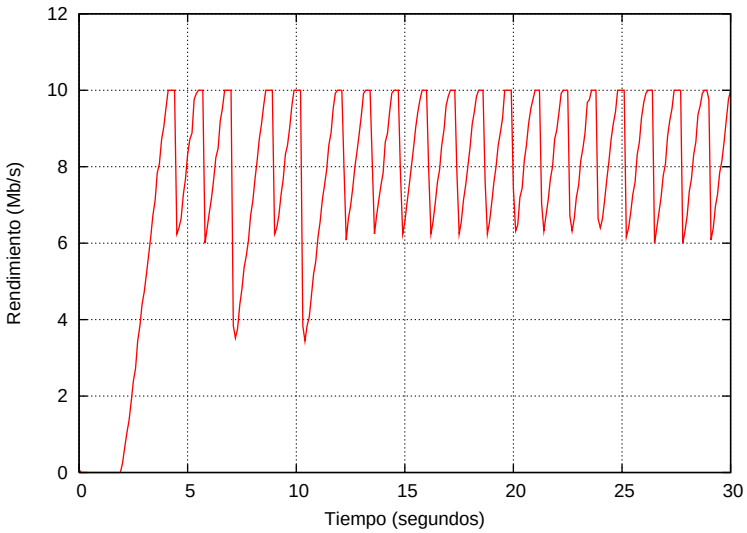
Cuadro 5.2: Especificaciones del hardware y software utilizados en la experimentación de entorno real

una longitud de 1,5m, se tienen 8 ns de retardo de propagación para ambos enlaces (*S1–DR* y *DR–C1*) que es igual al retardo de propagación especificado para los enlaces *S1–R1* y *R2–C1*.

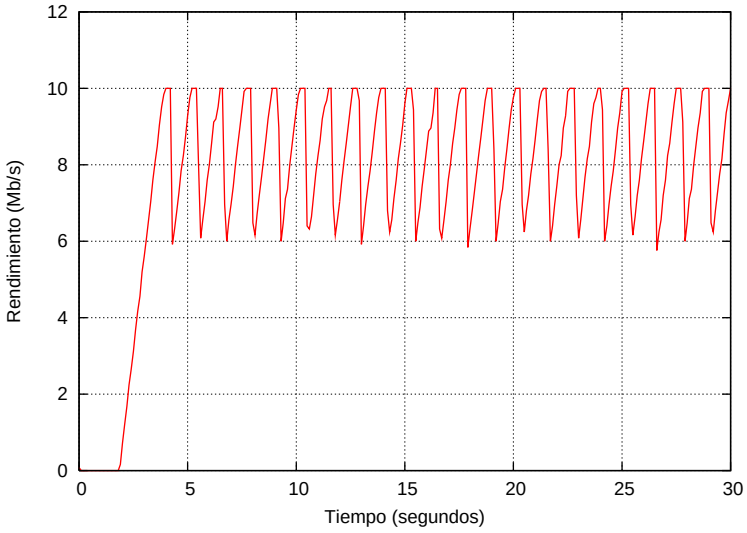
5.1.2. Transmisión continua

Se han llevado a cabo pruebas de transmisión continua desde el servidor de VoD *S1* al cliente *C1*, durante un intervalo de 60 segundos, y con un tamaño de paquetes de 1.000 bytes que incluye los encabezados Ethernet, IP, UDP, y ERAP, los datos del vídeo, la cola Ethernet y el IFG. Las figuras 5.3 (a) y (b) muestran el rendimiento (eje y) durante el tiempo de la prueba (eje x) para los experimentos en simulación y entorno real, respectivamente. Los gráficos muestran el mismo rendimiento y comportamiento durante todo el tiempo. La tasa de transmisión se incrementa hasta llegar a la capacidad máxima de transmisión del enlace *R1–R2*, momento en el cual el encaminador *R1* comienza a descartar paquetes, y ante la detección de pérdidas de paquetes, la tasa de transmisión se decrementa inmediatamente. Estas dos fases, de incremento y decremento de la tasa de transmisión, son repetidas continuamente.

Los gráficos en la figura 5.4 (a) y (b) muestran la tasa de transmisión supuesta por el emisor NTS. Cuando el NTS envía datos a una tasa de transmisión mayor a 10 Mb/s, la red no puede transferir los datos a esa velocidad, con lo cual la carga extra es soportada por las colas de los encaminadores. Cuando las colas están llenas, algunos paquetes son descartados y, cuando el emisor NTS detecta la falta de paquetes, inmediatamente decrementa la tasa de transmisión. De esta manera, se verifica que los valores de las muestras de anchos de banda (o tasa de transmisión



(a) Entorno simulado



(b) Entorno real

Figura 5.3: Rendimiento del NTS alcanzado a lo largo de la experimentación

supuesta) que obtiene el emisor NTS en un entorno de simulación se corresponden con los de la realidad.

Los resultados muestran que, para el caso de transmisiones de datos continuas, el uso del ancho de banda de la red, para el caso real y de simulación, son iguales. Por lo tanto, se considera validada la simulación de la transmisión continua de datos del NTS.

5.1.3. Comportamiento ante tiempos de inactividad

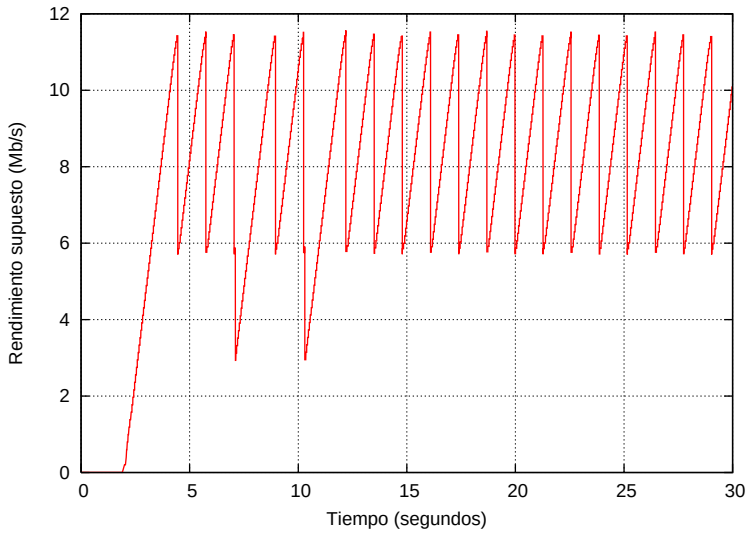
El LCS planifica la entrega de los datos de los vídeos para intervalos de tiempo. En cada intervalo, se sirven a los clientes con más necesidad de media, y la transmisión a los otros clientes se suspende momentáneamente. Por lo tanto, el NTS debe soportar los periodos de inactividad de las conexiones. Durante los periodos de inactividad de un cierto cliente, el NTS reduce el consumo de recursos (de CPU y memoria) deteniendo todos los temporizadores utilizados para gestionar la transmisión. Luego, la transmisión es reiniciada, y la tasa de transmisión continua desde el punto previo.

Se han realizado pruebas de transmisión discontinua desde el servidor de VoD *S1* al cliente *C1*, con un periodo de inactividad entre el segundo 1,45 y 5, y con un tamaño de paquete de datos de 1.000 bytes que incluye los encabezados Ethernet, IP, UDP, y ERAP, los datos del vídeo, la cola Ethernet y el IFG. Las figuras 5.5 (a) y (b) muestran el rendimiento (eje y) durante el tiempo de la prueba (eje x) para los experimentos en simulación y entorno real, respectivamente. Los gráficos muestran el mismo rendimiento y comportamiento durante todo el tiempo. En el segundo 1,45 se suspende la transmisión y se reinicia en el segundo 5; la nueva tasa de transmisión (en el segundo 5) es levemente mayor a la tasa de transmisión del punto previo (segundo 1,45). Esto es porque como el *RttTimer* debió ser disparado durante el tiempo de inactividad, entonces cuando la transmisión se reinicia, el temporizador es inmediatamente disparado, produciendo un decremento del IPG.

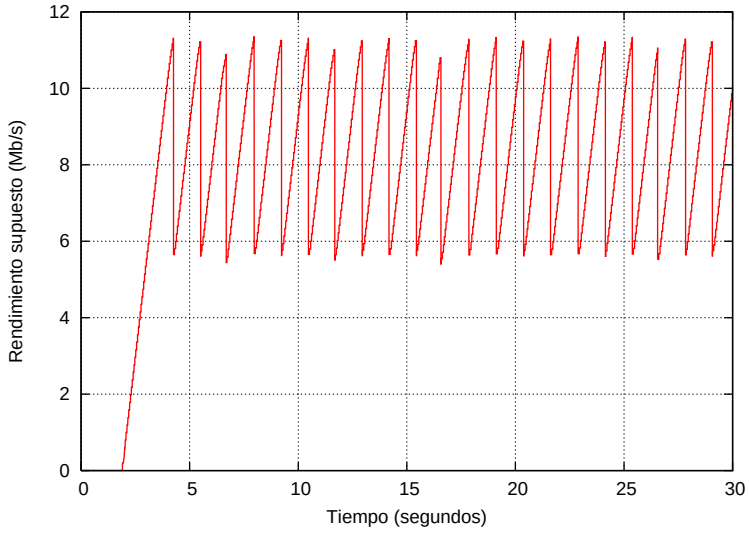
Los resultados muestran la igualdad, en el uso del ancho de banda de red en transmisiones de datos discontinuas, para el caso real y de simulación. Por lo tanto, se considera validada la simulación de la transmisión discontinua de datos del NTS.

5.1.4. Tasa de transmisión máxima

Se ha diseñado una prueba de estrés para evaluar el rendimiento del NTS a una alta carga de trabajo y determinar su consumo de CPU y precisión en el disparo de los temporizadores; note que ambos parámetros están relacionados ya que la

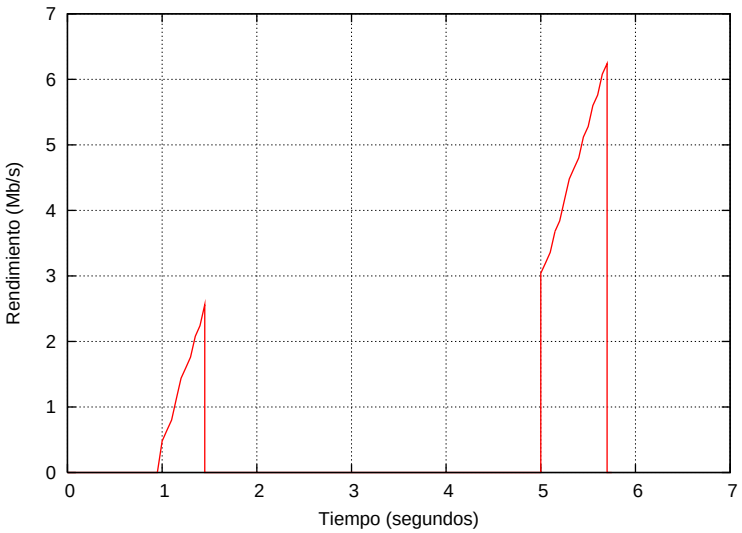


(a) Entorno simulado

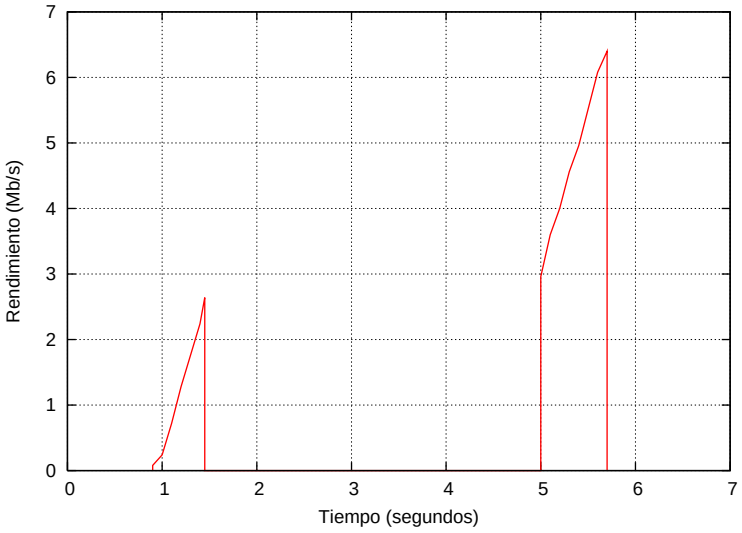


(b) Entorno real

Figura 5.4: Tasa de transmisión supuesta por el emisor NTS



(a) Entorno simulado



(b) Entorno real

Figura 5.5: Comportamiento en tiempo de inactividad

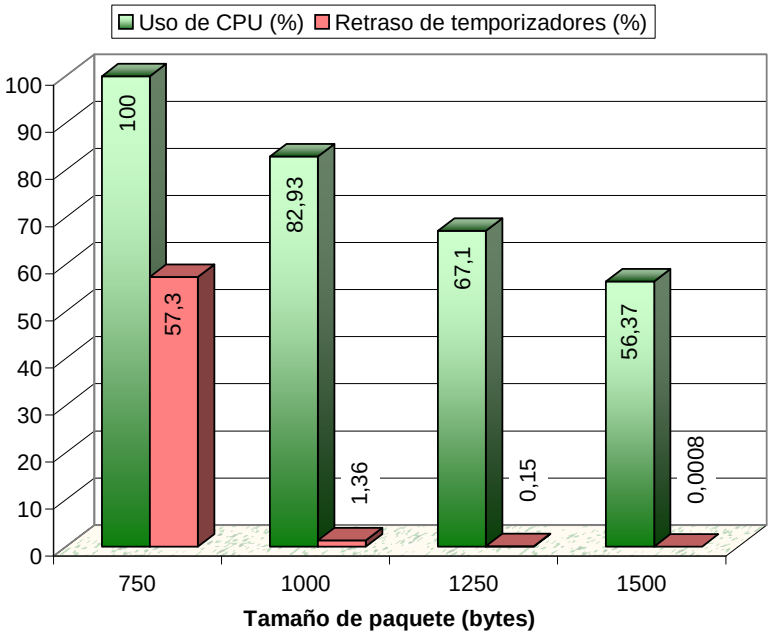


Figura 5.6: Rendimiento del NTS a 100 Mb/s con diferentes tamaños de paquete

CPU es el recurso más crítico para lograr tal precisión. El prototipo del NTS se ha configurado para enviar tráfico a 100 Mb/s, variando el tamaño de los paquetes entre 750, 1.000, 1.250 y 1.500 bytes. La figura 5.6 muestra el porcentaje de CPU utilizado y el retardo promedio de los temporizadores para los diferentes tamaños de paquete. Cuando el NTS usa un tamaño de paquete de 750 bytes, la CPU es utilizada al 100 % de su capacidad. Por lo tanto, los temporizadores son disparados con un retraso muy grande, en el orden del 56,37 %. Con paquetes de 1.000 bytes, la precisión de los temporizadores es buena (1,36 % de retraso) pero el consumo de CPU es aún muy alto, en el orden de 82,93 %. El consumo de CPU y la precisión es mejorada con paquetes de 1250 bytes (67 % de uso de CPU y 0,15 % de retraso de temporizadores), y aún más con paquetes de 1.500 bytes, alcanzando un retraso de los temporizadores de 0,0008 % y un uso de CPU de 56,37 %. Un consumo de CPU de 56,37 % es suficientemente bueno considerando que el NTS es el componente del servidor de VoD con más necesidad de recursos.

5.2. Comparación de los protocolos ERAP y RAP

ERAP presenta numerosas mejoras con respecto al protocolo del cual ha heredado su política de control de congestión, el RAP. Es importante mencionar que muchas de las deficiencias de RAP, solucionadas por ERAP, también ocurren en casi todos los protocolos de control de congestión existentes. En la sección 5.2.1 se presentan las pruebas que muestran que ERAP mantiene el mismo comportamiento que RAP al adaptar la tasa de transmisión ante casos de congestión de la red y, en la sección 5.2.2, se describen las diferencias entre ambos protocolos, contrastando sus rendimientos. Los resultados presentados muestran que ERAP mejora a RAP desde el punto de vista del consumo de recursos del servidor y la red, permitiendo concluir que el protocolo ERAP elimina las limitaciones del protocolo RAP, dando al servidor de vídeo más flexibilidad y oportunidad para atender las peticiones de los clientes.

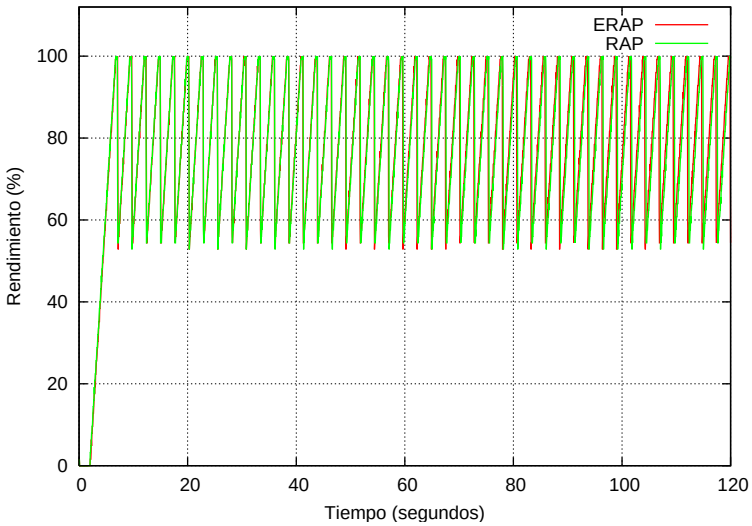
5.2.1. Política de control de congestión

RAP y ERAP han presentado prácticamente el mismo comportamiento en cuanto al uso del ancho de banda de la red, donde la mínima diferencia apreciada es debido a que RAP utiliza paquetes de ACKs un poco más grandes que ERAP (24 bytes contra 18). La figura 5.7 (a) muestra el porcentaje de uso del ancho de banda de la red para una simulación de 120 segundos, en la cual se enviaron datos de manera continua desde un servidor a un cliente, por una red con igual topología que la presentada en la figura *5.1* (*S1–R1–R2–C1*). En esta topología se consideraron los siguientes parámetros: ancho de banda *R1–R2* de 1 Mb/s y el resto de 2 Mb/s, retardos de enlaces de 10 ms, tamaño de paquetes de datos de 100 bytes, y tamaño de la cola *R1–R2* de 10 paquetes. La tasa de transmisión se aumenta hasta que el enlace *R1–R2* llega a su capacidad máxima de transmisión, entonces el encaminador *R1* comienza a descartar paquetes y, ante la detección de pérdidas de paquetes, la tasa de transmisión se decrementa inmediatamente. Estas dos fases, la de incremento y decremento de la tasa de transmisión, se repiten de manera reiterada.

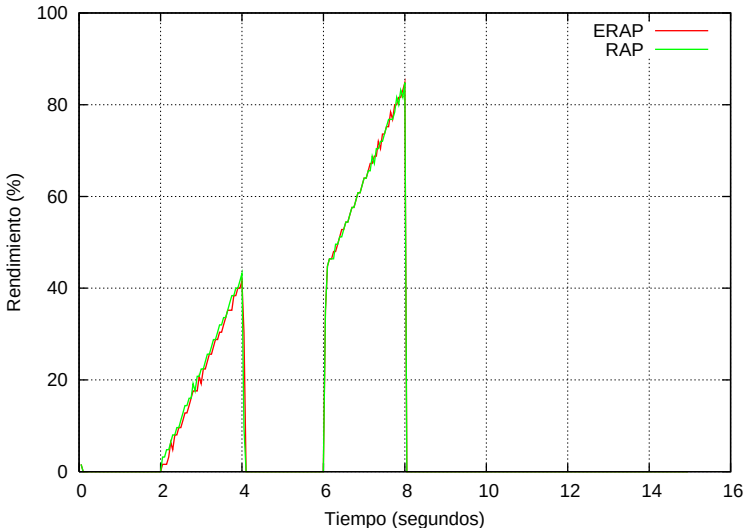
En la figura 5.7 (b) se estudia el comportamiento de ambos protocolos ante un período de inactividad comprendido entre el tiempo 4 y 6, y un tamaño de paquetes de datos de 100 bytes. La gráfica resultante muestra un comportamiento similar de los dos protocolos en cuanto al uso del enlace.

5.2.2. Diferencias entre RAP y ERAP

A continuación se analizan las diferencias de ambos protocolos con respecto a:



(a) Uso contínuo de la red durante 120 segundos



(b) Comportamiento en tiempos de inactividad

Figura 5.7: Comportamiento de la política de control de congestión de ERAP y RAP

Recursos destinados a la gestión de eventos

La desactivación de los temporizadores en momentos de inactividad que efectúa ERAP tiene dos ventajas: el ahorro de recursos de memoria y de CPU. ERAP reduce significativamente la cantidad de temporizadores o eventos a planificar con respecto a RAP ya que, en vez de tener $2N$ temporizadores, siendo N la cantidad total de conexiones, solo se tendrán $2X$ temporizadores, donde X es la cantidad de conexiones que han sido seleccionadas por el LCS para ser servidas en determinado intervalo o *slot* de tiempo. De esta manera, al reducir la cola de eventos, se reduce el consumo de memoria y se evita el disparo de temporizadores inútiles ahorrando ciclos de CPU.

Por lo tanto, en base a los datos expuestos, se concluye que ERAP presenta una gestión de eventos mucho más refinada que RAP, que permite reducir notablemente el consumo de recursos.

Recursos destinados a la recepción de paquetes de reconocimiento

ERAP mejora el uso de los recursos al centralizar la recepción de paquetes de reconocimiento (ACKs). Este protocolo sólo utiliza un *thread* para realizar la tarea de recepción de ACKs, en vez de los múltiples *threads* (uno por conexión) que utiliza RAP, con lo cual se ahorra una gran cantidad de recursos de memoria y CPU. Cada *thread* vacío (es decir, sin datos ni código) ocupa aproximadamente 10 MB de memoria (por defecto), con lo que si se utiliza una recepción de ACKs descentralizada para N conexiones (por ende, con N *threads* receptores), se requerirán $10N$ MB de memoria, y rápidamente el sistema comenzará a paginar[1].

En cuanto al uso de CPU, al tener un gran número de *threads*, la planificación y los cambios de contexto serán más costosos. Para su verificación, se realizaron pruebas en donde una aplicación envía cierta carga de tráfico UDP, que distribuye equitativamente entre N puertos de otro ordenador, en el cual N *threads* (uno por puerto) se encargan de recibir los paquetes. El gráfico de la figura 5.8 muestra la suma de los tiempos consumidos en modo usuario y sistema para cada conjunto de *threads* receptores (1, 50, 100, 150, 200, y 250 *threads*) con una carga de 10^6 paquetes (distribuidos entre los *threads* participantes), donde se aprecia un claro incremento del consumo de tiempo de CPU cuando aumenta la cantidad de *threads*. Otra cuestión a tener en cuenta es que el sistema operativo limita el número de *threads* que puede lanzar cada proceso, con lo cual también se vería restringida la cantidad de sesiones que puede soportar el servidor.

[1] La paginación es un proceso por el cual se efectúa un intercambio de páginas entre la memoria y el disco, el cual también es conocido como *swapping*.

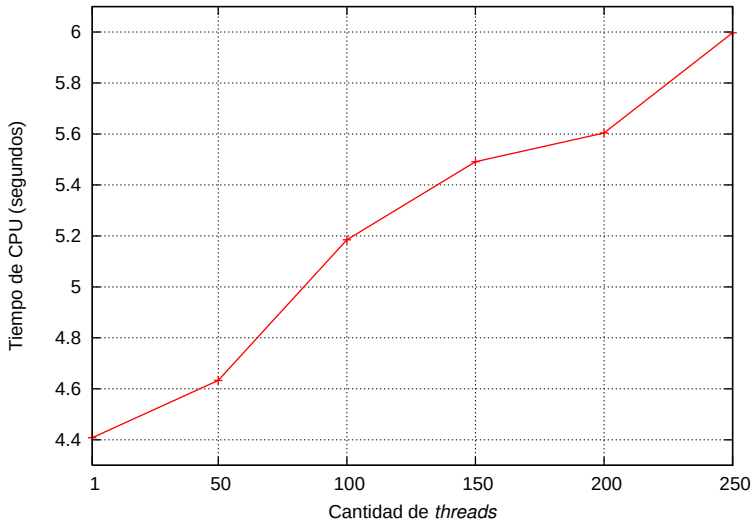


Figura 5.8: Uso de CPU según el número de *threads* receptores de paquetes

A partir de los argumentos presentados, se concluye que ERAP destina una cantidad de recursos, a la recepción de paquetes de reconocimiento, significativamente menor a RAP.

Sobrecarga de los protocolos

El tamaño reducido de los encabezados de ERAP, logrado principalmente utilizando diferentes encabezados para paquetes de datos y de reconocimiento (ACK), mejora el rendimiento del protocolo con respecto al RAP. Por un lado, ERAP tiene un encabezado de paquetes de reconocimiento de 18 bytes contra 24 bytes de RAP, que significa una reducción de la sobrecarga de RAP de un 25 %. Por otro lado, el encabezado de paquetes de datos de ERAP requiere solamente 4 bytes, contra los 24 bytes que utiliza RAP, resultando en una reducción de la sobrecarga de un 83,33 % con respecto a la de RAP. En la figura 5.9 se muestra una gráfica de la sobrecarga impuesta por ambos protocolos en paquetes de datos de diferentes tamaños.

Los resultados presentados permiten concluir que ERAP ofrece una sobrecarga del protocolo en la red mucho menor que RAP.

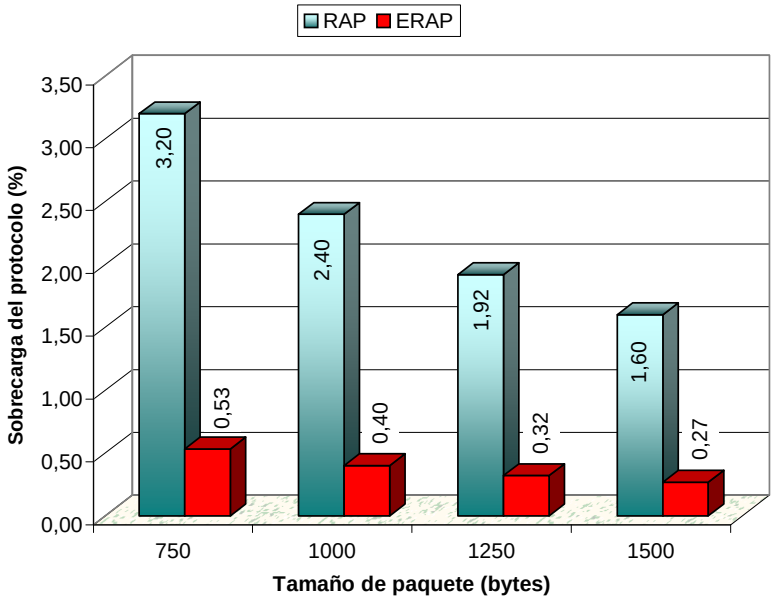


Figura 5.9: Sobrecarga del protocolo ERAP y RAP en los paquetes de datos

Regulación de la tasa de transmisión global

La centralización del envío de paquetes de todas las conexiones que incluye ERAP permite regular la tasa de tráfico a la capacidad que puede soportar el enlace de red. La falta de un mecanismo regulador de la tasa de transmisión en RAP, con operaciones de envío de paquetes UDP no bloqueantes, produce una gran cantidad de descartes en el nodo emisor debido al desbordamiento de la cola de la interfaz de red. Esto no solo origina una gran carga al servidor ya que debe retransmitir los paquetes descartados, sino que también desaprovecha ancho de banda disponible de red debido a que el protocolo, al suponer que fue la red quien ha perdido el paquete, disminuye la tasa de transferencia de las conexiones.

Se han realizado simulaciones para comparar el comportamiento de ERAP y RAP, en una situación en que la suma de todos los anchos de banda individuales de las conexiones entre el servidor y los clientes sea mayor que el ancho de banda total de salida del servidor. La figura 5.10 muestra la topología de la red simulada y la tabla 5.3 presenta los parámetros de la simulación. En este caso, el ancho de banda de salida del servidor es de 10 Mb/s, y la suma de todos los anchos de banda individuales de las conexiones es de 20 Mb/s.

El resultado de la simulación para el protocolo RAP es presentado en la figura 5.11 (a), donde se verifica la pérdida de paquetes por falta de regulación de la

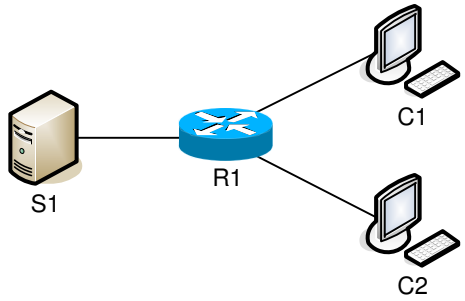


Figura 5.10: Topología de red utilizada para verificar la regulación de la tasa de transmisión

Parámetro de simulación	**Descripción**
Tamaño de paquetes	1.040 bytes
Ancho de banda de los enlaces	10 Mb/s
Retardo de los enlaces	10 ms
Tipo de cola de mensajes	*DropTail*
Tamaño de la cola de mensajes *R0–R1*	100 paquetes

Cuadro 5.3: Parámetros de simulación para verificar la regulación de tasa

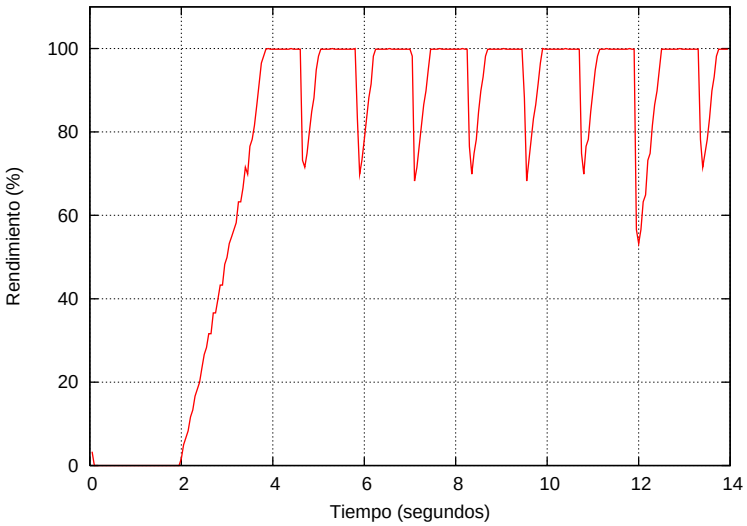
tasa de transmisión y el desaprovechamiento de la capacidad del enlace de red del servidor; mientras que ERAP presenta un muy buen comportamiento en las mismas situaciones, no pierde paquetes y utiliza todo el ancho de banda disponible del enlace de red, como puede observarse en la figura 5.11 (b).

Todas estas pruebas demuestran que ERAP tiene un rendimiento más elevado que RAP gracias a su mecanismo de regulación de la tasa de transmisión global.

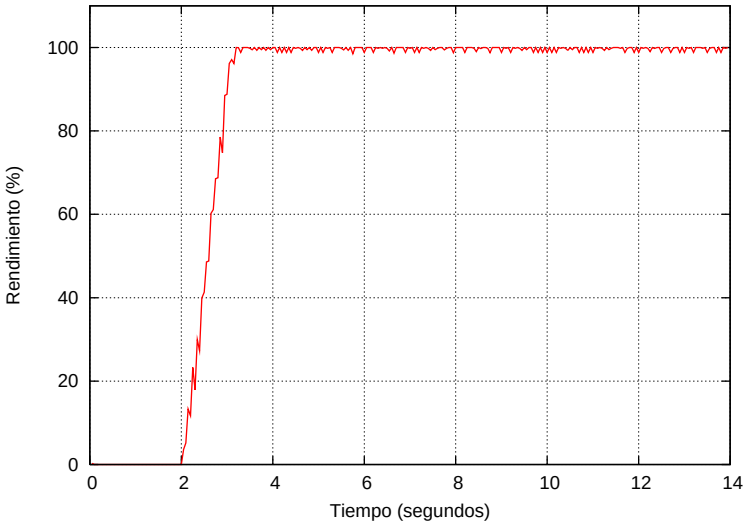
5.3. Adaptabilidad del sistema a las fluctuaciones del ancho de banda de la red

Se han realizado simulaciones para estudiar la aptitud del sistema VoD-NFR para planificar la entrega de los vídeos a los clientes. La idea de estas simulaciones es mostrar las ventajas que se logra al integrar adecuadamente el control de flujo del nivel de aplicación (LCS) con el control de flujo del nivel de la red (STM).

Se presentan dos aplicaciones contra las cuales se compara al sistema VoD-NFR. Estas dos aplicaciones envían el vídeo a los clientes, una mediante el protocolo de transporte RAP y la otra mediante TCP. Estas aplicaciones mantienen



(a) RAP no regula la tasa de transmisión total y pierde paquetes



(b) ERAP regula la tasa de transmisión total y no pierde paquetes

Figura 5.11: Comportamiento de ERAP y RAP cuando se alcanza el 100 % de uso del enlace de red del servidor

una total independencia entre la capa de aplicación y red, derivada del uso de los protocolos RAP y TCP, que impide planificar la entrega de los vídeos. Note que el propósito de las simulaciones no es comparar protocolos, sino mostrar la ventaja que obtiene el sistema VoD-NFR por medio de la integración y colaboración de las capas de aplicación y red.

En la figura 5.12 se presenta la topología de la red simulada con los anchos de bandas (expresados en Mb/s) correspondientes a cada enlace. En la tabla 5.4 se presentan los parámetros de simulación utilizados, en donde la carga útil se calcula de la siguiente manera:

- TCP: 1.040 (tamaño del paquete) - 20 (encabezado de IP) - 24 (encabezado de TCP) - 4 (encabezado STP)
- RAP: 1.040 (tamaño del paquete) - 20 (encabezado de IP) - 8 (encabezado de UDP) - 24 (encabezado de RAP) - 4 (encabezado STP)
- VoD-NFR: 1.040 (tamaño del paquete) - 20 (encabezado de IP) - 8 (encabezado de UDP) - 4 (encabezado de datos de ERAP) - 4 (encabezado STP)

Para el sistema VoD-NFR se ha definido una cola de mensajes para el nodo *n0* mucho más pequeña que para los otros dos sistemas. La razón tiene que ver con que VoD-NFR mantiene todos los paquetes (a transmitir) en su propio dominio y prácticamente no utiliza la cola del nodo debido a que los paquetes se envían a la velocidad que la red puede aceptarlos. Por lo tanto, es indiferente si la cola es de 1.000 paquetes o 10 paquetes.

En esta simulación se tienen tres clientes que solicitan el mismo vídeo en el mismo instante de tiempo. El servidor se encuentra en el nodo *n0*, el cliente 1 se encuentra en el nodo *n6*, el cliente 2 en el nodo *n7*, y el cliente 3 en el nodo *n8*.

Desde el segundo 10 hasta el segundo 15 se realiza un envío continuo de paquetes UDP, desde el nodo *n9* al nodo *n10*. Este flujo no tiene ningún control de congestión. De esta manera, se apropia durante el tiempo de duración del flujo, del ancho de banda del enlace comprendido entre los nodos *n4* y *n5*. Inicialmente, se realiza un *prefetch* donde el cliente almacena tantos datos como puede en un lapso de 10 segundos; se ha optado por esta política de *prefetch* ya que permite efectuar una comparación justa donde todos los clientes de todos los sistemas comienzan a consumir media en el mismo instante de tiempo.

A continuación se presentan las simulaciones: VoD-NFR frente a RAP, y VoD-NFR frente a TCP.

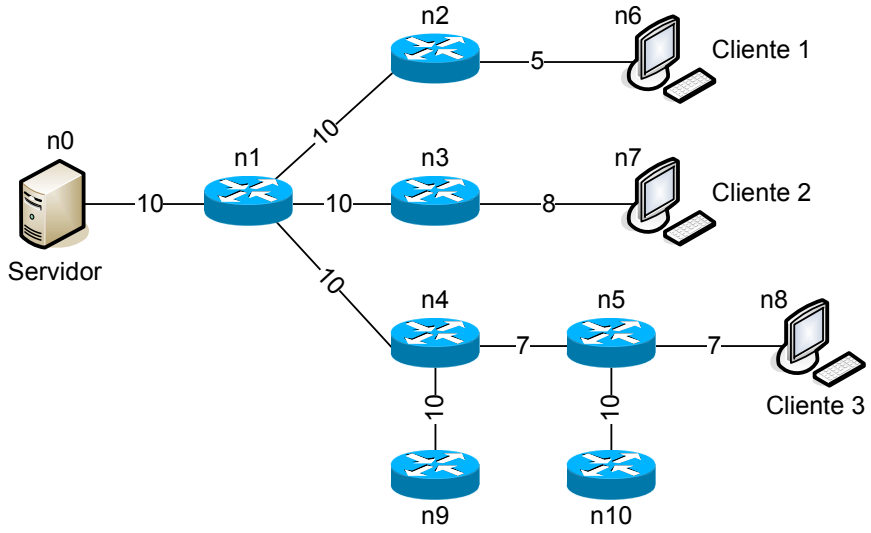


Figura 5.12: Topología de red utilizada para comparar a VoD-NFR contra una aplicación que utiliza flujos RAP y otra que utiliza TCP

Parámetro de simulación	**Descripción**
Tamaño de paquetes	1.040 bytes
Carga útil de los paquetes de datos	RAP: 984 bytes TCP: 992 bytes VoD-NFR: 1.004 bytes
Retardo de los enlaces	10 ms
Tipo de cola de mensajes	*DropTail*
Tamaño de la cola de mensajes del nodo *n0*	RAP y TCP: 1.000 paquetes VoD-NFR: 10 paquetes
Tiempo de simulación	240 segundos

Cuadro 5.4: Parámetros de la simulación que compara a VoD-NFR frente a aplicaciones con transmisiones RAP y TCP

5.3.1. VoD-NFR frente a RAP

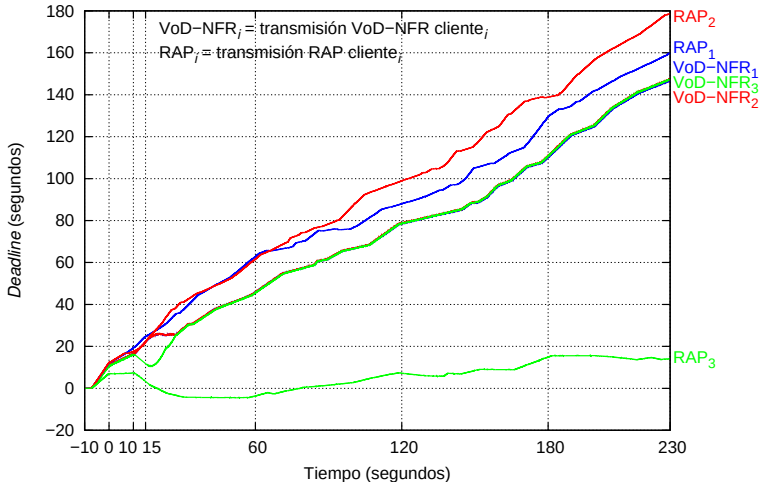
En esta sección, se muestran las ventajas que logra el sistema VoD-NFR que hemos presentado, frente a la aplicación con conexiones independientes RAP que no permite efectuar una planificación de alto nivel que tenga en cuenta los aspectos de las comunicaciones.

La figura 5.13 (a) muestra el resultado completo de la simulación mediante un gráfico que muestra el *deadline* para cada cliente a lo largo del tiempo de la experimentación, donde RAP_i y $VoD-NFR_i$ representan las transmisiones efectuadas por el cliente i, mediante la aplicación de flujos RAP y el sistema VoD-NFR, respectivamente. Los valores negativos de la escala del *deadline* representan la cantidad de media faltante para un instante dado de tiempo, es decir, media que el cliente ya debería haber recibido para no sufrir cortes en la visualización. En la figura 5.13 (b) se observa, con mayor detalle, la reacción de VoD-NFR y de RAP, al momento de producirse la congestión en el camino de comunicación con el cliente 3.

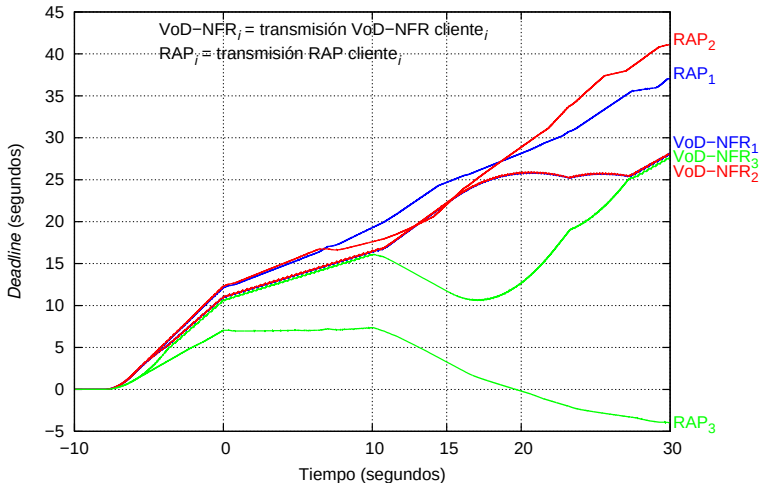
El sistema VoD-NFR recupera rápidamente las comunicaciones luego de que el tráfico, que se apropió (entre el segundo 10 y 15) del ancho de banda entre el enlace del nodo *n4* al *n5*, deja de existir. Mientras el flujo del cliente 3 tiene menor *deadline*, el servidor le dedica todos los recursos posibles para mejorar su situación. Como se observa en los gráficos, VoD-NFR disminuye el envío de media a los clientes 1 y 2, logrando un pronunciado ascenso del *deadline* del cliente 3. Aproximadamente en el segundo 25, los tres flujos vuelven a tener el mismo *deadline*. Ninguno de los clientes experimenta interrupciones durante la reproducción de los vídeos y se aprecia un muy buen funcionamiento general del sistema VoD-NFR.

Ahora, analicemos el caso de la aplicación que envía el vídeo a los clientes mediante conexiones RAP. El cliente 3, a los 20 segundos de haber comenzado la reproducción, empieza a experimentar cortes en la visualización del vídeo. En el tiempo 58, el cliente 3 ha tenido cortes en la reproducción por un total de 5 segundos, y recién desde ese momento comienza a recuperarse. Sin embargo, la presencia de los otros dos flujos, impiden que este flujo mejore su transmisión como debería. Note que el cliente 1 tiene menos ancho de banda que el cliente 3 (5 Mb/s contra 7 Mb/s). No obstante, teniendo el cliente 3 todo el ancho de banda disponible a partir del tiempo 15, no supera en ningún momento la pendiente de crecimiento del *deadline* experimentada por el cliente 1. Se observa una clara injusticia en la distribución del ancho de banda entre las diversas conexiones.

Si se compara el *deadline* promedio de los tres flujos al finalizar la simulación (tiempo 230), VoD-NFR también aventaja a la aplicación con conexiones RAP. VoD-NFR presenta un *deadline* promedio de 144,9 segundos $((145+144{,}5+$



(a) Simulación completa



(b) Ampliación de la subfigura (a), que muestra el momento de congestión de la red

Figura 5.13: VoD-NFR frente a RAP

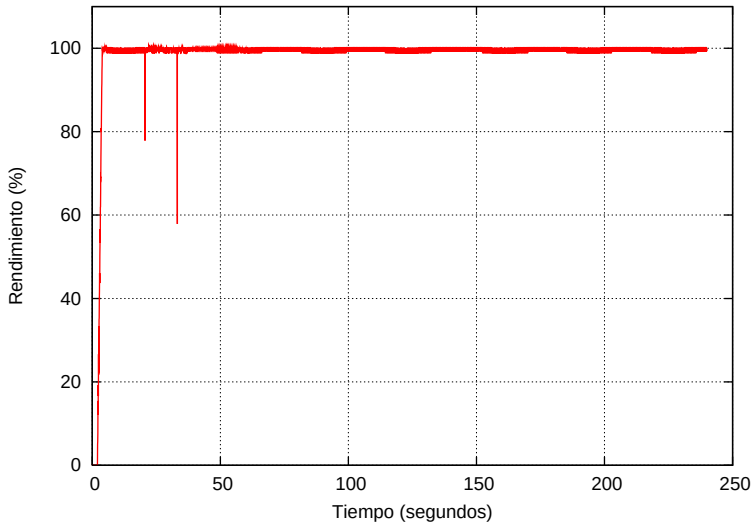


Figura 5.14: Ancho de banda utilizado del enlace entre el nodo *n0* y *n1*, correspondiente a la simulación del sistema VoD-NFR

145,1)/3), mientras que la aplicación con RAP presenta un *deadline* promedio de 117,6 segundos ((160+178,7+14,1)/3).

La ventaja en cuanto al mayor rendimiento promedio de VoD-NFR es debida a los siguientes factores: el menor tamaño de los encabezados de datos, la regulación de la tasa de transmisión del servidor que evita descartes de paquetes, y el mejor aprovechamiento del ancho de banda disponible. Puede verse el uso del ancho de banda del enlace comprendido entre los nodos *n0* y *n1*, para la simulación del sistema VoD-NFR y de la aplicación con flujos RAP, en las figuras 5.14 y 5.15, respectivamente.

5.3.2. VoD-NFR frente a TCP

En esta sección, se muestran las ventajas que logra el sistema VoD-NFR que hemos presentado, frente a la aplicación con conexiones independientes TCP que no permite efectuar una planificación de alto nivel que tenga en cuenta los aspectos de las comunicaciones.

La figura 5.16 (a) muestra el resultado completo de la simulación mediante un gráfico que muestra el *deadline* para cada cliente a lo largo del tiempo de la experimentación, donde TCP_i y $VoD-NFR_i$ representan las transmisiones efectuadas

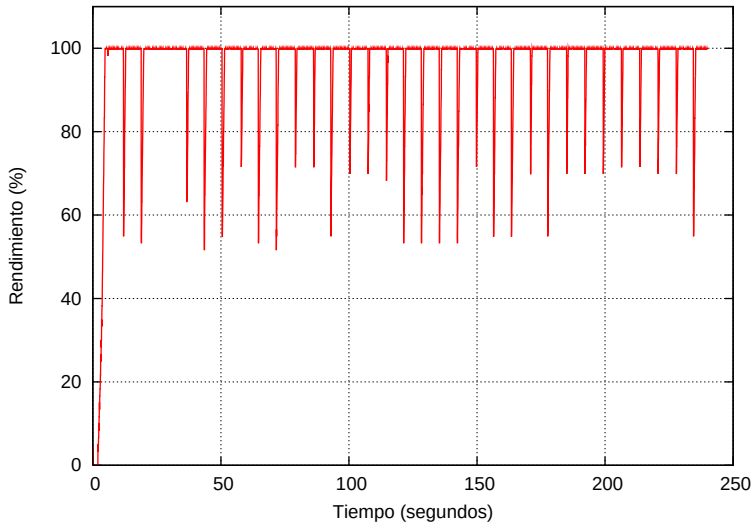
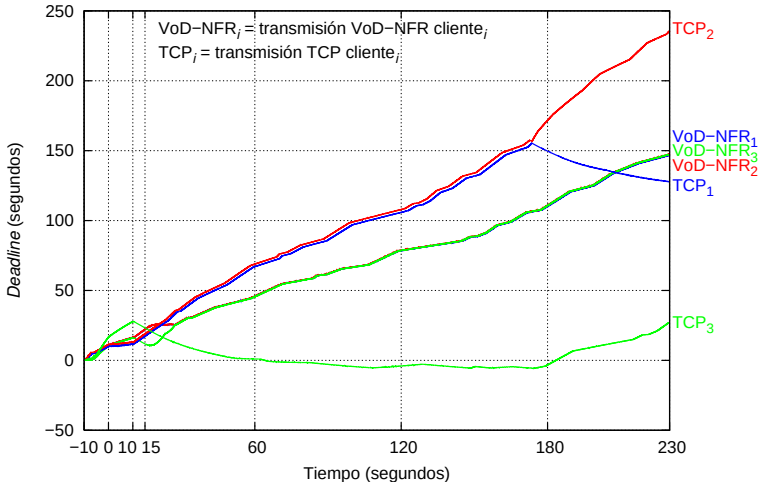


Figura 5.15: Ancho de banda utilizado del enlace entre el nodo *n0* y *n1*, correspondiente a la simulación de la aplicación con flujos RAP

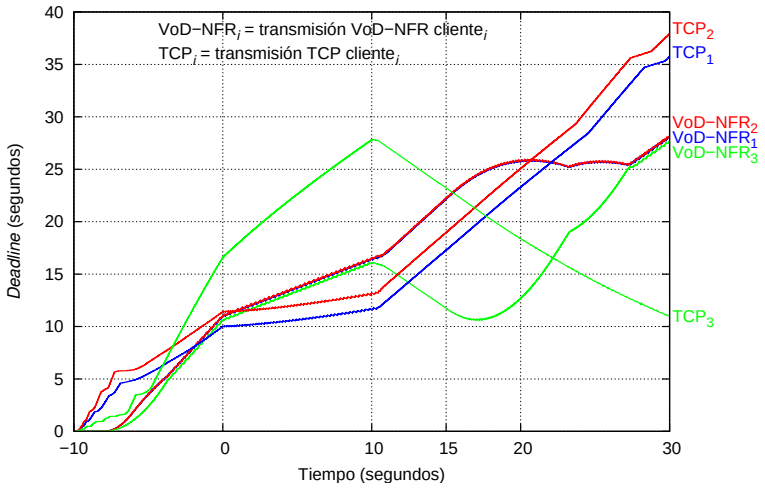
por el cliente *i*, mediante la aplicación de flujos RAP y el sistema VoD-NFR, respectivamente. En la figura 5.16 (b) se observa, con mayor detalle, la reacción de VoD-NFR y de TCP, al momento de producirse la congestión en el camino de comunicación con el cliente 3. El comportamiento del sistema VoD-NFR en esta simulación ya fue descripto en la comparación efectuada con la aplicación con comunicaciones RAP (ver sección 5.3.1).

A continuación se explica el comportamiento de la aplicación que envía el vídeo a los clientes mediante conexiones TCP. El cliente 3, a los 64 segundos de haber comenzado la reproducción, empieza a experimentar cortes en la visualización del vídeo. En el tiempo 174, el cliente 3 ha sufrido cortes en la reproducción por un total de 6 segundos. Desde este momento, comienza a recuperarse. Es decir, se requirieron 169 segundos para recuperarse de la congestión que duró solo 5 segundos. La presencia de los otros dos flujos, impiden que este flujo mejore su transmisión como se espera. No obstante, el incremento del flujo del cliente 3, es producido junto a una baja del flujo del cliente 1, y un leve aumento del flujo del cliente 2. Al igual que con el protocolo RAP, con TCP se aprecia una injusta distribución del ancho de banda entre las diferentes conexiones.

Si se compara el *deadline* promedio de los tres flujos al final de la simulación, el sistema VoD-NFR logra mejorar el rendimiento de la aplicación con conexiones TCP. VoD-NFR presenta un *deadline* promedio de 144,9 segundos $((145 + 144{,}5 +$



(a) Simulación completa



(b) Ampliación de la subfigura (a), que muestra el momento de congestión de la red

Figura 5.16: VoD-NFR frente a TCP

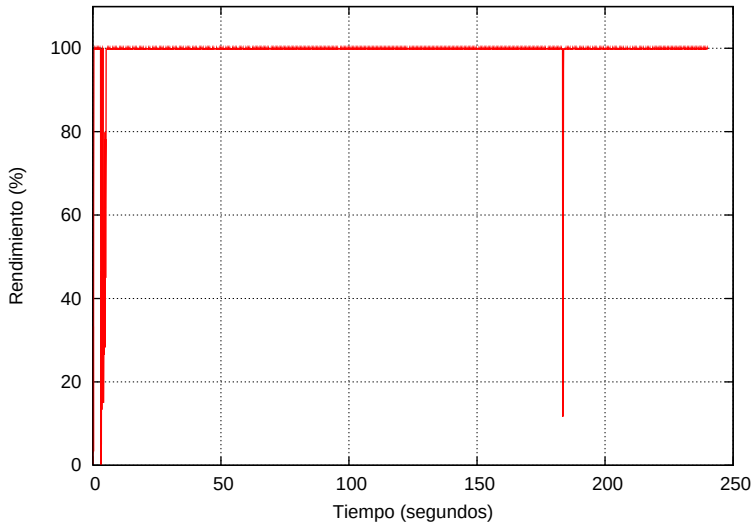


Figura 5.17: Ancho de banda utilizado del enlace entre el nodo *n0* y *n1*, correspondiente a la simulación de la aplicación con flujos TCP

145,1)/3), mientras que la aplicación con TCP presenta un *deadline* promedio de 130,2 segundos ((128 + 235,7 + 26,9)/3).

El mejor rendimiento del sistema VoD-NFR no es producto de un mayor uso del ancho de banda, ya que la aplicación con flujos TCP también aprovecha muy bien el ancho de banda disponible del enlace entre los nodos *n0* y *n1*. Puede verse el uso del ancho de banda del enlace comprendido entre los nodos *n0* y *n1*, para la simulación del sistema VoD-NFR y de la aplicación con flujos TCP, en las figuras 5.14 y 5.17, respectivamente. Por lo tanto, la ventaja en cuanto al mayor rendimiento promedio del sistema VoD-NFR encuentra explicación en el menor tamaño de los encabezados de datos, y en la regulación de la tasa de transmisión del servidor que evita descartes innecesarios de paquetes.

5.4. Tolerancia a fallos de VoD-NFR

Esta sección tiene como objetivo estudiar el comportamiento del sistema VoD-NFR ante fallos en la red, verificar sus mecanismos de detección, y analizar el proceso de recuperación. En la sección 5.4.1 se presenta la metodología y configuración del experimento, cuyos resultados se presentan en la sección 5.4.2.

5.4.1. Metodología y configuración del experimento

En este estudio de simulación con NS2, se ha utilizado el modelo GT-ITM [96] para generar una topología de red de 2 niveles con 8 dominios-*stub* y 1 dominio-tránsito. La red que forma el dominio-tránsito tiene 4 nodos interconectados mediante enlaces de 100 Mb/s. Cada dominio-*stub* está compuesto por enlaces de 100 Mb/s y se conecta con el dominio-tránsito mediante un enlace de la misma velocidad. Los retardos de propagación de los enlaces son generados automáticamente por GT-ITM y difieren unos de otros. Los clientes de vídeo se ubican en nodos conectados a los dominio-*stub* mediante enlaces de 10 Mb/s. Los servidores de vídeo también se ubican en nodos conectados a los dominio-*stub*, pero los enlaces son de 100 Mb/s. En la figura 5.18 se ilustra la topología generada, donde se observa cada nodo del dominio-tránsito (0–4) y los dominio-*stub* (4–43).

A partir de la infraestructura de red descripta, se asumen 25 clientes (nodos "*C*") que solicitan el mismo vídeo a alguno de los dos servidores (nodos "*S*") disponibles. Cada uno de estos servidores dispone del vídeo correspondiente en su catálogo. El vídeo es VBR y está codificado en MPEG2, con 6 canales de audio en formato AC-3 (*Dolby Digital 5.1*). La tasa de bits de este vídeo es de 2,61 Mb/s.

Cada cliente tiene una lista de servidores que determina el orden de preferencia a la hora de seleccionar uno de ellos para que le preste servicio. Este orden se basa en el principio de localidad, donde los servidores más próximos se prefieren ante los más lejanos. De acuerdo a este principio, se ha definido una zona de localidad para cada servidor, donde las dos zonas cubren la totalidad de la topología. Un cliente tiene como primera opción al servidor de su zona y como segunda opción al otro servidor. De esta forma, los clientes conectados a cualquiera de los nodos de los cuatro dominio-*stub*, que tienen enlaces directos con los nodos 0 y 2 del dominio-tránsito, tienen como primer servidor a aquel que está conectado al nodo 28 y como segundo al conectado al nodo 39. Los clientes conectados a alguno de los tres restantes dominio-*stub*, que tienen enlaces directos a los nodos 1 y 3 del dominio-tránsito, tienen como primer servidor a aquel que está conectado al nodo 39 y como segundo al conectado al nodo 28.

Todos los clientes solicitan el vídeo en el tiempo 0. En el LCS se ha definido una política de *prefetch* en la cual los clientes comienzan a reproducir el vídeo recién cuando disponen de 5 segundos de vídeo en *buffer*. Algunos clientes comienzan la reproducción antes que otros, dependiendo del estado de la comunicación que cada cliente tenga con su correspondiente servidor. La simulación se detiene a los 70 segundos. A los 10 segundos de comenzar la simulación, se induce la caída del enlace 0–2, el cual se vuelve operativo 20 segundos más tarde, es decir, en el tiempo 30. Se supone que la red tiene un encaminamiento estático, o un encaminamiento di-

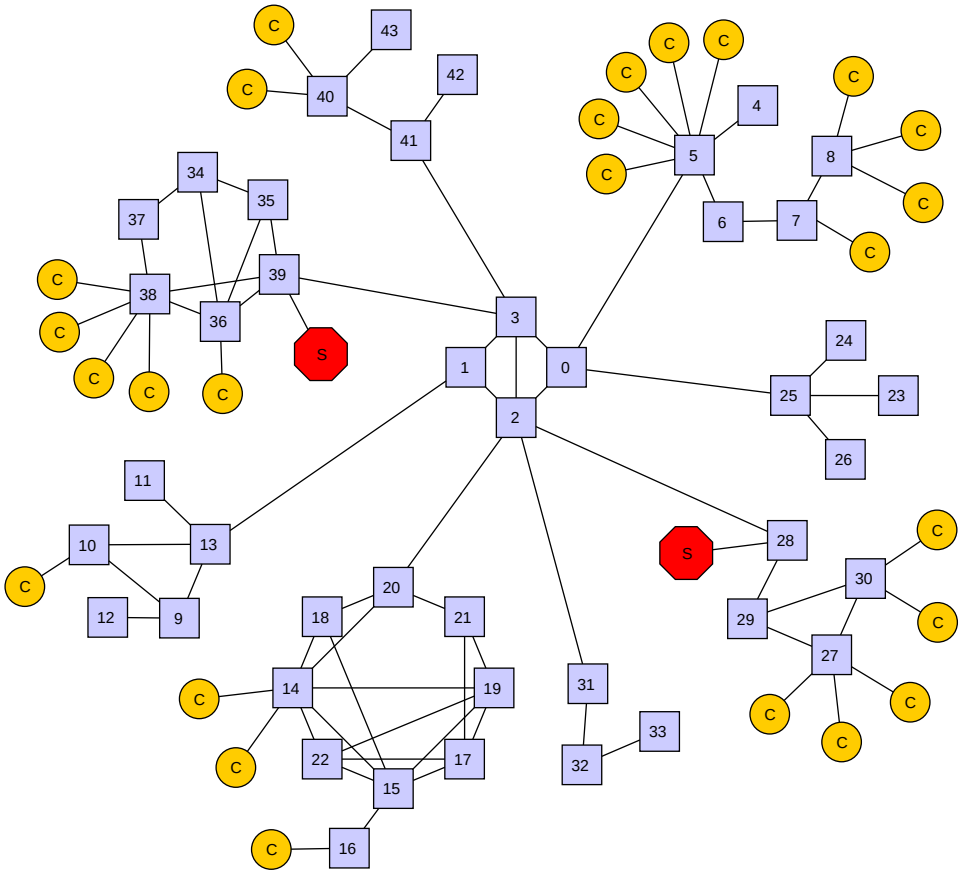


Figura 5.18: Topología GT-ITM utilizada para evaluar el comportamiento del sistema VoD-NFR

námico con una mala configuración de sus rutas que impiden redirigir los paquetes a su destino.

El propósito de producir este fallo en la red es el análisis del comportamiento del mecanismo de protección de los clientes (expuesto en la sección 2.7.1). Este mecanismo deberá ponerse en marcha ya que el mecanismo de detección de fallos del servidor (sección 2.7.2) no podrá actuar a causa de la incomunicación entre ambas entidades (cliente y servidor). El mecanismo de protección se ha configurado con un envío de un *ping* cada tres segundos.

En el tiempo 40, se produce una corriente UDP de 80 Mb/s que reduce a un 20 % la disponibilidad del ancho de banda del enlace 0–3. El objetivo de este fallo es hacer actuar el mecanismo de detección de fallos del servidor para poder analizar su comportamiento.

5.4.2. Resultados del experimento

En esta sección se presentan los resultados más representativos que permiten apreciar el funcionamiento y capacidad de recuperación de fallos del sistema VoD-NFR.

En el contexto de esta simulación, 9 clientes han migrado, 6 de ellos han efectuado dos migraciones y los restantes solo una. Estos 9 clientes son aquellos que se encuentran conectados a los nodos 5, 7 y 8, mientras que el resto de los clientes nunca ha migrado. Se han seleccionado tres clientes para analizar sus comportamientos: “cliente 1” es un cliente que nunca ha migrado y se encuentra conectado al nodo 38; “cliente 2” es un cliente que ha migrado una vez y está conectado al nodo 5; y “cliente 3” es un cliente que ha migrado dos veces y se encuentra conectado al nodo 5.

En la figura 5.19 (a), (b) y (c) se muestra el rendimiento supuesto por el servidor (curva suavizada exponencialmente), el *deadline* y la cantidad de datos recibidos para los tres clientes analizados, respectivamente. Estas tres gráficas pueden ser comprendidas al relacionarlas entre ellas.

En la gráfica del *deadline* puede observarse un descenso del mismo en el tiempo 10 para el cliente 2 y 3; este descenso se debe a que el cliente está consumiendo media pero no están ingresando nuevos datos del vídeo al *buffer* a causa de la caída del enlace 0–2. Obsérvese en la gráfica (a) cómo la curva suavizada exponencialmente de los clientes 2 y 3 convergen a un rendimiento de 0 Mb/s.

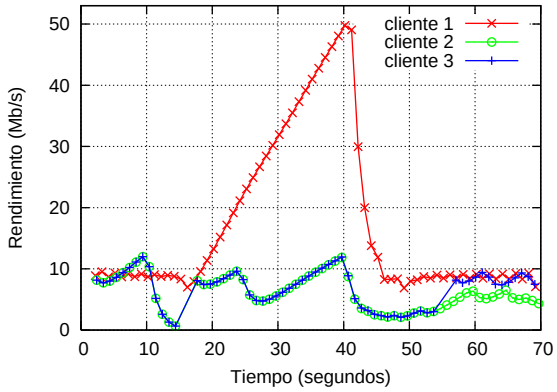
En el tiempo 15,25, los clientes 2 y 3 no reciben el *pong* de parte del servidor y migran en consecuencia; lo que significa que se ha tardado poco más de 5 segundos

para detectar la caída del enlace. Este valor es correcto ya que dado un envío de un *ping* cada 3 segundos, la detección estará acotada entre 3 y 6 segundos. Para verificar esto se analiza el caso del cliente 2. El cliente ha enviado un *ping* en el tiempo 9,25 y, antes de que se caiga el enlace en el tiempo 10, recibe el *pong*. El siguiente *ping* lo envía en el tiempo 12,25 y espera por el *pong* hasta el segundo 15,25; al no haber recibido el *pong*, el cliente decide migrar. Los 9 clientes antes mencionados (entre ellos los clientes 2 y 3) migran al servidor alternativo conectado al nodo 39. Mientras tanto, el cliente 1 no ha sufrido inconvenientes porque la caída del enlace no lo ha afectado y, en el segundo 15, ya tiene acumulado en su *buffer* poco más de 32 segundos de vídeo que aún tiene por visualizar (gráfica (b)).

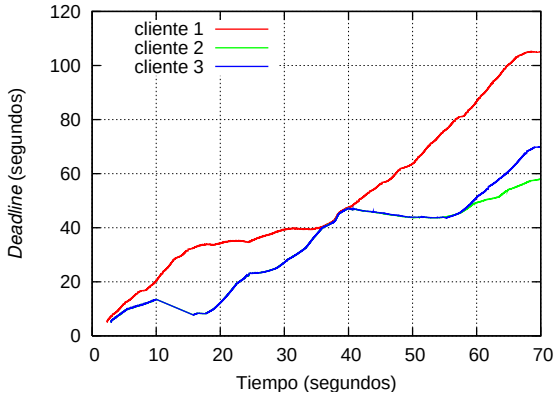
Poco antes del tiempo 16, el servidor conectado al nodo 39 ya está sirviendo a los 9 nodos migrados. Como se observa en la gráfica (a), a partir del tiempo 16 la comunicación de los clientes 2 y 3 ha mejorado nuevamente. En la gráfica (b) se puede observar cómo el servidor equilibra el estado de sus clientes, aumentando el envío al cliente 1 y 2 (y los restantes 7 clientes) y disminuyendo la transmisión de datos al cliente 3 (y el resto de sus clientes que tienen un *deadline* mayor), lo cual puede verse en la gráfica (c). En el tiempo 36, el servidor logra que los clientes 1, 2 y 3 tengan prácticamente el mismo *deadline*.

Note que entre el segundo 16 y el 40, el rendimiento supuesto para el cliente 1 se dispara hasta llegar a los 50 Mb/s; aunque esto parezca un error, realmente no lo es de acuerdo a la información que el servidor obtiene de la red. Veamos por qué sucede esto. El servidor disminuye el envío de datos, dentro del intervalo mencionado, a 3,85 Mb/s en promedio, es decir, a menos de la mitad de lo que soporta la conexión del cliente. Esta disminución del tráfico transmitido al cliente puede observarse en la gráfica (c), donde la curva presenta una pendiente bastante menor entre el segundo 16 y el 40 con respecto al resto del tiempo que dura la experimentación. Entonces, si en un determinado segundo el servidor transmite 3 Mbit a este cliente, pero lo hace a una tasa de envío de 30 Mb/s, los paquetes se transmitirán en ráfagas a una tasa mayor a la soportada por el enlace del cliente (10 Mb/s). Sin embargo, como se ha transmitido una cantidad muy pequeña de información, la cola del encaminador 38 (al cual se conecta el cliente 1) puede soportar la ráfaga sin descartar ningún paquete. Como no se pierden paquetes, el servidor entonces decrementa el IPG, o lo que es lo mismo, supone una tasa de transmisión mayor.

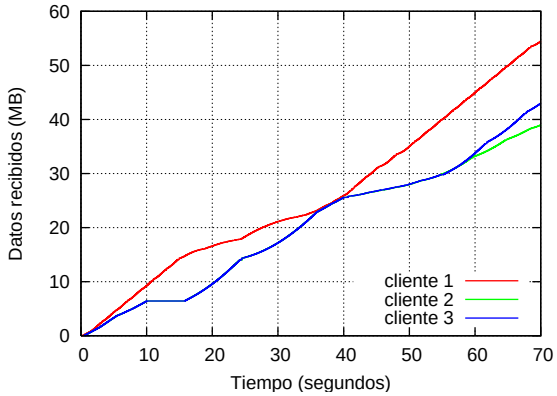
En el segundo 40, aparece la corriente UDP que, junto con la carga que el sistema VoD-NFR impone en la red (por las transmisiones de los vídeos), genera una congestión en el enlace 0–3. Esta corriente UDP, sin ningún tipo de control de congestión, se apropia del 80 % de la capacidad del enlace, produciendo una disminución del rendimiento de los clientes 2 y 3 que puede apreciarse en la gráfica



(a) Rendimiento supuesto por el servidor



(b) *Deadline*



(c) Cantidad de datos recibidos

Figura 5.19: Resultado de la simulación para los tres clientes representativos

(a). Aunque parezca extraño, el rendimiento del cliente 1 también baja. En realidad, lo que está sucediendo es que se están enviando más datos al cliente 1 que en la etapa anterior (ver gráfica (c)), con lo cual las colas de los encaminadores ya no pueden soportar la tasa de envío supuesta de 50 Mb/s. De esta manera, la tasa supuesta converge rápidamente a la tasa de envío real de 10 Mb/s.

Como la tasa promedio del vídeo es de 2,61 Mb/s, los 9 clientes que están utilizando el enlace 0–3 requieren que este disponga mínimamente de 23,49 Mb/s. Dado que los 20 Mb/s disponibles en el enlace son insuficientes, será necesario que el servidor migre algunos clientes. En este caso, aunque sólo era suficiente migrar dos clientes, el servidor ha migrado 6 de los 9 clientes, incluyendo al cliente 3 pero excluyendo al cliente 2. Esta situación se da porque el servidor desconoce que los 6 (o mejor dicho, 9) clientes estaban siendo afectados por el mismo caso de congestión. Estos 6 clientes migran a su siguiente servidor de la lista (circular), con lo cual vuelven a migrar al servidor inicial, conectado al nodo 28; la migración es posible debido a que el enlace 0–2 ya se encuentra operativo. En la gráfica (b) puede observarse que el cliente 3 vuelve a incrementar su *deadline* y, el cliente 2, que no ha migrado, también incrementa su *deadline* pero más suavemente porque los 20 Mb/s disponibles del enlace 0–2 son repartidos entre tres clientes.

En la figura 5.20 se muestra el rendimiento o utilización de los enlaces que conectan a los servidores con la red, donde el servidor 1 es aquel conectado al nodo 28 y el servidor 2 es el conectado al nodo 39. Por un lado, el servidor 1 hasta el tiempo 10 utiliza el 100 % de la conexión de red para transmitir los vídeos a sus 17 clientes. En el tiempo 10 se pierde la conectividad con 9 clientes y baja el aprovechamiento del enlace debido a que quedan solo 8 clientes a los que servir. Recién se vuelve a utilizar el 100 % del enlace cuando retornan 6 de los 9 clientes que habían migrado al servidor 2. Por otro lado, el servidor 2, que inicialmente tiene asignado ocho clientes, recibe 9 clientes más en el tiempo 16 que producen un aumento de la carga del servidor y una utilización del 100 % de la conexión de red para atender a sus 17 clientes. En el tiempo 40, la congestión que afecta a los nuevos clientes disminuye el uso del enlace, que aún disminuye más cuando 6 de sus clientes migran al servidor 1.

Los resultados de esta sección permiten concluir que el sistema VoD-NFR es capaz de detectar los fallos producidos en la red con la suficiente antelación, permitiendo tomar rápidas medidas para evitar cortes en la visualización de los vídeos. Naturalmente, la recuperación de fallos de red está supeditada a la existencia de una vía alternativa de comunicación, desde alguno de los servidores al cliente, que evada el punto de falla.

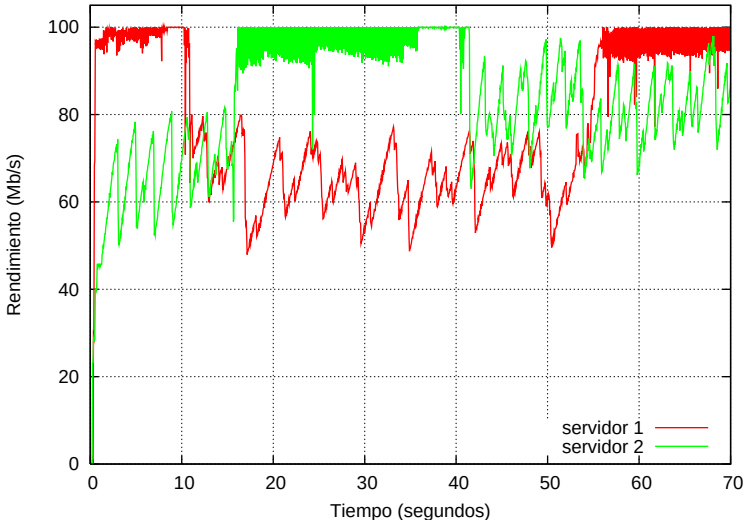


Figura 5.20: Utilización de la conexión de red de cada servidor

Capítulo 6

Conclusiones y líneas abiertas

En el transcurso de esta tesis doctoral, se ha presentado el nuevo sistema VoD-NFR, del que se ha descripto: su arquitectura y funcionalidad en el capítulo 2, su componente más crítico en el capítulo 3, su diseño en el capítulo 4, y los resultados de los estudios experimentales en el capítulo 5. Para finalizar, se exponen las conclusiones y aportaciones en relación a las hipótesis propuestas, y se hace referencia a posibles futuras líneas de investigación.

A medida que las redes digitales van ampliando sus servicios, ganando extensión y calidad de transferencia, nuevos productos se van presentando a los usuarios. En 1998 era prácticamente impensable que Internet llegaría en algún momento a permitir un ancho de banda y una velocidad tal como para poder transmitir señales audiovisuales de mediana o alta calidad. Diez años después, muchos países han llegado a tal meta, y algunos la han superado.

Los sistemas de vídeo bajo demanda a gran escala (LVoD, *Large-Scale Video-on-Demand*) brindan un servicio de visualización de vídeos a una gran cantidad de usuarios dispersos geográficamente, permitiendo numerosas opciones, dentro de las que se destaca la interactividad y personalizado de la programación ofrecida a los usuarios (servicio conocido como vídeo bajo demanda verdadero —T-VoD, *True Video-on-Demand*—). Esto significa que es posible elegir, a cualquier hora, la película que queremos ver como si fuera reproducida en nuestro dispositivo de vídeo particular, interactuando con el emisor enviándole nuestras preferencias de visualización (pausar, reanudar, avanzar rápidamente, etc).

Simplemente con mencionar que la famosa y multinacional cadena de videoclubs Blockbuster ha cerrado en España en el año 2006, es indudable que la tendencia es hacia el VoD por Internet. Sin embargo, hasta la actualidad, la mayoría de los sistemas de VoD fueron diseñados para trabajar en redes locales y dedicadas o que permiten hacer reserva de recursos. Sus arquitecturas no son aplicables a entornos LVoD, fundamentalmente debido a los nuevos requerimientos de escalabilidad, costo del sistema, y tolerancia a fallos. Cuando el entorno de red pasa de ser de una red local a una red como Internet, aumenta la probabilidad de fallos, disminuye el ancho de banda, y se pierde la calidad y clasificación de servicios la que es reemplazada por un modelo de servicio de “mejor esfuerzo”.

Es significativo que ninguno de los autores consultados haya tratado en profundidad el tratamiento de los fallos de la red para garantizar la QoS de un sistema de LVoD. La mayoría centran sus esfuerzos en el modelo de servicio y dejan de lado parámetros de suma importancia como son los de la red, de gran influencia en el rendimiento de estos sistemas. A partir de esta situación, se ha planteado el objetivo de esta investigación, que es la propuesta de una nueva arquitectura de un sistema de LVoD distribuido, que permita ofrecer un servicio de T-VoD, con comunicaciones unitransmisión sobre una red sin calidad de servicio como Internet. Este sistema, denominado VoD-NFR (*Video-on-Demand with Network Fault Recovery*), tiene como fin garantizar, ante fallos de la red y caídas de servidores, la entrega del contenido multimedia a los clientes por una red como Internet, sin degradar su calidad y sin sufrir interrupciones durante su visualización.

El objetivo de este trabajo se ha dividido en tres subobjetivos que involucran el diseño y desarrollo de:

1. Un planificador del tráfico de red (NTS), capaz de: 1) adaptarse a los estados de congestión de la red de una manera TCP-*Friendly*, 2) generar información del estado de las comunicaciones con los clientes, y 3) extremar el ahorro de recursos para soportar una elevada carga de trabajo.

2. Un planificador de canales lógicos (LCS), capaz de: 1) utilizar vídeos VBR, 2) adaptarse dinámicamente al ancho de banda disponible de las comunicaciones, 3) no degradar la calidad del vídeo para adaptarse a un ancho de banda menor al requerido por el vídeo, y 4) distribuir equitativamente el vídeo a los clientes priorizando a aquellos con mayor necesidad de media.

3. Un módulo de garantía de la calidad de servicio (QoSA), capaz de: 1) detectar inconvenientes en la comunicación entre servidores y clientes, y 2) con la debida antelación, tomar medidas que permitan continuar prestando un servicio sin pérdida de calidad.

A continuación, se presentan las conclusiones alcanzadas a partir de los resultados obtenidos, los artículos publicados que han surgido de este estudio, y las líneas abiertas que formarán parte de futuras investigaciones.

6.1. Conclusiones

- Se ha presentado el sistema VoD-NFR (capítulos 2, 3 y 4), de arquitectura totalmente distribuida, que permite ofrecer un servicio de T-VoD con comunicaciones unitransmisión sobre la red Internet, y que garantiza, ante fallos

de la red o caídas de servidores, la entrega del contenido multimedia a los clientes sin disminuir la calidad de los mismos y sin sufrir interrupciones durante su visualización (ver pruebas en capítulo 5).

- Se desarrolló un protocolo de control de congestión TCP-*Friendly*, que hemos denominado *Enhanced Rate Adaptation Protocol* (ERAP, sección 3.2), que genera información del estado de las comunicaciones con los clientes, y que minimiza el consumo de recursos permitiendo soportar la alta carga de trabajo de un sistema de VoD (sección 5.2).

- El módulo NTS se ha diseñado y desarrollado según los requerimientos del sistema VoD-NFR (sección 2.1.1, 3.1 y 4.4.4), y se ha evaluado extensamente a través de simulaciones, validadas mediante experimentación real (sección 5.1.2 y 5.1.3). La implementación real del NTS (sección 3.4) soluciona el problema de precisión de los temporizadores en un sistema operativo de propósito general, cuyos resultados en pruebas de estrés (sección 5.1.4) demuestran su viabilidad para ser incluido en un sistema de LVoD.

- El planificador de eventos de tiempo, desarrollado para el NTS, se ha integrado en un *middleware* de planificación de eventos (sección 4.1) que permite a una aplicación independizarse, en relación a la gestión de eventos, del entorno para el cual se desarrolla, ya sea este real o de simulación (NS2).

- Se ha diseñado y desarrollado el LCS (sección 2.1.1, 2.5 y 4.4.5) y se ha demostrando su capacidad de planificar la entrega de los vídeos a los clientes de manera justa, priorizando a los clientes con mayores necesidades de datos, logro que no sería tal sin el conocimiento del estado de la red provisto por el NTS. Esta integración del control de flujo del nivel de aplicación (LCS) con el control de flujo del nivel de red (NTS) permite al sistema VoD-NFR adaptarse a la variabilidad del ancho de banda de las comunicaciones. El esquema del LCS consigue no degradar la calidad del vídeo cuando el ancho de banda disminuye por sobre el requerido, aprovechando los momentos en que la red no está congestionada (sección 5.3).

- Se encontró evidencia de que la falta de control de flujo al nivel de aplicación influye en el funcionamiento de los protocolos de transporte (por lo menos de RAP y TCP), produciendo una distribución desbalanceada del ancho de banda entre las diferentes comunicaciones (sección 5.3.1 y 5.3.2); el flujo de una comunicación, al estar en competencia con flujos de otras comunicaciones, consumen considerable tiempo para adaptarse a las condiciones de la red. Esto afirma aún más la utilidad del LCS, que permite aumentar el ren-

dimiento individual de los flujos de las comunicaciones, acortando el tiempo de adaptación de los flujos a las condiciones de la red.

- Se ha diseñado un esquema tolerante a fallos (sección 2.7), inmerso en el sistema VoD-NFR, que utiliza dos mecanismos que colaboran para detectar fallos de la red o caídas de servidores, y un mecanismo de recuperación basado en la migración de servicios. Este esquema de tolerancia a fallos solapa la etapa de detección del fallo con la de recuperación, produciendo una ganancia de tiempo determinante que, junto con la detección temprana de la disminución del ancho de banda de las comunicaciones, permite mantener la QoS evitando que el cliente sufra cortes en la visualización del vídeo (ver pruebas en sección 5.4).

- El sistema VoD-NFR ha sido implementado en entorno real y de simulación (capítulo 4); en particular, la simulación ha permitido evaluar el sistema más extensivamente y a un bajo coste. Los resultados alcanzados (capítulo 5) muestran la aptitud del sistema para cumplir con sus objetivos, motivando la continuidad de la presente línea de investigación.

- Se desarrolló un *middleware* de red (sección 4.2) que presenta una interfaz de red común e independiente del medio que hace efectiva la comunicación: la red o el simulador NS2. Este *middleware* incrementa la funcionalidad del NS2 para permitir la transferencia de datos de usuario reales en el protocolo TCP, además de permitir generar conexiones TCP dinámicamente. La solución que se ha diseñado para estos dos problemas no modifica en absoluto el código TCP del simulador, y se basa en adicionar nuevas clases que dan soporte a la funcionalidad faltante.

6.2. Artículos publicados

El desarrollo del sistema VoD-NFR está inmerso en un grupo de investigación de VoD, en el cual se ha dedicado un gran esfuerzo a esquemas P2P, del cual ha surgido la siguiente publicación:

- Leandro Souza, Xiaoyuan Yang, Javier Balladini, Ana Ripoll, Fernando Cores. "Improving VoD P2P delivery efficiency over Internet using idle peers". International Conference on Signal Processing and Multimedia Applications (SIGMAP 2007), Barcelona, Spain. July 2007. ISBN: 978-989-8111-13-5.

El planificador del tráfico de red ha dado origen a las siguientes tres publicaciones, donde la primera ha sido premiada por el comité de la conferencia con la mención de “mejor artículo”:

- Javier Balladini, Leandro Souza, Remo Suppi. "A Network Traffic Scheduler for a VoD Server on the Internet". International Conference on Signal Processing and Multimedia Applications (SIGMAP 2006), Setúbal, Portugal. August 2006. ISBN: 972-8865-64-3, pp. 260–268. *Award Paper*.
- Javier Balladini, Leandro Souza, Remo Suppi. “A Stream Transport Module for a VoD Server on the Internet”. XVII Jornadas de Paralelismo, Albacete, España. Septiembre 2006. ISBN: 84-690-05551-0, páginas 513–518.
- Javier Balladini, Leandro Souza, Remo Suppi. "A Network Scheduler for an Adaptive VoD Server". E-Business and Telecommunication Networks, CCIS 9, pp. 237–251, 2008. Springer-Verlag Berlin Heidelberg 2008. ISBN-10: 3-540-70759-X.

Además, se ha escrito el siguiente artículo:

- Javier Balladini, Remo Suppi. “A real implementation of a Network Scheduler for a VoD system on the Internet”.

donde se presenta la implementación real del NTS, y que se encuentra en proceso de evaluación para ser publicado en la revista Multimedia Systems, publicada por Springer-Verlag y patrocinada por ACM SIGMM.

El planificador de canales lógicos ha dado origen a la siguiente publicación:

- Javier Balladini, Leandro Souza, Remo Suppi. "Un Planificador de Canales Lógicos para un Servidor de VoD en Internet". XII Congreso Argentino de Ciencias de la Computación (CACIC 2006), Potrero de los Funes, San Luis, Argentina. Octubre de 2006.

Finalmente, la tolerancia a fallos del sistema VoD-NFR se encuentra actualmente en proceso de publicación, para ser enviado a la revista Multimedia Systems.

6.3. Líneas abiertas

En base a la experiencia obtenida en el desarrollo de esta tesis, han surgido nuevas líneas de actuación que pueden contribuir a completar el nivel de refinamiento y la amplitud del problema de la tolerancia a fallos de red.

Hay fundamentalmente cuatro temas pendientes que requieren ser investigados:

Estudio de un caso real a gran escala que sea indicativo del grado de aprovechamiento que puede obtenerse del sistema VoD-NFR

En el presente trabajo, se ha demostrado que VoD-NFR es capaz de adaptarse a los problemas de las redes mediante la detección y consiguiente migración del servicio. Sin embargo, la eficacia que tenga VoD-NFR de recuperarse ante fallos depende de cuatro factores: la infraestructura de red que se disponga, la distribución de los servidores en ella, la distribución de los contenidos multimedia en los servidores, y la política de selección de servidores alternativos aplicada cuando el cliente debe abandonar el servicio prestado por su servidor principal. Por más que se analicen los últimos tres factores, nada podrá hacerse si la infraestructura de red existente presenta pocos caminos alternativos para transmitir los contenidos multimedia a los clientes. Es decir, si desde cualquier servidor para llegar a un cliente no puede evitarse la sección de la red que está congestionada, cualquier esfuerzo es totalmente inútil.

Por lo tanto, sería conveniente estudiar un escenario real donde puedan medirse las posibilidades que ofrezca la red para implantar un sistema como VoD-NFR, para luego poder evaluar el grado de aprovechamiento que puede obtenerse mediante su uso.

Detección de clientes afectados por el mismo caso de congestión

De acuerdo a la experimentación presentada en esta tesis, se ha observado que es posible que múltiples clientes afectados por un mismo caso de congestión migren todos juntos, sin tener en cuenta que tal vez es suficiente con migrar solamente algunos y aprovechar el ancho de banda liberado para servir a los demás clientes. Es necesario, entonces, estudiar y evaluar la necesidad de detectar estos casos y, si fuese necesario, proponer alguna solución.

Algoritmo de categorización de mejores servidores para los clientes

Cuando un cliente tiene que abandonar el servicio prestado por un servidor para continuar ser servido desde otro, es necesario decidir cuál será la mejor alternativa de todos los restantes servidores del sistema. Esta selección estará afectada por la distancia al cliente (cantidad de saltos y retardo) y la calidad de la conexión. Aunque inicialmente se haga un análisis aproximado de categorización de servidores para cada cliente (mediante una pequeña prueba de conectividad), indefectiblemente sería conveniente actualizar o modificar esta categorización de acuerdo a la enorme cantidad de información recolectada durante la prestación del servicio a los clientes.

Desarrollo del NTS con soporte de multitransmisión

El sistema VoD-NFR se ha desarrollado para su uso con comunicaciones unitransmisión. Sin embargo, en ciertos entornos acotados es posible disponer de comunicaciones multitransmisión IP que podrían ser aprovechadas, o utilizar otras alternativas como son el esquema *Overlay Network* y ALM (*Application Level Multicast*). Como ejemplo, se pueden mencionar dos arquitecturas basadas en *overlay networks*, OMNI [97] y Scattercast [98], mientras que dentro de los esquemas ALM pueden citarse a ALMI [99] y Emma [100].

Poner en marcha aspectos complementarios del proyecto

Con el fin de efectuar el proceso de despliegue del sistema VoD-NFR como un sistema final y de producción, es necesario complementarlo con el desarrollo del componente *Boletería* y de reproductores clientes. El componente *Boletería* es responsable de ofrecer un mecanismo de selección del vídeo y compra, así como de proveer un listado de los servidores candidatos para atender a un cliente determinado. Se debe establecer una política para generar y actualizar la lista de candidatos. La arquitectura de este componente podría ser distribuida para evitar que se vuelva un cuello de botella del sistema. En cuanto a los reproductores de vídeo, es conveniente realizar implementaciones para su uso en diferentes plataformas.

Bibliografía

[1] Gerald Lui-Kwan. In-flight entertainment: The sky's the limit. *Computer*, 33(10):98–101, 2000.

[2] Huadong Ma and Kang G. Shin. Multicast video-on-demand services. *SIGCOMM Comput. Commun. Rev.*, 32(1):31–43, 2002.

[3] Jean-Paul Nussbaumer, Baiju V. Patel, Frank Schaffa, and James P. G. Sterbenz. Networking requirements for interactive video on demand. *IEEE Journal of Selected Areas in Communications*, 13(5):779–787, 1995.

[4] Andrew Tanenbaum. *Computer Networks*. Prentice Hall PTR, fourth edition, 2002.

[5] James F. Kurose and Keith W. Ross. *Computer Networking: A top down approach featuring the Internet*. Addison-Wesley, third edition, 2004.

[6] IETF. The MBONE deployment working group (mboned). *URL: http://www.ietf.org/html.charters/mboned-charter.html. Mayo 2008.*

[7] Kevin C. Almeroth. A long-term analysis of growth and usage patterns in the multicast backbone (MBone). In *INFOCOM (2)*, pages 824–833, 2000.

[8] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications, RFC 3550, Network Working Group, July 2003.

[9] Dinkar Sitaram and Asit Dan. *Multimedia servers: applications, environments, and design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[10] Miguel Barreiro, Víctor M. Gulías, Juan J. Sánchez, and J. Santiago Jorge. The tertiary level in a functional cluster-based hierarchical vod server. In *Computer Aided Systems Theory - EUROCAST 2001-Revised Papers*, pages 540–554, London, UK, 2001. Springer-Verlag.

[11] Chor Ping Low, Hongtao Yu, J.M. Ng, Qingping Lin, and Y. Atif. An efficient algorithm for the video server selection problem. *Global Telecommunications Conference, 2000. GLOBECOM '00. IEEE*, 3:1329–1333 vol.3, 2000.

[12] Kun-Lung Wu, Philip S. Yu, and Joel L. Wolf. Segment-based proxy caching of multimedia streams. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 36–44, New York, NY, USA, 2001. ACM.

[13] R. Rejaie, M. Handley, H. Yu, and D. Estrin. Proxy caching mechanism for multimedia playback streams in the internet. *Proc. of the 4th International Web Caching Workshop*, 1999.

[14] Zhourong Miao and A. Ortega. Scalable proxy caching of video under storage constraints. *IEEE Journal on Selected Areas in Communications*, 20(7):1315–1327, Sep 2002.

[15] D. Zeinalipour-Yazti, V. Kalogeraki, and D. Gunopulos. pfusion: A p2p architecture for internet-scale content-based search and retrieval. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):804–817, June 2007.

[16] M. Hefeeda, A. Habib, D. Xu, B. Bhargava, and B. Botev. Collectcast: A peer-to-peer service for media streaming. *ACM/Springer Multimedia Systems Journal*, October 2003.

[17] Mohamed Hefeeda, Ahsan Habib, Boyan Botev, Dongyan Xu, and Bharat Bhargava. Promise: peer-to-peer media streaming using collectcast. In *MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia*, pages 45–54, New York, NY, USA, 2003. ACM.

[18] T.T. Do, K.A. Hua, and M.A. Tantaoui. P2vod: providing fault tolerant video-on-demand streaming in peer-to-peer environment. *Communications, 2004 IEEE International Conference on*, 3:1467–1472 Vol.3, 20-24 June 2004.

[19] Yang Guo, Kyoungwon Suh, Jim Kurose, and Don Towsley. P2cast: peer-to-peer patching scheme for vod service. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 301–309, New York, NY, USA, 2003. ACM.

[20] D.A. Tran, K.A. Hua, and T.T. Do. A peer-to-peer architecture for media streaming. *IEEE Journal on Selected Areas in Communications*, 22(1):121–133, Jan. 2004.

[21] Yinqing Zhao and C. C. Jay Kuo. Video server scheduling using random early request migration. *Multimedia Systems*, 10(4):302–316, 2005.

[22] Dusit Niyato. Load balancing algorithms for internet video and audio server. In *ICON '01: Proceedings of the 9th IEEE International Conference on Networks*, page 76, Washington, DC, USA, 2001. IEEE Computer Society.

[23] Yin-Fu Huang and Chih-Chiang Fang. Load balancing for clusters of vod servers. *Inf. Sci. Inf. Comput. Sci.*, 164(1-4):113–138, 2004.

[24] K.C. Almeroth. Adaptive workload-dependent scheduling for large-scale content delivery systems. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3):426–439, Mar 2001.

[25] J.Y.B. Lee. Concurrent push-a scheduling algorithm for push-based parallel video servers. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(3):467–477, Apr 1999.

[26] Apostolos Papagiannis, Dimitrios Lioupis, and Stylianos Egglezos. Design and implementation of a low-cost clustered video server using a network of personal computers. In *MSE '02: Proceedings of the Fourth IEEE International Symposium on Multimedia Software Engineering*, page 363, Washington, DC, USA, 2002. IEEE Computer Society.

[27] A. Dan, D. Sitaram, P. Shahabuddin, and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In *MULTIMEDIA '94: Proceedings of the second ACM international conference on Multimedia*, pages 15–23, New York, NY, USA, 1994. ACM.

[28] Simon Sheu and Kien A. Hua. Virtual batching: A new scheduling technique for video-on-demand servers. In *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 481–490. World Scientific Press, 1997.

[29] Kien A. Hua, Ying Cai, and Simon Sheu. Patching: a multicast technique for true video-on-demand services. In *MULTIMEDIA '98: Proceedings of the sixth ACM international conference on Multimedia*, pages 191–200, New York, NY, USA, 1998. ACM.

[30] Ying Cai and Kien A. Hua. Sharing multicast videos using patching streams. *Multimedia Tools Appl.*, 21(2):125–146, 2003.

[31] Lixin Gao and Don Towsley. Supplying instantaneous video-on-demand services using controlled multicast. In *ICMCS '99: Proceedings of the IEEE*

International Conference on Multimedia Computing and Systems Volume II-Volume 2, page 117, Washington, DC, USA, 1999. IEEE Computer Society.

[32] Kevin C. Almeroth and Mostafa H. Ammar. The use of multicast delivery to provide a scalable and interactive video-on-demand service. *IEEE Journal of Selected Areas in Communications*, 14(6):1110–1122, 1996.

[33] S. Viswanathan and T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. *Multimedia Systems*, 4(4):197–208, 1996.

[34] Aggarwal C. C., Wolf J. L., and Yu P. S. A permutation-based pyramid broadcasting scheme for video-on-demand systems. In *ICMCS '96: Proceedings of the 1996 International Conference on Multimedia Computing and Systems (ICMCS '96)*, page 0118, Washington, DC, USA, 1996. IEEE Computer Society.

[35] Kien A. Hua and Simon Sheu. Skyscraper broadcasting: a new broadcasting scheme for metropolitan video-on-demand systems. *SIGCOMM Comput. Commun. Rev.*, 27(4):89–100, 1997.

[36] Li-Shen Juhn and Li-Ming Tseng. Fast data broadcasting and receiving scheme for popular video service. *IEEE Transactions on Broadcasting*, 44(1):100–105, Mar 1998.

[37] Pâris J. F., Carter S. W., and Long D. E. A hybrid broadcasting protocol for video-on-demand. *Proceedings of the 2nd International Conference on ATM*, pages 132–139, 1999.

[38] Juhn L. and Tseng L. Harmonic broadcasting protocols for video-on-demand service. *IEEE Transactions on Broadcasting*, pages 268–271, 1997.

[39] Regant Y. S. Hung, H. F. Ting, and H. F. Ting. An optimal broadcasting protocol for mobile video-on-demand. In *CATS '07: Proceedings of the thirteenth Australasian symposium on Theory of computing*, pages 79–84, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[40] Zoe Antoniou and Ioannis Stavrakakis. Deadline credit scheduling policy for prerecorded sources. *Global Telecommunications Conference, 1999. GLOBECOM '99*, 1A:74–78, 1999.

[41] Zoe Antoniou and Ioannis Stavrakakis. An efficient deadline-credit-based transport scheme for prerecorded semisoft continuous media applications. *IEEE/ACM Trans. Netw.*, 10(5):630–643, 2002.

[42] Xin Wang and Henning Schulzrinne. Comparison of adaptive internet multimedia applications. *IEICE Transactions on Communications*, E82-B(6), June 1999.

[43] Andrew T. Campbell, Geoff Coulson, and David Hutchison. Transporting qos adaptive flows. *Multimedia Systems*, 6(3):167–178, 1998.

[44] Catherine Boutremans, Gianluca Iannaccone, and Christophe Diot. Impact of link failures on voip performance. In *NOSSDAV '02: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 63–71, New York, NY, USA, 2002. ACM.

[45] Sally Floyd and Kevin Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, 1999.

[46] S. Jacobs and A. Eleftheriadis. Real-time dynamic rate shaping and control for internet video applications. *IEEE First Workshop on Multimedia Signal Processing*, pages 558–563, 23–25 Jun 1997.

[47] Shanwei Cen, Calton Pu, and Jonathan Walpole. Flow and congestion control for internet media streaming applications. Technical report, 1997.

[48] Dorgham Sisalem and Henning Schulzrinne. The loss-delay based adjustment algorithm: A TCP-friendly adaptation scheme. In *Proceedings of NOSSDAV*, Cambridge, UK., 1998.

[49] Dorgham Sisalem and Adam Wolisz. LDA+: A TCP-friendly adaptation scheme for multimedia communication. In *IEEE International Conference on Multimedia and Expo (III)*, pages 1619–1622, 2000.

[50] Sally Floyd, Mark Handley, Jitendra Padhye, and Jorg Widmer. Equation-based congestion control for unicast applications. In *SIGCOMM 2000*, pages 43–56, Stockholm, Sweden, August 2000.

[51] M. Handley, J. Pahdye, S. Floyd, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol specification, RFC 3448, 2003.

[52] Reza Rejaie, Mark Handley, and Deborah Estrin. RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the internet. In *INFOCOM (3)*, pages 1337–1345, 1999.

[53] Reza Rejaie, Mark Handley, and Deborah Estrin. RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the internet. In *Technical report 98-681, CS-USC, August 1998*, 1998.

[54] R. Rejaie, M. Handley, and D. Estrin. Layered quality adaptation for internet video streaming. *IEEE Journal on Selected Areas in Communications*, 18(12):2530–2543, Dec 2000.

[55] M. Miyabayashi, N. Wakamiya, M. Murata, and H. Miyahara. Implementation of video transfer with tcp-friendly rate control. *Proceedings of International Technical Conference on Circuits/Systems, Computers and Communications*, pages 117–120, July 2000.

[56] Sridevi Palacharla, Ahmed Karmouch, and Samy A. Mahmoud. Design and implementation of a real-time multimedia presentation system using rtp. In *COMPSAC '97: Proceedings of the 21st International Computer Software and Applications Conference*, pages 376–381, Washington, DC, USA, 1997. IEEE Computer Society.

[57] Wu chang Feng, Ashvin Goel, Abdelmajid Bezzaz, Wu chi Feng, and Jonathan Walpole. Tcpivo: a high-performance packet replay engine. In *MoMeTools '03: Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pages 57–64, New York, NY, USA, 2003. ACM Press.

[58] Ajay Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, and Kevin Gibbs. Iperf, http://dast.nlanr.net/projects/iperf/, 2005.

[59] Aaron Turner and Matt Bing. tcpreplay, http://tcpreplay.synfin.net/trac/, 2004.

[60] Xin Wang and Ioannis Stavrakakis. Study of multiplexing for group-based quality of service delivery. In *Proceedings of the IFIP TC6 WG6.3/WG6.4 Fourth International Workshop on ATM Networks, Performance Modelling and Analysis, Volume 3*, pages 360–379, London, UK, UK, 1997. Chapman & Hall, Ltd.

[61] Simon S. Lam and Geoffrey G. Xie. Group priority scheduling. *IEEE/ACM Transactions on Networking*, 5(2):205–218, 1997.

[62] Jean M. McManus and Keith W. Ross. Video on demand over ATM: Constant-rate transmission and transport. In *INFOCOM*, pages 1357–1362, 1996.

[63] B. Qazzaz, R. Suppi, F. Cores, A. Ripoll, P. Hernandez, and E. Luque. Providing interactive video on demand services in distributed architecture. *Proceedings 29th Euromicro Conference, ISBN: 1089-6503-03*, CL-I:215–222, 2003.

[64] Bahjat Mohammad Khaleel Qazzaz. *Admission Control and Media Delivery Subsystems for Video on Demand Proxy Server*. PhD thesis, Universidad Autónoma de Barcelona, 2004.

[65] Steven Berson, Leana Golubchik, and Richard R. Muntz. Fault tolerant design of multimedia servers. *SIGMOD Rec.*, 24(2):364–375, 1995.

[66] M. Karol, P. Krishnan, and J.J. Li. Voip network failure detection and user notification. *Computer Communications and Networks, 2003. ICCCN 2003. Proceedings. The 12th International Conference on*, pages 511–516, 20-22 Oct. 2003.

[67] Tai T. Do, Kien A. Hua, and Mounir A. Tantaoui. Robust video-on-demand streaming in peer-to-peer environments. *Comput. Commun.*, 31(3):506–519, 2008.

[68] Alex C. Snoeren, David G. Andersen, and Hari Balakrishnan. Fine-grained failover using connection migration. In *Proc. of 3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, 2001.

[69] T. Anker, D. Dolev, and I. Keidar. Fault tolerant video on demand services. *Proceedings of 19th IEEE International Conference on Distributed Computing Systems*, pages 244–252, 1999.

[70] A. Maharana and G.N. Rathna. Fault-tolerant video on demand in rserpool architecture. *International Conference on Advanced Computing and Communications (ADCOM)*, pages 534–539, 20–23 Dec. 2006.

[71] Florin Sultan, Aniruddha Bohra, and Liviu Iftode. Service continuations: An operating system mechanism for dynamic migration of internet service sessions. In *Proc. Symposium in Reliable Distributed Systems (SRDS)*, oct 2003.

[72] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, November 2002.

[73] VLC media player. *URL: http://www.videolan.org/vlc/. Mayo 2008.*

[74] Michel Lespinasse and Aaron Holtzman. libmpeg2 - a free mpeg-2 video stream decoder. *URL: http://libmpeg2.sourceforge.net/. Mayo 2008*.

[75] H.J. Chen, A. Krishnamurthy, T.D.C. Little, and D. Venkatesh. A scalable video-on-demand service for the provision of vcr-like functions. *Proceedings of the International Conference on Multimedia Computing and Systems*, pages 65–72, 15-18 May 1995.

[76] Md.H. Kabir, E.G. Manning, and G.C. Shoja. An interactive scalable multimedia streaming scheme for vbr-encoded videos. *Proceedings of the 3rd Annual Communication Networks and Services Research Conference*, pages 145–150, 16–18 May 2005.

[77] I. Ahmad, Xiaohui Wei, Yu Sun, and Ya-Qin Zhang. Video transcoding: an overview of various techniques and research issues. *IEEE Transactions on Multimedia*, 7(5):793–804, Oct. 2005.

[78] Jayanata K. Dey-Sircar, James D. Salehi, James F. Kurose, and Don Towsley. Providing vcr capabilities in large-scale video servers. In *MULTIMEDIA '94: Proceedings of the second ACM international conference on Multimedia*, pages 25–32, New York, NY, USA, 1994. ACM.

[79] Kui Gao, Yuan Zhang, Wen Gao, and Simin He. Real-time scheduling supporting vcr functionality for scalable video streaming. *14th IEEE Proceedings on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 3:2711–2715, 7-10 Sept. 2003.

[80] Sassan Pejhan, Tihao Chiang, and Ya-Qin Zhang. Online rate control for video streams. *Circuits, Systems, and Signal Processing*, 20(3–4):361–373, 2005.

[81] ActiveX - VideoLAN Wiki. *URL: http://wiki.videolan.org/ActiveX. Mayo 2008.*

[82] Requirements for Internet Hosts — Communication Layers, RFC 1122, 1989.

[83] Ronald E. Walpole, Raymond H. Myers, Sharon L. Myers, and Keying Ye. *Probability & Statistics for Engineers & Scientists*. Prentice-Hall, Inc., New Jersey, eighth edition, 2007.

[84] Yinqing Zhao and C.-C. Jay Kuo. Design issues on request migration for video-on-demand services. *Proceedings of the 2004 International Symposium on Circuits and Systems (ISCAS)*, 2:II–49–52, 23–26 May 2004.

[85] P. Gevros, J. Crowcroft, P. Kirstein, and S. Bhatti. Congestion control mechanisms and the best effort service model. *IEEE Network*, 15(3):16–25, 2001.

[86] Steven McCanne and Sally Floyd. Ns2 - Network Simulator. http://www.isi.edu/nsnam/ns/, 2005.

[87] Lee Breslau et al. Advances in network simulation. *IEEE Computer*, 33(5):59–67, may 2000.

[88] V. Jacobson and Michael J. Karels. Congestion avoidance and control. *ACM Computer Communication Review; Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988*, 18, 4:314–329, 1988.

[89] Kenjiro Cho. A framework for alternate queueing: towards traffic management by pc-unix based routers. In *ATEC'98: Proceedings of the Annual Technical Conference on USENIX Annual Technical Conference*, pages 21–28, Berkeley, CA, USA, 1998. USENIX Association.

[90] Jin-Ho Kim, Saewoong Bahk, and Hyogon Kim. Performance impact of coarse timer granularities on qos guarantees in unix-based systems. *IEEE Trans. Comput.*, 52(1):51–58, 2003.

[91] B. Srinivasan, S. Pather, R. Hill, F. Ansari, and D. Niehaus. A firm real-time system implementation using commercial off-the-shelf hardware and free software. In *RTAS '98: Proceedings of the Fourth IEEE Real-Time Technology and Applications Symposium*, page 112, Washington, DC, USA, 1998. IEEE Computer Society.

[92] Marcos Paredes-Farrera, Martin Fleury, and Mohammed Ghanbari. Precision and accuracy of network traffic generators for packet-by-packet traffic analysis. *2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities -TRIDENTCOM 2006-*, page 6, March 2006.

[93] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[94] The ns Manual (formerly ns Notes and Documentation). *URL: http://www.isi.edu/nsnam/ns/doc/index.html. The VINT Project. A collaboratoin between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC. Mayo 2008.*

[95] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *SIGCOMM Comput. Commun. Rev.*, 27(1):31–41, 1997.

[96] E.W. Zegura, K.L. Calvert, and S. Bhattacharjee. How to model an internetwork. *INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, 2:594–602, 24-28 Mar 1996.

[97] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. Construction of an efficient overlay multicast infrastructure for real-time applications. *In Proceedings of INFOCOM, San Francisco, CA*, April 2003.

[98] Yatin Chawathe. Scattercast: an adaptable broadcast distribution framework. *Multimedia Systems*, 9(1):104–118, 2003.

[99] Dimitris Pendarakis, Sherlia Shi, Dinesh Verma, and Marcel Waldvogel. ALMI: An application level multicast infrastructure. In *Proceedings of the 3rd USNIX Symposium on Internet Technologies and Systems (USITS '01)*, San Francisco, CA, USA, mar 2001.

[100] Y. Nakamura, H. Yamaguchi, A. Hiromori, K. Yasumoto, T. Higashino, and K. Taniguchi. On designing end-user multicast for multiple video sources. In *ICME '03: Proceedings of the 2003 International Conference on Multimedia and Expo*, pages 497–500, Washington, DC, USA, 2003. IEEE Computer Society.

Acrónimos

CBR	Constant Bit Rate
ERAP	Enhanced Rate Adaptation Protocol
fps	frames per second
HDTV	High Definition Television
IFG	Inter-Frame Gap
IPG	Inter Packet Gap
LCS	Logical Channels Scheduler
NS2	Network Simulator 2
NTS	Network Traffic Scheduler
NTSC	National Television System Committee
P2P	Peer-To-Peer
PAL	Phase Alternating Line
QoS	Quality of Service
QoSA	Quality of Service Assurance
RAP	Rate Adaptation Protocol
RTP	Real-Time Transport Protocol
RTT	Round Trip Time
SC	Session Control

SCC Stream Control Channel

SecC Security Control

STM Stream Transmission Module

STP Stream Transmission Protocol

TCP Transmission Control Protocol

TES Time Event Scheduler

UDP User Datagram Protocol

VBM Video Buffer Map

VBR Variable Bit Rate

VDM Video Data Manager

Aquí me pongo a cantar
al compás de la vigüela,
que el hombre que lo desvela
una pena extraordinaria,
como la ave solitaria
con el cantar se consuela.

Pido a los santos del cielo
que ayuden mi pensamiento:
les pido en este momento
que voy a cantar mi historia
me refresquen la memoria
y aclaren mi entendimiento.

Vengan santos milagrosos,
vengan todos en mi ayuda,
que la lengua se me añuda
y se me turba la vista;
pido a mi Dios que me asista
en una ocasión tan ruda.

Florencio Molina Campos[2]

Yo he visto muchos cantores,
con famas bien obtenidas,
y que después de adquiridas
no las quieren sustentar.
Parece que sin largar
se cansaron en partidas.

Mas ande otro criollo pasa
Martín Fierro ha de pasar;
nada lo hace recular
ni los fantasmas lo espantan,
y dende que todos cantan
yo también quiero cantar.

Cantando me he de morir,
cantando me han de enterrar,
y cantando he de llegar
al pie del Eterno Padre;
dende el vientre de mi madre
vine a este mundo a cantar.

Que no se trabe mi lengua
ni me falte la palabra;
el cantar mi gloria labra
y, poniéndome a cantar,
cantando me han de encontrar
aunque la tierra se abra.

Me siento en el plan de un bajo
a cantar un argumento;
como si soplara el viento
hago tiritar los pastos.
Con oros, copas y bastos
juega allí mi pensamiento.

Yo no soy cantor letrao,
mas si me pongo a cantar
no tengo cuándo acabar
y me envejezco cantando:
las coplas me van brotando
como agua de manantial.

José Hernández[1] (fragmento del MARTÍN FIERRO I)

[1](1834-1886). Su infancia y adolescencia transcurrieron en los campos de Camarones y Laguna de los Padres (Argentina), donde conoció la vida de los gauchos. Participó en las luchas civiles y fue fiscal del Estado, Ministro de Hacienda, vicepresidente de la Cámara de Diputados y uno de los fundadores de la ciudad de La Plata, provincia de Buenos Aires. Además del Martín Fierro —obra cumbre de la literatura gauchesca—, escribió Vida del Chacho e Instrucción del Estanciero, entre otras.

[2]Imagen de la témpera "El truco", cedida por F. Molina Campos Ediciones (Buenos Aires, Argentina).