Proyecto de Grado 2006

Distribución de vídeo en vivo desde múltiples fuentes

Facultad de Ingeniería
Universidad de la República Oriental del Uruguay

Luis Stábile

Tutores: Pablo Rodríguez, Héctor Cancela

16 de octubre de 2007

Resumen

El presente es el informe final del Proyecto de Grado denominado "*Distribución de vídeo en vivo desde múltiples fuentes*" de la carrera de Ingeniería en Computación. El proyecto propone incorporar la estrategia de distribución de vídeo desde múltiples fuentes *(multi-source streaming)* en un sistema de código abierto y probar su desempeño en una arquitectura *Cliente-Servidores* y en una *red entre pares* (*peer to peer*). La estrategia la desarrollamos con el objetivo de mejorar la calidad percibida por los usuarios en contextos de fallas como Internet. Entre los aportes del trabajo se encuentra el estudio de sistemas de distribución de vídeo, formatos multimedia digital, estrategias para el envío de vídeo desde múltiples fuentes, mecanismos de sincronización de flujos de vídeo recibidos por los clientes y redes entre pares.

Palabras claves: *streaming, multi-source streaming, multimedia, peer-to-peer, vídeo.*

Índice general

Capítulo 1

Introducción

1.1. Motivación del Proyecto

En la actualidad, las redes de distrubución de vídeo (*VDN: Vídeo Delivery Network*) presentan un gran despliegue en Internet. De forma simplificada una *VDN* se encuentra formada por varios servidores y clientes distribuidos en la red. La calidad percibida por los usuarios es muy sensible a fallas simples en la red, ésto es de especial interés en contextos como Internet. Distintas técnicas son incorporadas a las *VDN* para lograr robustez, con el objetivo de mejorar la experiencia final de los usuarios. Una de estas técnicas, conocida como *path diversity*, ofrece la posibilidad de que un usuario reciba flujo de vídeo, con cierto nivel de redundancia, a través de varias rutas las cuales pueden originarse de una o varias fuentes simultáneas. La técnica permite minimizar el impacto de fallas en tramos de la red y en algunas de las fuentes.

El desarrollo del enfoque se inspira en que la ruta por defecto para el envío de datos entre dos nodos de Internet suele no ser la mejor opción. En [4] se ha observado que entre el 30 y 80 % de los casos existe una ruta de mejor calidad entre los nodos que la por defecto. La calidad se mide en términos de métricas como *round-trip-time*, *tasa de pérdida* y *ancho de banda*. El hecho de enviar información por muchas rutas simultáneamente aumenta el ancho de banda disponible, balancea la carga en la red y reduce considerablemente la probabilidad de pérdida de datos consecutivos causadas por ráfagas. En [2] se muestra que para el caso de vídeo es más sencillo recuperarse de múltiples pérdidas aisladas que del mismo número de pérdidas consecutivas.

Una ruta puede ser expresada en forma abstracta como una secuencia de nodos conectados a través de arcos. Como se menciona en [1], en la práctica estos caminos pueden no ser independientes y los nodos compartidos son candidatos a transformarse en puntos de embotellamiento. La detección de dichos puntos es actualmente un área de investigación activa y se estudia por ejemplo en [3].

Sin intervención de los operadores de la red, no es fácil gobernar las rutas para los flujos de una aplicación. Existen varias técnicas para resolver ésto en capa de aplicacion, en este trabajo nos concentramos en una: el envío desde múltiples fuentes. Se supone un sistema donde múltiples emisores tienen el contenido a ser distribuido. En ese caso entonces puede elegirse la fuente más apropiada para brindarle el contenido a un cliente, o mejor aún, puede enviarse

de todas las fuentes pero en mayor porcentaje o aquello de más importancia por la fuente más apropiada. También de esta forma puede usarse redundancia cruzada mitigando aún más los problemas que trae la emisión desde cada una de las fuentes en particular. El contenido de vídeo ya sea bajo demanda o en vivo es posible de ser distribuido desde múltiples fuentes.

1.2. Objetivo

El proyecto plantea el estudio de sistemas de distribución de vídeo, formatos multimedia digital, estrategias para distribuir vídeo desdes múltiples fuentes, mecanismos de sincronización de los flujos en el receptor y redes entre pares (*peer to peer*). El resultado es la incorporación de la técnica de distribución de vídeo en vivo desde múltiples fuentes a un sistema de código abierto y la realización de pruebas para el estudio de su desempeño en una arquitectura *Cliente-Servidores* y en una *red entre pares* (*peer to peer*). La estrategia la desarrollamos con el objetivo de mejorar la calidad percibida por los usuarios en contextos de fallas como Internet.

1.3. Organización del documento

El documento está organizado como sigue. En el capítulo 2 estudiamos el marco teórico del problema e introducimos conceptos que permiten comprender la problemática de nuestro trabajo y la solución propuesta. En el capítulo 3 describimos la solución que desarrollamos y los beneficios que detectamos frente a las alternativas manejadas. A continuación, en el capítulo 4 explicamos las decisiones de implementación, las librerías y estructuras de datos que utilizamos en el trabajo. En el capítulo 5 evaluamos el sistema desarrollado en una arquitectura *Cliente-Servidores* y en una *red entre pare*s. Finalmente en el capítulo 6 presentamos las conclusiones del proyecto.

Parte de los resultados del proyecto fueron publicados en "Multi-Source Video Streaming Suite", 7th IEEE International Workshop on IP Operations and Management (IPOM'07), San José, California, United States, 2007.

Capítulo 2

Estado del arte

En este capítulo describimos conceptos necesarios para comprender la problemática y la solución del sistema que desarrollamos. Explicamos la distribución de vídeo a través de Internet e introducimos los conceptos de compresión, contenedor, distribución de vídeo, distribución desde múltiples fuentes, estrategias para minimizar los problemas de distribución de contenido multimedia en vivo en contextos como Internet y redes entre pares. Finalmente introducimos el sistema de código abierto sobre el cual trabajamos y las razones por las que lo elegimos.

2.1. Digitalización de vídeo

La digitalización es el proceso mediante el cual, partiendo de una señal analógica, obtenemos una representación de la misma en formato digital. La digitalización de una imagen se basa a una división del espacio a modo de cuadrícula, donde la unidad más pequeña se denomina pixel [19]. Para cada uno de los pixels que tenemos en una imagen hay que guardar la información referente a la luminancia (brillo o niveles de gris) y, si es en color, también al nivel de cada una de las componentes, R(rojo), G(verde) y B(azul). Por tanto, tendremos para una misma imagen, varias matrices de información.

Cuando hablamos de digitalización de vídeo debemos tener en cuenta que entra en juego una tercera dimensión, el tiempo. Por tanto una secuencia de vídeo se genera mediante la proyección de un número de imágenes en un tiempo determinado, que dependerá del sistema sobre el que trabajemos (por ejemplo 24 imágenes por segundo (fps) en cine o 25 fps en el sistema PAL). El problema de este planteamiento es el alto volumen de datos que se crean. Es por ésto que se han desarrollado varios estándares de codificación y compresión como lo es la familia *MPEG*, entre otros, que permiten obtener una alta calidad de vídeo a la vez que diferentes tasas de transferencia (ver: 2.1.1).

Para obtener una imagen digital debemos tener en cuenta varios pasos:

Muestreo: Es en el único momento en el que tenemos contacto con la imagen original, o señal analógica, y es cuando decidimos con cuánta información queremos quedarnos. Es irreversible, puesto que toda aquella información que desechemos ya no podremos recuperarla y, por tanto, la calidad de la

imagen digital que generemos se verá afectada por el criterio que seleccionemos.

Cuantificación: Es el proceso mediante el cual se decide, para cada rango de colores (mundo analógico), cuál va a ser el color con el que va a ser representado en la imagen final. Por tanto, cuantos más niveles se tengan definidos mejor será la calidad. El problema que presenta es que, a mayor nivel de definición, mayor será el volumen de datos que deberemos guardar por cada uno de los pixels. La transformación realizada es también irreversible. El cuantificador más sencillo es el uniforme, aunque existen otros como el de *Max-Lloyd* que minimizan el error cuadrático medio.

Codificación Es el proceso de conversión de los valores cuantificados al sistema binario donde la organización final de los bits dependerá del formato que se escoja.

La codificación es el primer paso que nos es de interés. El formato de codificación con el que trabajamos es de la familia de *MPEG*.

2.1.1. Características

A continuación introducimos algunas características de los vídeos digitales.

Cantidad de *frames* por segundo

Indica la cantidad de imágenes por unidad de tiempo. El mínimo aceptado por el ojo humano para percibir el *vídeo* es alrededor de $15fps$. Los sistemas de codificación utilizado para la transmisión de señales de televisión analógica *PAL* y *NTSC* utilizan $25fps$ y $29{,}97fps$ respectivamente.

Resolución

Por resolución nos referimos a la dimensión de la imagen medida en pixels.

Calidad

La calidad puede ser medida con métodos formales como *PSNR* o informales como *calidad de vídeo subjetiva* usando observadores expertos.

Tasa de transferencia

La tasa de transferencia es una medida de la cantidad de información contenida en un flujo de vídeo por unidad de tiempo y es cuantificado usando bits por segundo.

Aspect ratio

Aspect ratio de una imagen es la proporción entre su anchura y su altura. Se calcula dividiendo la anchura por la altura de la imagen visible en pantalla, y se expresa normalmente como *X:Y*.

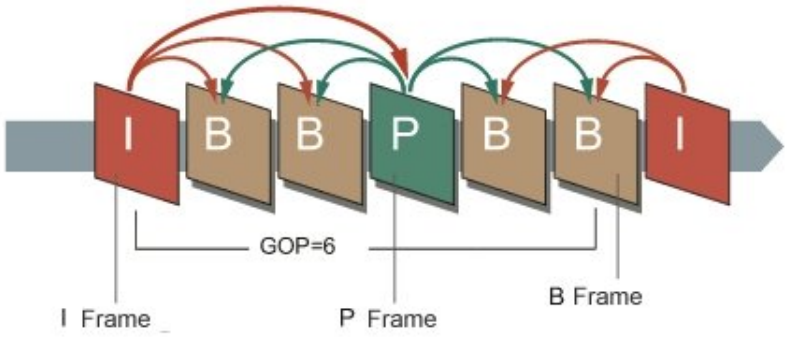

Figura 2.1: La estructura GOP.
En la imagen mostramos las dependencias entre frames para un GOP de tamaño seis. Cada dependencia indica por cada frame cuales son los que lo referencian.

2.1.2. Codificación con MPEG

MPEG (*Moving Picture Experts Group* [21, 22]) utiliza CoDec (*Codificador-Descodificador*) de compresión con bajas pérdidas[1] en donde las muestras tomadas de imagen y sonido son troceadas en pequeños segmentos (fotogramas). Los sistemas *MPEG* se basan en que la diferencia que hay ente un cuadro y el siguiente en una secuencia de vídeo es escasa y por lo tanto no es necesario grabar todos los fotogramas. Basta con grabar una imagen completa y luego guardar su evolución.

En los sistemas de codificación de imágenes en movimiento, tal como *MPEG-2* y *MPEG-4*, el contenido de imagen se predice antes de la codificación a partir de imágenes reconstruidas pasadas. Se codifican solamente las diferencias con estas imágenes reconstruidas y alguna información extra necesaria para llevar a cabo la predicción. A continuación estudiamos los sistemas *MPEG-2* y *MPEG-4*.

Sistema MPEG-2

El sistema *MPEG-2* [23, 19] crea un flujo de vídeo mediante tres tipos de datos (*frames intra*, *frames posteriores predecibles* y *frames predecibles bi-direccionales*) arreglados en un orden específico llamado *"la estructura GOP"* (GOP = Group Of Pictures o grupo de imágenes). El flujo esta compuesto por una serie de imágenes codificadas. Las tres maneras de codificar una imagen son: intra-codificado (*frame I*), predecible posterior (*frame P*) y predecible bi-direccional (*frame B*). En la figura 2.1 mostramos la estructura GOP y por cada frame se indica cuales son los que lo necesitan para la decodificación.

El audio del vídeo permite hasta seis señales de audio simultáneamente. Puede tratarse de audiciones *"mono"* (1 canal), *"estéreo"* (dos canales) o multicanal.

[1] Se denomina algoritmo de compresión con pérdida a cualquier procedimiento de codificación que tenga como objetivo representar cierta cantidad de información utilizando una menor cantidad de la misma, siendo imposible una reconstrucción exacta de los datos originales. La compresión con pérdida sólo es útil cuando la reconstrucción exacta no es indispensable para que la información tenga sentido. La información reconstruida es solo una aproximación de la información original.

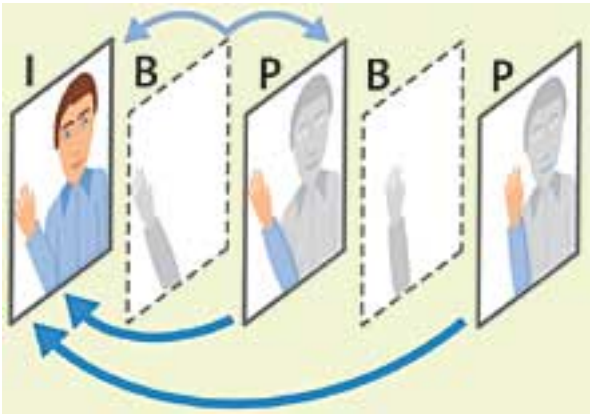

Figura 2.2: Tipos de frames.
Los frames I codifican redundancia espacial, mientras que los frames P y B codifican redundancia temporal. Los frames P y B son a menudo del 10 % *y* 2 % *respectivamente del tamaño de un frame I.*

El flujo de audio utiliza un único tipo y lo llamamos entonces simplemente bloque de audio.

La imagen del vídeo es separada en dos partes: luminancia (Y) y croma (también llamada señales de diferencia de color U y V). Las dos partes, a su vez, son divididas en *"Macro-bloques"* los cuales son la unidad básica dentro de una imagen. Cada *Macro-bloque* es dividido en cuatro 8x8 bloques de luminancia. El número de bloques de croma 8x8 depende del formato de color de la fuente. Por ejemplo en el formato común 4:2:0 hay un bloque de croma por macro-bloque por cada canal haciendo un total de seis bloques por *Macro-bloque*.

Los *frames I* están codificadas como una imagen utilizando *JPEG (Joint Photographic Experts Group [20])* y por lo tanto para decodificarla no hacen falta otras imágenes de la secuencia. Los *frames P* están codificadas como predicción del *frame I* o *P* anterior usando un mecanismo de compensación de movimiento. Para decodificar un frame de este tipo se necesita además de el mismo el *frame I* o *P* anterior. Los *frames B* se codifican utilizando el *frame I* o *P* anterior y el *P* o *I* siguiente como referencia y por tanto se necesitan para la decodificación. En la figura 2.2 presentamos una secuencia ilustrativa que muestra el contenido de cada tipo de frame.

En el caso de los *frames I*, la verdadera información de imagen es pasada a través del proceso codificador descrito abajo. Los *frames P* y *B* primero son sujetos a un proceso de *"compensación de movimiento"*, en el cual son correlacionados con la imagen previa (y en el caso del *frame B*, con la siguiente). Cada macro-bloque en el *frame P* o *B* es entonces asociada con un área en la imagen previa o siguiente que esté bien correlacionada con alguna de éstas. El *"vector de movimiento"* que mapea el *Macro-bloque* con su área correlacionada es codificado, y entonces la diferencia ente las dos áreas es pasada a través del proceso de codificación descrito a continuación. Cada bloque es procesado con una *"transformada coseno discreta"* (DCT) 8x8 que convierte del dominio del tiempo al dominio de la frecuencia. El coeficiente DCT resultante es entonces cuantificado de acuerdo a un esquema predefinido, reordenado y codificado.

Finalmente, se aplica un algoritmo de codificación Huffman de tabla fija.

Hay muchas estructuras de *GOP* posibles pero una comúnmente usada es la de 15 frames de largo, y tiene la secuencia *I_BB_P_BB_P_BB_P_BB_P_BB_*. Una secuencia similar de 12 frames es también común.

La tasa de bit de salida (tasa de transferencia) de un codificador *MPEG-2* puede ser constante (*CBR*) o variable (*VBR*), con un máximo determinado por el reproductor. Por ejemplo el máximo posible en un DVD de película es de 10.4 Mbit/s.

En la actualidad es utilizado para la codificación de audio y vídeo para señales de transmisión de Satélite Digital, Cable TV. Con algunas modificaciones es utilizado en SVCD, DVD y HDTV.

Sistema MPEG-4 [24, 22]

MPEG-4 toma muchas de las características de *MPEG-2*. A nivel de compresión *MPEG-4* es una versión mejorada del *MPEG-2*, sobre todo en la predicción de frame. El *MPEG-4* es capaz de segmentar fondo y figura, esto quiere decir que si tenemos un plano en el que aparece un fondo con objetos que se mueven delante de él, es capaz de recordar el fondo y grabar sólo los datos referentes a los objetos en una estructura llamada *VOP* (*Video Object Plain*). En lugar de *GOP* se definen una agrupación parecida que se denomina *GOV* (Group de *VOP*). En este caso generalmente no se mantienen una secuencia predefinida de *frames* (*GOV* para *MPEG-4*) , sino que la cantidad de *VOP* dentro de un *GOV* es variable.

Para nuestro trabajo no es necesario entrar en detalle sobre el sistema de codificación de *MPEG-4*. Todo lo que mencionamos para los distintos tipos de *frames* de *MPEG-2* se cumple para los distintos tipos de *VOP* de *MPEG-4*. Dado que los *frames* y *VOP* son tratados de la misma forma por nuestro sistema, a partir de aquí los llamaremos indistintamente *bloques* de vídeo.

2.2. Contenedor de vídeo

Para guardar los vídeos en disco o enviarlos a la red es necesario que los flujos de audio y vídeo estén almacenados en un mismo contenedor o cápsula. El contenedor es un formato de archivo que puede contener varios tipos de datos comprimidos por algún codificador de audio o vídeo. El proceso lo llamamos multiplexar (*MUX*) y consiste básicamente en unir vídeo, varias pistas de audio independientes y subtítulos para formar un único archivo contenedor. El proceso inverso lo llamamos demultiplexar (*DEMUX*) y consiste en separar las pistas de un archivo contenedor en archivos independientes. El archivo contenedor contiene, además de los datos, información necesaria para la sincronización de los mismos.

Algunos formatos contenedores son específicos para audio. Otros son capaces de almacenar tanto audio como vídeo. *MPEG-TS*, *MP4*, *AVI*, *OGM*, *MOV* y *ASF* son ejemplos de formatos contenedores. Los datos contenidos en un formato contenedor pueden provenir de diferentes codificaciones. En un caso ideal se podrá almacenar material audiovisual, comprimido en cualquier codificación, en cualquier contenedor. Desafortunadamente esto no sucede siempre, ya que

hay ciertas incompatibilidades de diseño entre ellos. En [37] se describen las compatibilidades entre codificación y contenedor.

Entre las características más importantes que debemos tener en cuenta al momento de escoger un formato de contenedor destacamos: los datos soportados (codificación), la sobrecarga que genera y soporte para *streaming media*. En general usaremos *MPEG-TS* y *OGM*.

2.3. Distribución de vídeo

En esta sección estudiamos conceptos básicos que son utilizados en la distribución de vídeo a través de Internet. Clasificamos la distribución según la emisión y la transmisión. Luego introducimos el concepto de *streaming media*.

2.3.1. Métodos de distribución

Emisión

La emisión de vídeo a través de una red se puede realizar en vivo *(live)* o a demanda *(VoD - Video on Demand)*. En la primera, la reproducción en el cliente se realiza a medida que se recibe información desde la fuente. En la segunda, el archivo multimedia debe estar almacenado (aunque no completamente) antes de la reproducción. El cliente tiene la posibilidad de comenzar a reproducir el mismo en el instante de tiempo del vídeo que deseé incluso retrocederla.

Transmisión

Por otro lado la transmisión puede realizarse de forma unidifusión *(unicast)* o multidifusión *(multicast)*. En la primera la distribución se realiza hacia cada cliente que la solicite mientras que en la segunda la distribución se realiza para un grupo. En la distribución multidifusión la cantidad de clientes no afecta la distribución del servidor; es lo mismo distribuir a uno que a varios clientes. En el caso de distribución a demanda este método no tiene sentido ya que cada cliente puede querer recibir diferentes partes del vídeo. El método unidifusión tiene la desventaja que por cada cliente el consumo de recursos es acumulativo. Existe una implementación nativa de *IP* para multidifusión pero no se encuentra implementada actualmente en Internet. No obstante se realiza parcialmente con la utilización de *CDN* especializadas o *redes entre pares*.

2.3.2. Streaming media

Cuando hablamos de *streaming media* nos referimos a la distribución de un flujo continuo de datos multimedia a través de Internet. El flujo es reproducido en el cliente a medida que la información es recibida. Se necesita que tanto el servidor de vídeo como las redes de datos sean capaces de mantener un flujo constante de información. Antes que existiera esta tecnología, era necesario descargar completamente el vídeo antes de comenzar su visualización.

Los usos de *streaming media* son muy amplios. Ejemplos típicos son la transmisión de eventos en vivo por Internet, las vídeo conferencias y el vídeo a demanda.

La tecnología de streaming media posee restricciones de tiempo sobre la transmisión ya que debe respetar los tiempos de reproducción de los bloques de audio y vídeo. Se requiere, por esta misma razón, una mayor coordinación entre la fuente y los clientes. El ritmo de transmisión esta marcado por los tiempos de reproducción del contenido.

Streaming media es caracterizado por datos que tienen severas restricciones de tiempo, que hacen que sea muy sensible a pérdidas y retardos en la red. *Streaming media* no solo sufre los mismos problemas asociados con la distribución de contenido estático; él también presenta desafíos debido a su naturaleza de tiempo real del contenido.

Debido a las restricciones antes mencionadas existen varias técnicas para resolver pérdidas en la red. Enfoques convencionales para la distribución con pérdidas de paquetes para datos estáticos, tales como retransmisiones, no son viables en un contexto de distribución de vídeo ya que un paquete que llega tarde es un paquete inútil. En general se aplican técnicas de reconstrucción o bien técnicas para minimizar el impacto de un paquete perdido. Los protocolos implementados sobre *TCP* [27] (como por ejemplo *HTTP* [28]) no son eficientes. Otros protocolos también utilizados funcionan sobre *UDP* [29] y son *RTP* [30] y *RTSP [31].*

Protocolos

UDP, *RTP* y *RTSP* (los protocolos empleados por algunas tecnologías de *streaming media*) hacen que las entregas de paquetes de datos desde el servidor se hagan con una velocidad mucho mayor que la que se obtiene por *TCP* y *HTTP*. Esta eficiencia es alcanzada por una modalidad que favorece el flujo continuo de paquetes de datos. Cuando *TCP* y *HTTP* sufren un error de transmisión, siguen intentando transmitir los paquetes de datos perdidos hasta conseguir una confirmación de que la información llegó en su totalidad. Sin embargo *UDP* continúa mandando los datos sin tomar en cuenta interrupciones. La principal desventaja de los protocolos sin conexión *(UDP, RTP, RTSP)* es que están más propensos a ser bloqueados por un *firewall*. Estudios como [12, 13] muestran que el protocolo más utilizado para distribuir vídeo en vivo en Internet es *HTTP*. A continuación mencionamos algunas características de los protocolos *RTP*, *RTSP* y *HTTP*.

RTP esta pensado para la transmisión de audio y vídeo en tiempo real y funciona sobre *UDP*. A cada paquete se le agrega una marca para que el receptor pueda saber los tiempos de reproducción que debe respetar. Además el protocolo numera los paquetes para determinar en el cliente si hubo pérdidas.

RTSP es el protocolo más utilizado en *streaming media*. Puede funcionar tanto sobre *TCP* como *UDP*, aunque en general se utiliza sobre este último. Brinda control de flujo multimedia por parte del usuario (*Play*, *Stop*, etc). Como se explica en [32] habitualmente se usa *RTP* distribuir el flujo y *RTSP* para controlarlo.

HTTP es el más lento de todos los protocolos mencionados en esta sección. Es utilizados por los servidores *Web* y permite ser transparente a la seguridad de los *firewalls* que no aceptan otros protocolos. A diferencia de los

anteriores, no puede enviar la información a tasa constante. Simplemente transmite tan rápido como pueda.

Técnicas de reconstrucción y minimización de impacto ante la pérdida de paquetes

En los protocolos que no utilizan retransmisión no se controla la pérdida de paquetes. Si bien esta falta de retransmisión es beneficiosa para las restricciones de tiempo, se debe tomar una decisión acerca de como sobrellevar la falta de algunos paquetes. Perkins, Hodson y Hardman en [5] plantean una serie de alternativas para este problema. Primero categorizan las acciones posibles para la pérdida de paquetes en dos grupos según quién las realice: reparación basada en el emisor y reparación basada en el receptor.

Con respecto a la reparación basada en el emisor podemos identificar la retransmisión como una técnica activa pero poco efectiva. Como pasivas distinguimos *FEC* (*Forwarding Error Correction*) e *Interleaving,* que son técnicas más adecuadas al contexto de *streaming media*. *FEC* consiste en el envío de paquetes extra cada k paquetes del flujo multimedia. Este paquete extra contiene chequeos de paridad permitiendo la reconstrucción total de un paquete de los k enviados pero a la vez implica el envío de información extra en la red. *Interleaving* consiste en enviar las unidades de información (supongamos que un paquete contiene m unidades de información) en un orden distinto, de manera que las unidades que inicialmente están contiguas ahora están a una distancia m y por tanto en distintos paquetes. El receptor conociendo esta distancia, rearma los paquetes cuando estos llegan. Ésto permite que ante una pérdida de un paquete, solo se pierdan unidades no contiguas, haciendo más fácil las tareas de reconstrucción, que si se pierde un paquete completo.

Las técnicas basadas en el receptor consisten todas en tomar decisiones con respecto a que hacer con los paquetes faltantes, ya que se sabe que no se retransmitirán. Una posibilidad es la inserción de un paquete reconstruido por interpolación.

Cuando la distribución de vídeo se realiza desde múltiples fuentes, un abanico de posibilidades se abren: por ejemplo la redundancia cruzada, que explicamos en 3.

2.4. Redes de distribución de contenido

Las redes de distribución de contenido (*CDN: Content Delivery Networks*) fueron desarrolladas para superar problemas de funcionamiento inherentes a Internet tales como la congestión en la red, la sobrecarga en servidores, que se presentan cuando muchos usuarios acceden a contenido popular [10, 7]. Las *CDN* mejoran la calidad percibida por los usuarios almacenando el contendido popular en servidores réplicas ubicados cerca de las áreas de solicitudes. Este enfoque provee un número de ventajas. Primero, ayuda a prevenir sobrecarga de los servidores dado que el contenido replicado puede ser distribuido de varios servidores. Además el contenido es distribuido a los usuarios por el servidor más cercano reduciendo así los tiempos de solicitud y respuesta, pérdidas de paquetes y el total de uso de recursos en la red [10].

Los modelos centralizados abundan en la Internet de hoy en día. Estas arquitecturas son pobres en términos de adaptabilidad y escalabilidad. Si un cliente establece conexión con un servidor que provee datos, servicios e información a todos sus usuarios es probable que sea sobrecargado y sus enlaces pueden ser fácilmente saturados. Es imposible con este enfoque adaptarse al crecimiento exponencial de tráfico en Internet. Los modelos centralizados para servidores de contenidos son inherentemente inescalables, incapaces de adaptarse a grandes cantidades de usuarios. Estos problemas llevan a que los usuarios del contenido no perciban una calidad aceptable sobre todo cuando se trata de contenido multimedia. La estructura de Internet como una interconexión de redes individuales es la clave de su escalabilidad pero no es suficiente para garantizar un rápido crecimiento en el número servicios. El tráfico en puntos de embotellamiento lleva a reducir considerablemente el rendimiento de la red en cuanto a retardos, tasa de pérdida y gran latencia.

El modelo de Internet centralizado o parcialmente distribuido en la distribución de contenido requiere que todos las solicitudes y respuestas de los usuarios viajan a través de muchas redes y enlaces probablemente congestionados. La primer solución adaptada a distribuir contendido a través de Internet es la técnica *mirroring* (réplica). Ésta técnica estáticamente duplica el contenido en diferentes puntos geográficos. Los usuarios manualmente seleccionan, de una lista de servidores, el servidor mejor situado en cuanto a su distancia.

Un enfoque más inteligente es *CDN* en donde la base de la tecnología es la distribución en puntos de la red en proximidad a las áreas de solicitud, mejorando así el rendimiento percibido por los usuarios. Las *CDN* son redes de capa de aplicación formadas por un conjunto dedicado de servidores (réplicas) distribuidos en las cercanías de los clientes, los cuales proporcionan contenido con unos valores reducidos de latencia [7]. Son redes cuyo propósito general apunta a reducir el tiempo de espera de los usuarios. Reducir el tiempo de respuesta implica reducir la latencia de la red así como el tiempo de proceso en los servidores. Intenta reducir la latencia evitando rutas congestionadas. lo que conduce a una reducción del tiempo de respuesta percibido.

Réplicas de contenido son geográficamente distribuidas, permitiendo así distribución rápida y confiable hacia los usuarios en cualquier ubicación. Están formadas por los proveedores de contenido que son servidores réplicas ubicados en puntos estratégicos del mundo y un mecanismo de redirección de las solicitudes a la mejor réplica. El diseño de una *CDN* requiere considerar las siguientes características:

- El tipo de contenido que es almacenado en los servidores réplicas
- Cómo el contenido se mantiene actualizado en los distintos servidores
- Un mecanismo que determine cuál es la mejor réplica para un cliente dado de modo que reducir el riesgo de encontrar puntos de embotellamiento en el camino. Debe considerar diferentes tipos de servicios con diferentes calidades de servicio.
- Un mecanismo de direccionamiento de las solicitudes a la mejor réplica.

Una *CDN* correctamente diseñada puede acelerar el acceso a contenido de los usuarios, reducir el tráfico en la red y los requerimientos de hardware en los proveedores de contenidos.

2.5. Codificación de múltiples descripciones

Codificación de múltiples descripciones (*MDC: Múltiple description coding*) es una técnica de codificación en la cual se divide un flujo multimedia en N flujos independientes ($N \geq 2$) que llamamos descripciones [9, 11, 19]. A su vez los paquetes de cada descripción pueden ser enviados a través de múltiples rutas (*path diversity*) (parcialmente) disjuntas. *MDC* presenta dos propiedades interesantes: (1) cada descripción puede ser independientemente codificada para dar un nivel de calidad aceptable, (2) Las múltiples descripciones contienen información complementaria y la calidad aumenta con la cantidad de descripciones correctamente recibidas. La idea de *MDC* es proporcionar robustez ante fallas en los flujos multimedia. Un arbitrario subconjunto de descripciones puede ser usado para decodificar el flujo original. Ante congestión en la red y pérdida de paquetes (los cuales son comunes en ambientes como Internet) no se interrumpe el flujo reproducido en el cliente, sólo se reduce la calidad percibida pero sigue siendo aceptable.

MDC es una forma de particionar datos, tal como el modelo de codificación de *MPEG-2* y *MPEG-4*. Estos últimos generan una capa base y N capas superiores. La capa base es necesaria para que el flujo pueda ser decodificado (*Frame I* en *MPEG-2* y *VOP* en *I MPEG-4*) y las capas superiores son aplicadas para mejorar la calidad. Además la primer capa depende de la base y cada capa $i+1$ depende de su subordinada i, por lo tanto sólo puede ser usada si esta última fue correctamente aplicada.

MDC permite a los proveedores de contenido enviar flujo a tasa adaptativa, es decir, los clientes que no pueden recibir a la tasa de transferencia del flujo multimedia, se suscriben a un subconjunto de esas descripciones, reduciendo así la calidad percibida.

A pesar de las ventajas que presenta *MDC* sobre los codificadores tradicionales que no particionan datos (*Single Description*), estos últimos aún predominan. Las razones son probablemente la gran complejidad de los codificadores y decodificadores, y la pérdida de alguna compresión eficiente dado que *MDC* consume mayor ancho de banda (considerando todas las descripciones) [8].

MDC es especialmente prometedor para aplicaciones de vídeo donde la retransmisión es inaceptable. Cuando se combina con transporte a través de *path diversity*, *MDC* permite balancear la carga y reducir la congestión en la red [9]. Surgió como un enfoque prometedor para superar las pérdidas en sistemas de distribución de vídeo. En la mayoría de las implementaciones, la codificación con múltiples descripciones genera dos descripciones de igual tasa de transferencia e importancia, por lo tanto cualquier descripción provee calidad baja pero aceptable y ambas descripciones juntas producen la calidad del vídeo original. Las dos descripciones son individualmente paquetizadas y enviados a través de la misma o diferentes rutas. En un caso más general, más de dos descripciones pueden ser generadas, las cuales pueden o no tener idénticas tasas de transferencia. La principal razón en el crecimiento de la popularidad de *MDC* es que puede proveer calidad adecuada sin requerir retransmisión para ninguna pérdida de paquete (a menos que la tasa de pérdida sea extremadamente grande). Los principales usos son en aplicaciones interactivas, tales como vídeo conferencias, en donde las retransmisiones son inaceptables. *MDC* trae consigo costos, dado que el codificador necesita más bits. Estos bits extras o redundantes son insertados intencionalmente para hacer flujos más resistentes a los errores de

transmisión.

MDC es particularmente prometedor en aplicaciones que son muy rigurosas a retrasos en la transmisión tales como distribución de vídeo.

2.6. Distribución de vídeo a través de múltiples rutas

Path diversity es una técnica de transmisión que envía datos simultáneamente a través de múltiples rutas en una red [1]. La técnica ha sido propuesta para para combatir las pérdidas y retardos que afectan la distribución de *streaming media*. Se ha demostrado que distribuir datos a través de *path diversity* puede mejorar la tolerancia a fallas así como proveer mayor ancho de banda y balancear la carga. Además la motivación para su uso es que la ruta por defecto en la distribución desde una única fuente entre dos nodos de la red suele no ser el mejor camino. En [4] se obtuvo como resultados que entre el 30 y 80 % de los casos hay una ruta alternativa con mayor calidad que la ruta por defecto, donde la calidad es medida en términos de métricas como *round-trip-time*, tasa de pérdidas y ancho de banda.

La ruta puede originarse de una (*single sender*) o múltiples fuentes (*multiple sender*). En primer enfoque, los paquetes son enviados en forma explícita a través de diferentes rutas. En ciertas circunstancias es posible especificar la secuencia de nodos a atravesar para cada uno de los paquetes. *Path diversity* puede ser alcanzado especificando explícitamente diferentes rutas para diferentes subconjuntos de paquetes. También puede alcanzarse enviando cada una de las diferentes secuencias de paquetes a distintos nodos (ubicados en puntos estratégicos de la red) los cuales realizan la operación de reenvío al destino deseado [2]. El enfoque *multiple sender* es un poco más complejo y consiste en la distribución de contenido a través de múltiples fuentes en donde cada una envía un subconjunto de paquetes. Una forma de alcanzar *multiple sender path diversity* es aprovechando la infraestructura distribuida de las *CDN* en donde los clientes reciben contenido de múltiples servidores réplicas. Además acoplando apropiadamente *MDC* con *CDN* se puede mejorar la confiabilidad ante pérdidas de paquetes y fallas en servidores. Este sistema es conocido como *codificación de múltiples descripciones en redes de distribución de contenido* y lo estudiaremos en *2.7*. Esta arquitectura encapsula los beneficios de *Path diversity* y los de *CDN*.

Las redes inalámbricas son más desafiantes para la distribución de *streaming media* pues generalmente brindan comportamiento más propenso a sufrir retardos y pérdidas. Enlaces inalámbricos son habitualmente puntos de embotellamientos en comparación con los enlaces cableados. En [18] se propone un sistema compuesto de *path diversity* entre múltiples *access points* y clientes móviles. Se concluye que el sistema puede proveer significantes beneficios comparado con el caso convencional donde se usa un único *access point*.

Múltiples rutas no garantizan ser independientes en la práctica. Una ruta puede ser expresada abstractamente como una secuencia de arcos conectados a través de nodos. Dos rutas a menudo comparten arcos y nodos. Los beneficios de *path diversity* no depende de si los caminos son completamente disjuntos perom sí de si los embotellamientos ocurren en secciones compartidas o disjuntas.

Secciones de embotellamiento compartidas reducen el impacto de *path diversity* mientras que secciones disjuntas no lo hacen. La identificación de puntos de embotellamiento y como evitarlos si es posible es un elemento importante en la efectividad para el uso de *path diversity*. La detección de puntos de embotellamientos compartidos es actualmente un área de investigación activa; ver por ejemplo [3].

Diferentes rutas pueden ofrecer diferentes anchos de banda, tasa de pérdidas y retardos característicos. Un sistema de *path diversity* debe considerarlo para explotar esas asimetrías. El primer paso es estimar las características de cada ruta. El uso de métricas para analizar esas características de los caminos es una área de investigación activa en la actualidad; muchas técnicas se han desarrollado y son aplicables en este contexto. No obstante, no profundizaremos en este punto. Rutas con diferentes tasas de pérdidas y retardos motivan a llevar a cabo técnicas para planificar la trayectoria de los paquetes. Por ejemplo los paquetes más importantes pueden ser enviados a través de rutas con más bajas tasas de pérdidas y los paquetes con fuertes restricciones de tiempo pueden ser enviados por las que tienen menores retardos.

En distribución desde múltiples fuentes (*multiple sender path diversity*) permite flexibilidad ante la dinámica de la red. En particular, es posible aumentar el número de fuentes y la cantidad de información redundante. Esta flexibilidad debe ser configurada para obtener calidad satisfactoria minimizando el costo. La arquitectura considerada involucra algún servidor distribuyendo vídeo, dividiendo el flujo del vídeo original dentro de muchos flujos con alguna nivel de redundancia. Cada cliente debe ser capaz de reconstruir el flujo de vídeo original a partir del conjunto de flujos recibidos de las distintas fuentes.

A continuación analizaremos los beneficios de *path diversity* [1]. Tales beneficios dependen de como los diversas rutas sean usadas. Por simplicidad, asumimos el caso ideal en el que todas las rutas son disjuntas y tienen idénticos atributos. Un beneficio directo es el incremento de ancho de banda disponible al usar múltiples rutas en lugar de una. Un beneficio complementario es balancear la carga al reducir el ancho de banda consumido por cada fuente dividiendo el flujo a través de varias rutas. *Path diversity* también reduce la frecuencia y el largo de ráfagas de pérdidas. Distribuyendo paquetes a través de múltiples rutas incrementa el espacio entre paquetes para cada ruta. Reducir las pérdidas en ráfagas provee un número de beneficios para *streaming media*. Por ejemplo, para el caso de vídeos, es más fácil recuperarse de múltiples pérdidas aisladas que del mismo número de pérdidas consecutivas [2]. Para dos rutas con la misma tasa de pérdida en promedio, mandar paquetes pares por una ruta e impares por la otra, tiene el mismo efecto desde el punto de vista del receptor pero reduce las pérdidas en ráfagas.

2.7. Codificación de múltiples descripciones en redes de distribución de contenido

Las *CDN* han sido ampliamente usadas por proveer baja latencia, escalabilidad, tolerancia a fallas y carga balanceada para la distribución de contenido *web* y más recientemente para *media streaming*. En [10] se propone un sistema que mejora el funcionamiento en la distribución de contenido en vivo en *CDN*

explotando la distribución desde múltiples fuentes. El proyecto aprovecha la infraestructura existente en *CDN* para alcanzar *path diversity* entre múltiples servidores réplicas y el cliente. *Path diversity* es provisto por las diferentes rutas que existen entre un cliente y los servidores réplicas más cercanos. Codificación de múltiples descripciones es junto con *path diversity* proveen mayor resistencia a pérdidas.

El sistema propuesto en [2] utiliza *MDC* para generar dos descripciones complementarias las cuales son distribuidas a través de los servidores réplicas más cercanos al cliente. Cuando un cliente solicita un flujo de vídeo, es direccionado a través del mecanismo de *CDN* a dos servidores, los cuales le enviarán descripciones complementarias. Esos servidores envían simultáneamente las descripciones al cliente a través de diferentes rutas. El sistema combina *MDC* con *path diversity* en el cual diferentes descripciones son explícitamente enviadas a través de diferentes rutas al cliente. *Path diversity* explota el hecho de que es poco probable que dos caminos sufran pérdidas simultáneamente. De esta forma, interrupciones en el vídeo se producen únicamente en casos pocos probables donde ocurran fallas simultáneas en ambas rutas. La probabilidad de que todas las rutas estén congestionadas es menos probable que la de que una única lo esté [11]. La arquitectura también presenta los beneficios de *CDN*, tales como tiempos de respuestas reducidos a los clientes, balancear la carga a través de los servidores, mayor tolerancia a fallas en los servidores y escalabilidad en el número de clientes. La ventaja de *CDN* es que provee una plataforma sobre la cual *path diversity* puede ser alcanzado sin técnicas explícitas para distribuir desde múltiples fuentes.

Por supuesto que el uso de varias rutas no garantiza independencia entre ellas. Generalmente partes de una ruta puede ser disjuntas mientras otras pueden coincidir. Entonces las ventajas de tener múltiples rutas depende de las características de las secciones comunes. Si la mayoría de las pérdidas ocurren en puntos comunes entonces hay pocas ventajas en el uso de *path diversity*. En cambio si las secciones comunes tienen relativamente poca falla entonces los beneficios de usar *path diversity* son altamente notorios.

En *CDN* donde cada cliente recibe datos de un único servidor sobre una única trayectoria, seleccionar la mejor ruta consiste en minimizar alguna noción de distancia entre el cliente y el servidor. No obstante, recibir datos de muchos servidores a un cliente como *MDC* en *CDN* requiere diferentes métricas: minimizar la distancia de cada cliente a múltiples servidores y a su vez maximizar la dispersión entre las mismas. Notar que minimizar la distancia a los servidores y maximizar la dispersión usualmente son contradictorias [1]. Evaluar esas técnicas es necesario para el diseño y operación de *CDN* que usa *Path diversity*. Además los problemas de selección de los servidores deben ser resuelto para cada solicitud de cada cliente.

Los resultados de [10] muestran que la reducción de las pérdidas cuando se usa una *CDN* sin *MDC* para distribuir flujo de vídeo es entre 20 y 40 %. No obstante, para ciertas topologías, *MDC* requiere un 50 % menos de servidores réplicas que técnicas convencionales de distribución de vídeo para alcanzar la misma calidad en los clientes. En resumen, podemos concluir que acoplar *MDC* con *path diversity* sobre la infraestructura de *CDN* puede proveer mejoras significantes en el funcionamiento sobre distribución de vídeo convencional, también sobre la infraestructura de *CDN*.

2.8. Redes entre pares

Las redes ente pares (*peer to peer*) son redes virtuales desarrolladas a nivel de aplicación sobre la infraestructura de Internet [19]. Los nodos, llamados *peers,* ofrecen sus recursos (ancho de banda, procesador, capacidad de almacenamiento) a los otros nodos, básicamente porque todos tienen intereses comunes. Las redes entre pares se caracterizan por no tener clientes ni servidores fijos, sino una serie de nodos que se comportan simultáneamente como clientes y como servidores de los demás nodos de la red. Este modelo contrasta con el modelo *Cliente-Servidor* basado en una arquitectura monolítica donde no hay una distribución de tareas, sino sólo una simple comunicación entre un usuario y una terminal, y en donde el cliente y el servidor no pueden cambiar roles. A medida que crece el número de nodos, crecen los recursos totales de la red [14]. En la actualidad estas redes tienen un gran despliegue y son muy usadas en aplicaciones de *file sharing* como *Bittorrent*, *eMule*, *KaZaA*.

En estas redes los nodos se conectan y desconectan con gran frecuencia en forma autónoma y completamente desincronizada. Esto significa que los recursos de la red en su totalidad son también dinámicos y por lo tanto la red debe ser robusta frente a estas fluctuaciones. El principal problema es como proveer buenos niveles de calidad en un contexto donde esta calidad depende de otros clientes que están distribuyendo vídeo, y que se conectan y desconectan frecuentemente. Este problema puede ser tratado usando redundancia en la distribución.

A continuación describimos seis características deseables de las redes entre pares.

Escalabilidad. Lo deseable es que cuantos más nodos estén conectados a la red, mejor será el funcionamiento. Así, cuando los nodos llegan y comparten sus propios recursos, los recursos totales del sistema aumentan. Esto es diferente en una arquitectura de *Cliente-Servidor* con un sistema fijo de servidores, en los cuales la adición de más clientes podría significar una transferencia de datos más lenta para todos los usuarios.

Robustez. La naturaleza distribuida de las redes también incrementa la robustez en caso de haber fallas en los nodos, permitiéndoles encontrar información sin hacer peticiones a ningún servidor centralizado.

Descentralización. Estas redes por definición son descentralizadas y todos los nodos son iguales. No existen nodos con funciones especiales, y por tanto ningún nodo es imprescindible para el funcionamiento de la red. En realidad, algunas redes comúnmente llamadas entre pares no cumplen con esta característica, como *Napster*, *eDonkey2000* o *BitTorrent.*

Costos repartidos. Se comparten o donan recursos a cambio de recursos. Según la aplicación de la red, los recursos pueden ser archivos, ancho de banda, procesador o almacenamiento de disco.

Anonimato. Es deseable que en estas redes quede anónimo el autor de un contenido, el servidor que lo alberga y la petición para encontrarlo siempre que así lo necesiten los usuarios.

Seguridad. Es una de las características deseables de las redes entre pares menos implementada. Los objetivos son: identificar y evitar los nodos maliciosos, evitar el contenido infectado, evitar el espionaje de la comunicación

entre nodos, crear de grupos de nodos seguros dentro de la red y proteger de los recursos.

En la actual Internet, el ancho de banda, las capacidades de almacenamiento y cómputo, son recursos caros. En aplicaciones que requieren una gran cantidad de recursos pueden utilizarse redes entre pares.

Una posible clasificación de las redes entre pares puede ser acorde a su grado de centralización. Las redes centralizadas se basan en una arquitectura donde todas las transacciones se hacen a través de un único servidor que sirve de punto de enlace entre dos nodos, y que a la vez almacena y distribuye la información sobre los nodos donde se almacenan los contenidos. Presentan falta de escalabilidad, enormes costos en el mantenimiento así como el consumo del ancho de banda. Todas las comunicaciones dependen exclusivamente de la existencia del servidor. Las redes entre pares descentralizadas son las más comunes, siendo las más versátiles al no requerir de un gestionamiento central de ningún tipo. Los nodos actúan al mismo tiempo como clientes y servidores. Todas las comunicaciones son directamente de usuario a usuario con la ayuda de un nodo (que es otro usuario) quien permite enlazar esas comunicaciones. En las redes entre pares mixtas hay una interacción entre un servidor central que administra los recursos como ancho de banda, enrutamientos y comunicación entre nodos, pero sin saber la identidad de cada nodo y sin almacenar información alguna, por lo que el servidor no comparte archivos de ningún tipo a ningún nodo. Tienen la peculiaridad de funcionar de ambas maneras, es decir, pueden incorporar más de un servidor que gestione los recursos compartidos, pero también en caso de que el o los servidores que gestionan todo, fallen o se desconecten, el grupo de nodos sigue en contacto a través de una conexión directa entre ellos con lo que es posible seguir compartiendo y descargando información. Los nodos son responsables de almacenar la información que permite al servidor central reconocer los recursos que se desean compartir, y para poder descargar esos recursos compartidos a los nodos que lo solicitan.

Estamos interesados en la aplicación de redes entre pares para distribución de flujos multimedia, las cuales presentan problemas más desafiantes que las tradicionales (*file sharing*). En estas últimas, un cliente primero descarga el archivo y luego lo usa (*open after download*). En la distribución de flujo multimedia los clientes consumen el contenido del archivo mientras el archivo esta siendo descargado (*play while download*)*[15, 17]*. Más específicamente, un conjunto de nodos almacenan cierto archivo y lo envían a los clientes solicitantes. En el otro lado, los nodos solicitantes reproducen los datos mientras se está recibiendo el flujo multimedia. En estos sistemas, la relación *distribuidor-solicitante* es típicamente *de muchos a uno,* en lugar de *uno a uno* como es en general en las redes entre pares. Dado que la cota para el ancho de banda ofrecido por el proveedor (ancho de banda de subida) puede ser menor que la tasa del archivo multimedia, es necesario involucrar múltiples distribuidores por sesión. Estas redes presentan el problema de la asignación de datos a múltiples servidores. Más específicamente, dado un solicitante y un conjunto de distribuidores, con ancho de banda de salida heterogéneo, se debe asignar un subconjunto de datos para cada uno. Entre múltiples solicitantes, quienes prometen mayor ancho de banda de salida deberían ser los más prioritarios porque ellos contribuirán más a la capacidad del sistema luego de convertirse en distribuidores.

Utilizar la infraestructura P2P para la distribucion de vídeo es una muy bue-

na idea debido a los altos requierimientos de ancho de banda de estos sistemas. La distribución de vídeo a demanda es muy similar a la distribucion de archivos [16], sin embargo, la distribución de vídeo en vivo tiene restricciones de tiempo real que implican resolver una serie de problemas técnicos en una infraestructura dinámica como lo son las redes entre pares. Al utilizar los recursos osciosos de los usuarios, los sistemas entre pares presentan una escalabilidad ideal, eliminado los altos costos de ancho de banda en servidores centralizados. El problema surge en como asegurar la calidad del vídeo en un contexto donde los clientes se conectan y desconectan dinámicamente. Distintas técnicas de redundancia se han aplicado para mitigar este problema, nuestro trabajo exploramos una de ellas: la distribución de vídeo en vivo desde múltiples fuentes simultáneas. En tal contexto, los servidores deben poseer la información y la sincronía de mandar la parte del stream que a cada uno le corresponde y el cliente la capacidad de juntar todas las partes.

Actualmente existen muchas implementaciones de distribución de vídeo sobre redes entre pares. Un ejemplo es *GnuStream*, un prototipo que está construido sobre la popular plataforma *Gnutella* [44]. Entre las principales características de *GnuStream* destacamos la capacidad en detectar cambios de estado de los nodos, y la recuperación ante fallas y degradación en el ancho de banda disponible. El sistema utiliza pruebas periódicas para detectar cambios en el estado de los nodos distribuidores, incluyendo desconexión, fallas y degradación en ancho de banda. Si un nodo que se encuentra distribuyendo flujo de contendido multimedia es detectado como sufriendo fallas o degradación en ancho de banda, *GnuStream* migrará todo o parte del contenido que le corresponde enviar a otro distribuidor. Además el conjunto de nodos distribuidores activos en una sesión cambia dinámicamente. Por más detalles del funcionamiento del sistema dirigirse a [17].

Es común encontrar redes entre pares estructuradas para distribución de vídeo en vivo basado en una algoritmos de reconstrucción más rápidos que los usados en aplicaciones de *file sharing*, como multidifusión a nivel de aplicación (*ALM: Application Level Multicast*). *ALM* fué propuesto para superar las limitaciones de multidifusión IP y son interesantes en distribución de vídeo en vivo debido a su eficiencia en comparación a los protocolos utilizados en aplicaciones de *file sharing*. Cada sistema basado en *ALM* tiene su propio protocolo de construcción y matenimiento del árbol de distribución[2]. Por ejemplo *NICE* y *Zigzag* adoptan un árbol jerárquico y son escalables a un gran número de nodos. Narada por otro lado apunta a aplicaciones con menor cantidad de nodos. Narada mantiene y optimiza una red *mesh*[3] de nodos, que produce buena performnce pero impone significante carga de mantenimiento. CoopNet emplea *MDC* y construye un árbol por cada descripción. Implementa un sistema híbrido entre una red entre pares y una *CDN*. En nuestro trabajo nos concentramos en la técnica de distribución desde múltiples fuentes, dejando de lado las problemáticas referentes a la topología de la red entre pares.

[2] Árbol construido por el protocolo de enrutamiento multidifusión para unir los receptores a las fuentes.

[3] es una topología de red en la que cada nodo está conectado a uno o más de los otros nodos. De esta manera es posible llevar los mensajes de un nodo a otro por diferentes caminos.

Figura 2.3: Aspecto visual de la interfaz de de usuario de *MPlayer*.

2.9. Software

El objetivo del proyecto es incorporar la técnica de distribución de vídeo en vivo desde múltiples fuentes a un sistema de código abierto. A continuación comentamos algunas de las alternativas de software que manejamos para implementar la técnica. Distinguimos entre cliente y servidor enfocándonos más en este último. Para el cliente, basta con cualquier reproductor que soporte *streaming media*.

RealNetwork

El programa *Mplayer* (Multimedia player, desarrollado por *RealNetwork*) [33] es un reproductor de código abierto muy conocido en el ambiente Linux, usado principalmente para la visualización de películas en múltiples formatos. Se le puede adicionar soporte para recibir flujo de vídeo vía protocolos *RTP/RTSP*. En la figura 2.3 mostramos el aspecto visual que tiene la interfaz de usuario. Este programa también puede ser llamado a través de línea de comandos y aplicarle parámetros como el *URL* del flujo solicitado. La principal desventaja de *Mplayer* es que funciona únicamente como cliente (no como servidor). La solución que brinda *RealNetwork* para distribuir vídeo es *Helix Service Delivery*.

MPEG4IP

El proyecto *MPEG4IP* [34] provee un conjunto de herramientas de código abierto, que permiten implementar servidores y clientes de streaming, basados en estándares libres de protocolos y extensiones propietarios. Su desarrollo está concentrado sobre *GNU/Linux*, pero también ha sido portado a otras plataformas. Dentro de los muchos programas que se incluyen en el proyecto se encuentran el reproductor *gmp4player* y el servidor *mp4live*, que permite codificar y transmitir flujos de vídeo y audio en tiempo real.

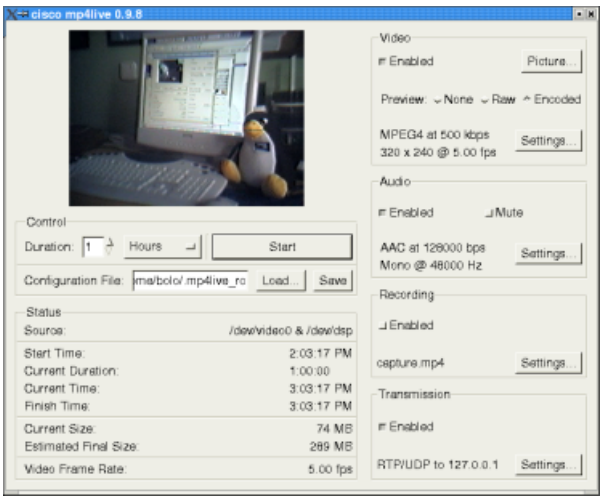

Figura 2.4: Aspecto visual de la interfaz de usuario de *mp4live*.

El servidor *mp4live* dispone de una Interfaz Gráfica de Usuario (Ver figura 2.4) que permite configurar los parámetros necesarios para generar el flujo multimedia. Éste puede ser transmitido por la red vía unicast o multicast, escrito localmente a un archivo o ambos simultáneamente. En el caso de ser transmitido por la red, se utiliza el protocolo *RTP*.

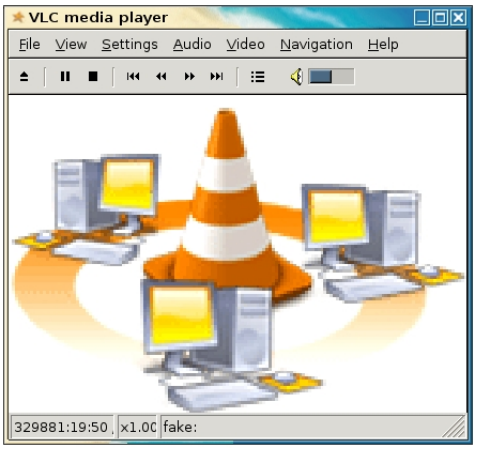

Figura 2.5: Aspecto visual de la interfaz de usuario de *VLC*.

VideoLAN

El proyecto VideoLAN [35] es una solución completa de software para la distribución de vídeo desarrollada bajo la licencia pública general *GNU (GPL)*. Soporta distribución en vivo y bajo demanda, transporte de datos a través de los protocolos IPv4 [25] e IPv6 [26], además de poder generar tráfico *unidifusión (unicast)* o *multidifusión (multicast)*. Es muy versátil dado que soporta una amplia gama de formatos, protocolos y plataformas (ver [37]).

Dentro de las herramientas desarrolladas por el proyecto VideoLAN, se encuentran *VLS* (VideoLAN Server) y *VLC* (inicialmente el cliente VideoLAN). A pesar de que partieron siendo programas complementarios (cliente y servidor), con el transcurso del tiempo, el desarrollo de *VLC* permitió que esta herramienta fuera una solución en si misma; es por ello que nuestra atención estará centrada en esta aplicación.

El programa *VLC* permite que se opere a través de línea de comandos o por su interfaz gráfica. Se recomienda, sin embargo, la utilización en línea de comandos ya que permite mayor control sobre los parámetros con los cuales se invoca.

Es importante destacar que las herramientas de VideoLAN sobresalen sobre los otros sistemas de distribución de vídeo ya que poseen buena documentación, foros y una interfaz de usuario amigable, lo que permite una rápida adaptación por parte de usuarios menos experimentados. En la figura 2.5 mostramos el aspecto visual que tiene la interfaz de usuario.

Capítulo 3

Enfoque del proyecto

3.1. Alcance

El alcance del proyecto es incorporar la estrategia de distribucón de vídeo en vivo desde múltiples fuentes a un sistema de código abierto y probar su desempeño en un escenario *Cliente-Servidores* real y en una *red* entre pares. La estrategia la desarrollamos con el objetivo de mejorar la calidad percibida por los usuarios en contextos de fallas como Internet, teniendo en cuenta los beneficios que explicamos en 2.6.

3.2. Visión global del sistema

En esta sección describimos la arquitectura del sistema para llevar a cabo la implementación de la estrategia de distribución de vídeo en vivo desde múltiples fuentes. Contamos con N servidores que envían vídeo en forma simultánea a M clientes. Cada servidor es identificado por un número entre 1 y N. Sin perder generalidad supondremos de aquí en más que M es 1. En la figura 3.1 mostramos una visión global del sistema que implementamos. Los módulos codificador, emisor, receptor y decodificador son para nosotros "cajas negras" (solo nos interesan las funcionalidades que prestan). Los módulos que incorporamos a *VLC* son: *mpath-server*, *mpath-bridge* y *mpath-client*.

El módulo *mpath-server* (ubicado en el servidor) es el encargado de decidir qué bloques de vídeo enviar al cliente. Deseamos que todo bloque sea enviado al menos una vez por algún servidor a los efectos de no perder información en las fuentes.

El módulo *mpath-bridge* (ubicado en el cliente) tiene como objetivo recoger los bloques recibidos desde un servidor. Hay una instancia del módulo *mpath-bridge* exclusivo para cada servidor.

Finalmente *mpath-client* obtiene los bloques recibidos de todos los servidores y reconstruye el flujo de vídeo. La tarea de reconstrucción consiste en ordenar en forma ascendente los bloques según su tiempo de decodificación[1] (*DTS: Decodification Timestamp*). El flujo de vídeo reconstruido puede ser reproducido localmente, enviado a disco o a la red.

[1] Tiempo absoluto en que un bloque debe ser decodificado.

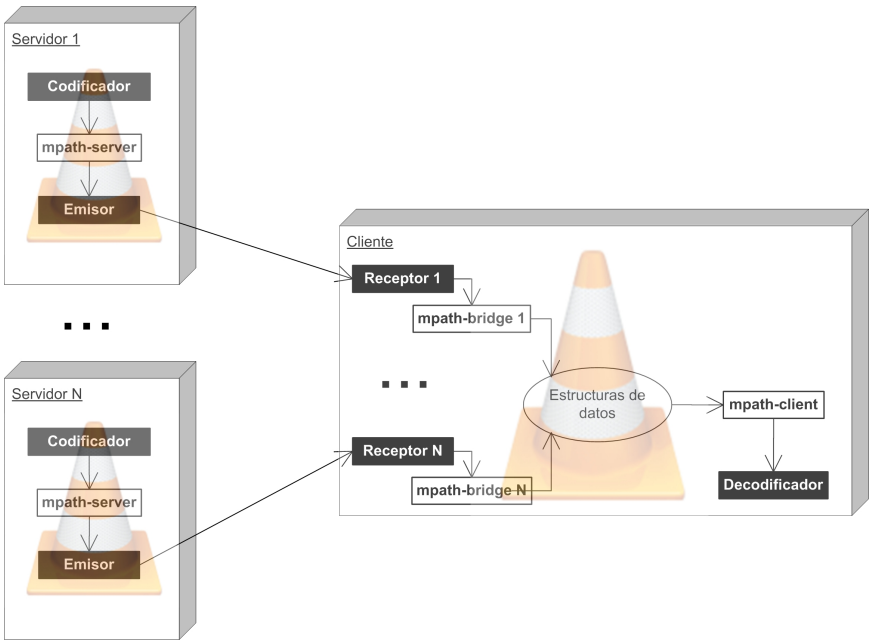

Figura 3.1: Visión global del sistema.

En el caso ideal en que todos los servidores comiencen a reproducir en el mismo instante de tiempo y ninguno se atrase o adelante, la idea anterior es viable. En la práctica ese escenario no ocurre. Notamos entonces la necesidad que tiene el cliente de saber cual es el desfasaje de tiempo entre los flujos de vídeo recibidos de los distintos servidores. Para resolver este problema introducimos el concepto de bloque redundante. Diremos que un bloque es redundante si el cliente detecta que el mismo ya lo ha recibido de algún otro servidor. Además diremos que un servidor redunda con un bloque si lo envía al cliente aunque no le corresponda. La correspondencia del envío de bloques la explicamos en 3.3.1.

3.3. Servidor

Como mencionamos anteriormente cada servidor debe escoger qué bloques del vídeo a distribuir enviará. Es claro que todos los bloques del vídeo original deben enviarse al menos una vez por algún servidor. Entre las alternativas que manejamos, destacamos dos métodos que llamamos *rotativo* y *por tipo*. Este último es el más sencillo y la idea es que cada servidor envíe solamente ciertos tipos de bloques. Con el objetivo de que todo bloque sea enviado una sola vez, cada tipo será enviado por un único servidor. En la figura 3.2 mostramos un ejemplo en donde hay tres servidores. Evidentemente este enfoque presenta muy poca escalabilidad en cuanto al número de servidores (con las codificaciones *MPEG-2* y *MPEG-4* tenemos 4 tipos de bloques) y escaso control del ancho de banda.

El método rotativo se basa en numerar cada bloque de audio y cada bloque de vídeo (en forma independiente) en orden de reproducción comenzando en *0*.

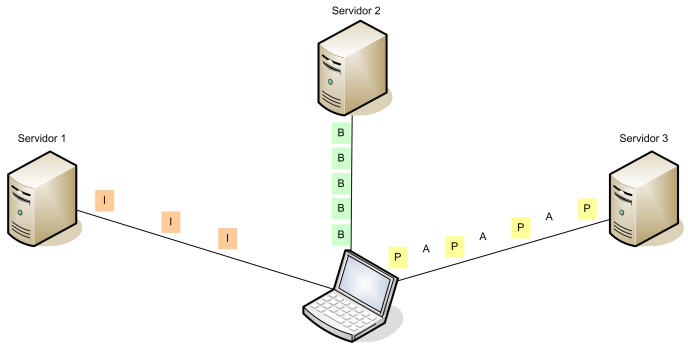

Figura 3.2: Envío de bloques usando el método por tipo.
El servidor 1 envía bloques de vídeo de tipo I; el 2 envía bloques de vídeo de tipo B y el 3 los bloques de vídeo de tipo P y bloques de audio.

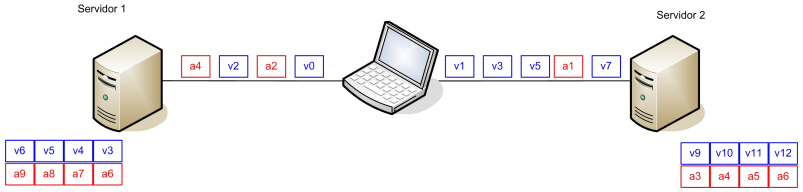

Figura 3.3: Envío de bloques usando el método rotativo.
Los servidores 1 y 2 envían los bloques pares e impares respectivamente.

De esta forma el primer bloque de audio es a_0 y el primer bloque de vídeo es v_0. Un bloque número b lo envía el servidor i (con $1 \leq i \leq N$) si se cumple la ecuación 3.1.

$$b \text{ mód } N = i - 1 \tag{3.1}$$

Como la ordenación de bloques de audio y la de vídeo es la misma en todos los servidores todo bloque se manda una sola vez. En la figura 3.3 mostramos un ejemplo del método mencionado. Notamos como ventajas que el método brinda mayor escalabilidad sobre el número de servidores, balancea la carga, pero no permite variar el ancho de banda entre un servidor específico y el cliente sin modificar la cantidad de servidores.

A continuación explicamos el método que implementamos y las ventajas frente a los anteriores.

3.3.1. Estrategia para el envío de bloques

En la práctica no es viable tratar a todos los servidores que distribuyen vídeo de la misma forma dado que no siempre son igualmente fidedignos. Hay servidores que están más propensos que otros a sufrir fallas. Por esta razón buscamos una estrategia que permita controlar el ancho de banda entre un servidor específico y el cliente. Deseamos que la cantidad de flujo de vídeo enviado por un servidor sea proporcional al grado de confiabilidad del mismo.

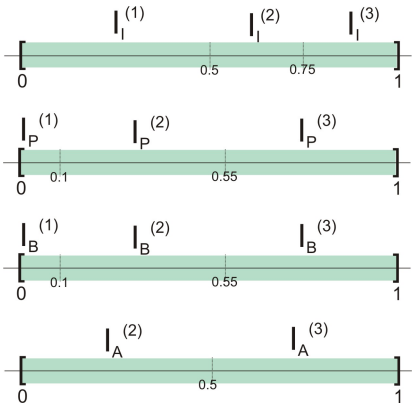

Figura 3.4: Ejemplo de partición del intervalo *[0, 1]* para el envío de frames. *Se cuenta con tres servidores. El primero envía 50 % de bloques I, 10 % de bloques P, 10 % de bloques B y ningún bloque de audio. Mientras que el segundo y el tercero envían cada uno el 25 % de bloques I, 45 % de bloques P y B y la mitad de bloques de audio.*

El método que elegimos para implementar lo denominamos *probabilístico* y la idea la explicamos a continuación. Controlamos el ancho de banda que consume el flujo de vídeo que envía un servidor en base al porcentaje de bloques de cada tipo que debe enviar. La fracción de bloques de tipo t enviados converge, en nuestro esquema, al porcentaje que queremos que el servidor transmita de ese tipo, cuando el número total de bloques enviados aumenta. De esta manera cada tipo de bloque se envía con una probabilidad igual al porcentaje que el servidor en cuestión debe enviar bloques de ese tipo. Definimos $P_j^{(i)}$ como la probabilidad de que los bloques de tipo j sean enviados por el servidor i. El número de servidor es el identificador y como mencionamos anteriormente es un entero comprendido entre 1 y N. Se debe cumplir la igualdad (3.2):

$$\sum_{i=1}^{N} P_j^{(i)} = 1 \,\mathrm{con}\, j \in \{I,\, P,\, B,\, A\} \tag{3.2}$$

Con el objetivo de que todo bloque sea enviado por un único servidor definimos $I_j^{(i)}$como un intervalo

$$\begin{cases} [a,b)\ si\ i < N \\ [a,b]\ si\ i = N \end{cases}$$

de longitud $P_j^{(i)}$. Además, decimos que $I_j^{(i)} < I_j^{(h)}$ si todo punto de $I_j^{(i)}$ es menor que todo punto de $I_j^{(h)}$. Dividimos el intervalo *[0, 1]* en intervalos $I_j^{(i)}$ de manera que se cumpla (3.3). Gráficamente equivale a pegar cada intervalo $I_j^{(i)}$a continuación uno de otro en orden ascendente en i para cada j. En la figura 3.4 se muestra un ejemplo para tres servidores.

$$(\forall i)(1 <= i \wedge i <= N \wedge I_j^{(i)} < I_j^{(i+1)}) \,\mathrm{con}\, j \in \{I,\, P,\, B,\, A\} \tag{3.3}$$

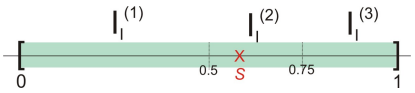

Figura 3.5: Ejemplo de envío de bloque de tipo I según la configuración de la figura 3.4.
El resultado de la variable aleatoria s pertenece al intervalo $I_I^{(2)}$. Decimos que al servidor 2 le corresponde enviar el bloque actual que es de tipo I.

Cada servidor al momento de enviar un bloque de tipo j debe sortear una variable aleatoria con distribución $U[0,1]$. Definimos s como el resultado de un sorteo. Se cumple que existe un único i tal que $s \in I_j^{(i)}$. Diremos al servidor i le corresponde enviar el bloque en cuestión. Si para un mismo bloque todos los valores de s coinciden en todos los servidores, el bloque será enviado por un único servidor. En la figura 3.5 mostramos un ejemplo para un bloque de tipo *I*.

Conseguimos que todos los resultados de los sorteos para bloques idénticos en diferentes servidores coincidan generando una secuencia de números pseudoaleatorios que comiencen con la misma semilla. En este punto distinguimos entre los bloques de audio y los de vídeo pues el orden esta definido para cada tipo de flujo. Generamos entonces una secuencia para los bloques de audio y otra independiente para los bloques de vídeo.

El método permite controlar el ancho de banda consumido por el flujo de vídeo de cierto servidor. Basta con definir el porcentaje de bloques de vídeo I, el de P, el de B y el de bloques de audio que le corresponde enviar. Permite asignarle mayor porcentaje de flujo a servidores más confiables y menos a los más vulnerables. Si lo comparamos con el método *rotativo,* para el actual balancear la carga de los servidores es una de las configuraciones posibles (las que todos mandan la misma proporción). En nuestro contexto nos interesará balancear la carga solo para el caso de que los servidores sean igualmente confiables. Es escalable en cuanto al número de servidores pues a diferencia del método *por tipo* no tiene restricciones en la cantidad.

3.3.2. Estrategia redundar con un bloque

La estrategia para el envío de bloques permite que todos los servidores envíen cada bloque una sola vez. El tiempo de decodificación (DTS) de cada bloque depende del tiempo en que el servidor comenzó a transmitir y de las condiciones en la red. El cliente, dado que ordena los N flujos recibidos según el DTS de los bloques, necesita saber el desfasaje entre los flujos recibidos de cada servidor para reproducirlos en el momento correcto. El cliente puede fácilmente conocer el desfasaje entre dos flujos si recibe el mismo bloque de dos servidores. A partir de esta observación introducimos el concepto de bloque redundante. Diremos que un bloque es redundante si el cliente detecta que el mismo ya lo ha recibido de otro servidor.

El objetivo de la estrategia para el envío de bloques redundantes es determinar que servidor (posiblemente ninguno) debe redundar con un bloque que no le corresponde enviar (por eso la hemos llamado anteriormente redundancia cruzada). Se basa en el método que explicamos en 3.3.1. Dado un bloque, cada

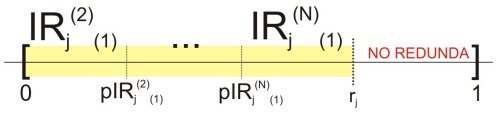

Figura 3.6: Partición del intervalo *[0, 1]* para la estrategia de redundar. *La figura muestra la partición del intervalo [0, 1] para determinar si un servidor redunda con un bloque de tipo j que le corresponde enviar al servidor número 1 de N. El punto* $pIR^{(2)}_{j\,(1)} = \frac{r_j \times P^{(i)}_j}{\sum_{k\neq 1} P^{(k)}_j}$ y $pIR^{(N)}_{j\,(1)} = \frac{r_j \times \sum_{k\neq 1 \wedge k\leq N} P^{(k)}_j}{\sum_{k\neq 1} P^{(k)}_j}$

servidor sabe qué servidor debe enviarlo. La estrategia determina cuál de los restantes debe redundar con dicho bloque. Nos interesa redundar únicamente con bloques vídeo de manera que los de audio serán enviado una única vez hacia el cliente.

Definimos r_I, r_P y r_B como el porcentaje de bloques de vídeo redundantes de tipo I, P y B respectivamente que se enviarán entre todos los servidores. En caso de que $r_j = 1$ $(\forall j)(j \in \{I, P, B\})$ cada bloque de vídeo será enviado dos veces entre todos los servidores (uno por el que le corresponda y otro por el que redunda). Como mencionamos, un servidor intentará redundar con un bloque solo si no le corresponde enviarlo. Dado un bloque de tipo *j*, contamos con una probabilidad r_j con $j \in \{I, P, B\}$ de que un servidor de los *N-1* restantes redunde con dicho bloque. Interesa saber a quien le corresponde enviar cierto bloque para no considerarlo al momento de redundar. Por cada tipo de bloque y por cada servidor que le corresponda enviar un bloque particionamos el resultado de un sorteo de una variable aleatoria $T \sim U[0, 1]$ en intervalos $IR^{(i)}_{j\,(V)}$de longitud $\frac{P^{(i)}_j \times r_j}{\sum_{k\neq V} P^{(k)}_j}$ donde j es el tipo de bloque, V es el servidor que le corresponde enviarlo e i es el servidor que redunda. Sea t el resultado de un sorteo de la variable aleatoria T; si $t > r_j$ ningún servidor redundará con el bloque, si $t \in IR^{(i)}_{j\,(V)}$ el servidor i redundará con el bloque de tipo j que le corresponde enviar al servidor V. En la figura 3.6 mostramos un ejemplo de partición del intervalo para los bloques que le corresponde enviar al servidor 1. Notar que el primer intervalo es $IR^{(2)}_{j\,(1)}$.

Nuevamente para que todos los sorteos de la variable aleatoria T sean idénticos en los distintos servidores, comenzamos la secuencia con la misma semilla. En el algoritmo 1 mostramos la lógica del módulo *mpath-server*.

3.4. Cliente

El cliente recibe flujos de vídeo de los distintos servidores y reconstruye el vídeo original para reproducirlo satisfactoriamente. Todos los bloques tienen una marca de tiempo (DTS) que indica el instante en el cual debe decodificarse. Diremos que una secuencia de bloques es correcta si la diferencia entre dos DTS de dos bloques arbitrarios es igual a la del vídeo original. Dado que los servidores comienzan a transmitir habitualmente en diferentes momentos, el DTS de los distintos bloques suele no ser correcto, no obstante la secuencia de bloques recibido por cada uno de los servidores si lo es. El objetivo final es a partir de

Algorithm 1 Envío de bloques.

```
1: Sea b el siguiente bloque a enviar;
2: if b es de tipo AUDIO then
3:    correspondencia ← Corresponde enviar audio b?;
4:    if correspondencia then
5:       enviarAudio(b);
6:    end if
7: else if b es de tipo VIDEO then
8:    correspondencia ← Corresponde enviar video b?;
9:    if correspondencia then
10:      enviarVideo(b);
11:   else
12:      redundancia ← Redundo con b?;
13:      if redundancia then
14:         enviarVideo(b);
15:      end if
16:   end if
17: end if
```

las N secuencias de bloques de audio y N de bloques de vídeo ordenarlas en dos secuencias correctas.

Como mencionamos los servidores habitualmente comienzan a transmitir en diferentes instantes y por ende los flujos que transmiten cada uno están desfasados. Tal desfasaje entre los servidores, si bien suele no ser constante, es la clave para poder reconstruir el vídeo original. El procedimiento para calcularlo lo llamamos estrategia de sincronización de flujos y lo explicamos en la siguiente sección. Una vez conocidos los desfasajes, basta con modificar correctamente las marcas de tiempo de los bloques, ordenarlos y reproducirlos.

3.4.1. Estrategia de sincronización de flujos

La sincronización de los distintos flujos de vídeo es crucial para cumplir el objetivo del proyecto. Una sincronización deficiente deteriora considerablemente la experiencia final del usuario la cual justamente queremos mejorar.

Existe un desfasaje intrínseco a cada flujo enviado por cada servidor provocado por las condiciones de la red o por los propios servidores. Para reconstruir el vídeo original a partir de los distintos flujos recibidos es necesario conocer el desfasaje entre cada servidor. Cada uno de los flujos recibidos por el cliente se identifica con un entero igual al número del servidor que lo envió.

El *DTS* es tiempo absoluto (no relativo al vídeo). La diferencia entre los *DTS* de dos bloques idénticos recibidos de dos servidores distintos indica el desfasaje entre ambos. A partir de lo anterior notamos la necesidad de guardar un histórico (que se mantiene dentro de una ventana de tiempo) de todos los bloques que ha recibido el cliente para mantener la información necesaria para calcular o recalcular los desfasajes. Utilizamos un mecanismo de comparación (que explicamos en el próximo capítulo) para determinar si dos bloques son idénticos. Por cada bloque recibido debemos saber la siguiente información:

- Identificador del bloque.

Algorithm 2 Recepción de bloques.

```
1: Sea b un bloque recibido del servidor i;
2: if b es de tipo AUDIO then
3:    almaceno(b);
4: else if b es de tipo VIDEO then
5:    redundante ← b es redundante?;
6:    if redundante then
7:       Sea DTS_j el DTS del bloque b no redundante y j el servidor;
8:       Sea DTS_i el DTS del bloque b redundante;
9:       actualizarDesfasaje(j, i, DTS_i − DTS_j)
10:   else
11:      almaceno(b, i, DTS_i);
12:   end if
13: end if
```

- Tiempo de decodificación.
- Servidor que lo ha enviado.

Cuando el cliente recibe un bloque chequeamos si el mismo lo ha recibido anteriormente. En caso afirmativo, calculamos el desfasaje que existe entre los servidores que lo enviaron (como explicamos en 3.4.1.1). De lo contrario almacenamos la información requerida para calcular a posteriori el desfasaje entre los flujos cuando detectemos que ese bloque se envió por otro servidor en forma redundante. Ver algoritmo 2.

3.4.1.1. Cálculo del desfasaje entre flujos

El procedimiento se lleva a cabo cuando el cliente recibe un bloque redundante. De esta forma cuenta con dos pares (id, DTS) correspondientes al mismo bloque recibido por dos servidores diferentes. Utilizamos una matriz $N \times N$ para almacenar los desfasajes entre los servidores que llamamos δ. La entrada $\delta_{i,j}$ indica el desfasaje entre los bloques recibidos por el flujo i y j. En otras palabras, es cuanto tiempo hay que sumarle a las marcas de tiempo de los bloques recibidos por el canal i para que sean iguales a los recibidos por el canal j. Naturalmente δ_i indicaa el desfasaje de los bloques respecto al servidor i.

La matriz cumple las propiedades que mostramos a través de las ecuaciones (3.4):

$$\begin{aligned} &\delta_{i,i} = 0 \text{ con } 1 \leq i \leq N \\ &\delta_{i,j} = -\delta_{j,i} \text{ con } 1 \leq i \leq N \wedge 1 \leq j \leq N \wedge i \neq j \\ &\delta_{i,j} = DTS_i - DTS_j \text{ con } 1 \leq i \leq N \wedge 1 \leq j \leq N \end{aligned} \tag{3.4}$$

Por cada bloque redundante podemos actualizar la matriz en la posición correspondiente. Analizando en detalle la matriz notamos que existe otra propiedad que nos ayudará a realizar el proceso de descubrimiento de los desfasajes entre los flujos de vídeo más eficientemente. Las filas i y j son iguales a menos de una constante igual al desfasaje entre los servidores i y j . De esta forma basta con conocer una entrada (i, j) con $i \neq j$ para cada servidor para poder calcular

Algorithm 3 Cálculo del desfasaje entre flujos.

Require: Conocer al menos una entrada para cada servidor;

while hay entradas desconocidas **do**
  for cada fila i **do**
    Sea j una entrada desconocida en la fila i
    {Se desconoce el desfasaje entre el servidor i y j}
    Sea h una entrada conocida en la columna j
    Calculo $\delta_{h,i}$ ente las filas i y h
    if $\neg desconocido(\delta_{h,i})$ **then**
      Calculo entradas desconocidas en h a partir de i sumando $-\delta_{h,i}$;
      Calculo entradas desconocidas en i a partir de h sumando $\delta_{h,i}$;
    end if
  end for
end while

la matriz completa. Definimos $1_{(M,N)}$ como la matriz de dimensión $M \times N$ con todas sus entradas 1 y obtenemos así la igualdad (3.5).

$$\delta_i = \delta_j + \delta_{j,i} * 1_{(1,N)} \tag{3.5}$$

Una vez conocida una entrada para cada servidor, es posible completar la matriz δ a través del el algoritmo 3. Dado que los desfasajes entre los flujos suelen variar, al cambiar una entrada en la matriz δ, la misma es recalculada convergiendo rápido al valor real.

3.4.1.2. Ventajas y desventajas

Como principales ventajas del método encontramos:

- El método no utiliza señalización externa para la sincronización.
- La redundancia en el contexto de Internet puede ser un elemento positivo durante períodos de baja congestión, como forma de abordar pérdidas en la red o fallas en los servidores de vídeo.

El procedimiento cuenta con la necesidad de cálculo extra para identificar los frames repetidos.

3.4.2. Reconstrucción del flujo original

A partir de los desfasajes y la secuencia de bloques recibidos de los diferentes servidores reconstruimos el flujo del vídeo original. Necesitamos algún mecanismo para utilizar un servidor como referencia y de esa forma reproducir al tiempo que el mismo indica en los bloques que envía. Para asegurarnos que a ese tiempo se han recibido todos los bloques de los restantes servidores, elegimos el más atrasado. A partir de la matriz δ, el mismo es para cualquier i el $\max_j\{\delta_{i,j}\}$. En caso de que no se conozca toda la matriz δ, i es la fila más confiable[2].

[2]La que tiene más entradas conocidas (o válidas).

Capítulo 4

Implementación

A continuación describimos decisiones de implementación, librerías y estructuras de datos utilizadas en el sistema. Los tres módulos que describimos anteriormente son incorporados a VLC como explicamos en el apéndice A.

4.1. Servidor

Como describimos anteriormente el servidor decide que bloques serán enviados al cliente. El procedimiento se realiza a través del sorteo de números pseudoaleatorios. Con el objetivo de mantener los mismos sorteos en cada uno de los servidores, utilizamos tres secuencias de números pseudoaleatorios independientes. La función *rand()* de la librería *stdlib.h* genera una única secuencia de números pseudoaleatorios con distribución uniforme una vez que definimos la semilla con *srand()*. Dado que necesitamos más de una secuencia, utilizamos *rand_r(seed)* [38] (definida en la misma librería) que a diferencia de *rand()* es reentrante. La función recibe como parámetro la semilla y a partir de ella genera el próximo número de la secuencia. En el servidor mantenemos el estado de cada una de las semillas utilizadas (audio, vídeo y redundancia). Las funciones *rand()* y *rand_r(seed)* devuelven un número entre 0 y $RAND\_MAX$ el cual normalizamos dividiendo entre este último. El período de la secuencia generada es 2^{32}.

Para evitar problemas de aproximación en la definición de la proporción de bloques que envía cada servidor, utilizamos una función que se encarga de normalizar la proporción de bloques de cierto tipo que envía cada uno de los servidores dividiendo el valor definido por la suma de todos los servidores. Es decir, el servidor i enviará $\frac{P_j^{(i)}}{\sum_{k=1}^{N} P_j^{(k)}}$ de bloques de tipo j *con* $j \in \{I, P, B, A\}$.

La representación de los intervalos que explicamos en la sección 3.3 para el envío de bloques la realizamos de una estructura de datos de tipo matriz que llamamos π. La misma es de tamaño $B \times N$ donde B es la cantidad de tipos de bloques y N la cantidad de servidores. Cada una de las filas indica el la cota superior del intervalo que le corresponde al servidor para cada tipo de bloque. Sea s el resultado de un sorteo con distribución $U[0,1]$, al servidor i le corresponde enviar el bloque de tipo j si se cumple que $\pi_{i-1,j} < s \leq \pi_{i,j}$. Definimos $\pi_{0,j}$ como 0.

De la misma forma que para la matriz π, tenemos N matrices de redundancia que llamamos ρ. La matriz ρ_V indica las probabilidades de redundancia para un bloque sabiendo que al mismo le corresponde enviarlo al servidor V. El procedimiento para determinar que servidor debe redundar es análogo al caso del envío por correspondencia.

4.2. Cliente

El cliente maneja una secuencia de canales a través de los cuales reciben bloques instancias del módulo *mpath-bridge*. La numeración de los canales depende del orden en que se presenten los servidores. En este punto al cliente solo le interesa saber a través de qué canales recibió un bloque dado. Es decir, no le interesa el servidor que le está enviando la información aunque se puede determinar fácilmente. Las distintas instancias de *mpath-bridge* y la instancia *mpath-client* comparten una estructura de datos en la que se almacenan los bloques no redundantes recibidos por los distintos canales. El acceso a la estructura compartida la realizamos a través del mecanismo de exclusión mutua *spinlock* [40] utilizando las primitivas definidas en las librerías de *VLC*.

A partir de los bloques recibidos por los distintos canales, la instancia del módulo *mpath-client* genera un flujo de audio y un flujo de vídeo. Las secuencias de bloques recibidas por los distintos canales se sincronizan al tiempo del servidor más atrasado. De esta forma sabemos que tenemos todos los bloques para reproducir al tiempo del servidor más atrasado. Esto en realidad no se cumple estrictamente pues depende de a qué instancia del *mpath-brdige* se le ha asignado procesador. Utilizamos el parámetro Δ (delay) que recibe el cliente para esperar el tiempo indicado en el mismo. Es decir para el tiempo t en realidad se reproduce $t - \Delta$. Observamos que para $\Delta > 100\,ms$ el sistema funciona correctamente. Para tiempos menores el vídeo reconstruido puede diferenciarse un poco del original degradando levemente la calidad de imagen percibida.

El proceso de sincronización al servidor más atrasado consiste en sumar a las marcas de tiempo de todos los bloques el desfasaje que existe entre el servidor que lo mandó y el más atrasado. Sea μ el servidor más atrasado e i el servidor al cual pertenece el bloque β a sincronizar. El tiempo del bloque β en el flujo del servidor μ es $DTS_\beta = DTS_\beta + \delta_{\mu,i}$. El servidor más atrasado es para cualquier fila ϕ de la matriz δ es aquel que tiene menor valor $\tilde{\mu} = \text{mín}_\mu\{\delta_{\phi,\mu}\}$. En caso que δ no esté completa, ϕ es la fila más confiable.

En un momento dado no siempre se ha recibido bloques del servidor μ. El proceso a través del cual se espera a que se reciba lo llamamos *rebuffering*. Dado que podríamos esperar por él indefinidamente, lo que hacemos es lo siguiente. Si no recibimos nada por el canal durante un tiempo menor a $MAX\_TIME\_REBUFFERING$, esperamos sin reproducir vídeo. De lo contrario reproducimos al tiempo de μ aunque no hayan bloques de ese servidor. La idea es no congelar la imagen e independizarse del flujo más atrasado. Con un valor de $MAX\_TIME\_REBUFFERING$ igual $10ms$ a se reproduce vídeo satisfactoriamente.

Usualmente los servidores suelen atrasarse o adelantarse en algunos microsegundos alterando el valor de los DTS que envían. Eso provoca que la matriz δ se recalcule innecesariamente pues tal desfasaje no altera la reproducción final. Para evitar la actualización de la entrada y el posible recálculo de la matriz

δ, utilizamos un umbral en el cual no se modifica la entrada en la matriz. Al umbral lo llamamos $TOLERANCE$. El cambio se ve reflejado en la matriz si la distancia entre la nueva entrada y la vieja es mayor a $TOLERANCE$.

Como mencionamos en el capítulo anterior, necesitamos un mecanismo de comparación de bloques para determinar si dos son idénticos. Además necesitamos guardar un histórico de los bloques no redundantes que hayan llegado al cliente. La estructura de datos utilizada para el histórico es tabla de hash (utilizamos la implementación de la librería de VLC) que se mantiene en una ventana de tiempo de $TIME\_SLIDE$ microsegundos. Para identificar los bloques guardamos en el histórico su $MD5$ (*Message-Digest algorithm 5* [39]). $MD5$ permite identificar bloques arbitrariamente grandes con 128 bits y para determinar si dos bloques contienen la misma información basta con comparar sus $MD5$ (32 dígitos hexadecimales).

Como vimos en el capítulo anterior, al recibir un bloque no redundante, se almacena información en el histórico la cual permitirá a posteriori calcular el desfasaje entre los servidores. Lo que realmente se almacena en el histórico es:

- $MD5$ del bloque (o identificador del bloque).
- Tiempo de decodificación del bloque.
- Canal por el que se recibió el bloque (permite conocer el servidor).

Una vez que todos los canales se han cerrado, el módulo *mpath-client* libera la memoria compartida y muestra la imagen recibida como parámetro en su invocación. La imagen se envía a través del módulo *fake* [43] de VLC que se encarga de mostrar la imagen. El procedimiento es el mismo que se utiliza en el proyecto *mosaic [42]*.

Para el reenvío de los flujos en nodos de la red entre pares, basta con configurar adecuadamente los módulos *mpath-bridge* en el archivo de configuración de los clientes (ver apéndice C). De esta forma se consigue armar una red entre pares.

Capítulo 5

Pruebas

En esta sección exponemos las pruebas que realizamos y los resultados que obtuvimos para evaluar la correctitud y aplicabilidad del sistema en un escenario real.

5.1. Correctitud

El objetivo de las pruebas de correctitud es evaluar si el sistema funciona como esperamos. Analizamos si los servidores envían la proporción de cada tipo de bloque especificado en el archivo de configuración. Además analizamos que el ancho de banda consumido por los servidores corresponde con la proporción de vídeo especificada en el archivo de configuración. En cuanto al cliente, ratificamos que recibe lo enviado por los servidores y lo procesa correctamente.

A continuación analizamos la proporción de cada tipo de bloques que cada uno de los servidores envía al cliente. La proporción de los bloques de cada tipo lo calculamos como la razón entre la cantidad de bloques enviado y el total de bloques de ese mismo tipo en el vídeo original. Cada servidor despliega estadísticas sobre lo que envía, ya sea por correspondencia o por redundancia. Invocando a *VLC* con la opción *-vvv*, al momento de enviar cada bloque se imprime en pantalla un mensaje con información sobre el tipo y tamaño en bytes. A través del programa awk[1] que mostramos en el apéndice E obtenemos por cada bloque de tipo j la cantidad enviada ($\#bloques_j$) y el tamaño promedio en bytes $|bloque_j|$. De esta manera podemos comparar la proporción que realmente envió con la especificada en el archivo de configuración.

Para el consumo de ancho de banda, tendremos en cuenta la razón entre ancho de banda consumido y la tasa de transferencia del vídeo original. Tal proporción debe asimilarse a un valor teórico que obtenemos con los datos del archivo de configuración. El ancho de banda real consumido por cada servidor lo obtenemos a través de las estadísticas que brindan los servidores y a través del programa *iftop [41]*.

Las estadísticas nos permiten calcular una medida de ancho de banda consumido sabiendo la duración del vídeo. A través de las estadísticas, el ancho de banda consumido por cada tipo de bloque j lo calculamos como mostramos en

[1] Lenguaje de programación particularmente diseñado para filtrar y manipular texto [19].

la ecuación (5.1) donde $\#bloques\_enviados_j$ es la cantidad de bloques enviados y $|bloques\_enviados_j|$ es el tamaño promedio.

$$bw_j = \frac{\#bloques\_enviados_j \times |bloques\_enviados_j|}{duración} \tag{5.1}$$

La correctitud en el cliente la determinamos a través de un evaluador visual el cual consiste en un usuario final que evalúa la calidad de la imagen y sonido percibida. Calificaremos la experiencia final del usuario con un valor entre *0* y *10*.

En la siguiente sección mostramos los casos de pruebas evaluados y sus resultados.

5.1.1. Casos de pruebas y resultados

En esta sección mostramos los resultados de los casos de pruebas. El escenario para todas las pruebas es:

- Medio y método de encapsulamiento ($HTTP/TS$).
- Audio (Se enviará por un único servidor).

Nos interesa variar los siguientes parámetros:

- Cantidad de servidores.
- Porcentaje de correspondencia de cada tipo de bloque por servidor.
- Porcentaje de redundancia en la red.
- Semillas de los sorteos.
- Archivos de vídeo.

5.1.1.1. Redundancia

La proporción de vídeo que los servidores en conjunto deben enviar como redundantes la definimos en el archivo de configuración. Para cada tipo de bloque de vídeo especificamos el porcentaje.

En las gráficas 5.1, 5.2 y 5.3 mostramos comparativamente el porcentaje de bloques de tipo I, P y B respectivamente para cada una de las 16 configuraciones que mostramos en las tablas E.2, E.3, E.4, E.5 y E.6 del apéndice E. En cada caso tenemos:

- El definido en el archivos de configuración y que llamaremos teórico.
- El porcentaje de bloques de vídeo redundantes enviados al reproducir el vídeo *IceAgeIntro.mpg*.
- El porcentaje de bloques de vídeo redundantes enviados al reproducir el vídeo *IceAgeTrailer.mpg*.

Para cada una de las configuraciones observamos que la redundancia global del vídeo en la red al reproducir *IceAgeIntro.mpg* e *IceAgeTrailer.mpg* es similar en todos los casos a la teórica definida en los archivos de configuración de los servidores. Concluimos que el método para redundar explicado en 3.3.2 funciona como esperamos.

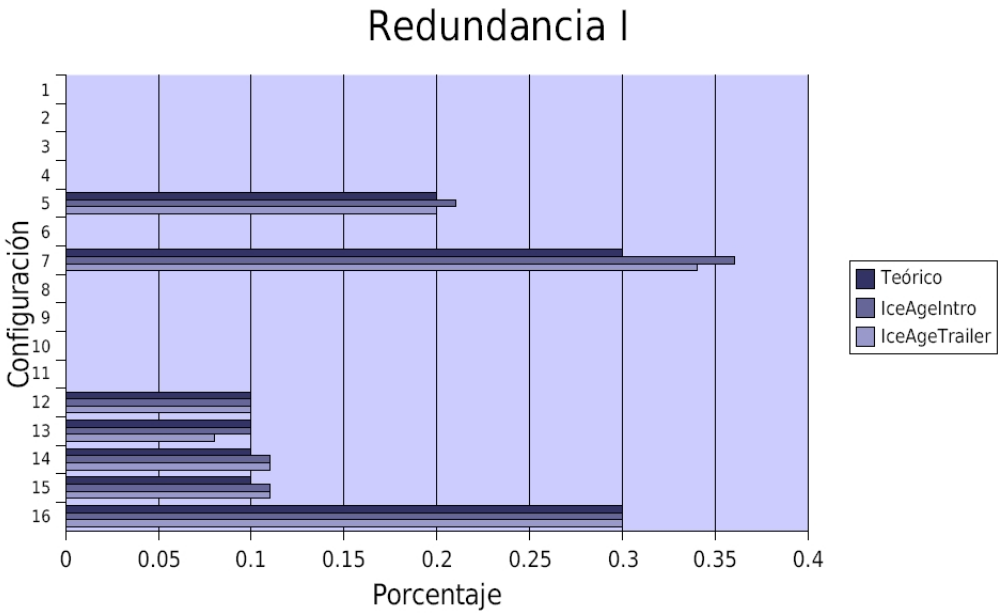

Figura 5.1: Redundancia para bloques de tipo I.

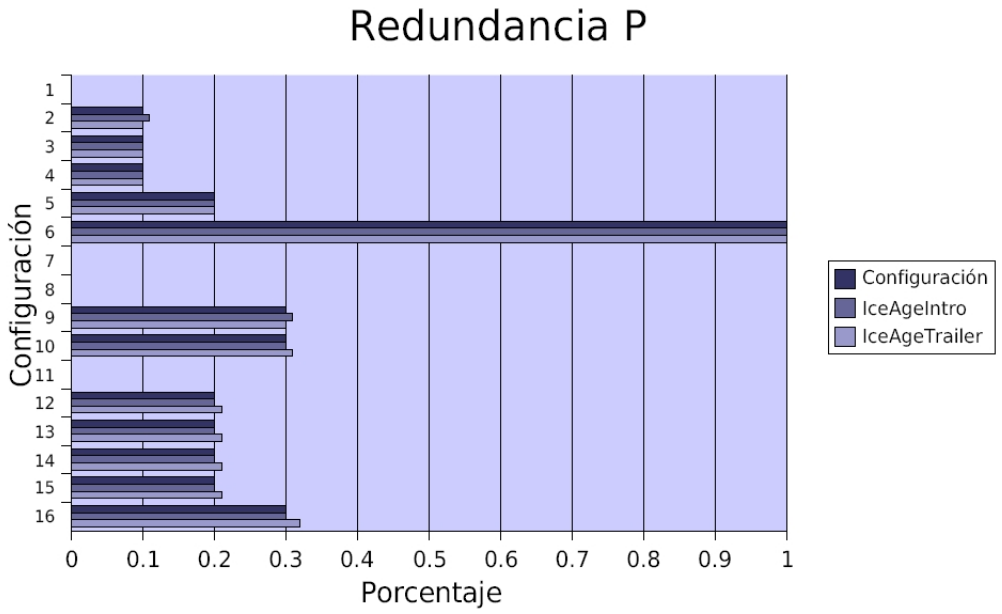

Figura 5.2: Redundancia para bloques de tipo P.

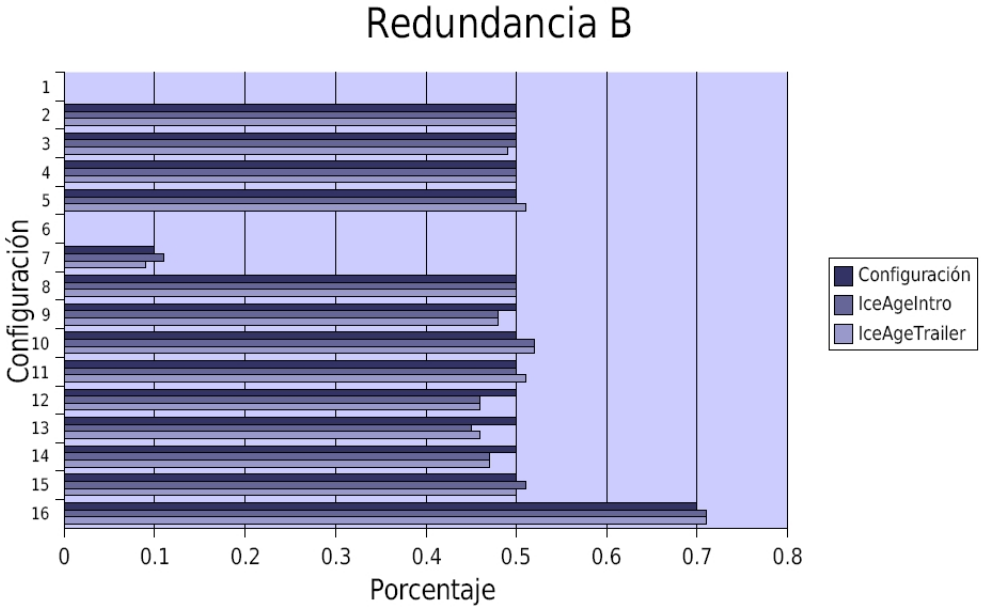

Figura 5.3: Redundancia para bloques de tipo B.

5.1.1.2. Ancho de banda

Utilizamos el programa *iftop* para obtener el ancho de banda consumido de la red por cada conexión. El ancho de banda consumido por cada servidor, según las estadísticas, lo obtenemos sumando la ecuación (5.1) para cada tipo de bloque, y el ancho de banda teórico para cada servidor se obtiene a través de la ecuación (5.2) donde *duracion* tiempo de reproducción del vídeo en estudio, $\#bloque_j$ es la cantidad de bloques en el vídeo original y $|bloque_j|$ es el tamaño promedio, c_j es el porcentaje de bloques enviados por correspondencia, r_j es la redundancia (para bloques de tipo j).

$$BW_{teórico} = \frac{\sum_{j \in \{I,P,B\}} (c_j + c_j \times r_j) \times |bloque_j| \times \#bloque_j}{duración} \tag{5.2}$$

En las gráficas 5.4, 5.5, 5.6 y 5.7 mostramos comparativamente el ancho de banda obtenido a través del programa *iftop*, el obtenido a través de estadísticas de los servidores y el teórico para las configuraciones mostradas en el apéndice E para los vídeos *IceAgeIntro.mpg* e *IceAgeTrailer.mpg*.

El ancho de banda obtenido con *iftop* se obtiene para un instante del vídeo y es el promedio de los $40s$ anteriores al momento en que se toma. Para los casos en que el servidor envía audio, el ancho de banda que observamos con *iftop* es mayor al resultado teórico.

Observamos que las gráficas se ajustan en general a los resultados teóricos esperados utilizando los cálculos a través de las estadísticas de los servidores y el programa *iftop*.

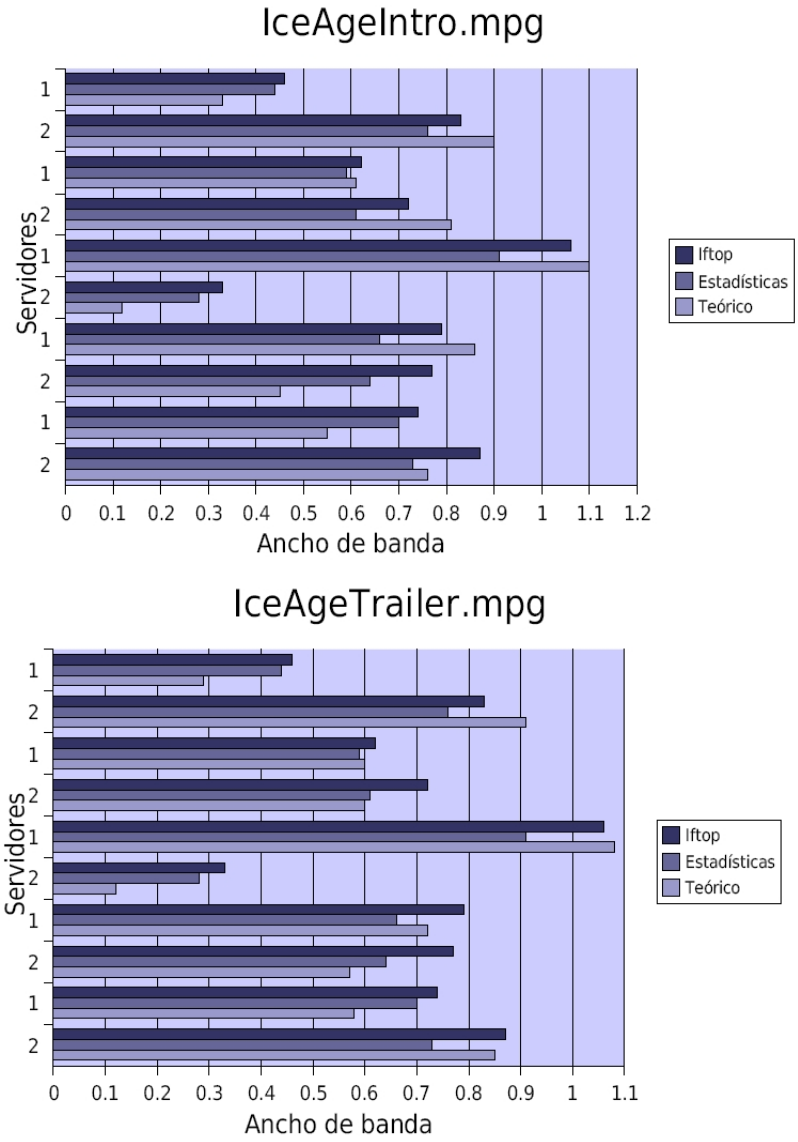

Figura 5.4: Ancho de banda consumido para dos servidores.
En las *gráficas mostramos comparativamente el ancho de banda enviado a la red, el obtenido con las estadísticas de los servidores y el valor teórico para los videos IceAgeIntro.mpg e IceAgeTrailer.mpg utilizando dos servidores.*

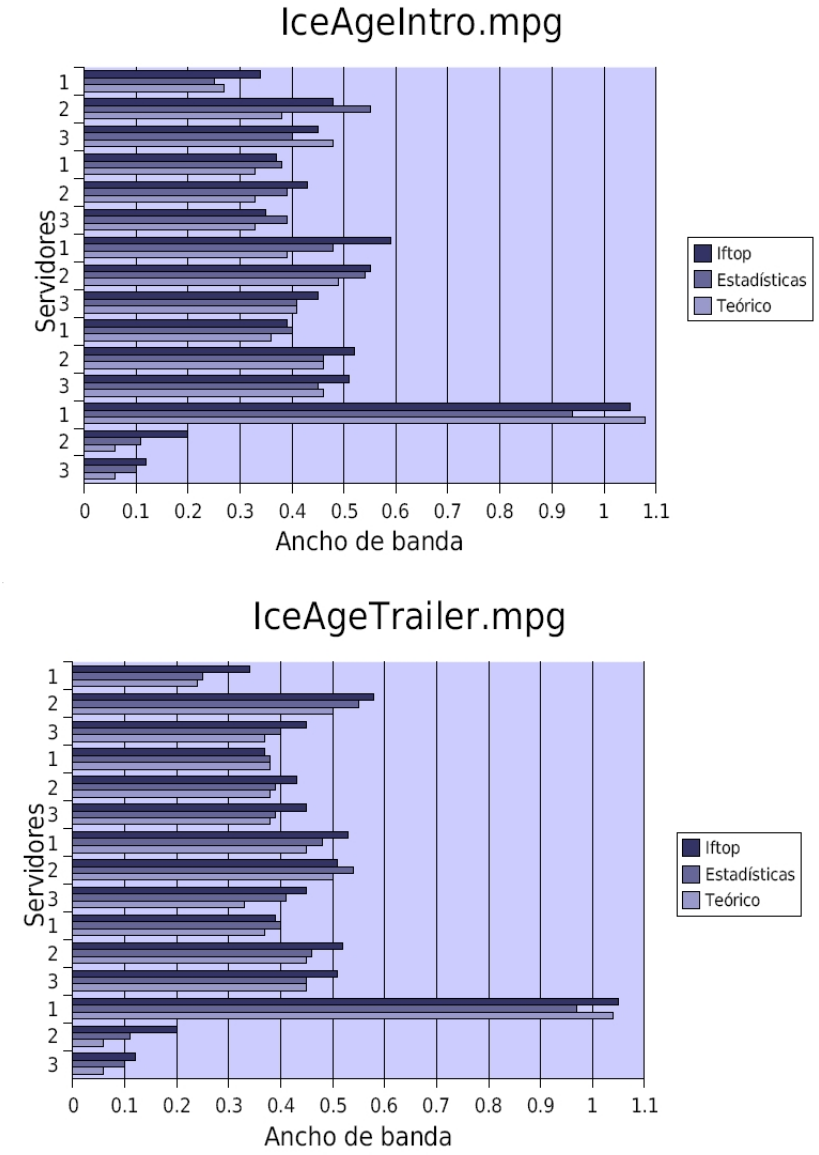

Figura 5.5: Ancho de banda consumido para tres servidores.
En las *gráficas mostramos comparativamente el ancho de banda enviado a la red, el obtenido con las estadísticas de los servidores y el valor teórico para los videos IceAgeIntro.mpg e IceAgeTrailer.mpg utilizando tres servidores.*

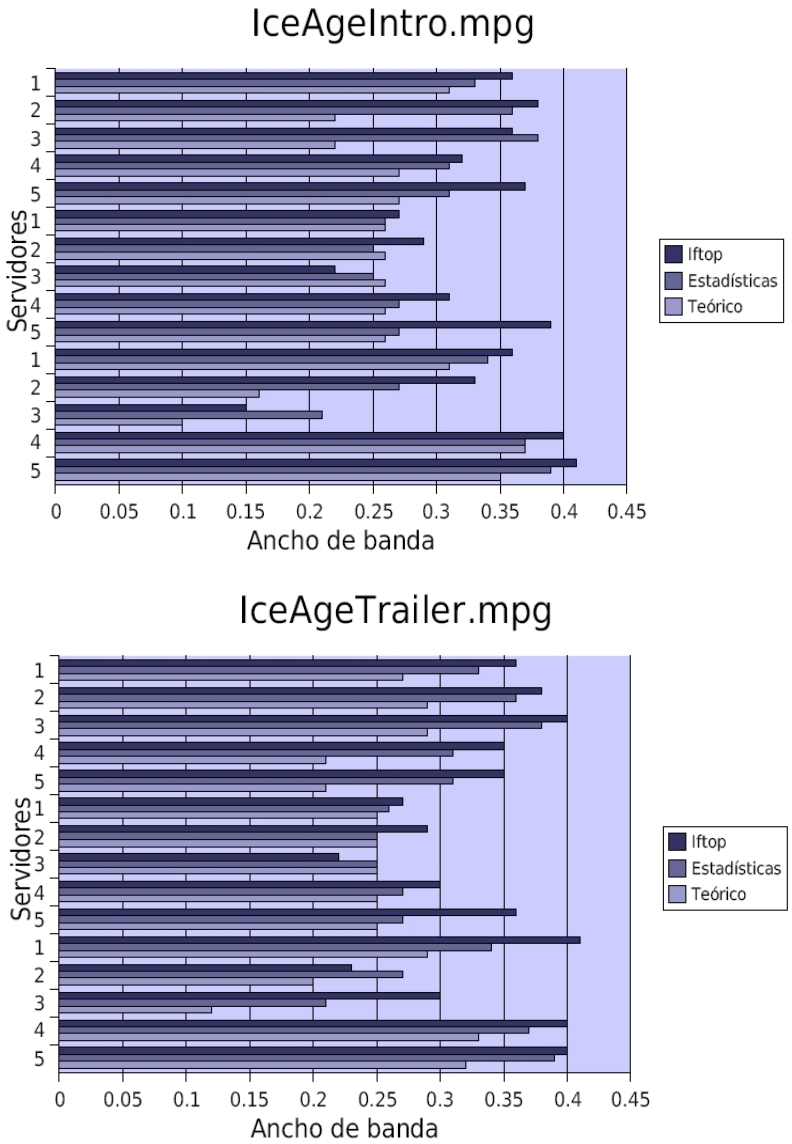

Figura 5.6: Ancho de banda consumido para cinco servidores.
En las *gráficas mostramos comparativamente el ancho de banda enviado a la red, el obtenido con las estadísticas de los servidores y el valor teórico para los videos IceAgeIntro.mpg e IceAgeTrailer.mpg utilizando cinco servidores.*

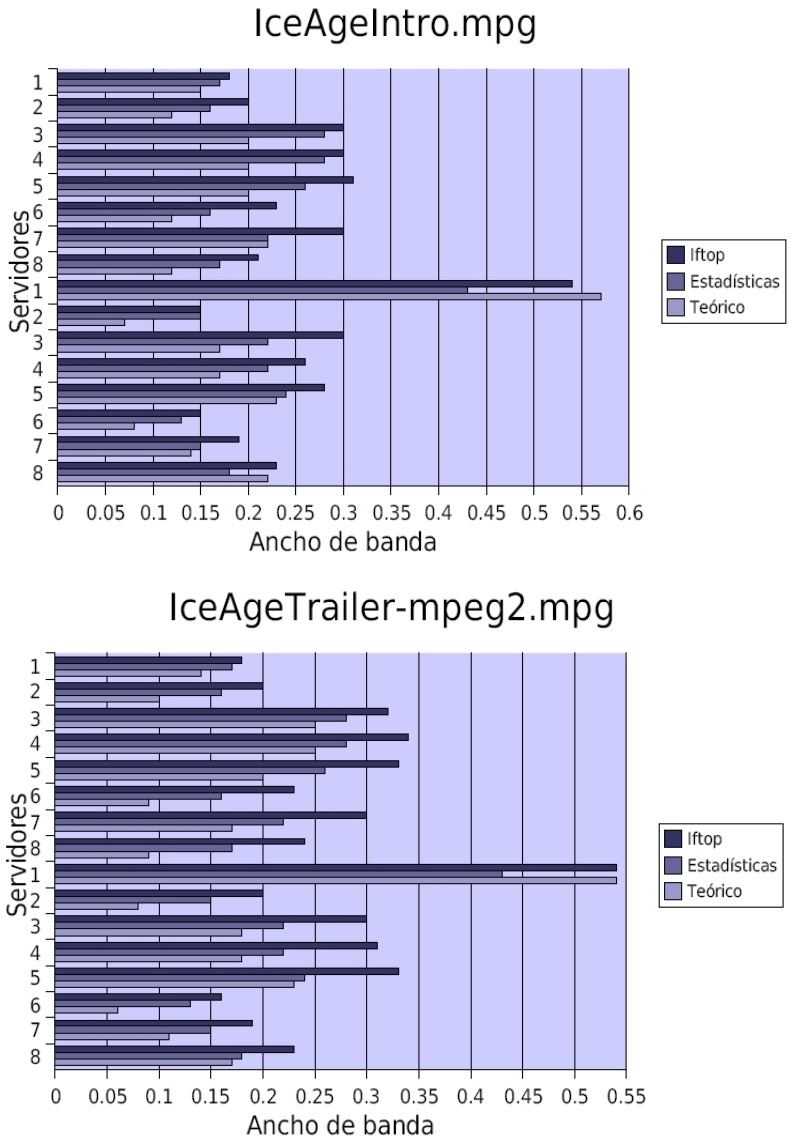

Figura 5.7: Ancho de banda consumido para ocho servidores.
En las *gráficas mostramos comparativamente el ancho de banda enviado a la red, el obtenido con las estadísticas de los servidores y el valor teórico para los videos IceAgeIntro.mpg e IceAgeTrailer.mpg utilizando ocho servidores.*

Archivo de configuración	IceAgeIntro.mpg	IceAgeTriller.mpg
1Server-1.conf	10	10
2Servers-1.conf	10	10
2Servers-2.conf	10	9
2Servers-3.conf	9	9
2Servers-4.conf	10	10
2Servers-5.conf	10	10
3Servers-1.conf	10	9
3Servers-2.conf	9	10
3Servers-3.conf	10	10
3Servers-4.conf	10	9
3Servers-5.conf	9	9
5Servers-1.conf	9	10
5Servers-2.conf	10	9
5Servers-3.conf	10	10
8Severs-1.conf	9	10
8Servers-2.conf	10	10

Cuadro 5.1: Calidad percibida por un evaluador visual.

5.1.1.3. Evaluador visual

La correctitud en el cliente la determinamos a través de un evaluador visual el cual consiste en un usuario final que evalúa la calidad de la imagen y sonido percibida. Calificaremos la experiencia final del usuario con un valor entre *0* y *10*. Las calificaciones son:

10 Excelente

7 Bueno

5 Regular

3 Pobre

0 Malo

En 5.1 mostramos para las distintas configuraciones, la calidad percibida por un usuario final.

Los resultados obtenidos a través de un evaluador visual muestran que en casi todos los casos la calidad de la imagen y sonido son excelentes. Concluimos que el cliente sincroniza correctamente los distintos flujos recibidos.

5.2. Desempeño en una arquitectura cliente servidores

El objetivo de las pruebas de desempeño en una arquitectura *Cliente-Servidores* es mostrar la utilidad del sistema en escenarios de fallas. La principal motivación del proyecto es mejorar la calidad percibida por el usuario final. En base a ésto mostramos que, cuando ocurren fallas en los servidores, la calidad percibida por

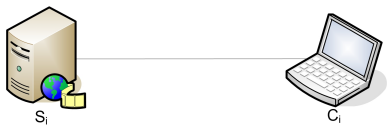

Figura 5.8: Distribución desde una única fuente con i entre 1 y 4.

el usuario final es superior cuando se usa *distribución desde múltiples fuentes* que cuando se recibe flujo de vídeo desde un único servidor.

Para las pruebas disponemos de cuatro servidores en los dos siguientes escenarios:

- Distribución desde una única fuente
- Distribución desde múltiples fuentes

Los servidores son idénticos a menos de su vulnerabilidad a fallas. Debido a la baja probabilidad de que tres o los cuatro servidores fallen mientras se distribuye un vídeo, no tenemos en cuenta esos casos. Llamamos *S1* y *S2* a los servidores no vulnerables. La fallas de los servidores *S3* y *S4* las simulamos a través de un módulo que llamamos *dropper* el cual fue implementado con ese fin. En el apéndice D describimos su funcionamiento. Los casos de pruebas que usamos están compuestos por una secuencia de pares (t_i, d_i) para cada servidor vulnerable, donde t_i es el tiempo en que se produce una falla y d_i es la duración de la misma.

Definimos q_i como la calidad percibida por el cliente i. Los casos de prueba los efectuamos con diferentes vídeos en distintas codificaciones (*MPEG-2* y *MPEG-4*). Como en las pruebas de correctitud, utilizamos (*HTTP/TS*). En 5.2.1 y 5.2.2 describimos las pruebas y los resultados para la distribución desde una única y desde múltiples fuentes. En 5.2.3 comparamos los resultados.

5.2.1. Desempeño desde una única fuente

En esta sección mostramos los casos de pruebas que llevamos a cabo para la distribución de vídeo desde una única fuente. Como mostramos en la figura 5.8, disponemos de cuatro servidores y cuatro clientes que reciben flujo de un único servidor. Suponemos que los servidores que no fallan distribuyen vídeo en forma satisfactoria de manera que $q_i = 10$ con $1 \leq i \leq 4$. Evaluamos únicamente los clientes que su respectivo servidor falla.

Por cada caso de prueba, la calificación final es el promedio de la calidad final percibida por cada cliente.

$$Calidad = \frac{\sum_{i=1}^{4} q_i}{4}$$

En la tabla 5.2 mostramos los casos de pruebas y los resultados obtenidos.

5.2.2. Desempeño desde múltiples fuentes

El escenario de pruebas es el que mostramos en la figura 5.9. El envío de bloques (audio y vídeo) la realizan todos los servidores en forma equitativa.

Fallas del servidor S3	*Falla del servidor S4*	q_3	q_4	*Calidad*
-	-	10	10	10
(25, 1)	-	6	10	9
(25, 3)	-	4	10	8.5
(25, 1); (30, 3)	-	3	10	8.25
(25, 3); (30, 3)	-	2	10	8
(25, 1)	*(25, 1)*	6	6	8
(25, 3)	*(25, 3)*	4	4	7
(25, 3)	*(28, 1)*	4	6	7.5
(25, 3)	*(28, 3)*	4	4	7
(25, 3)	*(27, 3)*	4	4	7
(25, 1); (30, 3)	*(26, 1)*	3	6	7.25
(25, 1); (30, 3)	*(26, 1); (29, 3)*	3	3	6.25

Cuadro 5.2: Casos de pruebas para distribución desde una única fuente.

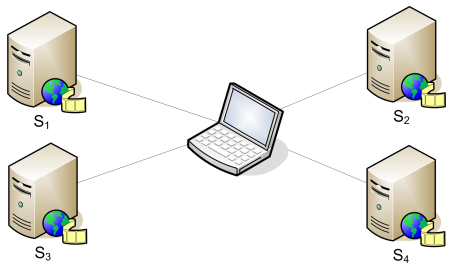

Figura 5.9: Distribución desde múltiples fuentes.

Como mostramos en la figura 5.9, disponemos de cuatro servidores y cuatro clientes que reciben flujo de todos los servidores.

En la tabla 5.3 mostramos los casos de pruebas y los resultados obtenidos.

5.2.3. Evaluación comparativa

A continuación presentamos en forma gráfica (figura 5.10) los resultados obtenidos. Observamos que en situaciones de fallas en servidores la calidad percibida por un usuario final es mayor cuando se utiliza el método de distribución desde múltiples fuentes con 20 % de redundancia. Además a medida que aumentamos el porcentaje de frames redundantes, la calidad aumenta considerablemente. Es de esperarse que también mejore la calidad si aumentamos la cantidad de servidores.

5.3. Desempeño en una red entre pares

El objetivo de las pruebas de desempeño en una red entre pares es mostrar que los clientes (nodos en este contexto) pueden distribuir selectivamente los flujos recibidos de otros nodos de la red. Para las pruebas utilizamos las configuraciones del apéndice E para tres servidores. En la figura 5.11 mostramos la red entre pares utilizada.

Fallas del servidor S3	*Falla del servidor S4*	Calidad (r = 0.2)	Calidad (r = 0.4)
-	-	10	10
(25, 1)	-	9.5	9.5
(25, 3)	-	9	9
(25, 1); (30, 3)	-	8.5	9
(25, 3); (30, 3)	-	8	8.5
(25, 1)	*(25, 1)*	9	9.5
(25, 3)	*(25, 3)*	8	9
(25, 3)	*(28, 1)*	8.5	9
(25, 3)	*(28, 3)*	8	9
(25, 3)	*(27, 3)*	8	9
(25, 1); (30, 3)	*(26, 1)*	7.5	8.5
(25, 1); (30, 3)	*(26, 1); (29, 3)*	7	8.5

Cuadro 5.3: Casos de pruebas para distribución desde múltiples fuentes.

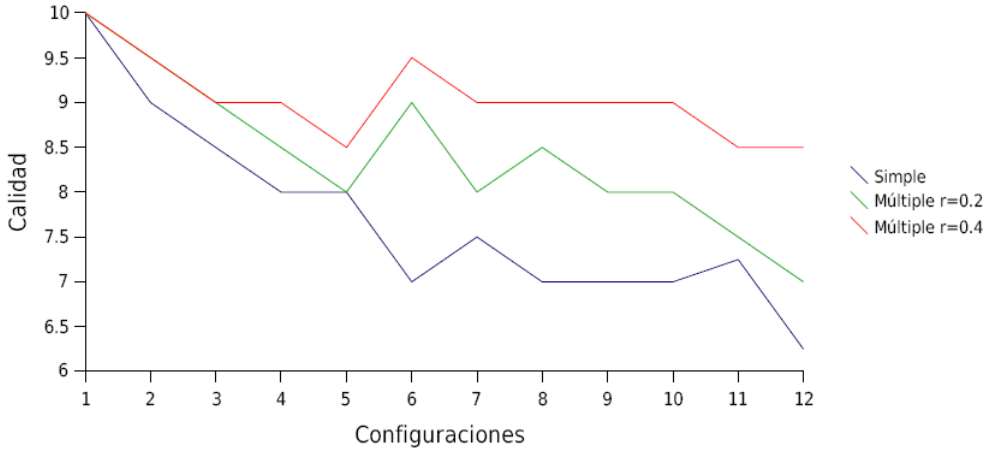

Figura 5.10: Desempeño en una arquitectura *Cliente-Servidor*.

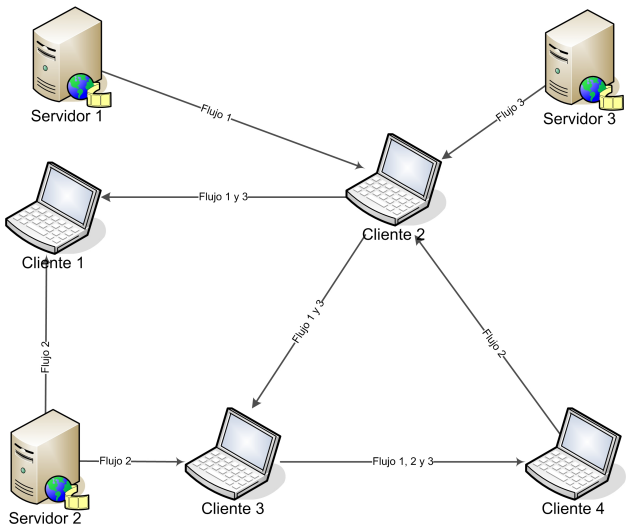

Figura 5.11: Red entre pares.

Configuración	IceAgeIntro.mpg	IceAgeTrailer.mpg
3Servers-1.conf	10	9.75
3Servers-2.conf	9.5	10
3Servers-3.conf	10	9.5
3Servers-4.conf	10	10
3Servers-5.conf	9.25	10

Cuadro 5.4: Calidad percibida por un evaluador visual en una red entre pares.

Nuevamente utilizamos los vídeos *IceAgeIntro.mpg* e *IceAgeTrailer.mpg* para evaluar la calidad de ellos en los nodos de la red entre pares. En la tabla 5.4 mostramos los resultados de calidad, tomando el valor promedio de los obtenidos a través de un evaluador visual en cada uno de los nodos.

Para cada una de las configuraciones observamos que el nivel de calidad es muy satisfactorio.

Capítulo 6

Conclusiones

Cumplimos con el objetivo y disponemos de un sistema de distribución de video que permite distribuir vídeo en vivo desde múltiples fuentes totalmente configurable en cuanto al consumo de ancho de banda de cada una de las fuentes. El sistema consiste en un conjunto de módulos *GNU* para *VLC* que permiten adaptarse a distintos protoclos de transporte, a distintos métodos de encapsulamiento y a distintos codificadores de la familia *MPEG*. Considerando las limitaciones que presenta la distribución de contenido multimedia en contextos de fallas como Internet, el sistema permite definir el nivel de redundancia del vídeo para reducir el impacto ante pérdidas.

El sistema esta apto utilizarse en distintos escenarios. En una red entre pares permite reducir el impacto ante desconexiones de los pares. La principal restricción en una *red entre pares* es el ancho de banda disponible en las fuentes (ancho de banda de subida) lo cual en nuestro sistema es totalmente configurable. En una *CDN* da grados de libertad en la elección de los servidores, brinda robustez ante fallas simples en la red y en las propias fuentes.

En escenarios de laboratorio (donde no existen fallas en servidores ni en la red) la calidad percibida comparando el método tradicional y el implementado es la misma una vez que los flujos de los distintos servidores se han sincronizado. El tiempo de sincronización depende de la cantidad de redundancia que los distintos servidores envían. Para disminuir el tiempo previo a la sincronización de los flujos en el cliente sin variar el consumo de ancho de banda luego de la sincronización, el servidor al comenzar la reproducción envía una ráfaga del vídeo original para que el cliente conozca su desfasaje rápidamente.

La calidad percibida por un usuario final, en contextos de fallas en los servidores, es mayor cuando utilizamos la estrategia de distribución implementada. Teniendo experiencia en contextos de fallas, es posible definir una configuración óptima variando en nivel de correspondencia de cada servidor, la cantidad de servidores y la redundancia global; de modo de obtener la calidad del vídeo original a el menor consumo posible de ancho de banda. En la práctica es natural asignarle a los servidores más vulnerables menor envío por correspondencia.

La situación en que ocurren fallas en los servidores en una arquitectura *Cliente-Servidores* se refleja fielmente a la dinámica en que los nodos se conectan y desconectan en una red entre pares. El desempeño observado en las pruebas para una red entre pares es excelente.

Poder enviar desde varias fuentes brinda la libertad de diseño a la red, ha-

ciéndola más resistente a desconexiones y a fallas simples en la red.

Actualmente estamos desarrollando prototipo de red entre pares para la difusión de TV en tiempo real. Varios desafíos técnicos y académicos se presentan. Nuestro trabajo resuelve uno de los aspectos más importantes: el mecanismo de distribución.

Glosario

En esta sección definimos algunos términos básicos para el entendimiento del material contenido en el Informe.

Access point Un punto de acceso inalámbrico (*WAP* o *AP* por sus siglas en inglés: *Wireless Access Point*) en redes de computadoras es un dispositivo que interconecta dispositivos de comunicación inalámbrica para formar una red inalámbrica. Normalmente un *WAP* también puede conectarse a una red cableada, y puede transmitir datos entre los dispositivos conectados a la red cableada y los dispositivos inalámbricos [19].

Codificador Decodificador (CóDec) Describe una especificación desarrollada en software, hardware o una combinación de ambos, capaz de transformar un archivo con un flujo de datos a otro archivo de menor tamaño usando algún mecanismo de compresión. Los códecs pueden codificar el flujo (a menudo para la transmisión, almacenamiento o cifrado) y recuperarlo o descifrarlo del mismo modo para la reproducción o la manipulación en un formato más apropiado. La mayor parte de códecs provocan pérdidas de información para conseguir un tamaño lo más pequeño posible del archivo destino [19].

Exclusión mutua Los algoritmos de exclusión mutua (comúnmente abreviada como *mutex* por *mutual exclusion*) se usan en programación concurrente para evitar que fragmentos de código conocidos como secciones críticas accedan al mismo tiempo a recursos que no deben ser compartidos [19].

MD5 Es un algoritmo de reducción criptográfico comúnmente utilizado para chequear la integridad de archivos. La codificación del MD5 de 128 bits es representada típicamente como un número de 32 dígitos hexadecimal [39]. Por ejemplo: MD5("*Distribución de vídeo en vivo desde múltiples fuentes*") = *1ee3f7093378caaa8f23cdbea5629761*.

Número pseudoaleatorio Un número pseudoaleatorio es un número generado a través de un proceso que parece producir números al azar, pero no lo hace realmente. Las secuencias de números pseudoaleatorios no muestran ningún patrón o regularidad aparente desde un punto de vista estadístico, a pesar de haber sido generadas por un algoritmo completamente determinista, en el que las mismas condiciones iniciales producen siempre el mismo resultado [19].

Spinlock Es cuando un hilo de ejecución (o *thread*) simplemente espera repetidamente hasta que se cumple una condición, como por ejemplo la llegada

de un paquete por la red o un semáforo que se haga disponible. Este tipo de bloqueos son muy eficientes sólo si lo más probable es que los hilos permanezcan bloqueados durante un corto intervalo de tiempo, pues evitan la sobrecarga que implica la replanificación de tareas del sistema operativo. Si el bloqueo se mantiene durante un período elevado de tiempo los *spinlocks* son muy costosos. Es por ello que los *spinlocks* se emplean típicamente para operaciones con elevada carga como puede ser analizar un volumen elevado de tráfico de red o atención de interrupciones hardware.

Tiempo de decodificación (DTS) Tiempo absoluto en que un bloque debe ser decodificado.

Transformada de coseno discreta (*DCT: Discrete Cosine Transform*) Es una transformada basada en la Transformada de Fourier discreta, pero utilizando únicamente números reales [19].

Multimedia Es un sistema que utiliza más de un medio de comunicación al mismo tiempo en la presentación de la información, como vídeo y sonido [19].

Firewall Es un elemento de hardware o software utilizado en una red de computadoras para controlar las comunicaciones, permitiéndolas o prohibiéndolas según las políticas de red que haya definido la organización responsable de la red.

Round trip time Es el retardo de ida y vuelta entre dos nodos de una red o el tiempo de latencia

Bibliografía

[1] John G. Apostolopoulos and Mitchell D. Trott, "Path Diversity for Enhanced Media Streaming". citeseer.ist.psu.edu/apostolopoulos04path.html.

[2] John G. Apostolopoulos, "Reliable video communication over lossy packet networks using multiple state encoding and path diversity", Proc. Visual Communication and Image Processing, VCIP '01, pp. 392-409, 2001. citeseer.ist.psu.edu/apostolopoulos01reliable.html.

[3] Dan Rubenstein and James F. Kurose and Donald F. Towsley, "Detecting shared congestion of flows via end-to-end measurement", Measurement and Modeling of Computer Systems, pp. 145-155, 2000. citeseer.ist.psu.edu/rubenstein99detecting.html.

[4] Stefan Savage and Andy Collins and Eric Hoffman and John Snell and Thomas E. Anderson, "The End-to-End Effects of Internet Path Selection", Proceedings of SIGCOMM, pp. 289-299, Boston, MA, 1999. citeseer.ist.psu.edu/savage99endtoend.html.

[5] C. Perkins and O. Hodson and V. Hardman, "A survey of packet loss recovery techniques for streaming audio", IEEE Network, vol. 12, no. 5, pp. 40-48, 1998. citeseer.ist.psu.edu/perkins98survey.html.

[6] L. Golubchik and J. C. S. Lui and T. F. Tung and A. L. H. Chow and W.-J. Lee and G. Franceschinis and C. Anglano, "Multi-path continuous media streaming: what are the benefits?", Technical Report CS-TR-2002-01, Computer Science and Engineering Department, vol. 49, pp. 429-449, 2002. http://dx.doi.org/10.1016/S0166-5316(02)00125-6.

[7] Novella Bartolini and Emiliano Casalicchio and Salvatore Tucci, "A Walk through Content Delivery Networks", MASCOTS Tutorials, pp. 1-25, 2003.

[8] Benjamín Molina and Carlos E. Palau and Manuel Esteve "Modeling content delivery networks and their performance", Computer Communications, vol. 27, no. 5, pp. 1401-1411, 2004.

[9] Wang, Yao and Reibman, A. R. and Lin, Shunan, "Multiple description coding for video delivery", Proceedings of the IEEE, vol. 93, no. 1, pp. 57-70, 2005. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1369698.

[10] John G. Apostolopoulos and T. Wong and W. Tan and S. Wee, "On multiple description streaming with content delivery networks, IEEE Infocom, 2002. citeseer.ist.psu.edu/apostolopoulos02multiple.html.

[11] John G. Apostolopoulos and W. Tan and S. Wee and G. Wornell, "Modeling path diversity for multiple description video communication", ICASSP, 2002. citeseer.ist.psu.edu/apostolopoulos02modeling.html".

[12] K. Sripanidkulchai and B. Maggs and H. Zhang. "An Analysis of Live Streaming Workloads on the Internet", Proceedings of Internet Measurement Conference (IMC), Taormina, pp. 41-54, 2004. citeseer.ist.psu.edu/751265.html.

[13] J. Van der Merwe and S. Sen and C. Kalmanek, "Streaming Video Traffic: Characterization and Network Impact", Proceedings of the 7th International Workshop on Web Content Caching and Distribution, Boulder, CO, USA, 2002. citeseer.ist.psu.edu/vandermerwe02streaming.html.

[14] Pablo Rodríguez Bocca, "Redes de Contenido: Taxonomía y Modelos de evaluación y diseño de los mecanismos de descubrimiento de contenido", Universidad de la República, Facultad de Ingeniería, Instituto de Computación. ISSN 0797-6410 INCO-RT-05-13, Montevideo, Uruguay, 2005. http://www.fing.edu.uy/inco/pedeciba/bibliote/tesis/tesis-rodriguezbocca.pdf

[15] D. Xu and M. Hefeeda and S. Hambrusch and B. Bhargava, "On Peer-to-Peer Media Streaming", Purdue Computer Science Technical Report, 2002. citeseer.ist.psu.edu/xu02peertopeer.html".

[16] Pablo Rodríguez-Bocca and Gerardo Rubino, "A QoE-based multi-source approach for P2P live video streaming" In 27th IEEE Conference on Computer Communications (INFOCOM'08), Phoenix, Arizona, United States, 2008.

[17] Xuxian Jiang and Yu Dong and Dongyan Xu and Bharat Bhargava, "Gnustream: A P2P Media Streaming System Prototype" In Proceedings of the International Conference on Multimedia and Expo (ICME), vol. 2, pp. 325-328, 2003. citeseer.ist.psu.edu/xu03gnustream.html.

[18] Allen K. Miu and John G. Apostolopoulos and Wai-tian Tan and Mitchell Trott, "Low-Latency Wireless Video Over 802.11 Networks Using Path Diversity", IEEE ICME 2003, Baltimore, MD.

[19] Wikipedia, www.wikipedia.org. Última visita: Octubre de 2007.

[20] JPEG Home Page http://www.jpeg.org/. Última visita: Octubre de 2007.

[21] MPEG Starting Points and FAQs http://www.mpeg.org/MPEG/starting-points.html. Última visita: Agosto de 2007.

[22] MPEG Home Page http://www.chiariglione.org/mpeg/. Última visita: Julio de 2007.

[23] Compresión MPEG-2 http://www.upv.es/satelite/trabajos/Grupo8_99.00/mpeg.html. Última visita: Julio de 2007.

[24] Compresión MPEG-4 http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm. Última visita: Julio de 2007.

[25] Internet Protocol version 4 http://www.faqs.org/rfcs/rfc791.html. Última visita: Octubre de 2007.

[26] Internet Protocol version 6 http://www.faqs.org/rfcs/rfc2460.html. Última visita: Octubre de 2007.

[27] Transmission Control Protocol, RFC 793 http://www.faqs.org/rfcs/rfc793.html. Última visita: Octubre de 2007.

[28] Hypertext Transfer Protocol, RFC 2616 http://www.faqs.org/rfcs/rfc2616.html. Última visita: Octubre de 2007.

[29] User Datagram Protocol, RFC 768 http://www.faqs.org/rfcs/rfc768.html. Última visita: Octubre de 2007.

[30] Real Time Protocol, RFC 1889 http://www.faqs.org/rfcs/rfc1889.html. Última visita: Octubre de 2007.

[31] Real Time Streaming Protocol, RFC 2326 http://www.faqs.org/rfcs/rfc2326.html. Última visita: Octubre de 2007.

[32] Multimedia Over IP: RTP, RTSP http://www2.ing.puc.cl/~jnavon/IIC3582/Present/3/mmip.htm#rtsp. Última visita: Octubre de 2007.

[33] MPlayer Home Pagehttp://www.mplayerhq.hu/. Última visita: Agosto de 2007.

[34] MPEG4IP Home Pagehttp://mpeg4ip.sourceforge.net/. Última visita: Agosto de 2007.

[35] VideoLan, www.videolan.org. Última visita: Octubre de 2007.

[36] VideoLAN Streaming Howto, http://www.videolan.org/doc/streaming-howto/en/streaming-howto-en.html. Última visita: Octubre de 2007.

[37] Streaming features list, http://www.videolan.org/streaming-features.html. Última visita: Octubre de 2007.

[38] Generador de números pseudoaleatorios, http://www.linuxmanpages.com/man3/rand_r.3.php. Última visita: Abril de 2007.

[39] "The MD5 Message-Digest, RFC 1321", http://www.faqs.org/rfcs/rfc1321.html. . Última visita: Abril de 2007.

[40] Understanding the linux kernel, O'Reilly, Tercera edición, Caplítulo Kernel Synchronization.

[41] "Display bandwidth usage on an interface", http://www.ex-parrot.com/~pdw/iftop/. . Última visita: Junio de 2007.

[42] Mosaic project http://wiki.videolan.org/Mosaic. . Última visita: Enero de 2007.

[43] Fake module `http://wiki.videolan.org/Fake`. . Última visita: Agosto de 2007.

[44] Gnutella, `www.gnutella.com`. . Última visita: Septiembre de 2007.

Apéndice A

Agregar un módulo a VLC

El procedimiento para agregar un módulo en *VLC* los describimos a continuación. *VLC* almacena los módulos dentro de un directorio especial llamado *modules* clasificado en diferentes categorías. Los que nos interesan para el proyecto son los de distribución de vídeo y se agrupan en un directorio llamado *stream_out*. Una vez que hemos clasificado el tipo de módulo, si no existe tal clasificación, debemos crear un directorio que describa las funcionalidades de los módulos que contendrá. En particular creamos un directorio que llamamos *multipath* dentro de *stream_out* para agrupar todas las funcionalidades del sistema. El siguiente paso es crear un archivo con nombre *Modules.am* (si no existe) e incluir los módulos de ese directorio. Indicamos todos los archivos que componen cada módulo siguiendo el formato:

$SOURCES\_mod\_1 = mod1\_archivo1.extension...mod1\_archivoP.extension$

En particular para el directorio *modules/mutipath* el contenido del archivo es el que se muestra en la figura A.1.

Por cada módulo debe haber un archivo que contenga el marco de definición. Tal marco describe al módulo y contiene: la categoría y subcategoría, nombre y descripción, y la definición de las funciones *Open*, *Close, Add, Del* y *Send*. La función *Open* se invoca al momento en que el módulo se levanta por primera vez y *Close* es la última en ejecutarse. La función *Send* es invocada cada vez que hay nuevos bloques para ser procesados por el módulo. Las funciones *Add* y *Del* se invocan cada vez que se agrega o elimina un tipo de flujo (audio, vídeo) al sistema.

Los archivos que deben compilarse con el sistema los indicamos en el archivo *configure.ac* (ubicado en el directorio raíz de *VLC*). Para que el módulo se compile por defecto junto con el sistema, debemos declararlo a través

---

SOURCES_stream_out_mpath_server = mpath_server.c
SOURCES_stream_out_mpath_client = mpath_client.c delayMatrix.c delayMatrix.h
SOURCES_stream_out_dropper = dropper.c

Figura A.1: Estructura del archivo *Modules.am*.

```
AC_CONFIG_FILES([
...
modules/stream_out/multipath/Makefile
...
])
...
VLC_ADD_PLUGINS([stream_out_mpath_client
stream_out_mpath_server stream_out_dropper])
```

Figura A.2: Líneas agregadas al archivo *configure.ac*.

de la macro *VLC_ADD_PLUGINS* la cual recibe como parámetro el nombre del módulo y se encuentra en la sección *default modules* del archivo. En caso de haber creado un nuevo directorio en *modules*, debemos ubicar la definición *AC_CONFIG_FILES* en donde se indican los *Makefiles* y agregar la ruta a nuestro directorio. Ese *Makefile* es el que compilará todos los módulos dentro del nuevo directorio.En la figura A.2 mostramos las líneas que agregamos al archivo *configure.ac*.

Finalmente debemos generar los *Makefiles* y compilar el sistema para que los módulos sean agregados satisfactoriamente. Para hay que ir al directorio raíz de *VLC* y ejecutar los siguientes comandos:

- *$>./bootstrap*
- *$>./configure opciones*
- *$>./compile*

Las opciones que recomendamos para el *configure* son las que se muestran en la figura A.3.

```
./configure –enable-optimize-memory –enable-mostly-builtin –enable-debug –enable-
sout –enable-httpd –enable-vlm –enable-dshow –enable-libcdio –enable-vcdx –enable-
cdda –enable-vcd –enable-ogg –enable-mkv –enable-mad –enable-libdvbpsi –enable-
ffmpeg –enable-real –enable-realrtsp –enable-a52 –enable-dts –enable-flac –enable-
libmpeg2 –enable-vorbis –enable-speex –enable-theora –enable-x264 –enable-x11 –
enable-xvideo –enable-glx –enable-opengl –enable-sdl –enable-freetype –enable-fribidi –
enable-directx –enable-fb –enable-caca –enable-oss –enable-esd –enable-alsa –disable-
skins2 –enable-wxwidgets –enable-visual –enable-galaktos –enable-loader
```

Figura A.3: Opciones para el comando *configure*.

Apéndice B

Configuración de los servidores

A continuación explicamos como configurar los servidores. Todos cuentan con un archivo en donde se definen las semillas utilizadas en los sorteos, el porcentaje de bloques que le corresponde enviar a cada uno y el porcentaje de bloques de vídeo redundantes que serán enviados por los servidores en su conjunto. Todos deben tener en su archivo la misma configuración.

La definición del medio de transferencia, el tipo de encapsulamiento y el destino (entre otros) se definen en el módulo *standard* de *VLC*.

B.1. Archivo de configuración

El archivo contiene la información referente al método *probabilístico*. Como explicamos en 3.3 necesitamos definir tres semillas. El valor de la semillas utilizada para el envío de vídeo y audio deben ir precedidas de *"Video seed "*, y *"Audio seed "* respectivamente. La semilla utilizada para sortear si un servidor redunda debe estar precedida de *"Redundancy seed "*. El valor de la semilla es un número entero de 32 bits.

El porcentaje de bloques de vídeo redundantes se define a través de la palabra *"Redundancy"* seguida de un vector tridimensional de la forma (i, p, b) con sus componentes comprendidos en el intervalo [0, 1]. Tales componentes se refieren al porcentaje de bloques de tipo I, P y B respectivamente.

El porcentaje de los bloques de cada tipo que le corresponde enviar a cada servidor se define de la siguiente manera:

$$Server\ n\ (i, p, b)\ a$$

El valor de n es un entero comprendido entre 1 y N (cantidad de servidores). Los valores de i, p, b y a son números reales comprendidos en el intervalo [0, 1] y corresponden a los bloques de vídeo de tipo I, P, B y bloques de audio respectivamente. La suma de los porcentajes de cada uno de los tipos de bloques que envían todos los servidores debe ser igual a 1. Si esto no se cumple y dicha suma es diferente de cero, se normalizará de forma de que todo bloque le corresponda enviar a un único servidor.

```
# Archivo de configuración de servidor

# Semilla para el envío de vídeo
Video seed 16

# Semilla para el envío de audio
Audio seed 2

# Semilla para el envío de bloques redundantes
Redundancy seed 3

# Envío de redundancia para los bloques de vídeo de
# tipo I, P y B respectivamente
Redundancy (0.3, 0.5, 0)

# Envío de bloques para los distintos servidores
Server 1 (0.2, 0.2, 0.2) 1
Server 2 (0.2, 0.2, 0.2) 0
Server 3 (0.2, 0.2, 0.2) 0
Server 4 (0.2, 0.2, 0.2) 0
Server 5 (0.2, 0.2, 0.2) 0
```

Figura B.1: Archivo de configuración para los servidor.
Todos los servidores envían la misma proporción de bloques de vídeo. Los bloques de audio son enviados únicamente por el número 1. Los servidores en su conjunto redundarán con el 30 % de bloques de vídeo de tipo I y el 50 % de bloques de vídeo de tipo P.

Las líneas que comienzan con '#' son comentarios. En la figura B.1 se muestra un ejemplo de un archivo de configuración para 5 servidores.

B.2. Módulo standard

El módulo *standard* permite enviar el flujo de vídeo a través de diferentes medios (archivo, red), utilizando algún método de encapsulamiento a un destino especificado. En [36] se detallan otras opciones que dejamos fuera del alcance del presente Informe.

Medio (access)

Esta opción es obligatoria y permite elegir el medio usado para guardar o enviar el flujo de vídeo la red. Entre los medios disponibles en [36] encontramos:

file: Guarda el flujo de vídeo a un archivo.

http: Envía el flujo de vídeo a la red a través del protocolo *HTTP*.

udp: El flujo de video se envía a través de *UDP* la red, a una dirección unicast o multicast.

Encapsulamiento (mux)

A través de esta opción obligatoria se define el método de encapsulamiento que será utilizado para el flujo de vídeo. Entre las opciones disponibles [36] encontramos:

ts: Encapsulamiento *MPEG2/TS*.

ps: Encapsulamiento *MPEG2/PS*.

mpeg1: Encapsulamiento *MPEG1*.

Destino (dst)

Esta opción permite dar información sobre la ubicación donde debería ser guardado o enviado el flujo de vídeo. El significado depende del parámetro usado para el medio.

- Si el medio es file (*access=file*), el destino es el directorio donde debe ser guardado.
- Si el medio es *HTTP* (*access=http*), el destino es la dirección de una interfaz de red local, puerto sobre la cual el servidor debe esperar solicitudes. Si no se especifica, *VLC* esperará solicitudes en todas las interfaces. Se especifica a través de *dirección:puerto*.
- Si el medio es *UDP* (*access=udp*), el destino es la dirección unicast o multicast y opcionalmente el puerto.

En [37] se muestran las compatibilidades entre los medios y formatos (CóDec) con los métodos de encapsulamiento.

B.3. Ejecución

A continuación describimos como ejecutar un servidor. Solo tendremos en cuenta las opciones vistas en B.2. Tanto el módulo de que implementa la estrategia de distribución desde múltiples fuentes (*mpath-server*) como *standard* se cargan y encadenan como cualquier módulo de *streaming media* de *VLC*. Por más información al respecto dirigirse a [36].

Una vez que se tienen los N servidores identificados (de *1* a N), el archivo de configuración y el vídeo a ser reproducido en cada uno, debe ejecutarse el comando que se muestra en la figura B.2. Lo comprendido entre comillas dobles no debe llevar espacios.

B.4. Ejemplo

En esta sección mostramos un ejemplo de ejecución. Disponemos de cuatro servidores idénticos que envían la misma proporción de flujo de vídeo a través de *HTTP/TS*. El servidor identificado con 1 envía el audio. Las semillas para los sorteos son 16, 2 y 3 para vídeo, audio y redundancia respectivamente. La redundancia global en la red será 30, 20 y 10 %, para los bloques de tipo I, P y B, respectivamente. En las figuras B.3 y B.4 mostramos el archivo de configuración y el comando para para ejecutar el servidor identificado con 1 respectivamente.

```
./vlc –sout "#mpath-server{
id=Identificación-servidor,
config=Archivo-configuración
}:
standard{
access=Opción-medio,
mux=Opción-encapsulamiento,
dst=Destino,
}"
video.mpg
```

Figura B.2: Ejecución de un servidor.
Opción-medio, Opción-encapsulamiento y Destino son los explicados en B.2. Identificación-servidor es un número entre 1 y N diferente para cada servidor y Archivo-configuración es la ubicación del archivo que se explicó en B.1.

```
# Archivo de configuración de servidor

# Semilla para el envío de vídeo
Video seed 16

# Semilla para el envío de audio
Audio seed 2

# Semilla para el envío de bloques redundantes
Redundancy seed 3

# Envío de redundancia para los bloques de vídeo de
# tipo I, P y B respectivamente
Redundancy (0.3, 0.2, 0.1)

# Envío de bloques para los distintos servidores
Server 1 (0.25, 0.25, 0.25) 1
Server 2 (0.25, 0.25, 0.25) 0
Server 3 (0.25, 0.25, 0.25) 0
Server 4 (0.25, 0.25, 0.25) 0
```

Figura B.3: Archivo de configuración para cuatro servidores.

```
./vlc –sout
"#mpath-server{id=1,config=~/configuraciones/4Serv-equi.conf}:
standard{access=http,mux=ts,dst=lulu:1231}" IceAgeIntro.mpg
```

Figura B.4: Comando para ejecutar el servidor identificado con 1.

B.5. Mensajes de error

A continuación mostramos los mensajes que despliega un servidor al detectar un error y las causas de los mismos.

Missing identification server (fatal error)

No se ha especificado el identificador del servidor.

The identification server must be between 1 and N

El identificador del servidor no se encuentra entre *1* y *N*.

Missing configuration file (fatal error)

No se ha especificado el archivo de configuración.

Configuration file not found

El archivo de configuración especificado no se ha encontrado.

Configuration file: Syntax error in line i:

El archivo de configuración tiene un error de sintaxis en la línea *i*.

Configuration file: Video seed not found

En el archivo de configuración no se ha especificado la semilla para los sorteos de correspondencia de bloques de vídeo a los servidores. Ver B.1.

Configuration file: Audio seed not found

En el archivo de configuración no se ha especificado la semilla para los sorteos de correspondencia de bloques de audio a los servidores. Ver B.1.

Configuration file: Redundancy seed not found

En el archivo de configuración no se ha especificado la semilla para los sorteos para determinar que servidor debe redundar con un bloque. Ver B.1.

Configuration file: Redundancy not found

No se ha especificado el porcentaje de bloques de vídeos redundantes que se enviarán desde los servidores en su conjunto. Ver B.1.

Configuration file: Server i not configured

No se ha especificado el porcentaje de bloques de cada tipo que debe enviar el servidor número *i*. . Ver B.1

Apéndice C

Configuración del cliente

En este capítulo explicamos como configurar el cliente para recibir flujo de N servidores. Utilizamos *VideoLAN Manager* (*VLM*) que es un pequeño manejador que permite definir múltiples flujos de entrada con una sola instancia de *VLC*.

C.1. Configuración

VLM se configura a través de un archivo en el cual se ingresan los comandos. En el archivo de configuración debemos definir un conjunto de elementos *VLM*. A nosotros solo nos interesa definir conexiones de tipo *broadcast* idenficadas por un nombre. Por cada conexión debemos definir la entrada y la salida. Con entrada nos referimos al medio, tipo de encapsulamiento y la *url* del flujo de vídeo. Con salida nos referimos al módulo que debe atenderlo.

Además de definir un canal por servidor, utilizamos uno especial en el cual se recibe una imagen. El objetivo de este canal es levantar el módulo *mpath-client*. Para tratar todos los canales que reciben flujo de los servidores de la misma forma, utilizamos el módulo *fake* de *VLC* para reproducir una imagen cuando termine el flujo de vídeo de los N servidores. El módulo *mpath-client* puede reproducir localmente el vídeo reconstruido (enviándoselo al módulo *display*), guardarlo a disco o enviarlo a la red (enviándoselo al módulo *standard* de *VLC*). Finalmente debemos activar cada canal con *play*. En la figura C.1 mostramos un ejemplo para cuatro servidores. Observar que se ejecutan cuatro instancias de *mpath-bridge* (una por cada flujo) y una de *mpath-client*.

C.2. Ejecución

Una vez que se tienen los N servidores reproduciendo vídeo se debe ejecutar el cliente. Necesitamos definirnos un archivo de configuración para *VLM* como mostramos en la figura C.1. Finalmente debemos ejecutar el comando de la figura C.2.

Para definir una red entre pares, el módulo *mpath-bridge* puede reenviar de la misma forma que *mpath-client* el flujo recibido por un nodo, a otro nodo utilizando los módulos *standard* y *duplicate* de *VLC*. En la figura C.3 mostramos

```
#Canal 1: Recibe el flujo del servidor 1
new channel1 broadcast enabled
setup channel1 input http/ts:localhost:1231
setup channel1 output #mpath-bridge{id=1}

#Canal 2: Recibe el flujo del servidor 2
new channel2 broadcast enabled
setup channel2 input http/ts:localhost:1232
setup channel2 output #mpath-bridge{id=2}

#Canal 3: Recibe el flujo del servidor 3
new channel3 broadcast enabled
setup channel3 input http/ts:localhost:1233
setup channel3 output #mpath-bridge{id=3}

#Canal 4: Recibe el flujo del servidor 4
new channel4 broadcast enabled
setup channel4 input http/ts:localhost:1234
setup channel4 output #mpath-bridge{id=4}

#Canal 5: Recibe una imagen
new channel5 broadcast enabled
setup channel5 input fake:
setup channel5 output #mpath-client{delay=0}:display

control channel1 play
control channel2 play
control channel3 play
control channel4 play
control channel5 play
```

Figura C.1: Archivo de configuración del cliente para cuatro servidores.

```
./vlc –color -I telnet –vlm-conf configuración –fake-file imagen –fake-aspect-ratio
"4:3"
```

Figura C.2: Ejecución del cliente
configuración es el archivo de configuración de VLM e imagen es la imagen a mostrar a través de fake.

una instancia del módulo *mpath-bridge* que reenviará el contenido a otro nodo de la red entre pares.

```
new channelK broadcast enabled
setup channelK input http/ts:localhost:1232
setup channelK output #duplicate{dst=mpath-bridge{id=K},dst=standard{access=http,mux=ts,url=lulu:1232}}
```

Figura C.3: Reenvío desde el módulo *mpath-brdige*.
En la última línea se especifica que el flujo recibido deberá duplicarse y ser enviado a los módulos mpath-bridge y standard. El módulo standard se encargará de reenviar el flujo al nodo de la red especificado.

C.3. Ejemplo

Siguiendo con el ejemplo de cuatro servidores que distribuyen vídeo, si llamamos *vlm_client.conf* al archivo de configuración de la figura C.1 y la imagen a mostrar es *VideoLAN.jpg*, basta con ejecutar el comando mostrado en la figura C.4.

```
./vlc –color -I telnet –vlm-conf vlm_client.conf –fake-file VideoLAN.jpg –fake-aspect-ratio "4:3"
```

Figura C.4: Ejemplo del comando para ejecutar el cliente.

Apéndice D

Módulo generador de fallas

En un contexto real nos interesa evaluar el desempeño del sistema cuando se producen fallas en servidores. Para generar dichas fallas implementamos un módulo que llamamos *dropper*. El módulo recibe como entrada un archivo de configuración en donde se especifican pares $(t_i,\ t_r)$. Los valores se expresan en segundos y son el tiempo relativo al vídeo en que se produce una falla y la duración de la misma respectivamente. Durante un período de fallas el módulo descarta los bloques que recibe para enviar. En la figura D.1 mostramos un ejemplo de un archivo de configuración utilizado en las pruebas de desempeño para el servidor 1 y la invocación.

```
#Primer falla se produce en el tiempo 25 s del vídeo y dura 1 s.
(25, 1)
#Segunda falla se produce en el tiempo 30 s del vídeo y dura 3 s.
(30, 3)
```

```
./vlc –sout
"#mpath-server{id=1,config=~/configuraciones/4Serv-equi.conf}:
dropper{config=~/fallas/drop.conf}:
standard{access=http,mux=ts,dst=lulu:1231}" IceAgeIntro.mpg
```

Figura D.1: Ejemplo de un archivo de configuración para el módulo *dropper* y el comando para ejecutar un servidor con el módulo.

Apéndice E

Pruebas de correctitud

En esta sección describimos los archivos de configuración utilizados, programas y algunos resultados obtenidos. En las tabla E.1 mostramos las semillas utilizadas.

Archivo \ Semillas	Vídeo	Audio	Redundancia
1Server-1.conf	16	2	3
2Servers-1.conf	16	2	3
2Servers-2.conf	162	21	13
2Servers-3.conf	12	14	2
2Servers-4.conf	162	21	13
2Servers-5.conf	162	21	13
3Servers-1.conf	1	1	1
3Servers-2.conf	1101	135	3
3Servers-3.conf	1012	345	3012
3Servers-4.conf	101	35	3012
3Servers-5.conf	1101	135	3
5Servers-1.conf	111	1352	23
5Servers-2.conf	111	1352	23
5Servers-3.conf	112	132	231
8Servers-1.conf	111	1352	123
8Server-1.conf	112	132	231

Cuadro E.1: Semillas utilizadas en las configuraciones.

En las siguientes tablas mostramos los porcentajes de envío por redundancia y correspondencia para los distintos servidores. Finalmente en la tabla E.1 mostramos el programa *awk* utilizado para recoger las estadísticas de los servidores.

Archivo		Frames I	Frames P	Frames B	Audio
1Server-1.conf	Redundancia	0	0	0	
	Servidor 1	1	1	1	1

Cuadro E.2: Configuraciones para 1 servidor.

Archivo		Frames I	Frames P	Frames B	Audio
2Server-1.conf	Redundancia	0	0.1	0.5	
	Servidor 1	0.9	0	0	0
	Servidor 2	0.1	1	1	1
2Server-2.conf	Redundancia	0	0.1	0.5	
	Servidor 1	0.5	0.5	0.5	1
	Servidor 2	0.5	0.5	0.5	0
2Server-3.conf	Redundancia	0	0.1	0.5	
	Servidor 1	0.9	0.9	0.9	0
	Servidor 2	0.1	0.1	0.1	1
2Server-4.conf	Redundancia	0.2	0.2	0.5	
	Servidor 1	0.9	0.1	0.9	0
	Servidor 2	0.1	0.9	0.1	1
2Server-5.conf	Redundancia	0	1	0	
	Servidor 1	0.7	0.3	0.3	1
	Servidor 2	0.3	0.7	0.7	0

Cuadro E.3: Configuraciones para 2 servidores.

Archivo		Frames I	Frames P	Frames B	Audio
3Server-1.conf	Redundancia	0.3	0	0.1	
	Servidor 1	0.8	0	0	0
	Servidor 2	0.1	1	0	0
	Servidor 3	0.1	0	1	1
3Server-2.conf	Redundancia	0	0	0.5	
	Servidor 1	0.33	0.33	0.33	0
	Servidor 2	0.33	0.33	0.33	1
	Servidor 3	0.33	0.33	0.33	0
3Server-3.conf	Redundancia	0	0.3	0.5	
	Servidor 1	0.5	0.5	0	1
	Servidor 2	0.1	0.5	0.5	0
	Servidor 3	0.4	0	0.5	0
3Server-4.conf	Redundancia	0	0.3	0.5	
	Servidor 1	0.5	0.3	0.1	0
	Servidor 2	0.25	0.35	0.45	1
	Servidor 3	0.25	0.35	0.45	0
3Server-5.conf	Redundancia	0	0	0.5	
	Servidor 1	0.9	0.9	0.9	0
	Servidor 2	0.05	0.05	0.05	1
	Servidor 3	0.05	0.05	0.05	0

Cuadro E.4: Configuraciones para 3 servidores.

Archivo		Frames I	Frames P	Frames B	Audio
5Server-1.conf	Redundancia	0.1	0.2	0.5	
	Servidor 1	0.8	0	0	0
	Servidor 2	0.05	0.5	0	0
	Servidor 3	0.05	0.5	0	0
	Servidor 4	0.05	0	0.5	0
	Servidor 5	0.05	0	0.5	1
5Server-2.conf	Redundancia	0.1	0.2	0.5	
	Servidor 1	0.2	0.2	0.2	0
	Servidor 2	0.2	0.2	0.2	0
	Servidor 3	0.2	0.2	0.2	1
	Servidor 4	0.2	0.2	0.2	0
	Servidor 5	0.2	0.2	0.2	0
5Server-3.conf	Redundancia	0.1	0.2	0.5	
	Servidor 1	0.7	0.1	0	0
	Servidor 2	0.1	0.3	0	0
	Servidor 3	0.05	0.2	0	1
	Servidor 4	0.1	0.2	0.5	0
	Servidor 5	0.05	0.2	0.5	0

Cuadro E.5: Configuraciones para 5 servidores.

Archivo		Frames I	Frames P	Frames B	Audio
8Server-1.conf	Redundancia	0.1	0.2	0.5	
	Servidor 1	0.4	0	0	0
	Servidor 2	0.3	0	0	0
	Servidor 3	0.1	0.4	0	0
	Servidor 4	0.1	0.4	0	0
	Servidor 5	0.05	0.2	0.2	1
	Servidor 6	0.05	0	0.2	0
	Servidor 7	0.05	0	0.4	0
	Servidor 8	0.05	0	0.2	0
8Server-2.conf	Redundancia	0.3	0.3	0.7	
	Servidor 1	0.6	0.3	0.3	0
	Servidor 2	0.05	0.1	0	0
	Servidor 3	0.05	0.2	0.1	0
	Servidor 4	0.05	0.2	0.1	1
	Servidor 5	0.05	0.2	0.2	0
	Servidor 6	0.05	0	0.1	0
	Servidor 7	0.05	0	0.2	0
	Servidor 8	0.1	0	0.3	0

Cuadro E.6: Configuraciones para 8 servidores.

```
#!/usr/bin/awk -f
BEGIN {
I_sent = 0; I_redu = 0;
P_sent = 0; P_redu = 0;
B_sent = 0; B_redu = 0;
I_size = 0;
P_size = 0;
B_size = 0;
};
{
if( $9 == "I" ) { I_sent++; I_size += $20; }
if( $9 == "P" ) { P_sent++; P_size += $20; }
if( $9 == "B" ) { B_sent++; B_size += $20; }
if( $11 == "I" ) { I_redu++; I_size += $22; }
if( $11 == "P" ) { P_redu++; P_size += $22; }
if( $11 == "B" ) { B_redu++; B_size += $22; }
};
END {
printf( "Cantidad de bloques enviados\n" );
printf( "----------------\n\n" );
printf( "\t\tI\tP\tB\n" );
printf( "Por correspondencia:\t%d,\t%d,\t%d\n", I_sent, P_sent, B_sent );
printf( "Redundantes:\t\t%d,\t%d,\t%d\n", I_redu, P_redu, B_redu );
printf( "Tamaño promedio:\t%d,\t%d,\t%d\ten bytes\n\n", \
( I_sent + I_redu != 0? I_size/( I_sent + I_redu ) : 0 ), \
( P_sent + P_redu != 0 ? P_size/( P_sent + P_redu ) : 0 ),\
( B_sent + B_redu != 0 ? B_size/( B_sent + B_redu ) : 0 ) );
};
```

Figura E.1: Programa *awk* utilizado para contar el número de bloques enviado de cada tipo.