PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Tesis de Maestría
en Informática

GoalBit: la primer red P2P de distribución de video en vivo de código abierto y gratuita

Daniel Osvaldo De Vera Rodríguez

2010

GoalBit: la primer red P2P de distribución de video
en vivo de código abierto y gratuita.
Daniel Osvaldo De Vera Rodríguez
ISSN 0797-6410
Tesis de Maestría en Informática
Reporte Técnico RT 10-15
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República.
Montevideo, Uruguay, octubre de 2010

TESIS

Presentada el día 18 de Octubre de 2010 en la

Facultad de Ingeniería, Universidad de La República

para obtener el título de

MAGISTER EN INFORMÁTICA

para

Daniel Osvaldo DE VERA RODRIGUEZ

PEDECIBA Informática
Instituto de Computación
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA

Título de la tesis :

GoalBit: la primer red P2P de distribución de video en vivo de código abierto y gratuita

Comité de examinadores:

Dra. Ana Paula	COUTO DA SILVA	Revisor
Dr. Gerardo	RUBINO	Examinadores
Dra. Libertad	TANSINI	
Dr. Héctor	CANCELA	Director académico
Dr. Pablo	RODRÍGUEZ-BOCCA	Director de tesis

Agradecimientos

En primer lugar quiero agradecer a mis tutores, Dr. Héctor Cancela Bosi y Dr. Pablo Rodríguez-Bocca por su apoyo y estímulo permanente.

Gracias al instituto de investigación INRIA por su apoyo económico en las distintas etapas de la maestría, así como a la Administración Nacional de Telecomunicaciones (ANTEL) por permitir mi dedicación al proyecto y acercarme a un escenario real en donde poner a prueba mi investigación.

Gracias a los demás investigadores del área que han contribuido de una forma u otra con este trabajo.

Agradezco a los doctores Gerardo Rubino, Ana Paula Couto Da Silva y Libertad Tansini que me brindan el honor de evaluar este trabajo.

Finalmente gracias a mi familia y mi novia por el apoyo brindado en todo este tiempo, en particular a mis padres, que con su esfuerzo me permitieron realizar la carrera de Ingeniería en Computación.

Abstract

Actualmente el tráfico sobre Internet se encuentra creciendo día a día, siendo la distribución de video el principal impulsor de dicho aumento. Tal como se presenta en [32], hoy en día el $40\,\%$ del tráfico total de Internet se corresponde con la distribución de contenidos audiovisuales, mientras que se espera que para el 2014 este alcance el $57\,\%$. A su vez, dentro de las diferentes formas de distribución de video, la distribución de video en vivo se encuentra creciendo en importancia. Debido a este contexto, en los últimos años se han desarrollado múltiples aplicaciones de streaming entre pares, o también llamados *P2PTV* (*Peer-to-Peer TV*). Algunos ejemplos de estas aplicaciones son PPlive [103], TVUnetwork [127], PPstream [104], SopCast [121], etc. Todas aplicaciones con protocolo e implementaciones cerradas.

Dada esta realidad, este trabajo se centra en el diseño e implementación de *GoalBit*, la primer red P2P para la distribución de video en vivo, gratuita y de código abierto.

Siguiendo el exitoso enfoque de BitTorrent [19], se diseña el *GoalBit Transport Protocol*, mediante el cual es posible distribuir un streaming de video en vivo sobre una red de pares. Se define el *GoalBit Packetized Stream* con el fin de especificar la forma de paquetizar un *streaming* de audio y video, encapsulándolo dentro del protocolo de transporte. En base a estas especificaciones, se apoya a la comunidad open-source en la implementación de un cliente de referencia llamado *GoalBit*.

Tanto a nivel de pruebas empíricas, como en emulaciones realizadas, se observó que la estrategia de selección de piezas, la cual define que piezas un par debe solicitar a otro, es un factor de gran relevancia en la continuidad y latencia del *streaming*. En este tesis se extiende el trabajo [109], introduciendo una nueva estrategia de selección de piezas y comparándola con las presentadas en dicho trabajo.

Por otro lado, se continua con el trabajo realizado en [110] sobre la medición de la calidad de experiencia mediante la tecnología PSQA, integrando dicha medición al cliente GoalBit, obteniendo de esta manera una visión global y completa del estado de la distribución del streaming en la red.

Finalmente, a modo de validar nuestras ideas, se desarrollan un conjunto de extensiones al protocolo GoalBit, entre los que se encuentran el prototipo de una red de distribución de contenidos y un prototipo de un monitor de contenidos genérico.

Tanto el cliente GoalBit, como sus extensiones fueron probados en escenarios reales, obteniendo resultados muy prometedores.

Índice general

Parte I

INTRODUCIÓN

Capítulo 1

Introducción

1.1 Motivación y Objetivos

Como es sabido, el actual crecimiento del tráfico sobre redes IP, y junto con él, el del tráfico sobre Internet, ha sido y será incesante. Tal como se presenta en [32], se espera que el tráfico global sobre redes IP sea cuadriplicado entre el año 2009 y el 2014. A su vez, el principal impulsor de este crecimiento, es el tráfico asociado a la distribución de video, el cual considerando todas sus posibles formas (TV, video bajo demanda, Internet y P2P) representará más del $91\,\%$ del tráfico global para fines del 2014. Respecto a la distribución de video sobre Internet, el pronóstico es muy similar al anterior: en la actualidad este se corresponde con el $40\,\%$ del tráfico total, y se espera que en el 2014 este alcance al $57\,\%$.

Este crecimiento (tanto del tráfico global como del video) puede ser explicado en base a la existencia de ciertas tendencias y de ciertos facilitadores. Existen 3 claras tendencias las cuales influyen directamente en el crecimiento de tráfico IP: la hiper-conectividad (uso de múltiples aplicaciones, las cuales consumen concurrentemente desde la red), la imposición de la alta definición en el video y la migración a nuevas tecnologías (por ejemplo, migración de usuarios de sistemas de televisión clásicos a sistemas de tipo IPTV). Por otro lado, estas tendencias se sostienen en base a un conjunto de facilitadores, entre los que se encuentran: el crecimiento del número de pantallas digitales disponibles (con lo bajos costos de los televisores y monitores LCD, junto con la aparición de teléfonos móviles con grandes pantallas, lectores de tipo *"e-book"*, consolas de juegos, GPS, etc. se espera que para fines del 2014 la superficie total de pantallas digitales disponibles sea un $70\,\%$ mayor respecto a la actual), el aumento del ancho de banda disponible a nivel de usuario (se espera que el mismo se cuadriplique para el 2014) y el aumento en la capacidad de cómputo de los diferentes dispositivos.

Todos estos factores llevan a que la distribución de video tome un rol protagónico dentro del mundo tecnológico de hoy en día, exigiendo cambios a todos los involucrados en este proceso (como productores de contenidos, ISPs, usuarios finales, etc.) a muy diferentes niveles: desde la existencia de nuevos modelos de negocio, de nuevas infraestructuras, de nuevos dispositivos, hasta la aparición de nuevos problemas con sus consecuentes soluciones.

Ahora bien, mientras la mayoría de las aplicaciones se están tornando hacia el video, este a su vez se encuentra cambiando en sí mismo. En particular, la distribución de video en vivo está

creciendo en importancia respecto a las otras formas de distribución (se espera que para el 2014 el $10\,\%$ del total del tráfico de video en Internet, se corresponda con distribución de video en vivo). Este hecho trae aparejado grandes desafíos (por no decir problemas o costos) a las ISPs. Por la naturaleza intrínseca de este tipo de distribución, la cual posee restricciones de tiempo real, se presentan problemas tanto al momento de su codificación, como (y principalmente) en su posterior distribución a los usuarios finales.

La codificación de video en vivo usualmente requiere el uso de *hardware* especializado (*encoders* implementados en *hardware*), dado que (y dependiendo de la codificación requerida) los *encoders* implementados mediante *software*, no poseen suficiente capacidad para llevar a cabo dicha tarea. Evidentemente, el uso de *hardware* específico para cumplir esta tarea, implica costos más altos para el ISP. Esto no ocurre con la codificación de video a ser distribuido bajo demanda, en donde sí se suelen usar *encoders* implementados en *software*, los cuales en base a la aplicación de reiterados procesos sobre el video, logran la codificación deseada. Por ejemplo, el proceso de codificación de un video de 5 minutos puede demorar incluso más de 10 minutos. Notar que en el caso de la codificación de video en vivo, el codificación de 5 minutos de video debe requerir ni más ni menos que esos 5 minutos.

Respecto a la distribución de video en vivo, y en comparación con la distribución bajo demanda, esta presenta un comportamiento mucho menos uniforme, en donde suelen ocurrir grandes ráfagas de conexiones de usuarios ante ciertos eventos (como por ejemplo durante la transmisión de un partido de fútbol) y muy pocas durante otros períodos de tiempo (luego que termina la transmisión del partido, la cantidad de conexiones baja considerablemente). Observar que en la distribución bajo demanda, debido al hecho de que los contenidos se encuentran disponibles durante días, meses o incluso años (aquí no existe el concepto de "en vivo"), el acceso de los usuarios a dichos contenidos se encuentran mucho más distribuidos en el tiempo. Dada la actual infraestructura de Internet, en donde todas las conexiones son punto a punto (*"unicast"*), es que la existencia de grandes ráfagas de conexiones (a las que se encuentra expuesta la distribución de video en vivo), conlleva a varios problemas, entre los que podemos destacar: los problemas de escala, los problemas de costos y a los problemas de cuellos de botella.

Por un lado, dada la variabilidad de la demanda de los usuarios, es complejo dimensionar correctamente una red de distribución de video en vivo, en base a recursos fijos (es decir, en base a la disposición de una cierta cantidad de servidores con el fin de atender la demanda). Usualmente ocurren períodos en donde los recursos se encuentran ociosos (en donde la demanda no requiere de todos los recursos disponibles), y períodos en donde los recursos no dan a basto, períodos en donde la red no escala correctamente y justamente, períodos en donde mayor es el interés de los usuarios por acceder al contenido.

También sucede que el ancho de banda es el recurso más costoso en Internet (al menos por ahora, dado que en los últimos años los costos se han reducido hasta casi en un $90\,\%$) y a su vez el más requerido por las redes de distribución de contenido. En particular el costo asociado a la distribución de video es proporcional a la cantidad de usuarios en la red (debido al uso de conexiones *unicast*). Por lo cual de la distribución de video (en general) una actividad muy costosa.

Además de esto, el hecho de escalar la red (por ejemplo aumentando la cantidad de servidores) sin una estricta planificación, puede causar congestiones en ciertos sectores de Internet,

lo que implica una mayor tasa de pérdidas de paquetes y por lo tanto un peor servicio brindado a los usuarios finales. Motivo por el cual, escalar una solución de distribución de video, no implica simplemente agregar más recursos, sino que es necesario ubicar dichos recursos lo más cercano al usuario final con el fin de evitar los cuellos de botella.

Como solución a estos problemas es que aparecen los sistemas de *streaming* entre pares, o también llamados *P2PTV* (*Peer-to-Peer TV*). Estos sistemas crean redes virtuales a nivel de aplicación sobre la infraestructura de Internet, en donde los nodos de la red, llamados *pares* o *peers*, ofrecen sus recursos (ancho de banda, capacidad de procesamiento y capacidad de almacenamiento) a otros nodos de la red, básicamente porque entre estos comparten intereses en común. Como consecuencia de esto, al aumentar la cantidad de usuarios en la red, también aumenta la cantidad de recursos disponibles, solucionando el problema de escalabilidad. Por otro lado, el ancho de banda requerido por el ISP (o por quien se encuentre a cargo de la distribución del contenido), disminuye considerable, sucediendo lo mismo con los costos asociados a la distribución. Además de esto, los nodos que sirven a un *peer* pueden ser elegidos según un criterio de proximidad, evitando de esta manera los cuellos de botella en la red.

Como desventaja, se tiene que estos sistemas deben tratar con recursos muy dinámicos (los *peers* se conectan y desconectan de la red con una alta frecuencia, en forma autónoma y asincrónica), lo que impone restricciones a nivel de control, dado que la red debe ser robusta para soportar estas fluctuaciones.

En la actualidad existen algunas redes P2PTV comerciales muy exitosas, tales como: PPlive [103], TVUnetwork [127], Octoshape [7], SopCast [121], TVAnts [126], etc. Pero no se registra ninguna red de tipo P2PTV gratuita y de código/protocolo abierto.

Por otro lado, considerando este contexto y las ventajas ofrecidas por los sistemas de tipo P2PTV, es que en Febrero del 2010, la IETF [54] anuncio la creación de un grupo de trabajo (llamado PPSP: *Peer to Peer Streaming Protocol*), con el fin de crear y estandarizar un protocolo de *streaming* P2P tanto para contenidos en vivo como para videos bajo demanda.

Debido al contexto presentado, y en base a las experiencias recolectadas en nuestro anterior proyecto GOL!P2P [110], se plantea como principal objetivo de esta tesis, la definición de un protocolo abierto de tipo P2P para la distribución de video en vivo (llamado *GoalBit Transport Protocol*), junto con el apoyo a la comunidad *open-source* en la implementación de un primer cliente de referencia (llamado GoalBit). Además de esto, se contribuirá con el grupo PPSP de la IETF, en la definición de un protocolo estándar de *streaming* P2P, aportando nuestros conocimientos y nuestra experiencia adquirida.

El protocolo *GoalBit Transport Protocol* (GBTP), se basa en el protocolo BitTorrent [19] (el protocolo P2P más exitoso y popular para el intercambio de archivos), el cual fue adaptado con el fin de soportar la distribución de video en vivo, junto con sus grandes restricciones de tiempo real (adaptando la comunicación entre los *peers*, las políticas y estrategias aplicadas en el protocolo, etc.). Como complemento a la especificación del protocolo GBTP, se define el *GoalBit Packetized Stream* (GBPS), especificación que define como encapsular el contenido audiovisual dentro del protocolo GBTP. Respecto al cliente GoalBit (implementación de referencia), este fue desarrollado en base a la integración de 3 diferentes aplicaciones de código abierto: Videolan Media Player (VLC) [131] (aplicación usada para todo lo referente a la manipulación, codificación y reproducción de video), Enhanced CTorrent [43] (usada como cliente

BitTorrent de base) y OpenTracker [97] (usada como *tracker* BitTorrent de base).

Por otro lado, uno de los principales desafíos al montar una red de distribución de video, es el hecho de asegurar que el servicio brinda al menos un mínimo de la calidad esperada por los usuarios. Una buena medida de esto es la llamada calidad de experiencia o también QoE (en ingles: *Quality of Experience*), la cual se define como la performance global de un sistema visto desde la perspectiva de un usuario (o en otras palabras esta es una medida subjetiva de que tan bien o mal un sistema funciona, desde el punto de vista de un usuario final). A su vez, para el caso de un servicio de distribución de video, la QoE principalmente se corresponde con la calidad percibida por los usuarios al momento de reproducir el video (que tan bien este se ve). Por lo tanto, es más que importante (sino necesario), el hecho de monitorear continuamente una red de distribución de video con el fin de detectar rápidamente fallas y poder corregirlas. Resulta también muy importante, el hecho de lograr un monitoreo completo de la red, basándose en la calidad percibida por los usuarios finales.

Si bien hoy en día existen múltiples soluciones de monitoreo como [46, 47, 63, 84, 85], todas suelen ser soluciones de alto porte, muy costosas (en términos económicos) que a su vez, no brindan una visión completa de la calidad percibida en la red, sino que en su mayoría aplicando alguna técnica de *"probing"* (es decir, agregando dispositivos de monitoreo en lugares estratégicos de la red), nos brindan una vista parcial del estado de la distribución de los contenidos.

Debido a estos motivos, se propone integrar el trabajo realizado en [110] sobre la metodología PSQA (en ingles: *Pseudo-Subjective Quality Assessment*) al diseño del protocolo *GoalBit Transport Protocol* y a su implementación de referencia (el cliente GoalBit), con el fin de poder brindar reportes automáticos de la calidad percibida por los usuarios finales al ejecutar un *streaming* GoalBit.

Finalmente, con el objetivo de presentar el potencial de la tecnología P2P, junto con una forma de validar nuestras ideas, plateamos como objetivo de esta tesis el desarrollo de un prototipo de una red de distribución de contenidos en vivo (o en ingles: *Content Distribution Network* (CDN)), el cual incluya las siguientes características:

- Red de transporte basada en GBTP.
- Distribución dinámica de recursos (frente a eventos de gran demanda, poder incrementar automaticamente los recursos asociados a dicho evento).
- Administración sencilla de la red (brindar un sencilla administración de la red, mediante una interfaz *Web*).
- Adaptación de la señal GBTP a otras tecnologías (por ejemplo brindar la posibilidad de adaptar un streaming GBTP con el fin de poder ser distribuido a cliente Flash).
- Monitoreo de la calidad de experiencia existente en la red (en base a reportes PSQA realizados por el $100\,\%$ de los usuarios finales).
- Monitoreo genérico del estado de las diferentes señales (en base a la técnica de *probing*), como complemento al monitoreo basado en PSQA y como monitoreo de base para tec-

nologías en las cuales no se encuentra integrado los reportes PSQA (tal como es el caso del *streaming* Windows o del *streaming* Flash).

- La implementación de todos los componentes de la red sean de código abierto.

1.2 Contribuciones

Las contribuciones de esta tesis se pueden dividir en 2 grandes temas: redes P2P para la distribución de video en vivo, y monitoreo de redes de distribución de contenido. A continuación se describen dichas contribuciones agrupadas según estos temas. Las publicaciones relacionadas con los resultados obtenidos, se listan en el final de este capítulo.

1.2.1 Redes P2P para la Distribución de Video en Vivo

Los principales resultados de esta tesis consisten en los aportes realizados al campo de las redes P2P para la distribución de video en vivo.

Se presenta un completo estado del arte de las actuales redes de tipo P2PTV, entre las que se destacan PPLive [103] con más de $200,000$ usuarios concurrentes en un mismo canal [116], SopCast [121] con más de $100,000$ usuarios concurrentes, Octoshape [95], entre otras. Para cada uno de éstas, se brindan detalles sobre sus protocolos e implementaciones (muchas veces en base a información obtenida mediante estudios de ingeniería inversa y otras en base a información brindada por los propios desarrolladores).

Se define el protocolo *GoalBit Transport Protocol* (GBTP), un protocolo abierto, de tipo P2P para la distribución de contenidos en vivo. Este protocolo es una extensión de BitTorrent y establece como el contenido debe ser transportado sobre la red de pares. A su vez, se define la especificación *GoalBit Packetized Stream* (GBPS), en donde se establece el modo en que el video debe ser paquetizado y encapsulado en las piezas GBTP. Debido a sus diseños, tanto GBPS como GBTP no presentan restricción alguna sobre las codificaciones (audio y/o video) y formatos contenedores a encapsular/transportar. Estas especificaciones (junto con algunos resultados empíricos) fueron publicadas en [br-lanc09].

Se apoyó a la comunidad *open-source* en la implementación de un primer cliente de referencia de las especificaciones GBTP/GBPS, llamado GoalBit. Este hecho, además de ser un gran aporte a la comunidad *open-source*, es un importante aporte a la comunidad académica y en particular a la generación de nuevos proyectos de investigación sobre las redes de tipo P2PTV. Con GoalBit se dispone de un protocolo y de una herramienta accesible, en donde es posible desarrollar y probar nuevas políticas, nuevas estrategias, etc. El desarrollo de GoalBit es basado en aplicaciones abiertas y muy populares como el reproductor de video VLC [131].

En base a la experiencia adquirida en el proceso de diseño e implementación de GoalBit, se contribuyó con el grupo PPSP de la IETF en la definición de un protocolo estándar de *streaming* P2P.

Tanto a nivel de pruebas empíricas, como en emulaciones realizadas, se observó que la estrategia de selección de piezas, la cual define que piezas un par debe solicitar a otro, es un factor de gran relevancia en la calidad percibida por los usuarios finales. Por este motivo, se presenta un modelo matemático de cooperación entre pares, sobre el cual se realiza un detallado

análisis de las posibles estrategias de selección de piezas y sus propiedades. Para cada una de estas se calculan sus valores de latencia (tiempo de *buffering* inicial) y de continuidad en la reproducción del *streaming*. En base a estos conceptos y con el fin de encontrar una estrategia de selección de piezas óptima, se plantea un problema de optimización combinatorio, el cual es resuelto en base a la aplicación de metaheurísticas como búsqueda local y colonia de hormigas, obteniendo una estrategia de selección de piezas, con considerables mejoras respecto a las existentes anteriormente en la literatura. Este análisis, desarrollado en equipo [1], es base de la Mastría de Pablo Romero [109] y fue presentado parcialmente en [col-latincom09] y [ruicumt09]. En particular, en esta tesis se introduce y compara una nueva estrategia, la cual es finalmente usada en GoalBit.

Se realiza un prototipo de una red de distribución de contenido basado en GoalBit, el cual provee funcionalidades como: una simple administración de los contenidos transmitidos y sus recursos asociados, asignación dinámica de recursos según la demanda de cada contenido, adaptación del *streaming* a diferentes infraestructuras (tal como adaptación al *streaming* Flash o Windows), etc.

1.2.2 Monitoreo de Redes de Distribución de Contenido

Además de las contribuciones referentes al campo de las redes P2P para la distribución de video en vivo, esta tesis realiza importantes aportes en lo referente al monitoreo en redes de distribución de contenidos, principalmente al área de monitoreo de la calidad de experiencia.

Siguiendo la técnica publicada en [usa-ipom07], y haciendo uso de la función de PSQA diseñada previamente en la tesis de doctorado de Pablo Rodriguez-Bocca [110], se construyó un primer prototipo con el fin de monitorear en tiempo real una red de distribución de video usando PSQA. Este prototipo permite cambiar de manera interactiva los parámetros de congestión de una red (configurando pérdidas a nivel de paquetes de red o a nivel de frames de videos), pudiéndose observar en parelelo el efecto de dichas pérdidas sobre el video (mediante la reproducción del mismo) y sobre la función PSQA (mediante la presentación de una gráfica de actualización constante). Dicho prototipo fue presentado en [usa-sigm08], en donde fue condecorado con el premio *"Best Student Demonstration Award"*, siendo posteriormente publicado en [acm-per08].

Partiendo de los mismos conceptos implementados en el prototipo anteriormente mencionado, se integró el monitoreo PSQA tanto al protocolo GBTP, como a su implementación de referencia. Siendo GoalBit la primer red de distribución de video en vivo en realizar mediciones en tiempo real de la calidad de experiencia aplicando la metodología PSQA.

Basado en el cliente GoalBit, se desarrolló un prototipo de un monitor genérico de redes de distribución de video, con el cual es posible visualizar el estado tanto de señales en vivo, como de videos a ser servidos bajo demanda, sea cual sea su codificación y protocolo de distribución. Con el fin de presentar el potencial de este monitor, se lleva a cabo un profundo y detallado monitoreo sobre YouTube [145], la mayor red de distribución de videos bajo demanda de la actualidad.

[1] El equipo de investigación está compuesto por los siguientes miembros: María Elisa Bertinat, Daniel de Vera, Darío Padula, Franco Robledo, Pablo Rodríguez Bocca, Pablo Gabriel Romero y Gerardo Rubino

1.3 Resumen del Documento

Lo que sigue en este documento se encuentra organizado tal como se describe a continuación.

Parte I INTRODUCCIÓN. Luego de este capítulo introductorio, el Capítulo 2 presenta conceptos generales, necesarios para comprender este trabajo. Más en detalle, se introducen breves conceptos sobre el video digital, las redes de distribución de video y se presenta un servicio de distribución de video, llamado AdinetTV [3], el cual será referenciado a lo largo del documento.

El Capítulo 3 se centra en la presentación de un resumen del estado del arte de las redes P2P para la distribución de video, junto con la introducción de ciertos conceptos referidos a éstas. Se presentan los problemas de costos y escalabilidad existentes en las redes de distribución de contenido clásicas, y como las redes P2P minimizan los mismos. Con el fin de comprender los diferentes tipos de redes P2P, se presenta una taxonomía en la cual las redes pueden ser clasificadas. Se realiza una breve introducción al protocolo de intercambio de archivos BitTorrent [19], en el cual se basan muchas de las redes de distribución de contenidos en vivo. Se presenta un completo estado del arte de las redes P2P para la distribución de video en vivo. Para cada una de éstas se brindan detalles sobre su protocolo y su implementación. Finalmente se presenta el grupo de trabajo *Peer to Peer Streaming Protoco*l (PPSP) de la IETF, su problemática asociada y su estado de avance.

El Capítulo 4 sumariza los diferentes métodos para evaluar la calidad de video. Se presentan las evaluaciones subjetivas, las objetivas y un enfoque híbrido, llamado *Pseudo-Subjective Quality Assessment* (PSQA). Se brindan detalles para cada una de éstas, haciendo principal foco en la metodología PSQA y su aplicación. Finalmente se presenta un resumen de las actuales soluciones para el monitoreo de redes de distribución de contenido.

Parte II *STREAMING* GOALBIT. En el Capítulo 5 se especifica el *streaming* GoalBit. Se presenta su arquitectura, su protocolo de transporte (llamado *GoalBit Transport Protocol*), la paquetización del *streaming* de video asociado con este protocolo (llamada *GoalBit Packetized Streaming*), su implementación y las diferentes estrategias y políticas aplicadas en el *streaming* GoalBit.

En el Capítulo 6 se realiza un detallado análisis de las posibles estrategias de selección de piezas y sus propiedades. Se presenta un modelo matemático de cooperación entre pares, sobre el cual se plantea un problema de optimización combinatorio. Este problema es resuelto mediante la aplicación de metaheurísticas como búsqueda local y colonia de hormigas, obteniendo una estrategia de selección de piezas con muy buenas propiedas en lo referente a latencia y continuidad en la reproducción. Como resultado de este capítulo, y en base a emulaciones realizadas con GoalBit, se compara una de las estrategias de selección de piezas óptimas del modelo, con la actual estrategia desarrollada en GoalBit.

El Capítulo 7 brinda algunos resultados empíricos obtenidos al usar GoalBit para distribuir contenidos. Se presenta un profundo análisis del protocolo GBTP, mostrando su comportamiento y su *overhead* agregado a la transmisión. Finalmente se presentan los resultados obtenidos durante la transmisión de un evento muy popular, realizado por un usuario final.

Parte III EXTENSIONES DEL STREAMING GOALBIT. El Capítulo 8 describe la Plataforma GoalBit, una extensión del *streaming* GoalBit con el fin de lograr una mejor utilización y administración de los recursos de la red. En este, se presenta su arquitectura, su implementación y su implantación a modo de prueba en AdinetTV.

El Capítulo 9 detalla la integración del monitoreo de la calidad de experiencia mediante PSQA, en el *streaming* GoalBit. Se presenta su arquitectura, su implementación y algunos de los resultados obtenidos al hacer uso de la plataforma GoalBit en AdinetTV.

En el Capítulo 10 se introduce el diseño e implementación de un monitor de video genérico, llamado el *Monitor GoalBit*, con el cual es posible monitorear el estado tanto de contenidos en vivo, como de contenidos bajo demanda, independientemente de sus protocolos de *streaming* y sus tipos de codificación. Con el fin de demostrar el potencial de este monitor, se realiza un completo análisis de YouTube [145], detallando las principales características de los videos presentes en esta red.

Finalmente en el Capítulo 11 se presentan las conclusiones generales de este trabajo y sus perspectivas a futuro.

1.4 Publicaciones y Demostraciones

Esta sección contiene una lista de las publicaciones realizadas durante el transcurso de esta tesis, junto con las demostraciones realizadas en conferencias.

1.4.1 Publicaciones

[br-lanc09] María Elisa Bertinat, Daniel De Vera, Darío Padula, Franco Robledo, Pablo Rodríguez-Bocca, Pablo Romero y Gerardo Rubino *GoalBit: The First Free and Open Source Peer-to-Peer Streaming Network*, en: *5th international IFIP/ACM Latin American conference on Networking (LANC'09).*, Pelotas, Brazil, 2009.

[col-latincom09] María Elisa Bertinat, Daniel De Vera, Darío Padula, Franco Robledo, Pablo Rodríguez-Bocca y Pablo Romero *Systematic Procedure for Improving Continuity and Latency on a P2P Streaming Protocol*, en: *1th IEEE Latin-American Conference on Communications (LatinCom'09).*, Medellín, Colombia, 2009.

[ru-icumt09] María Elisa Bertinat, Daniel De Vera, Darío Padula, Franco Robledo, Pablo Rodríguez-Bocca, Pablo Romero y Gerardo Rubino *A COP for Cooperation in a P2P Streaming Protocol*, en: *IEEE International Conference on Ultra Modern Telecommunications (ICUMT'09).*, St.Petersburg, Rusia, 2009.

[acm-per08] Daniel De Vera, Pablo Rodríguez-Bocca y Gerardo Rubino *Automatic quality of experience measuring on video delivering networks*, en: *ACM SIGMETRICS Performance Evaluation Review, Volume 36, Issue 2.*, páginas 79-82, 2008.

[usa-ipom07] Daniel De Vera, Pablo Rodríguez-Bocca y Gerardo Rubino *QoE Monitoring Platform for Video Delivery Networks*, en: *7th IEEE International Workshop on IP Opera-*

tions and Management (IPOM'07)., San José, California, Estados Unidos, 2007.

1.4.2 Demostraciones

[usa-sigm08] Daniel De Vera, Pablo Rodríguez-Bocca y Gerardo Rubino *Automatic quality of experience measuring on video delivering networks*, en: *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'08). Ganadora del Best Student Demonstration Award*, Annapolis, Maryland, Estados Unidos, 2008.

Capítulo 2

Conceptos Generales

Este capítulo se basa fuertemente en el estado del arte presentado en [110] y introduce conceptos generales necesarios para lograr una completa comprensión de este trabajo.

2.1 Video Digital

El video digital es una representación digital de la señal de video, en lugar de una analógica. Este se refiere a la captura, manipulación, distribución y almacenamiento de la señal de video en formatos digitales, permitiendo de esta manera un mecanismo más flexible de manipular y/o presentar el video por un computador.

El video digital dio a lugar una gran cantidad de nuevas aplicaciones, tales como la emisión de video sobre cables digitales, vía satélite o terrestre, video conferencias, grabación digital del video, etc. En particular, la transmisión de video sobre redes de paquetes comenzó en los mediados de 1990, junto con el crecimiento de Internet.

2.1.1 Codificación de Video

La codificación o compresión de video, es una técnica diseñada para eliminar las similitudes o redundancias existentes en una señal de video. Una señal digital de video es una secuencia de imágenes digitales. Normalmente las imágenes consecutivas dentro de la secuencia tienen redundancia temporal, dado que estas presentan algún movimiento sobre ciertos objetos. A su vez, dentro de las imágenes también existe redundancia espacial, debido a que en general existe una correlación entre píxeles vecinos. Además de eliminar estas redundancias, los codificadores de video generan pérdidas de información, dado que estos únicamente codifican la información relevante a la percepción del video, dejando de lado información imperceptible y mejorando de esta manera el rango de compresión.

Un método de compresión es definido mediante su codificador y su decodificador, llamados conjuntamente CODEC (del ingles: *enCOder/DECoder*).

Hoy en día, existen varios *codecs* usados en forma masiva. La mayoría de ellos basados en estándares [65] propuestos por el *"Moving Pictures Expert Group"* (MPEG) y algunos otros propietarios. Dentro de los *codecs* basados en estándares de MPEG, se encuentra MPEG-2,

MPEG-4, H.264 (MPEG-4 parte 10 [66]), XVID [142] (MPEG-4 parte 2 [67]), etc. Dentro de los *codecs* propietarios, algunos ejemplos son RealVideo [107], Windows Media Video, On2 [96], etc.

Tanto los *codecs* basados en estándares como los *codecs* propietarios comparten los mismos principios de compresión, por lo que el entendimiento de uno de ellos, genera un entendimiento básico de toda el área de compresión.

A continuación veremos una breve reseña del funcionamiento de los *codecs* basados en el estándar MPEG. Un concepto clave dentro del estándar es el llamado *"frame"*, este representa la codificación de una imagen del video. Resumidamente, existen 3 diferentes tipos de *frames*: los *I-frames* (en ingles: *Intra frames*), los *P-frames* (en ingles: *Predicted frames*) y los *B-frames* (en ingles: *Bidirectional frames*).

Los *I-frames* codifican una imagen completa, se utilizan para decodificar los otros *frames* que componen el video y pueden ser utilizados como puntos de acceso aleatorio para comenzar a decodificar un video a partir de un cierto momento. En general, la codificación de estos *frames* ocupa más espacio que la de los otros tipos de *frames*.

Los *P-frames* son imágenes predichas con referencia a un *I-frame* o *P-frame* anterior, por lo que es necesario la decodificación del *frame* de referencia antes de poder decodificadar el *P-frame* en cuestión.

Los *B-frames* son imágenes predichas con referencia a dos *frames* que pueden ser de tipo I o P, uno anterior y uno posterior. De esta manera es necesario la decodificación de los dos *frames* de referencia así como la reordenación de los *frames* para poder decodificadar un *B-frame*.

La Figura 2.1 muestra las dependencias entre los diferentes tipos de *frames*.

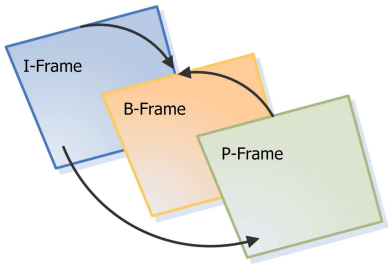


Figura 2.1: Dependencias entre tipos de *frames*. Un *I-frame* representa una imagen. Los *P-frames* son imágenes predichas con referencia a un *I-frame* o *P-frame* anterior. Los *B-frames* son imágenes predichas con referencia a dos *frames* que pueden ser de tipo I o P, uno anterior y uno posterior.

La secuencia de *frames* es dividida en grupos de *frames* llamados GOP (en ingles: *Group of Pictures*). Un GOP es un conjunto de *frames* que pueden ser decodificados de forma independiente al resto (no se necesita la referencia a ningún *frame* por fuera de este conjunto), este es definido como la secuencia de *frames* comenzando desde un *I-frame* hasta el *I-frame* siguiente. Usualmente un error en un *I-frame* se propaga hasta el siguiente *I-frame* (debido a las dependencias existentes entre los *frames* dentro de un mismo GOP). A su vez, errores en los *P-frames* son propagados hasta el siguiente *I-frame* o *P-frame* y los errores en *B-frames* no propagan errores. Se puede observar que al aumentar la cantidad de *I-frames* en la codificación

del video, se aumenta la robustez del video ante posibles errores. Sin embargo al aumentar la cantidad de *I-frames*, consecuentemente se aumenta el tamaño del video. Dentro de la especificación MPEG-2, el tamaño promedio de los GOPs es de 20 *frames*, mientras que en MPEG-4 este valor alcanza los 250 *frames* (esto se debe a que MPEG-4 posee una mayor expresibilidad y flexibilidad, permitiendo así una mejor compresión usando más *P-frames* y *B-frames*).

2.1.2 Multiplexación del audio y video

En la subsección anterior vimos el proceso de codificación del video, en donde partiendo desde una señal analógica de video se genera una señal digital. Este mismo proceso también ocurre con el audio (y si bien, el proceso de codificación del audio es diferente al de video, el resultado es el mismo: la generación de una señal digital y comprimida del audio). Tanto a la señal codificada de audio como a la de video, se le llaman flujos o *streams* elementales.

Normalmente los flujos elementales asociados a un mismo contenido son almacenados y/o transmitidos multiplexados entre sí (por ejemplo una película y sus audios en diferentes idiomas). Para llevar a cabo esta tarea existen diferentes formatos contenedores (también llamados *muxers*), entre los que se encuentran MPEG-TS, MPEG-PS, ASF (conocido como AVI), MP4, OGG, etc.

Como es de imaginar, más allá de que todos estos contenedores cumplen una función común (la de multiplexar y sincronizar varios *streams* elementales), existen grandes diferencias entre estos. Mientras que algunos fueron diseñados especialmente para la transmisión de contenidos en vivo (como por ejemplo MPEG-TS) otros fueron diseñados para un escenario de video bajo demanda (como por ejemplo MPEG-PS, ASF, MP4, etc.). Esto no significa que para transmitir contenidos en vivo no se pueda usar contenedores como ASF o MPEG-PS, ni que para almacenar un video no se pueda usar MPEG-TS, todos los formatos contenedores son genéricos y permiten ser usados bajo cualquier escenario.

Las especificaciones diseñadas para distribuir contenido en vivo insertan periódicamente información acerca de la multiplexación dentro del *streaming* resultante (como por ejemplo: cantidad de *streams* multiplexados, ID de cada *stream*, etc.) e información acerca de la sincronización de los flujos elementales (referencia de relojes, etc.). Las otras especificaciones, almacenan esta información en un paquete especial llamado *"header del muxer"*, el cual como se presenta en la Figura 2.2, debe ser el primer paquete enviado al usuario (sin este paquete no es posible demultiplexar el *streaming*, siendo así imposible reproducir el contenido).

2.1.3 Distribución del audio y video

Una vez que el audio y el video fueron multiplexados, generándose así un único *stream* de datos, este puede ser almacenado y/o distribuido. En caso que el contenido sea distribuido sobre una red, este es encapsulado en paquetes de algún protocolo de transporte y enviado por la misma. Hoy en día existen varios protocolos de transporte adecuados para distribuir un *streaming* de video; algunos de ellos son: UDP [55] (en ingles: *User Datagram Protocol*), RTP [56] (en ingles: *Real-time Transport Protocol*), HTTP [57] (en ingles: *Hypertext Transfer Protocol*), MMS [81] (en ingles: *Microsoft Media Server*) y RTMP [4] (en ingles: *Real Time Messaging*

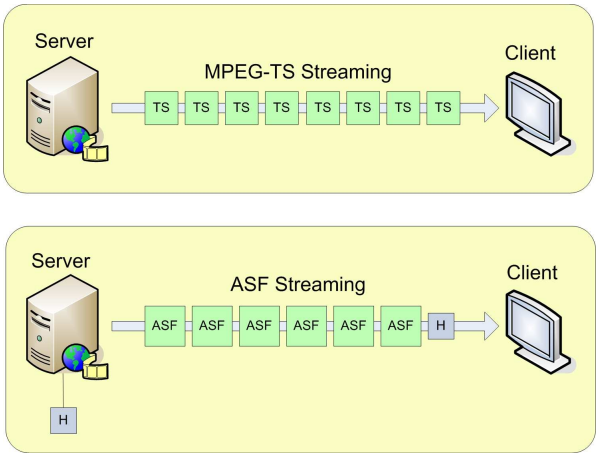


Figura 2.2: Diferencias entre formatos contenedores.

Protocol). Cada uno de estos protocolos presenta ciertas ventajas en algunos escenarios y ciertas desvantajas en otros.

UDP es un protocolo no orientado a conexión, el cual por ejemplo es apropiado para las transmisiones en vivo. RTP fue especialmente diseñado para distribuir contenidos sobre redes de paquetes, este se basa en UDP y como su nombre lo sugiere se especializa en la realización de transmisiones en vivo. Finalmente HTTP es un protocolo orientado a conexión, el cual garantiza el correcto envío de cada paquete dentro del *streaming*. Este protocolo presenta una gran ventaja frente a los otros, y es el hecho de no ser filtrado por los *firewalls*, lo que lo llevo a ser un protocolo muy usado dentro de Internet.

2.2 Redes de Distribución de Video

Una **red de distribución de video** o también llamadas **VDN** (en ingles: *Video Delivery Network*), es un sistema que distribuye video de forma transparente a usuarios finales [110]. Hoy en día, estas redes se encuentran en pleno crecimiento debido al incesante aumento del ancho de banda disponible a nivel de las redes de acceso (en Internet, en redes celulares, en redes IP privadas, etc.), generando así nuevos posibles modelos de negocio para los proveedores de contenido, ISPs, entre otros.

Los proveedores de servicios han desarrollado varios tipos diferentes de servicios usando el *streaming* de video, como por ejemplo: *Broadcast TV* (BTV), *Video-on-Demand* (VoD), video conferencias, monitoreo de seguridad, etc. Desde el punto de vista de la red, las VDNs presentan varias arquitecturas diferentes. Las más exitosas hoy en día pueden ser clasificadas dentro de las siguientes categorías: *Digital Cable and Satellite* (televisión digital por cable o vía satélite), *Internet TV* (televisión sobre Internet), P2PTV (televisión sobre redes P2P), IPTV (televisión sobre redes IP) y Mobile TV (televisión sobre redes móviles).

A continuación se presenta una arquitectura genérica para todas las VDNs, para luego mostrar una breve reseña de cada una de las posibles arquitecturas.

2.2.1 Arquitectura Genérica

Desde un punto de vista genérico y a alto nivel, todas las VDNs comparten la misma arquitectura (ver Figura 2.3). El flujo de datos dentro de esta arquitectura esta compuesta por 5 etapas diferentes: la adquisición, la codificación, la paquetización, la distribución y decodificación.

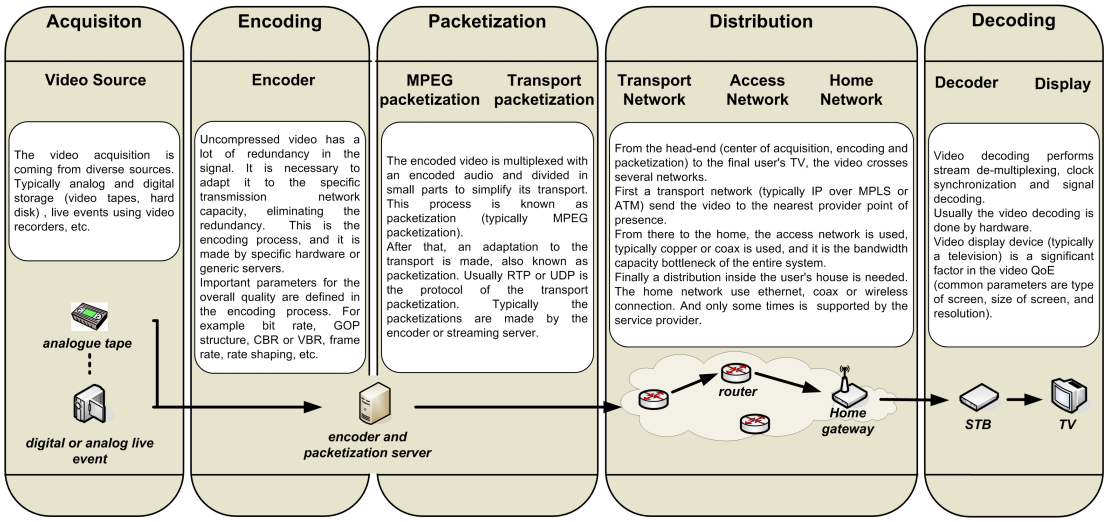


Figura 2.3: Arquitectura genérica de las redes de distribución de video. El flujo de datos dentro de esta arquitectura esta compuesta por 5 etapas diferentes: la adquisición, la codificación, la paquetización, la distribución y decodificación.

Existen varias posibles fuentes para la adquisición del video, por ejemplo algunas de ellas son: almacenamiento analógico o digital (discos duros, cintas de video), eventos en vivo (filmados a través de una cámara de video), etc. Debido a que la señal adquirida contiene gran cantidad de redundancia, esta es eliminada (o al menos reducida) en la etapa de codificación (ver “Codificación de Video” dentro de la Sección 2.1). En esta etapa se adapta tanto la señal de video como la de audio al medio de transmisión (normalmente mediante el uso de un *hardware* específico o de servidores dedicados). Algunos de los posibles parámetros a definir en esta etapa son: *codec* de video, *codec* de audio, codificación a tasa constante o a tasa variable, tasa de codificación (en ingles: *bitrate*), tamaño de GOP, etc.

Luego de la etapa de codificación, el audio y el video son multiplexados, creándose así un único flujo de transmisión (ver “Multiplexación del audio y video” dentro de la Sección 2.1). Este flujo de transmisión, es a su vez encapsulado dentro de algún protocolo de transporte (ver “Distribución del audio y video” dentro de la Sección 2.1).

En la etapa de distribución, el contenido suele viajar sobre diferentes redes antes de llegar al usuario final. Primero por una red de transporte (generalmente IP sobre MPLS o ATM), la cual se encuentra más próxima al proveedor del servicio. Desde allí hasta el hogar del usuario, el contenido viaja a través de una red de acceso (típicamente una red montada sobre cables coaxiales o de cobre), esta red suele ser el cuello de botella en la distribución del contenido. Finalmente y ya dentro del hogar del usuario, el contenido viaja a través de la red residencial (en general montadas sobre Ethernet, cables coaxiales o wireless).

En la etapa de decodificación, el flujo de transporte es demultiplexado, separando el flujo de audio del de video. Luego estos son decodificados en forma independiente para finalmente ser presentados al usuario (normalmente a través de un televisión o un monitor).

2.2.2 Arquitecturas Específicas

A continuación se presentan un conjunto de arquitecturas específicas, las cuales respetan la arquitectura genérica anteriormente presentada. En general estas arquitecturas específicas varían entre sí exclusivamente respecto a los protocolos aplicados en la etapa de distribución (los procedimientos, las especificaciones y los protocolos aplicados en el resto de las etapas suelen ser compartidos).

Internet TV. Este tipo de VDN, se encuentra montada sobre Internet, típicamente sobre un conjunto de *datacenters* los cuales absorben toda la carga de los usuarios. Algunos ejemplos bien conocidos son: YouTube [145], msnTV [92], JumpTV [70], JustinTV [71], etc. Normalmente estas redes utilizan protocolos de transporte basados en TCP, como por ejemplo HTTP, MMS o RTMP.

P2PTV. Otro método de streaming muy popular en Internet, consiste en usar la capacidad ociosa de los usuarios para ayudar a los servidores de *streaming* en la distribución de los contenidos. Estos sistemas son llamados entre pares o P2P (en ingles: *Peer-to-Peer*). Actualmente, algunos de los más existosos son: PPlive [103], TVUnetwork [127], PPstream [104], SopCast [121], TVAnts [126], etc.

Estos sistemas crean redes virtuales a nivel de aplicación sobre la infraestructura de Internet. Los nodos en la red son llamados *pares*, estos ofrecen sus recursos (ancho de banda, capacidad de procesamiento y capacidad de almacenamiento) a otros nodos en la red, básicamente porque entre estos comparten intereses en común. Al aumentar la cantidad de usuarios en la red, también aumenta la cantidad de recursos disponibles. Esto es llamado *escalabilidad* en Internet. Claramente el usar estos sistemas para distribuir video parece ser una buena idea, más teniendo en cuenta los altos requerimientos en términos de ancho de banda existen en la distribución de video. En este trabajo desarrollamos GoalBit, una red P2PTV.

IPTV. IPTV es una arquitectura de red en donde servicios de televisión digital (BTV) y video bajo demanda (VoD) son distribuidos usando el protocolo IP, sobre una red dedicada y cerrada (notar que en IPTV no se utiliza Internet como red de distribución). En este tipo de arquitectura, las 3 primeras etapas de la arquitectura genérica (adquisición, codificación y multiplexado) son realizadas en el centro de datos llamado *head-end*, y para decodificar la señal se usan *set-top boxes*, los cuales se conectan directamente a los televisores de los usuarios. Típicamente el servicio de IPTV es ofrecido por ISPs, junto con el servicio de voz sobre IP (VoIP) y el acceso a Internet (comúnmente llamado *"Triple Play"*). Algunos de los proveedores de infraestructura para IPTV existentes en el mercado actual son: Alcatel [5], Siemens [119] y Cisco [31].

MobileTV. MobileTV es el nombre usado para describir un servicio de distribución de video (tanto en vivo como video bajo demanda) a teléfonos móviles. Este servicio es comúnmente brindado por operadores móviles. Existe un proceso de estandarización de la infraestructura MobileTV hacia una de tipo IMS [17] (en ingles: *IP Multimedia Subsystem*). Se plantea también esta generalización para otras arquitecturas como IPTV o InternetTV, creando así una comunicación multimedia universal.

Televisión Digital por Cable o vía Satélite. La televisión digital por cable emite televisión a los usuarios vía señales de frecuencia de radio transmitidas sobre redes de cable coaxial o fibra óptica y la televisión digital satelital transmite mediante satélites. Normalmente en estas arquitecturas se usan los estándares de MPEG-2 o MPEG-4 (para codificar y multiplexar).

2.2.3 Calidad de Experiencia en una VDN

Uno de los principales desafíos al montar una red de distribución de video, es el hecho de asegurar que el servicio brinda al menos un mínimo de la calidad esperada por los usuarios. La calidad de experiencia o también llamada QoE (en ingles: *Quality of Experience*), es la performance global de un sistema, visto desde la perspectiva de un usuario[1]. QoE es básicamente una medida subjetiva de que tan bien o mal un sistema funciona, desde el punto de vista de un usuario final. Por lo tanto la QoE es un buen indicador de si el sistema cumple con sus objetivos y en que medida. Para identificar los factores que tienen un rol relevante en la QoE de un sistema, ciertos aspectos cuantitativos deben ser tomados en cuenta [41]. Para el servicio de distribución de video, el más importante es la calidad percibida al reproducir el video.

Existen múltiples factores en cada una de las etapas de la arquitectura genérica de las VDNs (ver subsección anterior: Arquitectura Genérica) que pueden afectar directamente en la calidad percibida al reproducir el video y por lo tanto en la QoE medida en el sistema. En la etapa de adquisición, el hecho de disponer de una buena calidad en la señal original es muy importante. En la etapa de codificación, la señal debe ser codificada teniendo en cuenta el medio de distribución y sus restricciones (en particular el ancho de banda disponible). Por lo que se debe definir el *bitrate* de la codificación, respecto a la arquitectura objetivo. En InternetTV o P2PTV, arquitecturas montadas sobre Internet en donde los usuarios usan conexiones de banda ancho residenciales, el ancho de banda disponible varia de 500 a 1500 Kbps. En MobileTV, la red de acceso tiene un ancho de banda disponible de unos 100 Kbps en las redes 2G y de 2 Mbps como máximo en las redes 3G. En IPTV y televisión digital por cable o vía satélite, el ancho de banda disponible varia de 12 a 24 Mbps En la etapa de distribución, factores como el *delay*, *jitter* y pérdidas en la red pueden afectar directamente en la QoE (de hecho los factores relacionados con la etapa de distribución, son los que representan mayor riesgo para la calidad percibida). Finalmente en la decodificación del video, algunos de los factores que pueden afectar a la calidad del video presentado son: el tipo de pantalla, el tamaño de la misma, la resolución, etc. Como se puede observar en este pequeño resumen, claramente

[1] A veces se confunde la calidad de experiencia con la calidad de servicio (QoS). QoS se remite a medir la performance a nivel de red y desde el punto de vista de la red, QoE mide la performance desde el punto de vista de un usuario final [41].

existen un amplia cantidad de factores que pueden impactar directamente en la QoE, por lo tanto es más que importante poder medirla (cosa que no siempre es sencillo) y en lo posible tomar acciones cuando esta se encuentra por fuera de los valores normales.

2.3 AdinetTV: un Servicio de Distribución de Video de Referencia

AdinetTV [3] es un servicio de distribución de televisión en vivo y video bajo demanda, focalizado en contenidos Uruguayos, de una ISP de mediano porte. En la Figura 2.4, se presenta una captura de la interfaz *Web* de AdinetTV.

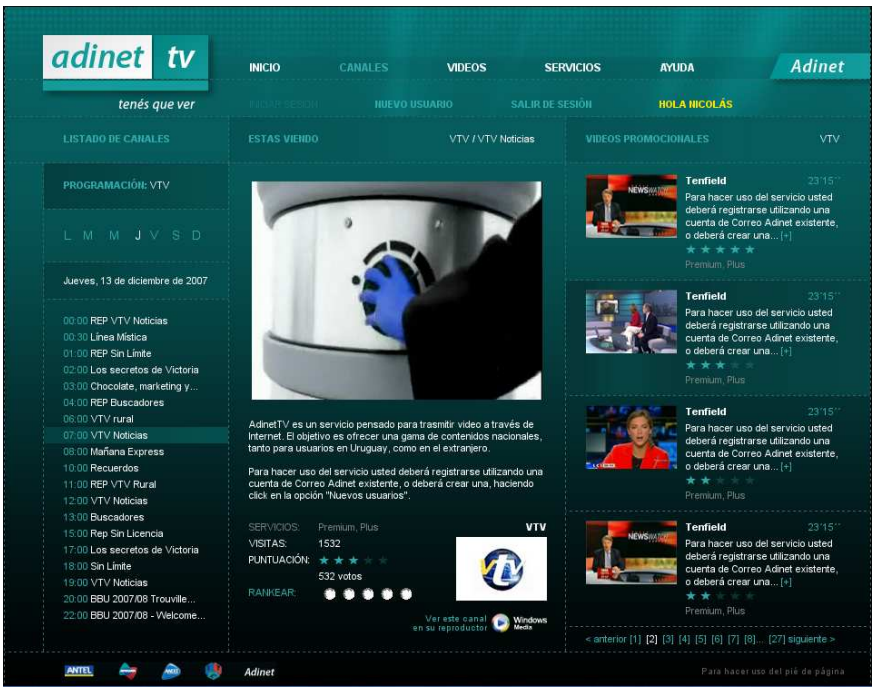
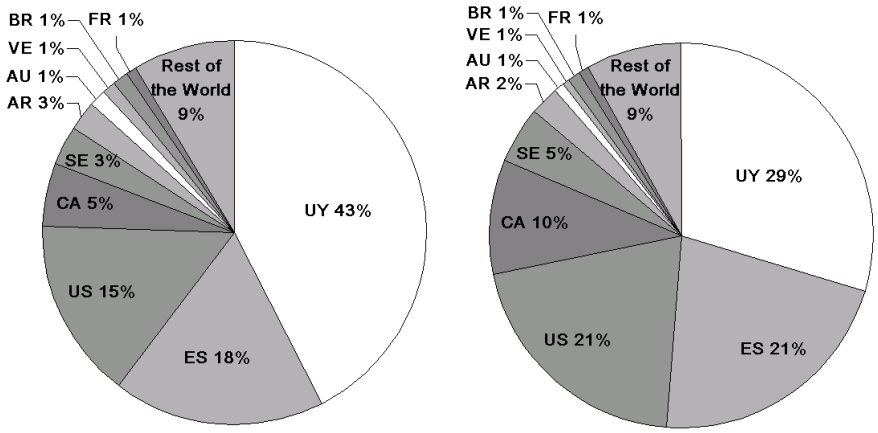


Figura 2.4: AdinetTV: un servicio de distribución de video de referencia.

AdinetTV posee una arquitectura de tipo CDN (*Content Delivery Network*), con decenas de servidores y un ancho de banda de subida total de 2 Gbps. El *bitrate* del video en vivo varía dependiendo de la cantidad de usuarios simultáneos en la red, de entre 100 Kbps a 350 Kbps (por lo tanto, la red soporta desde 5,500 a 20,000 conexiones concurrentes).

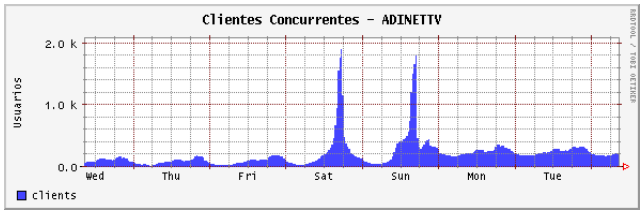
Hasta el momento, AdinetTV tiene más de 120,000 usuarios registrados. El 43 % de los usuarios se conectan a AdinetTV desde Uruguay, mientras que el 57 % restante lo realizan de países como España o Estados Unidos, en donde existen grandes comunidades de Uruguayos. La Figura 2.5(a) presenta un porcentual del origen de las conexiones a AdinetTV, mientras que la Figura 2.5(b) presenta un porcentual del tiempo acumulado de las conexiones según su origen. Como se puede observar en las mismas, los usuarios del exterior de Uruguay, permanecen mayor tiempo conectados a los *streams* de AdinetTV que los usuarios de dicho país.



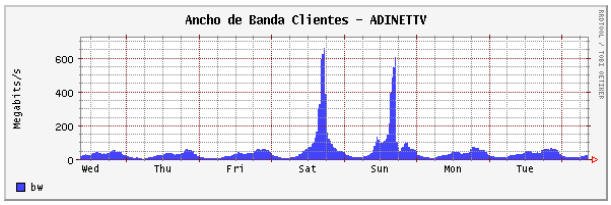
(a) Porcentual del origen de las conexiones. (b) Porcentual del tiempo acumulado de las conexiones según origen.

Figura 2.5: Uso geográfico de AdinetTV.

Durante eventos populares, como partidos de fútbol, AdinetTV ha tenido picos de $4{,}000$ usuarios simultáneos. La Figura 2.6(a) muestra los usuarios simultáneos durante una semana y la Figura 2.6(b) muestra el consumo de ancho de banda en ese período.



(a) Usuarios simultáneos.



(b) Consumo de ando de banda.

Figura 2.6: Estadísticas de AdinetTV.

Con el fin de estimar la distribución de la frecuencia de conexiones/desconexiones de usuarios, así como del tiempo de vida de una conexión, se analizaron archivos de *logs* de 4 meses de los servidores de *streaming* de AdinetTV. La Figura 2.7 presenta la distribución del tiempo de vida de una conexión en AdinetTV. Para una cantidad x de usuarios, la curva que da $F(x) = y$ representa que x usuarios tuvieron un tiempo de conexión $\geq y$.

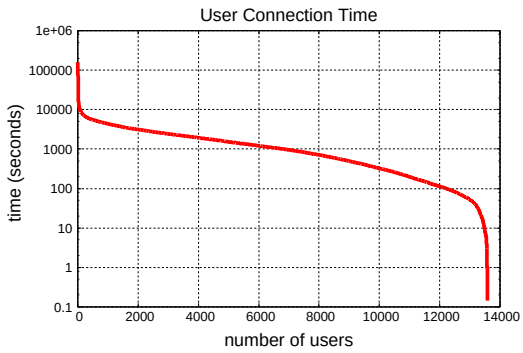


Figura 2.7: Distribución del tiempo de vida de una conexión en AdinetTV. Para una cantidad x de usuarios, la curva que da $F(x) = y$ representa que x usuarios tuvieron un tiempo de conexión $\geq y$

Capítulo 3

Redes *Peer-to-Peer* (P2P)

3.1 Escalabilidad y Costos en Internet

Al momento de desplegar una red de distribución de video sobre Internet, surgen 2 grandes problemas: la escalabilidad y los costos. Al aumentar la cantidad de usuarios en la red, es muy difícil mantener la calidad del servicio y en caso de lograrlo el costo (en términos monetarios) es alto. Esto se debe a que la única comunicación disponible en Internet es la punto a punto (desde un servidor a un usuario) y por lo tanto se le debe enviar una copia diferente del video a cada usuario. También sucede que el ancho de banda es el recurso más caro en Internet[1] y a su vez el más requerido por las redes de distribución de contenido. En particular el costo asociado a la distribución de video es proporcional a la cantidad de usuarios en la red. Además de esto, el hecho de escalar la red (por ejemplo aumentando la cantidad de servidores) sin una estricta planificación, puede causar congestiones en ciertos sectores de Internet, lo que implica una mayor tasa de pérdidas de paquetes y por lo tanto una menor calidad de experiencia en el sistema.

Pero el hecho de que las comunicaciones en Internet sean punto a punto, no es una limitación del protocolo IP (sobre el que se encuentra montado la Internet). Este protocolo provee conexiones de uno a uno, llamadas *unicast*, también provee conexiones de uno a muchos, llamadas *multicast* y finalmente también provee conexiones de uno a todos, llamadas *broadcast*. En *IP Broadcast*, los datos son enviados una única vez (la fuente envía una única copia) a todos los usuarios en la red, por escala es imposible de aplicarlo en Internet. En *IP Multicast*, los datos son enviados una única vez (la fuente envía una única copia) a todos los usuarios que se hayan registrado en la fuente como interesados en recibir dichos datos. El problema de porque únicamente se encuentra habilitado la comunicación *IP Unicast* en Internet, se debe a las reglas de negocio tradicionales establecidas en los ISPs (en ingles: *Internet Service Provider*), basadas puramente en la falta de maduración en los estándares y en la preservación del ingreso[2]

[1] Si bien hoy en día el ancho de banda es el recurso más caro en Internet, no esta claro hasta cuando esta situación se mantendría así. En los últimos años los costos se han reducido hasta casi en un 90 %, lo que ha dado lugar al despliegue de muchas nuevas redes de distribución de contenido.

[2] Existen algunas excepciones a esta regla: MBONE [44] (en ingles: *Multicast Backbone*) fue una red IP virtual, creada para transmitir video sobre *IP Multicast* desde las reuniones de la IETF (en ingles: *Internet Engineering Task Force*). MBONE fue usada por varias universidades y institutos de investigación. Recientemente, la red Abilene

3.1.1 *Content Delivery Networks* (CDN): logrando escalabilidad

Sin la presencia de *IP Multicast* en Internet, usualmente las redes de distribución de video se encuentran montadas sobre una granja de servidores, comunicándose con los usuario mediante conexiones *IP Unicast*. La granja de servidores brinda alta disponibilidad y permite realizar balanceo de carga entre sus servidores. Supóngase que esta tiene un ancho de banda de B Kbps contratado el cual es compartido entre todos los servidores de la granja (normalmente de manera uniforme). Si el *stream* consume bw Kbps, la cantidad total de conexiones simultáneas (usuarios simultáneos) tiene que ser menor de B/bw. Es posible aumentar este número, aumentando el ancho de banda disponible B Kbps (aumentando los costos de la red) o reduciendo la calidad de *stream* (reduciendo bw).

Actualmente, los redes de distribución de video basadas en Internet tienen una estructura tradicional de *"Content Delivery Network"* (CDN). Una CDN [53, 105, 130] es un sistema que posee servidores ubicados en puntos estratégicos dentro de internet (geográficamente distribuidos), que cooperan transparentemente entre sí para distribuir contenidos a usuarios finales. Básicamente todos los nodos dentro de la CDN pueden servir los mismo contenidos. La preferencia de un usuario a un nodo en particular, es definida usando protocolos (normalmente propietarios) entre los servidores para identificar cual es el más apto para desarrollar la tarea. Cada CDN tiene sus propias reglas para determinar cual es el mejor servidor, usualmente basadas en criterios como la proximidad al usuario, congestiones en la red, etc.

Las redes de distribución de video más exitosas hoy en día en Internet, usan una arquitectura de CDN, tal es el caso de YouTube [145], msnTV [92], Jumptv [70], etc. Tal como es de esperar detrás de las CDN comerciales existen estudios académicos que sustentan y demuestran su buen desempeño [35, 73, 136].

Desde el punto de vista de la escalabilidad y de los costos, las CDN resuelven el problema de la escalabilidad, pero no el problema de los costos (dado que el costo en ancho de banda es proporcional a la cantidad de usuarios simultáneos). Además de esto, las CDNs generalmente evitan parcialmente los cuellos de botella en la red (los enlaces internacionales), típicamente fallan con los contenidos altamente populares a una localidad. Un ejemplo de esta situación podría ser un partido de fútbol entre las selecciones de Uruguay y Brasil, el cual tiene muchos usuarios dentro de Uruguay dispuestos a ver el partido, a los cuales no los ayudaría que existieran muchos servidores fuera de Uruguay dado que todos estos compartirían el mismo enlace congestionado, el del ISP que brinda servicio a Uruguay.

3.1.2 Redes *Peer-to-Peer*: reduciendo los costos

Desde el punto de vista de la escalabilidad, parecería ser necesario que los servidores se encuentren lo más cercano posible al usuario, por otro lado desde el punto de vista de los costos, sería ideal distribuir la carga referente a la distribución del video entre todos los interesados (los usuarios).

Un método de *streaming* que se ha vuelto muy popular, consiste en usar la capacidad ociosa de los usuarios para ayudar a los servidores de *streaming* en la distribución del video, mediante

Network [2] ha sido creada en Estados Unidos por la comunidad Internet2 y esta soporta *IP Multicast* nativamente. Otro ejemplo de los esfuerzos de implantar el uso de *Multicast* es la iniciativa *BBC Multicast* [13].

la actualmente madura tecnología *Peer-to-Peer* (P2P). Estos sistemas crean redes virtuales a nivel de aplicación sobre la infraestructura de Internet. Los nodos en la red son llamados *pares* o *peers*, estos ofrecen sus recursos (ancho de banda, capacidad de procesamiento y capacidad de almacenamiento) a otros nodos en la red, básicamente porque entre estos comparten intereses en común. Como consecuencia de esto, al aumentar la cantidad de usuarios en la red, también aumenta la cantidad de recursos disponibles. Además de esto, los pares que sirven a un cierto *peer* pueden ser elegidos según un criterio de proximidad, evitando de esta manera los cuellos de botella en la red (incluso mejor que con los servidores dentro de una CDN). Como desventaja se tiene que los *peers* se conectan y desconectan de la red con una alta frecuencia, en una forma autónoma y asincrónica. Por lo tanto los recursos disponibles en la red son completamente dinámicos, lo que impone restricciones a nivel de control, dado que la red debe ser robusta para soportar estas fluctuaciones. El gran desafío a la hora de diseñar una red P2P es el lograr ofrecer la calidad requerida por los usuarios bajo un ambiente de gran dinamismo.

Actualmente existen algunas redes P2PTV comerciales muy exitosas, como por ejemplo: PPlive [103], TVUnetwork [127], PPstream [104], SopCast [121], TVAnts [126], etc. Además de las redes comerciales existen algunos estudios académicos como: SpreadIt [37], CoopNet [99] y SplitStream [29].

Los desafíos en el diseño de una red P2P varían respecto al tipo de servicio de video a desplegar. El caso de video bajo demando (VoD) tiene características muy similares a los tradicionales sistemas P2P para compartir archivos. Sin embargo, la transmisión de video en vivo tiene grandes restricciones respecto a los sistemas P2P tradicionales, las cuales deben ser solucionadas en un ambiente de alto dinamismo de recursos.

En la próxima sección, se presentan diferentes clasificaciones de las redes P2P.

3.2 Taxonomía de las Redes *Peer-to-Peer*

Las redes *Peer-to-Peer (P2P)* son la clase de sistemas y aplicaciones que usan recursos distribuidos para llevar a cabo alguna función crítica de forma descentralizada. Los recursos pueden ser: procesamiento, almacenamiento (contenidos) y/o transmisión (ancho de banda) y las funciones críticas pueden ser: computación distribuida, información compartida, colaboración entre *peers*, etc.

Existen múltiples definiciones acerca de las redes P2P (en [83] se presenta una discusión sobre este tema). Una de ellas (dada por Clay Shirky [118]) expresa que: una red **Peer-to-Peer** es una red de contenido que toma ventajas de los recursos ociosos disponibles en los nodos de Internet (es decir, en los usuarios finales).

A continuación se presentan algunas de las características de las redes P2P.

- Las redes P2P están compuestas por nodos con un gran nivel de autonomía. En particular los *peers* tienen la posibilidad de entrar y salir de la red en el momento que ellos lo dispongan y de una manera muy flexible.
- Teniendo en cuenta la libertad de movimiento que poseen los nodos, es necesario que

exista un mecanismo de incentivos por participación. En general los *peers* al compartir sus recursos obtienen beneficios para poder utilizar recursos de otros *peers*.

- Otra consecuencia de la autonomía es el *anonimato*. Muy a menudo, los *peers* dentro de una red P2P no desean ser conocidos. Por ejemplo por razones legales cuando el contenido distribuido se encuentra sujeto a derechos de copia o en otros casos simplemente porque este conocimiento no es necesario.

Con el fin de entender los diferentes tipos de aplicaciones, diseños, arquitecturas y algoritmos que coexisten en el mundo P2P, se presenta a continuación una taxonomía de las redes P2P basada en 3 dimensiones (o características): tipo de aplicación, tipo de auto-organización y tipo de descentralización. Dicha taxonomía se basa plenamente en la presentada en [110].

3.2.1 Caracterización por Tipo de Aplicación

La primer clasificación de las redes P2P que se utiliza en la literatura, es en base al tipo de aplicación de la red[82, 83].

- **Compartición de contenido**: redes de archivos compartidos, distribución multimedia, almacenamiento distribuido, *caching*, manejo de información (descubrimiento, agregación, filtrado, organización, etc.).
- **Colaboración**: comunicación (*chat*, mensajería instantánea), redes de juegos (*gaming*), redes sociales, etc.
- **Computación distribuida**: computación distribuida en Internet/Intranet. Algunos ejemplos de este tipo de aplicación son: la búsqueda de vida extraterrestre, el proyecto genoma, análisis de mercado, análisis de riesgos financieros, etc.

Un muy buen articulo sobre esta clasificación es [83].

3.2.2 Caracterización por Descentralización

Conceptualmente, dentro de una red P2P existen 3 posibles roles que un *peer* puede cumplir. Los nodos que poseen el contenido, llamados *"source"*, los nodos que demandan este contenido, llamados *"requester"* y finalmente los nodos que intercambian mensajes de control con el fin de lograr una vision global de la red (por ejemplo: poder saber quien tiene un cierto contenido), llamados *"router"*. Notar que un mismo nodo según el diseño de la red, podría ser *source*, *requester* y *router* al mismo tiempo.

La descentralización de la red, se encuentra determinada por la capacidad de esta para trabajar con varios nodos del mismo tipo (*source*, *requester* y *router*). Considerando que en una red P2P normalmente existen muchos nodos de tipo *source* (debido al deseo de los *peers* en compartir su contenido), el concepto de descentralización se limita a la capacidad de la red a trabajar con múltiples *requesters* y en particular *routers*.

A continuación se presentan tres diferentes categorias en las que las redes P2P pueden ser clasificadas desde el punto de vista de su descentralización.

- **Pura:** Esta es la máxima expresión de un sistema P2P, en donde todos los nodos tienen los mismos roles (*source*, *requester* y *router*) y las mismas responsabilidades. En este caso la red P2P es una red completamente distribuida, lo que implica que la misma puede funcionar con una cantidad arbitraria de nodos de cada clase. Por ejemplo un *requester* le puede pedir un contenido a un nodo cualquiera y este nodo debe saber como llegar a alguna de las fuentes de dicho contenido. Ejemplos de este tipo de redes son: Gnutella [50] y Freenet [33, 124].

- **Híbrido:** El modelo centralizado, recibe el nombre de híbrido, debido a que no es un sistema P2P puro. En este caso existe un nodo central (o una granja de servidores) que rutea todos los mensajes de control de la red. Los *requesters* preguntan por un cierto contenido al servidor central y este le informa los *sources* del contenido. En un red híbrida pueden haber uno o muchos *requesters*. Por ejemplo en Napster [26, 94] o en BitTorrent [19] existe un único nodo de tipo *router*, y muchos *sources* y *requesters*, mientras que en Seti@home [117, 137] existe un único *router*, muchos *sources* (quienes ofrecen sus recursos de procesamiento) y un único *requester* (quien recibe los resultados procesados en los nodos *sources*).

- **Jerárquico:** El modelo jerárquico se encuentra entre el modelo puro y el híbrido, en este existen varios nodos de tipo *router*, que entre sí forman un red de *backbone* basada en mensajes de control con el fin de presentar una visión global de la red a todos sus participantes. Este modelo combina la eficiencia en las búsquedas de contenido de las redes híbridas, junto con el balanceo de carga y la no existencia de un único punto de falla, presente en las redes puras [143, 144]. El ejemplo más conocido de este tipo de red es KaZaA [72].

3.2.3 Caracterización por Auto-organización

A grandes rasgos, una red P2P puede implementar funcionalidades referidas al descubrimiento y/o a la distribución. Las funcionalidades de descubrimiento se refieren a búsquedas de información en la red. Por ejemplo en una red de compartición de contenido, los usuarios podrían realizar búsquedas de contenidos, o en una red de colaboración (supongamos una red de mensajería instantánea), los usuarios podrían buscar otros contactos en la red, o en una red de computación distribuida, los pares podrían consultar la siguiente información a procesar. Las funcionalidades de distribución, se refieren a la propia distribución de la información en la red. Por ejemplo en una red de compartición de contenido, los usuarios podrían realizar descargas mediante esta red, o en el caso de una red de colaboración (nuevamente supongamos una red de mensajería instantánea), los usuarios podrían enviarse mensajes entre sí, o en el caso de una red de computación distribuida, los pares podrían distribuir sus resultados.

No todas las redes implementan ambas funcionalidades, por ejemplo BitTorrent [19] (la cual se presenta en la siguiente sección) no implementa la funcionalidad de descubrimiento de contenidos, pero en el caso en que estas lo realizan, existen 2 diferentes maneras de implementarlo: a) a través de dos redes de *overlay*[3] diferentes, una dedicada al descubrimiento del

[3]Una red de *overlay*, es una red construida sobre otra red ya existente. Estas suelen ser redes virtuales creadas

contenido y otra dedicada a la distribución del contenido; b) a través del uso de una única red de *overlay*, en donde se lleve a cabo tanto la lógica del descubrimiento del contenido como la de la distribución. Por motivos de uniformidad en el documento vamos a suponer que en el caso b), conceptualmente también existe una red de *overlay* de descubrimiento del contenido.

Con el fin de implementar el *overlay* de descubrimiento, los peers intercambian ciertos mensajes de control y ruteo. La caracterización por auto-organización se refiere a la forma en que las redes construyen su *overlay* de descubrimiento[28, 79] (notar que esta caracterización no aplica a las redes P2P que únicamente implementan la funcionalidad de distribución de contenido).

La auto-organización del *overlay* de descubrimiento puede ser: estructurado o no estructurado.

- **No estructurado.** La topología de los nodos en un *overlay* no estructurado es organizada en forma arbitraria (no determinista) usando algún algoritmo de inundación (*broadcast* o *random walks*) con el fin de descubrir el contenido. Los nodos participantes se mantienen en contacto usando mensajes de tipo *keep-alive*. Usualmente este tipo de redes no escalan bien debido a la gran cantidad de mensajes de control existentes en la red, por otro lado estas redes presentan un alto nivel de anonimato. El ejemplo más conocido de estas redes es Gnutella [50].

- **Estructurada.** Un red estructurada, usa un protocolo globalmente consistente para rutear una cierta búsqueda sobre un nodo o contenido. En general, estas redes implementan alguna de las variantes de la tecnología *Distributed Hash Table* (DHT), en donde cada nodo mantiene (y responde acerca de) un conjunto específico de contenidos (o un conjunto de índices a contenidos). De esta manera se logran búsquedas de contenido deterministas y eficientes. En general los protocolos de estas redes aseguran descubrir un cierto contenido, consultando como máximo una cantidad logarítmica del tamaño total de la red de nodos. Algunos ejemplos conocidos son: Kademlia [80], CAN [106], Chord [122], Pastry [112, 113] y Tapestry [148]

3.3 BitTorrent: una Red P2P de Referencia

A continuación se presenta el protocolo BitTorrent [19], en el cual, tal y como veremos más adelante en este documento, se basa el protocolo de transporte de GoalBit, así como el de muchas de las redes P2P orientadas a la distribución de video en vivo.

BitTorrent es un protocolo abierto, de tipo *Peer-to-Peer*, el cual siguiendo la taxonomía anteriormente presentada, se encuentra dentro de las redes de compartición de contenido de tipo híbrida. Este ha tenido un gran éxito en Internet, al punto tal, que hoy en día, las mayores y más populares redes P2P para compartir archivos, se encuentran montadas sobre BitTorrent. A su vez y basado en el hecho de tener un protocolo abierto, existen una gran cantidad de aplicaciones que implementan BitTorrent (llamados clientes BitTorrent), algunas de las más conocidas son: μTorrent [20] (este cliente es implementado por los propios desarrolladores de BitTorrent,

a nivel de aplicación. Por ejemplo las redes P2P suelen ser un *overlay* de Internet (una red virtual creada sobre una red ya existente).

es de código cerrado y únicamente disponible en Windows y MacOS), Vuze (anteriormente llamado Azureus) [134], BitComet [18], Miro [86], etc. También existen un conjunto de librerías que proveen funcionalidades BitTorrent, como por ejemplo: BTSharp [25] (escrita en C#, basada en el *framework* .NET), libTorrent Rakshasa [75] (escrita en C++), libtorrent Rasterbar [76] (también escrita en C++), etc. Todo esto, ha llevado a que BitTorrent se consolide en el mundo de las redes P2P, convirtiéndose en un referente en lo relativo al diseño y desarrollo de aplicaciones P2P.

Como se puede observar en la Figura 3.1, existen 3 tipos diferentes de componentes en la arquitectura de BitTorrent: los *seeders*, los *peers* normales (también llamados *leechers*) y el *tracker*.

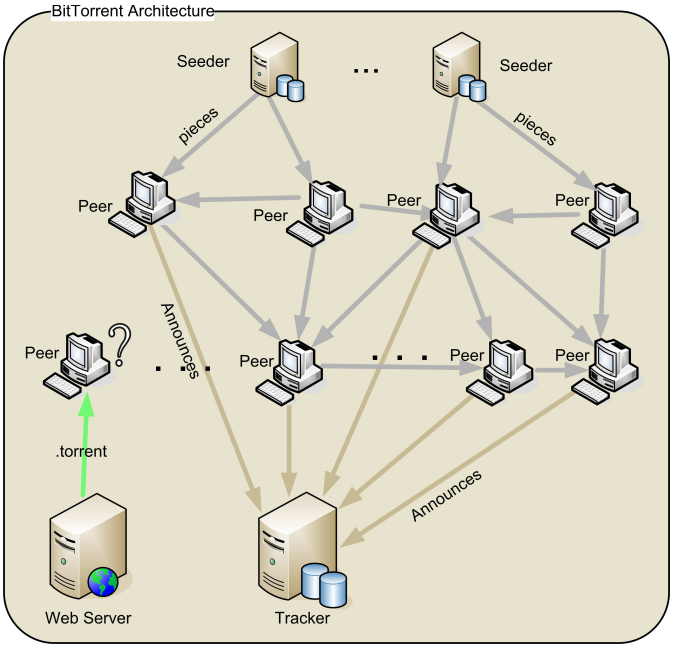


Figura 3.1: Arquitectura BitTorrent. Los 3 componentes de la red y sus relaciones son presentadas en esta imagen (los *seeders*, los *leechers* y el *tracker*).

El o los *seeders* son *peers* quienes poseen de forma completa el archivo a distribuir (o bien porque ellos son el origen de la descarga, o bien porque ya lo han descargado en su totalidad). Este tipo de *peer*, se caracteriza por no realizar descargas de otros *peers*, sino que estos únicamente sirven el contenido a otros pares de la red. Los *peers* normales o también llamados *leechers*, son *peers* que aún no han completado la descarga. Estos *peers* realizan descargas de otros *peers* y sirven el contenido ya descargado. Por su parte, el *tracker* mantiene la lista de todos los pares que se encuentran descargando (o compartiendo) un archivo en un instante de tiempo dado.

Tal como se comento en la SubSección 3.2.3, BitTorrent no implementa funcionalidades de descubrimiento del contenido. En BitTorrent cada archivo o grupos de archivos a distribuir se

identifica mediante un archivo de meta-datos llamado archivo Torrent (el cual suele tener extensión .torrent). Los archivos torrent se encuentran publicados en la *Web*, en donde los usuarios pueden realizar búsquedas y de desde allí descargarlos. Estos archivos son el punto de entrada de una ejecución BitTorrent.

En el protocolo de BitTorrent, cuya completa especificación se pueden encontrar en [1], el o los archivos a distribuir son divididos en piezas de tamaño fijo, las cuales representan la unidad de transferencia entre pares. Los archivos torrent básicamente contienen la siguiente información: él o los archivos a ser compartidos, el tamaño de las piezas, la enumeración y MD5 de cada una de las piezas incluidas en la transferencia, la URL del *tracker*, etc.

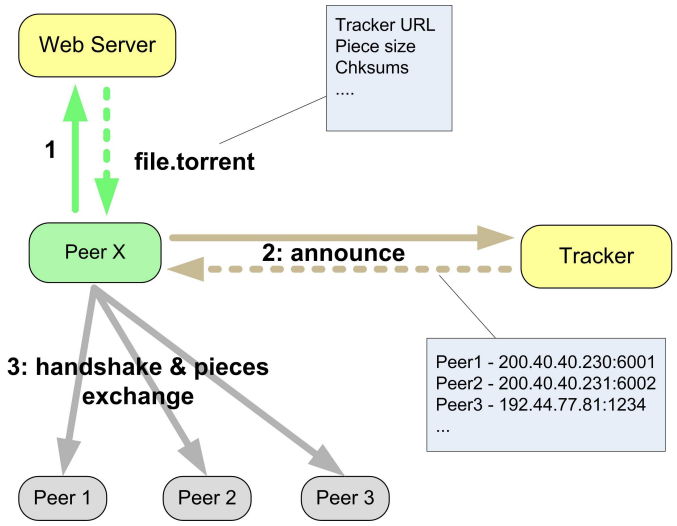


Figura 3.2: Flujo de ejecución en BitTorrent.

Como se presenta en la Figura 3.2, en primera instancia, cuando un cliente comienza a realizar una descarga (o reanuda una descarga previa), este se anuncia en el *tracker* mediante HTTP o HTTPS. El *tracker* le retorna un subconjunto de otros *peers* que se encuentran descargando el mismo archivo (llamado *swarm*). A su vez, la comunicación con el *tracker* es realizada de forma periódica con el fin de presentar actividad en el mismo. Si pasado más de cierto tiempo, un *peer* aún se ha anunciado en el *tracker*, este último lo puede eliminar de su lista de *peers* activos.

Una vez que un *peer* consigue su *swarm*, este intenta contactar a todos los *peers* incluidos en él. Toda la comunicación entre *peers* es realizada en base al intercambio de once mensajes binarios, transmitidos sobre el protocolo TCP. Al iniciar una conexión entre dos *peers*, estos realizan un *handshake*, en donde se comunican todas las piezas que tienen y las que les falta. A su vez, cuando un *peer* recibe una nueva pieza, este le informa dicho evento a todos los *peers* con los cuales se encuentra interactuando (tiene una conexión establecida). De esta manera, cada *peer* conoce con exactitud las piezas que tienen el resto de los *peers* en su *swarm* y solicita (de manera explícita) las piezas que requiere para completar la descarga. La política de

selección de piezas (que piezas debe solicitar un *peer* a otro) aplicada en BitTorrent es llamada *rarest-first* (en español: primero las más raras). Esta establece que los *peers* deben solicitar primero las piezas menos distribuidas entre todos los pares de su *swarm*, garantizando de esta manera, una alta diversidad en las piezas, y previniendo el problema de la falta de ciertas piezas en la red. Notar que la política de selección de piezas no aplica para el caso de los *seeders*, dado que estos no solicitan piezas.

La política de selección de *peers* (a que *peers* un cliente le permite realizar descargas de sí mismo) aplicada en BitTorrent es llamada *tit-for-tat* (en español: ojo por ojo). La misma define que un cliente primero le debe enviar piezas a los *peers* de los cuales este se encuentra descargando piezas a una mayor tasa. Dentro de esta política, con el fin de lograr una rápida inserción de los nuevos *peers* en la red, y a modo de realizar una exploración de la misma (en busca de *peers* con mejores condiciones) se define una excepción al *tit-for-tat*, llamada *optimistic-unchoking* (en español: habilitación optimista). El *optimistic-unchoking*, establece que cada ciertos intervalos de tiempo, un cliente debe permitir el envío de piezas a otro *peer*, sin tener referencias de él (el cliente nunca ha descargado una pieza del *peer* a habilitar). En el caso de los *seeders* no es posible aplicar *tit-for-tat*, dado que estos no realizan descargas. Usualmente la política de selección de *peers* aplicada en los *seeders* es realizada de forma uniforme, teniendo en cuenta factores como el tiempo desde la última habilitación realizada a un *peer*, la cantidad de *bytes* enviados, etc.

Al intentar aplicar BitTorrent a la distribución de video en vivo, surgen varios problemas, primero y como se explicó anteriormente, que BitTorrent es un protocolo definido para transmitir uno o más archivos y no un flujo continuo de datos (como es el video en vivo). Dejando este problema de lado, la política de selección de piezas (*rarest-first*) tampoco se adecua al contexto de la transmisión de video de vivo dado que esta genera grandes tiempos de espera en el comienzo de la reproducción del video (posee una gran latencia inicial).

GoalBit toma como base BitTorrent y define una serie de cambios en el protocolo y en sus políticas, con el fin de permitir la distribución y reproducción de video en vivo.

3.4 Redes P2P para la Distribución de Video en Vivo

Las redes P2P más populares de hoy en día son las diseñadas para compartir archivos. Algunas de estas son: BitTorrent [19] (presentada anteriormente), KaZaA [72] o eMule [42]. Una hipótesis de base aplicada en estos sistemas, es el hecho de que los usuarios permanecen conectados a la red (poniendo sus recursos a disposición del sistema) por cierto tiempo. Por ejemplo en [115] se reporta una duración media, cercana a una hora para los sistemas Napster [94] y Gnutella [50].

Las redes P2P de distribución de video en vivo deben satisfacer restricciones más complejas, debido a que la reproducción de video tiene restricciones de tiempo real. Como referencia para el diseño de nuestro sistema P2P, usamos un servicio de distribución de video (AdinetTV, presentado en la Sección 2.3), en donde los usuarios en promedio permanecen 15 minutos conectados.

Algunas recapitulaciones sobre la estructura y el diseño de las redes P2P para la distribu-

ción de video en vivo, se presentan en [62, 116, 123]. Basandonos en estas y extendiendo lo escrito en el capitulo 2.3.3.2 de [110], en las siguientes subsecciones se presentan un conjunto de redes P2P para la distribución de video organizadas según la estructura de su *overlay* de distribución.

3.4.1 *Overlay* de Distribución basados en Árboles

Las redes P2P basadas en un *overlay* de distribución en forma de árbol, normalmente construyen una cadena de *proxies* de distribución con los usuarios conectados. Existe un nodo inicial que publica la señal en vivo, al cual llamaremos *broadcaster*. Este le envía el video a un cierto *peer*, quien toma la señal, la consume y se la reenvía a otro *peer* dentro de la red. Así sucesivamente, los *peers* van construyendo el *overlay* de distribución. Las cadenas muy largas (en donde los *peers* se reenvían el *stream* de uno a otro) presentan grandes problemas, si bien teóricamente cubren la distribución del video. Por un lado, estas generan un atraso considerable en los nodos más lejanos al *broadcaster*. Por otro lado, estas son cuasi impracticables debido a la dinámica de los *peers* dentro de la red (esta cadena se cortaría muy a menudo). En este tipo de arquitectura cuando un nodo se desconecta de la red, toda la cadena que depende de él queda desconectada del *stream*, hasta que su sucesor vuelva a conectarse.

La idea de la distribución basada en árboles es la de reducir lo más posible el largo de la cadena y por lo tanto la probabilidad de que un predecesor se desconecte. Estas redes implementan un grafo de distribución, en donde la raíz es el *broadcaster* y los nodos reenvían el *stream* en lo posible a más de un nodo. La profundidad del árbol (cadena más larga partiendo desde la raíz) puede ser logarítmica respecto a la cantidad total de nodos en la red. Notar que la estructura del árbol depende en gran medida del ancho de banda disponible en cada uno de los nodos, así como del *bitrate* del *stream*.

3.4.1.1 GOL!P2P

Dentro de este tipo de aplicaciones se encuentra nuestro proyecto anterior, llamado GOL!P2P [110]. El prototipo GOL!P2P es una red P2P híbrida para la distribución de contenidos en vivo. Un control centralizado mantiene (y define) una estructura básica de la red en forma de árbol, mientras que se aplica la técnica de ***multi–source streaming*** [111] para distribuir el video. El *multi–source streaming* consiste en dividir el flujo original de video en varios sub-flujos, por lo que un usuario para reproducir el video, lo debe recibir de múltiples fuentes. Según el diseño de división y dependiendo del grado de redundancia aplicado en los sub-flujos, podría pasar que un usuario reconstruya el flujo de video original sin la necesidad de recibir todos los sub-flujos. A modo de resumen, en GOL!P2P los usuarios se conectan periódicamente con un servidor central el cual define la topología de la red para cada sub-flujo, existiendo un árbol de distribución diferente por cada sub-flujo. Esta topología es comunicada a los usuarios, quienes son los responsables de llevarla a cabo, conectando con los *peers* correspondientes para cada tipo de sub-flujo.

3.4.1.2 PeerCast

PeerCast [101] es una red P2P diseñada para transmitir radio y video sobre Internet. Este presenta un protocolo y código abierto, siendo gratiuto su uso tanto para los consumidores como para los productores. PeerCast mantiene una arquitectura basada en árboles, en donde por cada contenido en vivo (VoD no es soportado por PeerCast) y partiendo desde el servidor de *broadcasting*, se construye un árbol de distribución con los *peers* conectados a dicho canal.

Un *peer* al ingresar a la red se contacta con el servidor de *broadcasting*, quien le responde OK o NOK, en base a su capacidad ociosa. Si el servidor de *broadcasting* tiene suficiente capacidad ociosa, este va a incluir al *peer* dentro de la lista de sus hijos y le va a comenzar a enviar el *streaming*. En el caso contrario, el servidor de *broadcasting* va a elegir como máximo a 8 *peers* de su listado de hijos y se los va a enviar al *peer* en cuestión. En este caso, el *peer* registrará el listado de pares enviados por el servidor de *broadcasting* y los contactará uno por uno, hasta encontrar un nodo que tenga capacidad disponible. El contenido entre *peers* es transmitido sobre UDP (un padre envía el *streaming* a sus hijos). A su vez, los hijos notifican a sus padres periódicamente (utilizando el mecanismo *keep-alive*), para que los padres mantengan actualizada su lista de hijos.

3.4.1.3 SplitStream

SplitStream [29] provee una infraestructura colaborativa que puede ser usada para distribuir tanto grandes archivos (como por ejemplo actualizaciones de *software*) como *streaming* de video, en un forma totalmente descentralizada. SplitStream fue construido sobre Scribe [27], un sistema escalable de tipo publicador-suscriptor, que emplea Pastry [112] como el *overlay* de descubrimiento.

3.4.2 *Overlay* de Distribución basados en Mallas

La mayoría de las redes P2P para compartir archivos hoy en día, implementan un *overlay* de distribución de tipo “malla” (o como son llamado en ingles *mesh-based overlay*). En este *overlay* no existe una topología definida en la red. Dentro de la red se encuentra la información acerca de qué *peers* se encuentran descargando qué archivo (en un momento dado). Por ejemplo, en BitTorrent Puro [19] esta información se encuentra centralizada en un servidor *Web* (llamado *tracker*); en BitTorrent Trackerless esta se encuentra distribuida sobre una red estructurada denominada Kademlia [80]; en eMule [42] esta se distribuye sobre un conjunto de servidores (usualmente usando el *software* Lugdunum [78]); en KaZaA [72] esta la mantienen algunos *peers* de alta disponibilidad llamados *super-peers*. Usando esta información, cada *peer* al conectarse a la red obtiene un subconjunto de otros *peers* que están descargando el mismo archivo (dicho subconjunto es llamado *swarm*). Los archivos son distribuidos en piezas (llamadas *chunks*). Cada *peer* conoce las piezas que tiene el resto de los *peers* en su *swarm*, y de esta manera los *peers* se solicitan entre sí las piezas que requieran.

Las redes de tipo *mesh*, hace ya algún tiempo que se comenzaron a usar para la distribución de video en vivo. Pero las tradicionales aplicaciones P2P para compartir archivos no pueden ser usadas directamente para llevar a cabo este fin, dado que por ejemplo (y entre otras cosas), las políticas de pedido de piezas no satisfacen ciertas restricciones que naturalmente tiene la

reproducción de video (una de ellas, que para comenzar a reproducir el video los primeros *chunks* deben ser descargados primero, lo cual no es necesario en las aplicaciones para descargar archivos). Entonces, dos grandes características que debe satisfacer la política de descarga de piezas de una aplicación P2P para la distribución de video en vivo son: lograr un rápido comienzo de la reproducción del video y lograr una buena continuidad en la misma.

Algunos estudios académicos acerca del diseño de las redes P2P de tipo *mesh* para la distribución de video son: [36, 123, 132, 139, 140].

A continuación se presentan algunas de las más populares redes P2P comerciales para *streaming* de contenidos en vivo de la actualidad. Todas ellas de tipo *mesh*.

3.4.2.1 PPLive

PPLive [103] es el *software* más popular de P2PTV (especialmente para usuarios Asiáticos), con más de $200,000$ usuarios concurrentes en un mismo canal [116] (alcanzando un *bitrate* agregado de 100 Gbps). Nacido como un proyecto de estudiantes en la Universidad de Ciencia y Tecnología Huazhong de China, PPLive usa un protocolo propietario y su código no se encuentra disponible. A su vez, PPLive únicamente brinda servicios a usuarios consumidores, es decir, que no es posible realizar un *broadcasting* propio con esta aplicación.

Respecto a su protocolo, aplicando ingeniería reversa, los trabajos [6, 52] demuestran que PPLive implementa una red de tipo *mesh*. Este es similar al protocolo de BitTorrent, presentando una comunicación inicial con un *tracker* y posteriores interacciones entre *peers*, llevando a cabo el intercambiando de piezas. A diferencia de BitTorrent, los *peers* en PPLive implementan una política "chismosa" (en ingles: *gossip-based policy*) con el fin de conseguir nuevos *peers*. Esto significa que los *peers* se comunican entre sí (se "chusmean") otros *peers* con los que están interactuando, con el fin de ampliar sus listas de contactos. Por otra parte, en esta red el proceso de selección de *peers* (el armado del *swarm* de un usuario) es realizado de forma aleatoria, lo cual implica costos más altos para los ISPs y posibles congestiones en la red.

Respecto a la aplicación, en PPLive se usan dos diferentes tipos de codificaciones para el video: Window Media Video (VC-1) y Real Video (RMVB) a una tasa promedio de 250 Kbps a 400 Kbps. A su vez, esta posee un reproductor multimedia nativo para presentar el video a los usuarios. En la aplicación existen 2 *buffers* diferentes, uno usado dentro del motor de distribución (en donde se guardan los *chunks* descargados por el cliente P2P) y otro usado por el reproductor de video (de donde se presenta el contenido a los usuarios).

Respecto a la ejecución de PPLive, luego que un usuario selecciona un canal, este descarga una lista de *peers* y los contacta uno a uno. Este proceso inicial demora en promedio $10 \sim 15$ segundos. Luego de este proceso, es necesario unos $10 \sim 15$ segundos más para llenar los *buffers* de la aplicación y así comenzar con la reproducción del video. Por lo tanto en promedio se requieren de aproximadamente $20 \sim 30$ segundos para comenzar la reproducción del video (este tiempo depende en gran medida de la popularidad del canal seleccionado, pudiendo ser de hasta 2 minutos para los más populares). Luego de múltiples experimentos en [52] se estima que el tamaño del *buffer* del reproductor es de al menos $5{,}37$ MBytes (más de 2 minutos a 350 Kbps) y el del motor de distribución puede variar entre $7{,}8$ MBytes y $17{,}1$ MBytes (entre 3 y 7 minutos a 350 Kbps).

3.4.2.2 Octoshape

Octoshape [95], de origen danés, es una plataforma de *streaming* P2P mallada, usada, entre otros, por la CNN para distribuir sus contenidos bajo picos de más de un millón de usuarios concurrentes. Octoshape brinda servicios de manera gratuita a consumidores, pero no así a publicadores (no es posible realizar un *broadcasting* propio con Octoshape de manera gratuita). Tanto su protocolo como su código es cerrado.

Según [7] en esta red, los *peers* que se encuentran viendo un canal son insertados como metadata del propio contenido, por lo que no existe la figura de un *tracker* en la red. A su vez, el contenido es dividido en múltiples sub-flujos (más puntualmente, existe un sub-flujo por cada *peer* en la red). Esta división es realizada en los *peers* con el fin de no cargar al *broadcaster* (nodo fuente del contenido). Ningún sub-flujo es idéntico entre sí, y con cualquier K número de sub-flujos un *peer* puede reconstruir el flujo original. El número K, depende de la tasa de codificación del *streaming* original, y de la tasa de cada uno de los sub-flujos. Por ejemplo, si el video original fue codificado a 400 Kbps, y los subflujos son codificados a 100 Kbps, un *peer* va a requerir al menos 4 sub-flujos para poder reconstruir el flujo original. Cada *peer* mantiene una lista de referencias a otros *peers*. A su vez, cada *peer* mantiene una lista de *peers* "listos a ser consumidos" (en ingles *a standby list*), quienes van a ser requeridos en caso de que falle alguno de los *peers* de los cuales se está consumiendo el cotenido actualmente. Los *peers* al recibir un pedido de conexión (de otro *peer*), pueden aceptarlo o no, en base a sus capacidades de subida. Octoshape reporta que usando su protocolo de *streaming*, se puede obtener hasta un $97\,\%$ de ahorro en el ancho de banda consumido por el *broadcaster* (en el caso de que los usuarios tengan en promedio una capacidad de ancho de banda de subida superior a la tasa de codificación del video) [7].

3.4.2.3 SopCast

SopCast [121], creado en la Universidad Fudan de China, es otra red de P2PTV muy popular, la cual ha soportado más de $100,000$ usuarios concurrentes (según sus desarrolladores). Esta brinda soporte tanto a consumidores (usuarios que quieren ver ciertos contenidos) como a productores (usuarios que quieren distribuir sus propios contenidos). Tanto el protocolo SopCast, como su aplicación son cerradas.

Siguiendo procedimientos de ingeniería inversa [6, 116] muestran que SopCast implementa un *overlay* de tipo *mesh*, con un protocolo (al igual que en el caso de PPLive) similar al de BitTorrent. Una primera diferencia de SopCast respecto a PPLive o BitTorrent, es que este usa UDP como protocolo de transporte.

La aplicación SopCast mantiene un *buffer* local (ídem al *buffer* del motor de distribución en PPLive) y cuando este se encuentra lleno ejecuta un reproductor de video genérico (por ejemplo Windows Media Player) quien consume el *streaming* desde un servidor HTTP implementado en la propia aplicación SopCast (en el puerto 8902). En comparación con PPLive, el tamaño del *buffer* de SopCast es un poco menor y varía entre $1{,}8$ MBytes y $2{,}3$ MBytes (1 minuto bajo un *streaming* de 360 Kbps). Sin embargo el tiempo de espera en comenzar la ejecución es sensiblemente mayor, en este caso entre 1 y 3 minutos.

3.4.2.4 CoolStreaming

CoolStreaming [34] (también conocido como DONet) es el predecesor de PPLive y SopCast. Este fue cerrado hace algún tiempo debido a problemas legales. Al igual que con PPLive y SopCast, este presenta un protocolo propietario y código fuente cerrado.

La siguientes descripciones de los protocolos implementados en CoolStreaming, se basan en publicaciones realizadas por los propios autores de este [141, 146, 147].

A nivel conceptual existen 2 diferentes protocolos implementados en CoolStreaming, uno referido a la obtención de *peers* con quienes interactuar (llamados nodos en los artículos de referencia) y otro referido al intercambio de información y contenido entre *peers* (llamados *partners*).

Respecto al primero, cada nodo posee un identificador único (podría ser su dirección IP) y mantiene un conjunto de referencias a otros nodos en la red (llamado *membership cache*). Cuando un nodo ingresa en la red, este contacta al nodo origen que se encuentra emitiendo el contenido, para el cual su identificador (dirección IP) es globalmente conocido. El nodo origen selecciona aleatoriamente a uno de los nodos incluidos dentro de sus referencias, llamado "encargado", y se lo retorna al nodo recién ingresado. A continuación el nodo recién ingresado consultará al encargado asignado, de donde obtendrá el listado de referencias a otros nodos de la red. Notar que de esta manera se uniformizan las referencias a los nodos que posee cada uno de los *peers* y se minimiza la carga en el nodo origen. Finalmente se aplica un protocolo *Gossip* (al igual que en PPLive) con el fin de mantener actualizado el listado de referencias de los nodos, agregando los nuevos nodos ingresados a la red y eliminando los nodos que abandonaron la misma.

Respecto al protocolo de intercambio de piezas, el contenido es dividido en piezas de tamaño fijo (normalmente de 1 segundo de video). Cada *peer* tiene asociada una ventana deslizante. Para el caso normal la ventana tiene un tamaño de 120 piezas, es decir de 2 minutos. Además, el *peer* tiene asociado un mapa de las piezas que existen y de las que no existen dentro de dicha ventana, utilizando una estructura de 120 *bits*, en donde el valor de cada *bit* es 1 si se tiene la pieza de la posición de dicho *bit* y 0 en el caso contrario. Al comienzo cada *peer* selecciona nodos aleatoriamente de su lista de contactos e intenta conectase con ellos, como resultado de este proceso, el *peer* va a construir un grupo de 4 *partners* con quienes va a intercambiar los mapas de sus correspondientes ventanas. En base a estos mapas, los *partners* arman una agenda de pedido de piezas, definiendo qué piezas se le piden a qué *peers*, utilizando la misma estructura de 120 *bits*, en donde el valor de cada *bit* es 1 si se solicita el envío de la pieza de la posición de dicho *bit* o 0 en el caso contrario. La estrategia aplicada por los *peers* en el armado de la agenda de pedidos, consiste en solicitar primero las piezas menos difundidas en la red a los *peers* con mayor ancho de banda disponible (observar la similitud con la política *rarest-first* aplicada en BitTorrent, Sección 3.3). Las agendas de pedido son enviadas a los *partners* correspondientes, y estos usando un protocolo estándar de transporte (en la implementación de CoolStreaming se optó por TFRC [59]) envían las piezas solicitadas. Por otro lado los *partners* con los que interactúa un nodo no son fijos, sino que estos son evaluados bajo intervalos regulares de tiempo, en donde el "peor" nodo es descartado (el que menos ha aportado), dando su lugar a un nodo tomado en forma aleatoria de la lista de contactos (nuevamente notar la similitud con las políticas *tit-for-tat* y *optimistic-unchoking* de BitTorrent).

Finalmente, en ambos protocolos (el de intercambio de *peers* y el de intercambio de piezas) se utiliza TCP como protocolo de transporte.

3.4.3 Resumen

En la Tabla 3.1 se presenta un breve resumen con algunas de las características de las redes P2P para la distribución de contenidos en vivo presentadas en esta sección. A modo comparativo también se agregó nuestra red GoalBit. Como se puede observar en dicha tabla, en la actualidad existe una clara tendencia al uso de redes malladas para el desarrollo de sistemas de P2PTV. Esto se debe a que las redes de distribución con estructura de árbol, suelen tener problemas cuando la dinámica de los *peers* es muy alta, dado que continuamente es necesario reconstruir el o los árboles de distribución, situación que es muy frecuente dentro de las grandes redes P2P, puesto que los *peers* entran y salen del sistema con una alta frecuencia. Por otro lado, las redes malladas se ajustan de mejor manera a la dinámica de los *peers*, pues la descarga del video no se encuentra sujeta a un único *peer*, como en el caso de las redes de tipo árbol. En las redes de malla todos los *peers* son nodos de tipo *source*, sin importar las restricciones de ancho de banda de subida que estos posean, no como en las arquitecturas de tipo árbol, en donde, si la capacidad de subida de un *peer* es menor que la tasa a la que se encuentra codificado el video, este nodo no puede ser tenido en cuenta en la distribución. Como desvantaja las redes malladas presentan tiempos de *buffering* mayores que las de tipo árbol, lo cual claramente aumenta el atraso global en la reproducción del video. Tal como se presenta en las conclusiones de [116], este tipo de redes son adecuadas para distribución de contenidos tolerantes a retrasos (en ingles *delay-tolerant*), pero se encuentran lejos de ser aptas para la distribución de video con restricciones críticas de tiempo real, como por ejemplo para comunicaciones bidireccionales como las video-conferencias, donde 20 segundos de atraso no es aceptable. Más allá de esto, la gran mayoría de los contenidos son tolerantes a retrasos: no importa ver un gol en un partido de fútbol 2 o 10 segundos despues, siempre que todos los usuarios lo vean aproximadamente al mismo tiempo; razón por lo cual las redes malladas han triunfado sobre las arborescentes. También si analizamos la tabla presentada anteriormente, se puede observar que no existe ninguna red mallada con un protocolo libre, razón que nos motivo a desarrollar GoalBit.

3.5 PPSP: *Peer to Peer Streaming Protocol*

En Febrero del 2010, la IETF [54] anunció la creación del grupo de trabajo PPSP (*Peer to Peer Streaming Protocol*), con el fin de definir y estandarizar un protocolo de *streaming* P2P tanto para contenidos en vivo como para video bajo demanda. PPSP y GoalBit tienen muchos objetivos comunes, es por esto que desde el comienzo se intentó apoyar su desarrollo.

3.5.1 Motivación y Alcance

Esta sub-sección se basa en [61]. Como se presentó anteriormente, existe una creciente integración del *streaming* P2P a la infraestructura global de distribución de contenidos. Los protocolos de *streaming* P2P propietarios, no solo resultan en esfuerzos de desarrollo repetitivos, sino que presentan grandes problemas de integración, al intentar posicionar al *streaming* P2P como una

Cuadro 3.1: Resumen de las redes P2P para la distribución de contenidos en vivo presentadas

Red	Tipo de red	Tipo de protocolo	Estado del proyecto	Actual modelo de negocio
GOL!P2P	Arborescente	Abierto	Abandonado. Llegó a etapa de prototipado.	-
PeerCast	Arborescente	Abierto	No activo. Última versión liberada en Diciembre del 2007.	Uso gratuito para consumidores y productores.
SplitStream	Arborescente	Cerrado	Abandonado.	-
PPLive	Mallada	Cerrado	Muy activo. Última versión liberada en Junio del 2010.	Uso gratuito para consumidores
Octoshape	Mallada	Cerrado	Muy activo.	Uso gratuito para consumidores
SopCast	Mallada	Cerrado	Muy activo. Última versión liberada en Marzo del 2010	Uso gratuito para consumidores y para productores
CoolStreaming	Mallada	Cerrado	Cerrado en Junio del 2005 por motivos legales.	-
GoalBit	Mallada	Abierto	Muy activo. Última versión liberada en Junio del 2010	Uso gratuito para consumidores y para productores

solución global. Un buen ejemplo de esto es el éxito de la *Web*, en donde HTTP (el protocolo estándar de la *Web*) hace posible desplegar un completo sistema de distribución, que consiste no solo de dispositivos de borde (servidores *Web* y clientes *Web*), sino que también está compuesta por dispositivos como los *cache Web* y nodos de CDN. Todos estos dispositivos se comunican siguiendo protocolos estándares y proveen grandes beneficios tanto para los consumidores de servicios, como para los publicadores de los mismo y en definitiva para toda la infraestructura de la red.

En el contexto de las redes P2P, los nodos de borde (*Caches*, CDNs) son considerados como medios prometedores, tanto para mejorar la *performance* del *streaming* P2P como para reducir el tráfico entre las redes de los ISPs, debido a la posibilidad de ubicar *peers* estables con buenas capacidades de ancho de banda, llamados *super-peers*.

Por otro lado, las redes móviles e inalámbricas se han convertido en importantes componentes a ser soportados correctamente en el futuro de la Internet. Junto con ello, el *streaming* a dispositivos móviles se ha convertido en un servicio clave a ser ofrecido por los ISPs. Mas allá que actualmente la mayoría de los despliegues de MobileTV siguen el modelo cliente/servidor, existe una clara intensión de integrar el modelo P2P a este tipo de sistemas. Sin embargo, la Internet móvil puede presentar severas limitaciones respecto al ancho de banda disponible para realizar el *streaming* P2P. Pueden haber varios cuellos de botella en donde el ancho de banda es limitado y el costo de la transmisión puede ser alto: a) entre las terminales móviles y los nodos de acceso; b) entre los nodos de acceso; c) entre la red móvil y la red fija.

En este caso el estándar PPSP, también puede ayudar en la solución de estos desafíos. Con PPSP, dispositivos como *gateways* móviles y puntos de acceso pueden actuar como *super-peers*, reduciendo así el consumo de ancho de banda en los diferentes cuellos de botella.

Al momento de diseñar un sistema de *streaming* P2P, se debe tener en cuenta que múltiples protocolos y especificaciones son requeridas. Con el fin de aclarar este punto y poder establecer el alcance del proyecto PPSP, en la Figura 3.3 se presentan los componentes de un sistema genérico de *streaming* P2P.

La capa de transporte es la responsable de la transmisión de datos entre los *peers*. TCP, UDP u otros protocolos pueden ser usados. Dentro de la capa de comunicación se encuentran

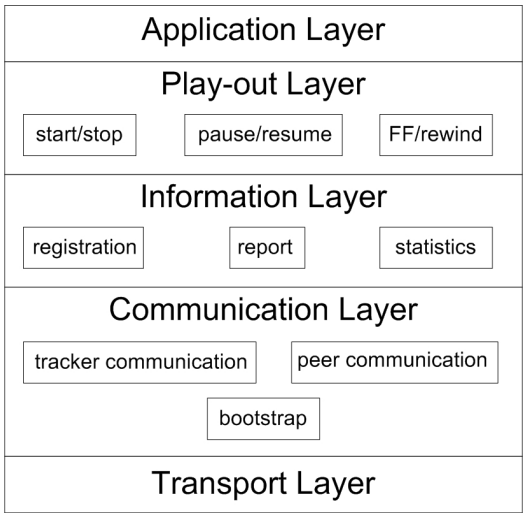


Figura 3.3: Componentes de sistema de streaming P2P.

3 componentes: la comunicación con el *tracker*, la comunicación entre *peers* y el proceso de inicialización, de donde los *peers* obtienen una referencia al contenido. En la capa de información están incluidos los siguientes componentes: registro, permitiendo a los nodos registrar la información de sus contenidos en el sistema; reporte, permitiendo a los *peers* reportar su estado del *streaming* al *tracker* (ancho de banda de subida y bajada consumido, cantidad de *peers* con los que se está interactuando, etc.); y estadístico, permitiendo al *tracker* tener una visión global del sistema. La capa de reproducción es la encargada de controllar la ejecución del *streaming*: *play*, *stop*, etc. Finalmente, la capa de aplicación es la capa superior dentro de todos los componentes del sistema P2P.

Los protocolos que el grupo de trabajo PPSP tiene como objetivo estandarizar son el de comunicación con el *tracker*, el de comunicación entre *peers* (ambos de la capa de comunicación) y el de reporte (de la capa de información). Es decir, aquellos procesos en donde existe comunicación entre *peers* y entre *peers* y el *tracker*:

1. Protocolo PPSP *tracker*
   Este protocolo va a definir:

   (a) Formato y codificación de la información a intercambiar entre el *tracker* PPSP y los *peers* PPSP, como por ejemplo: la lista de *peers*, el estado del *streaming*, la capacidad de un *peer*, etc.

   (b) Mensajes a intercambiar entre el *tracker* PPSP y los *peers* PPSP, definiendo como los PPSP *peers* reportan su estado y realizan pedidos al *tracker*, y como éste responde a los pedidos.

2. Protocolo PPSP *peer*
   Este protocolo va a definir:

(a) Formato y codificación de la información a intercambiar entre *peers* PPSP.

(b) Mensajes a intercambiar entre *peers* PPSP con el fin de llevar a cabo la ejecución del *streaming*.

3.5.2 Planificación

El grupo de trabajo PPSP, tiene definida la siguiente agenda:

- Diciembre 2010: Envío de la definición del problema al *Internet Engineering Steering Group* (IESG) en modo informacional.
- Abril 2011: Envío de la investigación sobre las actuales arquitecturas de sistemas de *streaming* P2P al IESG en modo informacional.
- Abril 2011: Envío del documento de requerimientos al IESG en modo informacional.
- Agosto 2011: Envío de la definición del protocolo PPSP *peer* al IESG como propuesta de estándar.
- Agosto 2011: Envío de la definición del protocolo PPSP *tracker* al IESG como propuesta de estándar.
- Diciembre 2011: Envío de la guía de uso al IESG en modo informacional.

Por lo tanto, se debería esperar que el estándar sea publicado durante el año 2012.

Capítulo 4

Evaluación de la Calidad de Video

Este capítulo, en base a [110], resume los diferentes tipos de evaluaciones subjetivas y objetivas, y presenta un enfoque híbrido entre estas, llamado PSQA. Finalmente éste brinda un breve resumen de las actuales soluciones para el monitoreo de redes de distribución de video.

4.1 Evaluaciones Subjetivas

No hay un mejor indicador acerca de la calidad de video que un observador humano, por el simple hecho de que el video es creado para ser visto por seres humanos. Desafortunadamente, la calidad dada por un observador depende de su propia experiencia. Por esto, es que se dice que la calidad de video es percibida. Por definición, la calidad percibida de video es un concepto subjetivo. El mecanismo usado para evaluarla es llamado pruebas subjetivas. Existen diferentes técnicas de evaluación de la calidad subjetiva, las cuales dependen, de los aspectos de calidad que se estén evaluando y el tipo de aplicación. Las técnicas de evaluación de calidad subjetiva pueden ser categorizadas en dos grandes grupos: las de tipo cualitativo y las de tipo cuantitativo.

Las evaluaciones cualitativas [22, 23] tienen como resultado descripciones acerca de la calidad percibida por los observadores. Usualmente éstas no se trasladan bien a una escala numérica. Normalmente estas evaluaciones suelen ser usadas para observar cómo los individuos reaccionan ante una variación de la calidad percibida, por ejemplo con el fin de definir el esquema de precios para un servicio multimedia. En nuestro caso, nos vamos a enfocar en las evaluaciones de tipo cuantitativas.

Las evaluaciones cuantitativas consisten en armar un conjunto de observadores (seres humanos), quienes evalúan la calidad percibida al reproducir una serie de pequeños videos (a cada video, cada observador le asigna un valor dentro de una escala numérica). De esta manera, en primera instancia vamos a obtener la calidad percibida por un usuario promedio para cada video. Luego, procesando estos datos se puede obtener una evaluación más pesimista u optimista si es necesario. Estos procesos de evaluación suelen llamarse *Mean Opinion Score* (MOS) [68].

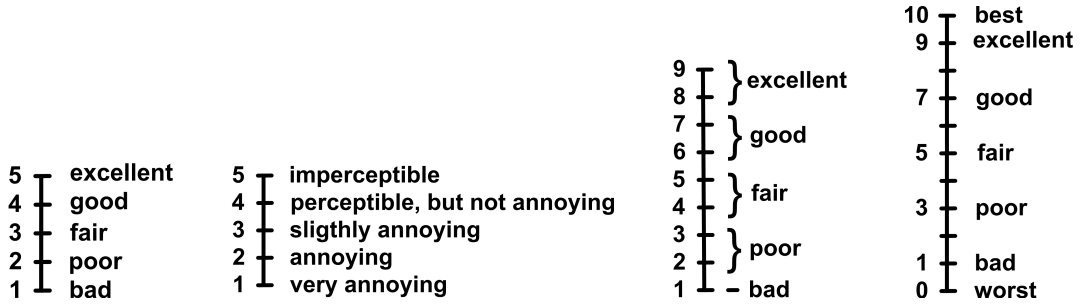
Existen métodos estándares sobre cómo llevar a cabo las evaluaciones subjetivas, como por ejemplo ITU-R BT.500-11 [68], en donde se incluyen las siguientes variantes:

- *Single Stimulus* (SS),

- *Double Stimulus Impairment Scale* (DSIS),
- *Double Stimulus Continuous Quality Scale* (DSCQS),
- *Single Stimulus Continuous Quality Evaluation* (SSCQE),
- *Simultaneous Double Stimulus for Continuous Evaluation* (SDSCE)
- y *Stimulus Comparison Adjectival Categorical Judgement* (SCACJ).

Las diferencias entre éstas suelen ser mínimas y principalmente dependen del tipo de aplicación a considerar. A continuación se presentan detalles sobre dos de estos métodos: SS y DSIS.

Single Stimulus El método de *Single Stimulus* (SS) es diseñado para realizar evaluaciones absolutas sobre la calidad de secuencias audiovisuales. La duración de las secuencias debe ser cercana a los 10 segundos. Las secuencias deben comenzar y terminar con una escena gris, de no más de 500 ms, para hacer la evaluación más natural. También, la secuencia de video no puede terminar en una escena incompleta y el audio y video deben estar perfectamente sincronizados. Luego de que cada secuencia es vista, se les solicita a los observadores que califiquen la calidad percibida en dicha secuencia. Las secuencias de video son presentadas de a una y son calificadas de forma independiente bajo una escala absoluta. Las diferentes escalas sugeridas en el estándar son: la escala de 5 puntos (Figura 4.1(a)) usualmente usadas en evaluaciones de audio, y las escalas de 9 puntos (Figura 4.1(c)) y 11 puntos (Figura 4.1(d)), normalmente usadas en las evaluaciones de video. Dependiendo de la exactitud buscada, el número de observadores requerido por el estándar varia entre 4 y 40. En nuestro trabajo, se desarrolló una evaluación subjetiva basada en este método, sobre una escala de 11 puntos. Por más detalles, ver Sección 4.3.2.



(a) 5–point scale. (b) 5–point impairment scale. (c) 9–point scale. (d) 11–point scale.

Figura 4.1: Escalas del estándar ITU para los métodos de evaluación subjetivos.

Double Stimulus Impairment Scale El método de *Double Stimulus Impairment Scale* (DSIS) es diseñado para realizar evaluaciones sobre la degradación generada sobre una secuencia de

video, debido a algún proceso de codificación o de transmisión. Las secuencias son presentadas de a pares en paralelo, en donde una de ellas es la secuencia de referencia y la otra es la secuencia transformada. Los participantes de la evaluación son informados acerca de cual es el video de referencia y se les solicita que califiquen el video transformado usando la escala presentada en la Figura 4.1(b). La diferencia entre la reproducción de ambos video debe ser menor a 2 segundos y la duración de las secuencias debe ser menor o igual a 10 segundos.

Como se puede suponer, la realización de cualquiera de estas evaluaciones consumen mucho tiempo y son costosas en términos de trabajo humano. Motivo por el cual, éstas son difíciles de repetir a menudo. Otro gran problema que tienen este tipo de evaluaciones, es el hecho de no poder ser parte de un proceso de evaluación automático de la calidad de video (por ejemplo, intentando monitorear la calidad de un *streaming* en vivo).

En la siguiente sección, se presentan las evaluaciones objetivas sobre la calidad de video.

4.2 Evaluaciones Objetivas

Considerando los inconvenientes asociados a las evaluaciones subjetivas, principalmente el hecho de basarse en el trabajo de un conjunto de personas, los investigadores e ingenieros naturalmente buscaron procedimientos automáticos para llevar a cabo estas tareas. Estos son llamados evaluaciones objetivas. Las métricas objetivas, básicamente son fórmulas o algoritmos (usualmente algoritmos de procesamiento de señales) que miden, en cierto sentido, la calidad de un *streaming* de video. Salvo algunas pocas excepciones, las evaluaciones objetivas proponen diferentes maneras de comparar la señal distribuida con la señal original, usualmente computando una especie de distancia entre estas señales.

Los métodos de evaluación objetiva de la calidad de video más usados son:

- *Peek Signal to Noise Ratio* (PSNR),
- *Video Quality Metric* (VQM) de ITS[11, 133],
- *Moving Picture Quality Metric* (MPQM) de EPFL,
- *Color Moving Picture Quality Metric* (CMPQM) [128, 129],
- *Normalization Video Fidelity Metric* (NVFM) [129].

Estos métodos difieren en su complejidad, desde el más simple (PSNR) al más sofisticado basado en el *Human Vision System* (HVS), como por ejemplo CMPQM o NVFM. A continuación se presentan detalles sobre dos de estos métodos: PSNR y VQM.

Peak Signal–to–Noise Ratio El más común y simple de los métodos objetivos para la evaluación de la calidad de video es *Peak Signal–to–Noise Ratio* (PSNR). Definimos el error cuadrático medio (*Mean Square Error* o MSE) entre una secuencia de video original o y una secuencia distorsionada d como:

$$\text{MSE} = \frac{1}{K.M.N} \sum_{k=1}^{K} \sum_{m=1}^{M} \sum_{n=1}^{N} [o_k(m,n) - d_k(m,n)]^2$$

Donde cada secuencia de video tiene K *frames* de $M \times N$ píxeles y $o_k(m,n)$ y $d_k(m,n)$ son la luminancia del píxel en la posición (m,n) del k^{th} *frame* de la secuencia. El PSNR es definido como:

$$\text{PSNR} = 10.\log_{10} \frac{L^2}{\text{MSE}}$$

En donde L es la máxima luminancia en un *frame*. Cuando los píxeles son representados usando 8 *bits* por muestra se tiene $L = 255$.

La primera y gran ventaja del PSNR es que este es simple de computar. Sin embargo, en nuestro contexto, este no es apropiado como una medida de la QoE, ya que no se correlaciona bien con las evaluaciones subjetivas, donde los resultados del PSNR pueden diferir en gran forma con la calidad percibida por los usuarios [12].

Video Quality Metric de ITS El método *Video Quality Metric* (VQM) [11, 133] es desarrollado por el *"Institute for Telecommunication Sciences"* (ITS). En primera instancia, este método extrae ciertas características del video original y del video distorsionado. Las características son una medida objetiva que caracteriza los cambios perceptuales en la secuencia, analizando tanto información espacial, como información temporal y de crominancia. Luego un conjunto de parámetros de calidad son calculados, comparando las características originales con las de la señal distorsionada. Usando estos parámetros, una clasificación asigna una medida de calidad. La clasificación es una combinación lineal, calculada usando funciones que modelan el comportamiento visual humano.

Este método de evaluación es práctico cuando no es posible tener la señal original y la señal distorsionada al mismo tiempo, por ejemplo cuando se realiza distribución de video en una red. Pero de cualquier manera, todavía es necesario tener la información sobre las características de la señal original al analizar la señal distorsionada. VQM es aceptado como un estándar de evaluación de la calidad objetiva por ANSI [9] y algunos estudios [12] muestran una buena correlación con las evaluaciones subjetivas cuando se tratan con codificaciones de bajo *bitrate*, en los otros casos, este no se correlaciona de buena manera.

A modo de resumen, las evaluaciones de calidad objetivas normalmente implican tener tanto la señal original como la señal distorsionada en el mismo lugar, por lo que su uso no es posible en ambientes distribuidos con señales de tiempo real. Dejando esto de lado, el problema más importante, es que en muchos casos estas evaluaciones no se correlacionan de buena manera con la percepción humana, por lo que su uso como reemplazo a las evaluaciones subjetivas es muy limitado.

4.3 Un Enfoque Híbrido: *Pseudo-Subjective Quality Assessment* (PSQA)

El *Pseudo-Subjective Quality Assessment* (PSQA) [90, 110] es una técnica híbrida entre las evaluaciones subjetivas y las objetivas, la cual permite aproximar los valores obtenidos mediante evaluaciones subjetivas de forma automática. Por este motivo, el PSQA junta las ventajas de los dos tipos anteriores de pruebas: brinda una evaluación simple con un bajo costo computacional, sin la necesidad de tener la secuencia original, que además se correlacionan en buena medida con la calidad percibida por los usuarios. Como consecuencia, esta técnica es ideal para aplicaciones de distribución de video en tiempo real, como en las que se enfoca nuestro proyecto.

A muy alto nivel, la idea consiste en realizar una evaluación subjetiva, compuesta por un panel de observadores humanos, de un conjunto de secuencias de video. Luego con los resultados de esta evaluación, se entrena una red de aprendizaje automático (los mejores resultados se obtuvieron con redes neuronales [48, 110]) con el fin de capturar la relación existente entre los parámetros causantes de la distorsión de las secuencias y la calidad percibida por los usuarios.

La evaluación PSQA fue propuesta en [87] y ha sido aplicada a diferentes contextos: conversaciones interactivas [88, 89], video H.263 [69] usado en teleconferencias [90, 91], sobre redes de tipo DiffServ IP [125], etc.

4.3.1 La Metodología PSQA Aplicada Sobre Video

Resumiendo el trabajo presentado en [110], la metodología PSQA consiste en 3 diferentes etapas (observar Figura 4.2):

1. La definición de parámetros que afectan a la calidad del video y la generación de una base de datos con secuencias distorsionadas por dichos parámetros.
2. La realización de una evaluación subjetiva de las secuencias incluidas en la base de datos.
3. El entrenamiento de una red de aprendizaje automático, con el fin de correlacionar los parámetros definidos con la calidad percibida por los usuarios.

A continuación veremos una reseña de cada una de estas etapas.

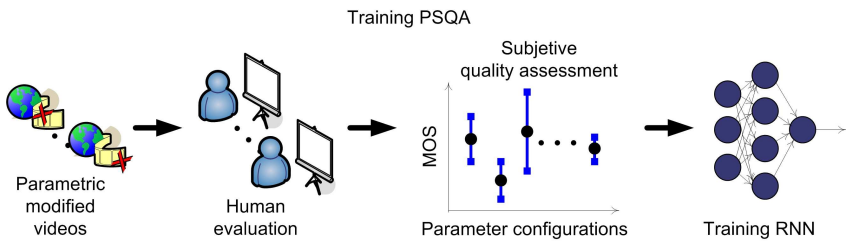


Figura 4.2: La Metodología PSQA.

Definición de Parámetros y Generación de Secuencias. El primer paso dentro de la metodología PSQA, es la selección de ciertos parámetros que pensamos que pueden afectar en la calidad percibida por usuarios finales. Esta etapa es una de las más importantes, dado que la precisión y robustez del método depende en gran medida de los parámetros seleccionados. En particular depende de qué tanto estos parámetros se correlacionan con la calidad del video. Es posible clasificar los potenciales parámetros a tener en cuenta en los siguientes grupos: parámetros de ambiente, parámetros de la fuente, parámetros de transmisión y parámetros del receptor. Algunos ejemplos de parámetros de ambiente son: el nivel de ruido ambiental, la luz donde se realice la prueba, la definición del monitor o televisión, los parlantes a usar, etc. Normalmente este tipo de parámetros no son tenidos en cuenta en el diseño de PSQA. Los parámetros de la fuente se refieren a características propias del video, como por ejemplo: la naturaleza de la escena (cantidad de movimiento, color, contraste, etc.), parámetros referidos con la codificación del mismo (tipo de codec, *bitrate*, *framerate*, tamaño de GOP, etc.), etc. Algunos de los ejemplos de parámetros de transmisión, los cuales normalmente se refieren a cuestiones a nivel de red, son: ancho de banda disponible en la red, pérdidas de paquetes en la red, *delay* en la red, *jitter* en la red, etc. Finalmente los parámetros del receptor, se refieren a ciertas técnicas que pueden ser implementadas en el receptor con el fin de no degradar la calidad del video. Algunos ejemplos son: tamaño de *buffer* del cliente, técnicas de ocultación de errores (interpolación, regeneración de datos perdidos), etc.

Una vez que se tienen definidos los parámetros sobre los cuales construir la aplicación PSQA, se deben buscar configuraciones representativas para cada uno de ellos. Por ejemplo, si un parámetro es el *bitrate* del video, se debería definir las posibles configuraciones en base a la aplicación objetivo, si se trata de una aplicación de *streaming* sobre Internet, valores razonables serían 100Kbps, 350kbsp y 500Kbps. Finalmente se debe generar una base de datos de secuencias con (en lo posible) todas las posibles combinaciones de configuraciones de parámetros. Esta base de datos va a ser usada para realizar las evaluaciones subjetivas.

Evaluación Subjetiva En la segunda etapa de la metodología PSQA, se evalúan los parámetros definidos previamente, mediante pruebas subjetivas. Normalmente para realizar esta evaluación se sigue alguno de los métodos definidos en la Sección 4.1. Luego de realizada la evaluación, con los resultados se suele calcular el *Mean Opinion Score* (MOS) y su desviación estándar para cada secuencia. Con lo cual, es posible filtrar opiniones de usuarios que estuvieron muy desviados de la media en la mayoría de los videos, seguramente debido a causas como la falta de atención o de entendimiento al realizar la prueba. De esta manera, luego de haber filtrado opiniones no coherentes con la mayoría de los usuarios, se obtiene para cada secuencia de video un valor acerca de su calidad percibida.

Entrenamiento de una RNN En la tercer etapa de la metodología PSQA, se entrena y valida una herramienta de aprendizaje automático. En particular, los mejores resultados se han obtenido usando redes neuronales o en ingles: *Random Neural Network* (RNN) [48]. En esta etapa se deben dividir los resultados de forma aleatoria en dos grupos, uno usado para entrenar la RNN y el segundo usado para validar el entrenamiento de la misma. Normalmente se suele usar el 70 % u 80 % de los datos para entrenar la red y el 30 % o 20 % restante para validar la misma.

También en esta etapa se deben definir parámetros propios de las RNN como la topología a usar (cantidad de capas), los pesos, etc. Una vez entrenada y validada la RNN, ya se tiene una aplicación PSQA pronta para ser usada.

Modo de Uso de una Aplicación PSQA Luego de construida la aplicación PSQA, el modo de uso es muy sencillo. Simplemente se debe capturar el valor de los parámetros dentro de la señal en cuestión, ingresarlos en la RNN entrenada y obtener el valor de la calidad percibida (observar la Figura 4.3).

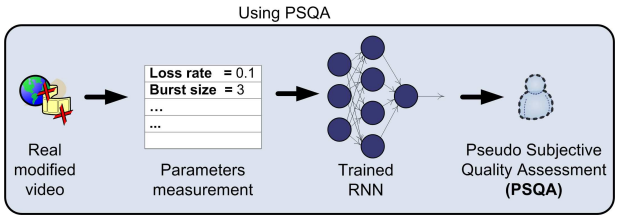


Figura 4.3: Modo de Uso de una Aplicación PSQA.

4.3.2 Un Diseño PSQA para una Red de Distribución de Video MPEG-4

En esta subsección, se presenta una aplicación PSQA para una red de distribución de video MPEG-4, desarrollada dentro del marco del doctorado de Pablo Rodríguez-Bocca [110], la cual, tal y como veremos más adelante, se encuentra integrada en nuestra red P2P.

La aplicación es presentada en base a las etapas existentes dentro de la metodología PSQA.

Definición de Parámetros y Generación de Secuencias. El diseño de esta aplicación PSQA, combina factores propios del video, tal como la cantidad de movimiento y el tipo de codificación, con aspectos referidos a la distribución, como pérdidas en la red.

Respecto a los parámetros referidos a la transmisión, los siguientes 3 parámetros son tenidos en cuenta:

Tasa de Pérdidas de *I-Frames*. En este caso, dado que la duración de las secuencias debe ser de como máximo 10 segundos, básicamente existe un único GOP dentro de la secuencia (y por lo tanto un único *I-Frame*), lo que implica que el valor de este parámetro puede ser 1 o 0. Lo llamamos LR_I.

Tasa de Pérdidas de *P-Frames*. Este parámetro mide el porcentaje de *P-Frames* perdidos por GOP. Lo llamamos LR_P.

Tasa de Pérdidas de *B-Frames*. Al igual que en el parámetro anterior, este mide el porcentaje de *B-Frames* perdidos por GOP. Lo llamados LR_B.

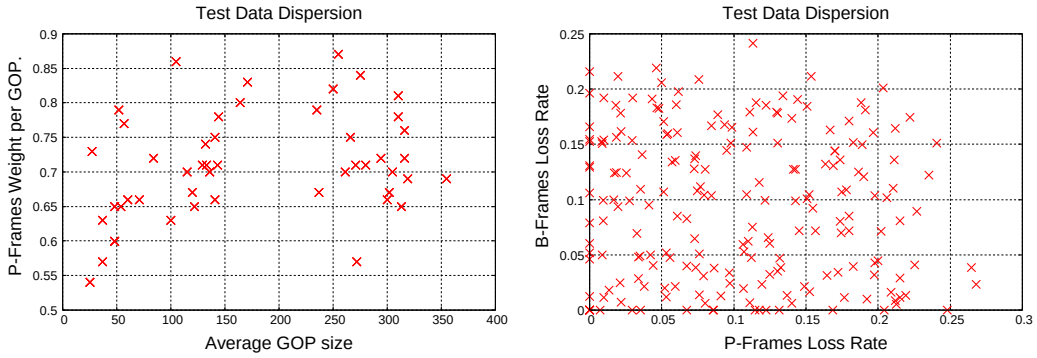
Como fue presentado en la Sección 2.1, los *I-Frames* son imágenes de video codificadas en JPEG y mediante los *P-Frames*/*B-Frames* se codifican las redundancias espaciales y temporales

existentes en el video. En base a esta información, 2 parámetros referidos a la fuente son tenidos en cuenta:

Tamaño Promedio de GOP. El *encoder* decide cuando agregar un *I-Frame* en base a las características del video, por lo que este parámetro refleja si el video presenta muchos cambios abruptos o no. Lo llamamos $\overline{\text{GOP}}$.

Peso de los *P-Frames* por GOP. Este parámetro indica el peso en *bytes* de los *P-Frames* dentro del GOP. Dado que el movimiento es codificado en base a *P-Frames*, esta es una manera muy simple de representar la cantidad de movimiento existente dentro de la escena.

Para generar la base de datos de videos, se usan 50 secuencias de video MPEG-4 diferentes, de unos 10 segundos de duración, con tamaño de entre 238 KB y 2.78 MB. Estas son elegidas con el fin de cubrir los rangos de los parámetros de fuente. La dispersión de los parámetros de fuente es presentada en la Figura 4.4(a). Con estas 50 secuencias se generan 204 secuencias distorsionadas en base a los parámetros de transmisión, que al igual que en el caso de los parámetros de la fuente, se intenta cubrir lo más posible los rangos de los parámetros de transmisión. En la Figura 4.4(b) se presenta la dispersión de los parámetros LR_P y LR_B. Notar que los parámetros LR_P y LR_B se encuentran acotados por el valor 0,3, esto se debe a que tasas de pérdida mayores a este número, recaen en una calidad mínima en la reproducción del video.



(a) Tamaño Promedio de GOP y Peso de los *P-Frames* por GOP.

(b) LR_P y LR_B.

Figura 4.4: Dispersión en las secuencias de video.

Evaluación Subjetiva En la evaluación subjetiva se aplica el método *Single Stimulus* presentado en la Sección 4.1. En este caso 10 personas evalúan la calidad percibida en 204 secuencias de video. En la Figura 4.5, se presentan los resultados obtenidos para cada secuencia junto con el MOS. La desviación estándar en las evaluaciones es de 0.15, lo que significa que en promedio los evaluadores difirieron en un $\pm 15\,\%$ en sus opiniones.

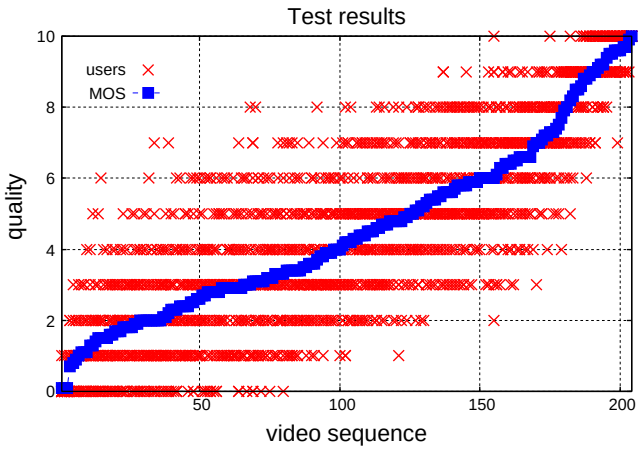


Figura 4.5: Resultados de la evaluación subjetiva

Entrenamiento de una RNN Al entrenar la RNN, se prueba con diferentes topologías, obteniendo los mejores resultados al aplicar una topología de 3 capas, tal como se presenta en la Figura 4.6. Mediante esta, se obtiene un MSE en la validación de 0.254898.

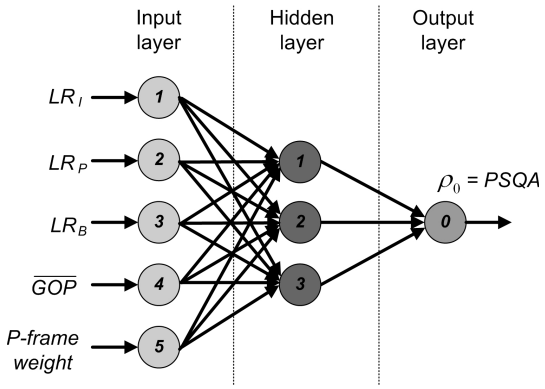


Figura 4.6: Topología usada en la RNN.

Algunos de los resultados brindados por la RNN luego de entrenada son muy brevemente presentados a continuación (para obtener una descripción completa ver [110]). Como se puede observar en la Figura 4.7, la calidad se degrada rápidamente al aumentar las pérdidas en los *I-Frames* o *P-Frames*.

En la Figura 4.8(a) y en la Figura 4.8(b), se puede observar como la calidad disminuye lentamente al aumentar las pérdidas de los *B-Frames* (tal cual como es esperado).

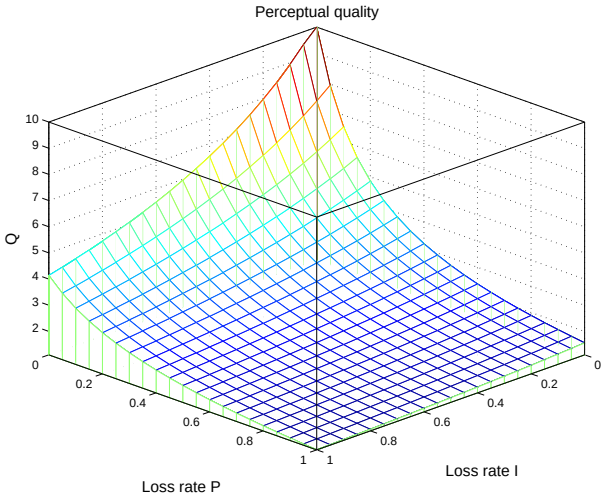
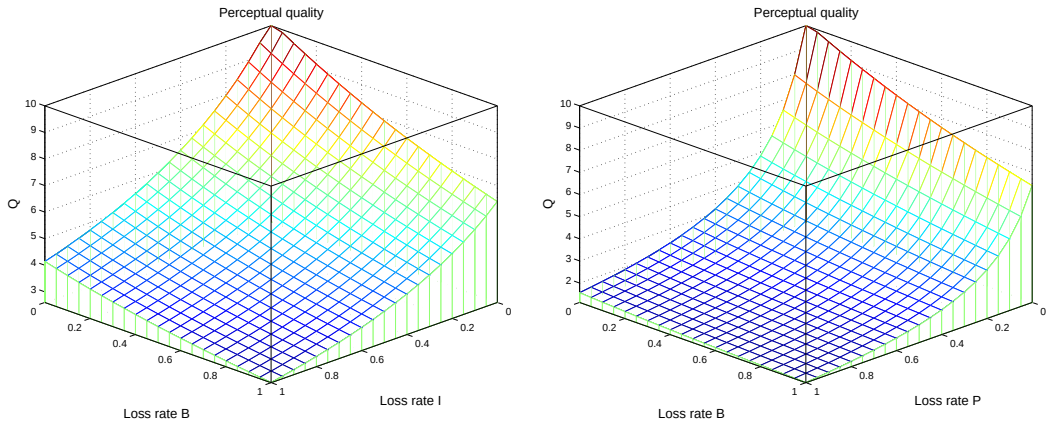


Figura 4.7: La calidad se degrada rápidamente al aumentar las pérdidas en los *I-Frames* o *P-Frames*.



(a) LR_B vs. LR_I. (b) LR_B vs. LR_P.

Figura 4.8: La calidad disminuye lentamente al aumentar las pérdidas de los *B-Frames*.

4.4 Soluciones para el Monitoreo de Redes de Distribución de Video

En esta sección se presentan algunas de las soluciones actuales para el monitoreo de redes de distribución de video, las cuales en general, suelen ser soluciones de gran porte, muy costosas en términos económicos, principalmente usadas en sistemas de tipo IPTV (ver Sección 2.2.1).

Al momento de monitorear una red de video, existen al menos 2 diferentes enfoques posibles para llevar a cabo dicho monitoreo (los cuales pueden ser aplicados de manera conjunta): monitorear las señales a nivel de codificación (es decir, dentro del *head-end* del sistema) o monitorear los contenidos a nivel de usuario final (es decir, luego del proceso de distribución del mismo).

Algunos ejemplos de soluciones a nivel de *head-end*, son: [46, 85] (llamados "*multiviewer*"). Estos dispositivos reciben varios flujos de video y presentan una imagen consolidada de los mismos a través de un panel de control. Con ellos se monitorea la adquisición y la codificación de los diferentes flujos de video.

Ejemplos de dispositivos ubicados a nivel de usuario (llamados *"probes"*) son: [45, 64]. Estos dispositivos suelen conectarse a la salida del *set-top box*, con el fin de capturar información acerca de la recepción y estado del *streaming*. Dentro de la información capturada se encuentran: la presencia o no de video, nivel de audio y luminancia, cantidad de movimiento, formatos de audio y video, capturas de pantalla, etc. A su vez, estos dispositivos permiten acceder remotamente a esta información, mediante SNMP [58] o algún protocolo propietario. Otro ejemplo de *probe* se presenta en [120], mediante el cual es posible re-distribuir la señal recibida en el *set-top box* sobre Internet, a una tasa conveniente.

Ver [47, 63, 84] por el detalle de algunas soluciones integrales, las cuales abarcan desde el monitoreo en el *head-end* hasta el monitoreo a nivel de usuarios.

Se puede notar que la mayoría de las soluciones presentadas, terminan por tener un panel de monitoreo, en donde se pueden observar todos los contenidos que están siendo distribuidos (a nivel del *head-end*) junto con alguna información complementaria del estado en que están siendo recibidos por algunos usuarios finales (tomando los reportes desde los *probes*). Estas soluciones requieren de personal dedicado y no incluyen medidas objetivas automáticas. A su vez, claramente la visión brindada por estas soluciones es una vista incompleta de la red. Debido a esta última razón, junto con el hecho de la falta de una plataforma de monitoreo simple desarrollada únicamente en base a *software*, es que como resultado de esta tesis proponemos desarrollar un monitor simple, pero a su vez genérico, el cual, en base a los reportes PSQA de cada uno de los clientes (calculados de forma automática), permita tener una visión completa de la calidad de experiencia existente en la red.

Parte II

STREAMING GOALBIT

Capítulo 5

Especificación del *Streaming* GoalBit

En este capítulo se presenta la especificación del *Streaming* GoalBit [51], brindando detalles sobre su arquitectura, su protocolo, el diseño de las políticas aplicadas en éste y su implementación de referencia. Esta información fue publicada en [br-lanc09] (ver Sección 1.4).

La especificación de GoalBit consta de dos partes: el *GoalBit Transport Protocol* (GBTP) y el *GoalBit Packetized Stream* (GBPS). El protocolo de trasporte GBTP[1] define la forma en que los *peers* intercambian el contenido entre sí. Es una extensión del protocolo BitTorrent, presentado anteriormente en la Sección 3.3, con el fin de soportar la distribución de un flujo continuo de video. GBPS especifica como se debe encapsular el contenido audiovisual dentro del protocolo GBTP. Por como han sido diseñados, tanto GBPS como GBTP no presentan restricción alguna sobre las codificaciones (audio y/o video) y formatos contenedores a encapsular/transportar.

En la Sección 3.4, fueron presentadas algunas de las redes P2P para la distribución de video en vivo más populares en la actualidad, entre ellas: PPLive [103] con más de $200,000$ usuarios concurrentes en un mismo canal [116], SopCast [121] con más de $100,000$ usuarios concurrentes, Octoshape [95], PPstream [104], TVAnts [126] y TVUnetwork [127]. Todas estas redes son comerciales y de código/protocolo cerrado.

Una primera y gran diferencia de la red GoalBit respecto a las redes mencionadas anteriormente, es el hecho de ser una red no comercial, de protocolo abierto, junto con una implementación de referencia de código abierto. De hecho, GoalBit es la primera red *open-source* de este tipo en Internet.

El cliente GoalBit (la implementación de referencia), al igual que algunas de las aplicaciones P2PTV comerciales, es una aplicación completa para la distribución de video en vivo, la cual implementa desde la adquisición, codificación y distribución del contenido, hasta la reproducción del mismo. Algunas de las principales características del cliente son:

- Al momento de distribuir contenidos, estos pueden ser capturados de distintas fuentes (mediante una tarjeta capturadora, una cámara web, un stream HTTP/MMS/RTP/UDP, un archivo en disco, etc.), pudiendo ser codificados y multiplexados bajo diferentes es-

[1] En este documento se le llama protocolo de transporte a aquellos que son utilizados para enviar video en una red TCP/IP, esto es lo usual para este contexto, como por ejemplo RTP (Real-time Transport Protocol). No confundir con los protocolos incluidos en la capa de transporte del modelo de OSI (TCP, UDP, etc).

pecificaciones. Algunos ejemplos son MPEG-2, MPEG-4 (H.264, DivX), VC-1 (WMV) o Theora para el caso del video, AAC, VORBIS, MPGA o WMA para el del audio y MPEG-TS, MPEG-PS, ASF, OGG y MP4 en lo referente a lo multiplexación.

- Implementa de forma completa la especificación GBPS y el protocolo GBTP.
- Posee un reproductor nativo de video con el fin de presentar los contenidos a los usuarios finales.
- Es de código abierto, basado en la adaptación e integración de las siguientes aplicaciones: Videolan Media Player (VLC) [131] (para todo lo referente a la manipulación, codificación y reproducción de video), Enhanced CTorrent [43] (como cliente BitTorrent de base) y OpenTracker [97] (como tracker BitTorrent de base).

5.1 La Arquitectura de GoalBit

En esta sección se presenta la arquitectura de la red, sus componentes y las funciones de cada uno de estos.

Como se puede observar en la Figura 5.1, existen 4 tipos diferentes de componentes en la red: el *broadcaster*, los *super-peers*, los *peers* normales y el *tracker*. El *broadcaster* es el responsable de obtener el contenido y de volcarlo a la red. Los *super-peers* son *peers* con buena capacidad, los cuales cumplen principalmente la función de la distribución inicial del contenido. Los *peers* normales son los usuarios finales, quienes se conectan a la red con el único fin de reproducir un cierto contenido. El *tracker* cumple el mismo rol que en BitTorrent, siendo el encargado de administrar los *peers* en la red. La información acerca de que contenidos se están transmitiendo es distribuida mediante la *Web*. Usualmente, algunos servidores *Web* mantienen un conjunto de archivos `.goalbit` (nuestros archivos de configuración, similar a los archivos `.torrent` en BitTorrent), desde donde los usuarios los descargan.

A continuación veremos en detalle cada uno de estos componentes.

5.1.1 El *Broadcaster*

El *broadcaster* recibe el contenido de alguna fuente (desde una señal analógica, un *stream* existente, un archivo de video almacenado en disco, etc.), lo procesa (codificación y multiplexación del audio y del video) y lo vuelca a la red *Peer-to-Peer*.

Este componente cumple un rol crítico dentro de la cadena de distribución del contenido. Si este abandona la red, obviamente finaliza la distribución. Como consecuencia, en un buen diseño, este no debe encontrarse sobrecargado con varias tareas. Su principal tarea es la de creación y transmisión del contenido y no la de la distribución masiva del mismo.

5.1.2 Los *Super-peers*

Como comentamos anteriormente, el *broadcaster* no debe ser el responsable de llevar a cabo la distribución inicial del contenido hacia los *peers* normales. Esta es la principal tarea de los *super-peers*. Los *super-peers* son *peers* de alta disponibilidad y gran capacidad de ancho de

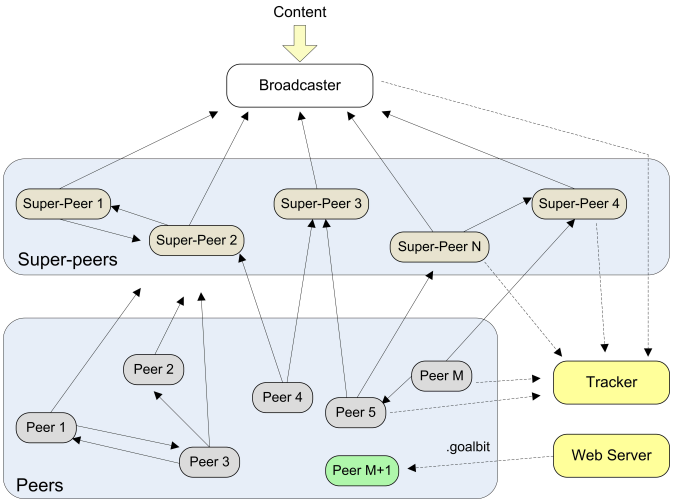


Figura 5.1: Arquitectura GoalBit. Los 4 componentes de la red y sus relaciones son presentadas en esta imagen (el *broadcaster*, los *super-peers*, los *peers* y el *tracker*).

banda, quienes obtienen el *streaming* del *broadcaster* y de otros *super-peers*, y lo distribuyen hacia los *peers* normales.

Como primer diferencia con BitTorrent, en nuestro sistema el concepto de *seeder* (definido en la Sección 3.3) depende del tipo de *peer*. Para los *super-peers* el *seeder* es el *broadcaster*, mientras que para los *peers* normales los *seeders* son el *broadcaster* (al que normalmente no tienen acceso directo) y todos los *super-peers* (con quienes sí se comunican frecuentemente).

5.1.3 Los *Peers*

Los *peers* son los usuarios finales del sistema, quienes se conectan a un *streaming*, con el único fin de poder reproducirlo. Representan a la mayoría de los nodos dentro del sistema. Usualmente la cantidad de *peers* en la red debería ser mucho mayor a la cantidad de *super-peers*. Los *peers* se caracterizan por presentar un comportamiento muy dinámico respecto al tiempo: se conectan y desconectan de la red con una alta frecuencia.

Finalmente, y no menor, los *peers* son la "razón de ser" de la red, el sistema es construido para que ellos lo utilicen.

En futuras versiones de GoalBit, se espera promover un nodo de *peer* a *super-peer*, cuando dicho nodo presente un buen y sostenido desempeño en la red (en la actualidad el tipo de *peer* es definido al inicio de la ejecución y este permanece incambiado hasta el final de la misma). Este cambio permitirá realizar un mejor aprovechamiento de los recursos disponibles en la red, puesto que hoy en día los únicos *super-peers* en la red suelen ser provistos por los propietarios de los *broadcasters*.

5.1.4 El *Tracker*

El *tracker* mantiene una lista de todos los nodos (*broadcaster*, *super-peer* y *peers*) conectados a la red (ver Figura 5.2).

Cada vez que un *peer* se conecta a la red, este se anuncia en el *tracker*, quien registra a dicho *peer* en el contenido correspondiente. A su vez, el *tracker* le retorna a este un listado de otros *peers* también registrados bajo el mismo contenido. Usando este listado, el *peer* establece comunicación con otros *peers*, comenzando de esta manera el intercambio de piezas.

Como es de imaginar el *tracker* cumple un rol central en la distribución del video en la red. Si este se encuentra fuera de servicio, la distribución es imposible, al menos para los *peers* nuevos que no conocen otros *peers* de los cuales obtener el contenido.

Con el fin de mitigar este problema (y pasar de ser una red híbrida a una red pura) es posible reemplazar al *tracker* por la implementación de un *overlay* de descubrimiento estructurado (como por ejemplo una red Kademlia [80], ver Sección 3.2), en donde se mantenga almacenada la información manejada por el *tracker*. Este tipo de redes, en donde no existe un *tracker* central, se suelen llamar `trackerless`. Actualmente en GoalBit, no se encuentra implementada la funcionalidad de *trackerless*.

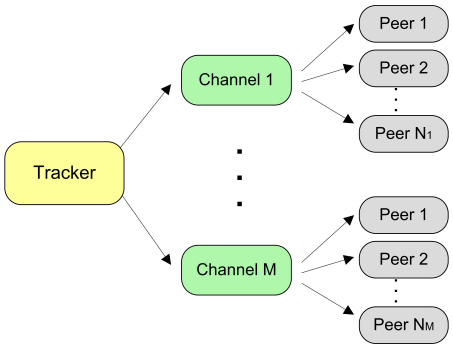


Figura 5.2: Estructura de datos almacenados en el *Tracker*.

5.2 *GoalBit Transport Protocol*

Como se comentó anteriormente el *GoalBit Transport Protocol* (GBTP) es una versión extendida del protocolo BitTorrent, optimizada para la transferencia de video en vivo. En esta sección introducimos algunas de sus principales características y presentamos su completa especificación.

5.2.1 Conceptos Generales

En GBTP, al igual que en BitTorrent, cada pieza del *stream* es identificada por un único número llamado *ID de pieza*. Este identificador es usado por los *peers* para llevar a cabo el intercambio

de piezas. Dado que el GBTP es diseñado para la transmisión de un flujo continuo y no de un archivo, la numeración de las piezas es cíclica en el intervalo $[0, MAX\_PIECE\_ID]$[2].

Cada *peer* posee una ventana deslizante en donde se alojan las piezas descargadas. Esta ventana tiene un valor mínimo, llamado *base*, y un largo que define el máximo ID de pieza dentro de la ventana. Los *peers* únicamente pueden descargar y compartir piezas cuyo identificador se encuentre incluido dentro sus respectivas ventanas.

Para llevar a cabo la reproducción del video, los *peers* tienen un reproductor que consume piezas en forma secuencial dentro de sus respectivas ventanas deslizantes. Se define como índice de ejecución, o también llamado linea de ejecución, al identificador de la siguiente pieza a consumir. Otros conceptos importantes manejados dentro del protocolo son el de *buffer* activo y el de índice de *buffer* activo (o ABI). Como se puede observar en la Figura 5.3, se define como *buffer* activo a la secuencia consecutiva de piezas disponibles a partir de la linea de ejecución. En otras palabras, este concepto se corresponde con el video descargado, que el reproductor va a poder reproducir de manera ininterrumpida en el instante actual. Finalmente definimos el índice de *buffer* activo como el mayor ID de pieza dentro del *buffer* activo.

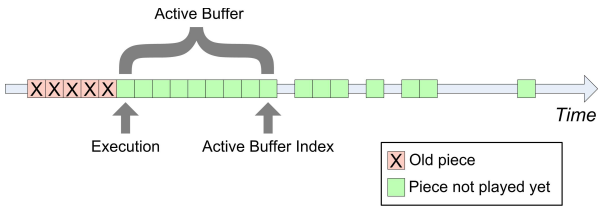


Figura 5.3: Definición de *buffer* activo e índice de *buffer* activo.

Información Almacenada en el *Tracker*. Como vimos anteriormente, un *tracker* gestiona múltiples contenidos. Para cada uno, el *tracker* almacena una referencia de los *peers* conectados a dichos contenidos. A continuación se presentan detalles acerca de la información almacenada en el *tracker* por cada contenido y por cada *peer*.

Como se puede observar en la Figura 5.4, cada contenido posee un identificador llamado *info_hash*: un *string* en formato 20-byte SHA1. A su vez, cada contenido almacena información sobre los *peers* conectados a él. Para cada *peer* se almacenan dos tipos diferentes de información: datos estáticos y datos dinámicos. La datos estáticos están compuestos por la IP del *peer*, el puerto en donde escucha por nuevas conexiones y el tipo de *peer* (*broadcaster-peer*, *super-peers*, *peers* o *broadcaster-super-peer*[3]). Dentro de los datos dinámicos (o también llamados información de estado) se pueden encontrar por ejemplo: el ABI del *peer*, la fecha del último anuncio del *peer*, la última calidad de experiencia reportada (usando la función de PSQA definida en la subsección 4.3.2), etc.

[2] Actualmente el parámetro MAX_PIECE_ID es definido como 2^{31}.

[3] El tipo *broadcaster-super-peer* es usado en redes donde no existen *super-peers*. Esta es una versión simplificada de la arquitectura GoalBit, en donde el *broadcaster* sirve directamente a los *peers*.

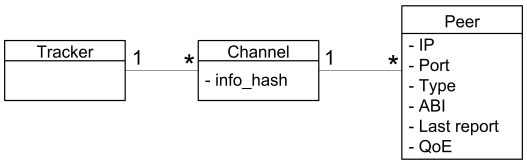


Figura 5.4: Información almacenada en el *tracker*.

Archivos GoalBit. Los archivos GoalBit (con extensión `.goalbit`) son descriptores de contenidos. Dentro de estos archivos se define la información necesaria para que un usuario pueda ejecutar un *streaming* (la dirección del *tracker*, el identificador del contenido, la metadata del contenido, etc.). Estos archivos se corresponden con los archivos Torrent de BitTorrent (archivos con extensión `.torrent`), con la diferencia que los archivos GoalBit son archivos de texto plano, estructurados mediante XML. Por cada contenido se almacena la información presentada en la Tabla 5.1. Dada la posibilidad de que un archivo GoalBit describa múltiples contenidos, es posible definir un contenido por defecto, el cual será ejecutado al abrir dicho archivo.

Cuadro 5.1: Información de un contenido almacenada dentro de un archivo GoalBit.

Parámetro	Descripción
channel_id	Identificador del contenido (una cadena de caracteres).
chunk_size	Tamaño de pieza para la ejecución.
tracker_url	URL del *tracker* donde está la información del contenido.
bitrate	*Bitrate* del *stream*.
name	Nombre del contenido.
description	Descripción del contenido.
thumb	Logo del contenido.

5.2.2 Flujo de Ejecución de un Contenido

El flujo de ejecución llevado a cabo en GoalBit es muy similar al llevado a cabo en la descarga de un archivo mediante BitTorrent.

A grandes rasgos, como se puede observar en la Figura 5.5, el proceso de ejecución se puede resumir en 3 diferentes etapas: la obtención de un archivo GoalBit, la comunicación inicial con el *tracker* y la comunicación entre *peers*. En la primera, el usuario obtiene el archivo GoalBit, quizás descargandolo de la *Web*, quizás vía *mail*, etc. Como vimos anteriormente, dentro de este archivo se encuentra la información necesaria para comenzar la ejecución del *streaming*: URL del *tracker*, tamaño de pieza, identificador del contenido, etc. El *peer* se anuncia a sí mismo en el *tracker*, quien le retorna un listado de otros *peers* (llamado *swarm*) y un índice (un ID de pieza) de donde comenzar la ejecución. Con esta información, el *peer* se contacta

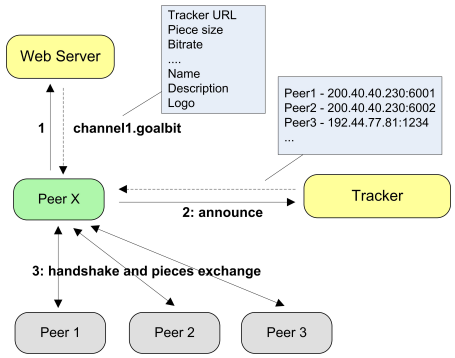


Figura 5.5: Flujo de ejecución al usar GoalBit.

con los otros *peers* del *swarm*, llevando a cabo el intercambio de piezas. Finalmente, el *peer* comienza la reproducción del contenido cuando su *buffer* se ha completado.

A continuación se detalla la comunicación entre los *peer* y el *tracker* y la comunicación entre los mismos *peers*.

5.2.3 Comunicación Tracker-Peer

Existen dos tipos de intercambios entre los *peers* y el *tracker*: la comunicación inicial y la comunicación periódica, llevada a cabo durante la estadía del *peer* en la red. Estas comunicaciones son sobre `HTTP/HTTPS` y son basadas en un único mensaje llamado *announce*. En la Tabla 5.2 se presentan los parámetros enviados por el *peer* en la solicitud del *announce* al *tracker* y en la Tabla 5.3 los incluidos en la respuesta enviada por el *tracker* al *peer*.

Inicialmente el *peer* contacta al *tracker* con el fin de obtener un listado de *peers* y un índice en donde comenzar a pedir piezas. Normalmente, en este *announce* el *peer* envía *numwant* = *55* y *abi* = *MAX_PIECE_ID + 1*. En la respuesta, el *peer* obtiene un conjunto de *peers* (de a lo sumo *numwant peers*), un índice inicial (un ID de pieza) y un tipo de *peer* asignado.

Dado que el *tracker* necesita mantener información acerca del estado de cada *peer*, debe existir una comunicación regular entre estos para transmitir dicha información. En la comunicación periódica (y dejando de lado ciertos casos particulares), el *peer* envía un *announce* con *numwant* = 0, *abi* = el ABI actual del *peer* y *qoe* = la calidad de experiencia medida por el *peer* en el último período. En este caso, no existe información relevante para el *peer* en la respuesta del *announce*. En la comunicación periódica y en caso de necesitarlo, los *peers* pueden pedir un nuevo listado de *peers* al *tracker* (cambiando el parámetro *numwant*). La comunicación periódica es regulada por el *tracker*, quien define el mínimo intervalo de tiempo entre 2 *announces* consecutivos mediante el parámetro *interval*. El valor de tiempo utilizado por defecto es de 30 segundos.

En ambos tipos de comunicación (la inicial y la periódica), el *tracker* construye diferentes listas de *peers* dependiendo del tipo de *peer* que se está anunciando: para el caso de los *broadcasters* se envía un *swarm* vacio; en el caso de los *super-peers*, el *swarm* incluye algunos *broadcasters* y varios *super-peers*; finalmente en el caso de los *peers*, este incluye algunos

Cuadro 5.2: Parámetros enviados por el *peer* en la solicitud del *announce* al *tracker*.

Parámetro	Descripción
protocol	Versión del protocolo usada, actualmente "goalbit-0.5".
info_hash	Identificador del contenido (en formato 20-byte SHA1).
peer_id	Identificador del cliente. Este es un *string* aleatorio de 20 *bytes* de largo, generado por el *peer* al iniciar la sesión.
event	Este parámetro puede valer *"started"* o *"stopped"*. Ya sea en la comunicación inicial o en la periódica se debe enviar *"started"*. Únicamente en la desconexión de un *peer* se debe enviar *"stopped"*.
port	Puerto en donde el cliente escucha por conexiones entrantes.
uploaded	Cantidad de *bytes* subidos por el cliente desde el comienzo de la ejecución.
downloaded	Cantidad de *bytes* descargados por el cliente desde el comienzo de la ejecución.
numwant	Cantidad de *peers* que el cliente desea obtener del *tracker*. En caso de que este parámetro valga 0, el *tracker* no retorna ninguna referencia.
abi	ABI del *peer* al momento de enviar el *announce*. En el caso de la comunicación inicial (en donde el cliente aun no tiene un ABI definido) se debe enviar *MAX_PIECE_ID + 1*.
peer_type	Tipo de *peer*: *broadcaster-peer* = 1, *super-peer* = 2, *peer* = 3, *broadcaster-super-peer* = 4.
peer_subtype	Subtipo de *peer* (valor usado para ciertos casos particulares).
qoe	Calidad de experiencia medida por el cliente.
compact	Si este parámetro vale 1, indica que el cliente acepta una respuesta compacta, en donde cada *peer* enviado en la respuesta del *tracker*, es representado por 6 *bytes*: los primeros 4 *bytes* codifican la IP y los últimos 2 el puerto.
tracker_id	En la respuesta del *announce*, el *tracker* puede devolver un $tracker\_id$, si esto ocurre en los sucesivos *announces* el cliente debe incluir este valor.

Cuadro 5.3: Parámetros incluidos en la respuesta del *announce*.

Parámetro	Descripción
interval	Intervalo de tiempo (en segundos) que el cliente debe esperar hasta enviar un próximo *announce*.
tracker_id	Este *string* debe ser enviado en el siguiente *announce* al *tracker*.
broadcaster_num	Cantidad de *broadcaster-peers* en el contenido.
super-peer_num	Cantidad de *super-peers* en el contenido.
peer_num	Cantidad de *peers* en el contenido.
max_abi	Mayor ABI reportado por un *seeder*.
peer_type	Tipo de *peer* asociado al *peer* en cuestión.
offset	Índice de inicio (le indica al cliente en donde comenzar a pedir piezas). Este parámetro es usado únicamente en la comunicación inicial.
peers	Listado de *peers*.

super-peers y varios *peers*.

Existen múltiples estrategias para construir el *swarm* de un *peer* en el *tracker*, algunas de estas se encuentran detalladas en 5.2.5.3.

5.2.4 Comunicación *Peer-to-Peer*

Como vimos anteriormente, el *tracker* le envía a cada nuevo *peer* en la red, un subconjunto de otros *peers* que se encuentran reproduciendo el mismo contenido y un índice dentro del flujo de piezas en donde comenzar a reproducir dicho *streaming*. Una vez con esta información, los *peers* comienzan el intercambio de piezas. La comunicación entre estos, es llevada a cabo en base al intercambio de mensajes binarios sobre el protocolo TCP. El puerto es configurable, por defecto se usa el 2706.

Los diferentes tipos de mensajes intercambiados entre *peers* se pueden clasificar en dos grandes grupos: los de intercambio de información contextual y los de intercambio de piezas. El primero, incluye mensajes cuyo fin es informar acerca de la situación contextual de los *peer*, esto es: que *peer* permanece conectado y que piezas posee. Dentro de este grupo se incluyen 5 diferentes tipos de mensajes: HANDSHAKE, BITFIELD, HAVE, WINDOW UPDATE y KEEP-ALIVE. A su vez, como veremos más adelante, el intercambio de piezas sigue un proceso no trivial, en donde 8 tipos de mensajes son usados: INTERESTED, NOT INTERESTED, CHOKE, UNCHOKE, REQUEST, CANCEL, PIECE y DONT HAVE. Todos estos mensajes comprendidos dentro del segundo grupo.

A continuación se presentan detalles sobre cada uno de estos mensajes organizados según su grupo.

5.2.4.1 Intercambio de Información Contextual

En la Figura 5.6 se presenta un resumen de los mensajes incluidos en el intercambio de información contextual.

Inicialmente, el *peer* envía mensajes de tipo HANDSHAKE (ver mensaje *Handshake* en la Tabla 5.4) a cada uno de los *peers* recibidos desde el *tracker*. Entre otras cosas, dentro de este mensaje se envía información como el tipo de *peer* y la ventana deslizante (la base y el largo). Estos datos son usados por los otros *peers* para aceptar o rechazar la conexión del nuevo *peer*. En caso de aceptar dicha conexión, el receptor de este mensaje debe responder enviando otro mensaje de HANDSHAKE con sus propios datos.

Una vez que la comunicación entre dos *peers* está establecida, estos deben intercambiar mensajes de tipo BITFIELD (ver los detalles en la Tabla 5.5). En este mensaje se comunica las piezas que un *peer* tiene en la ventana deslizante del otro (los datos de la ventana del *peer* remoto son tomados del mensaje HANDSHAKE). De esta manera un *peer* conoce las piezas que inicialmente poseen el resto de sus contactos dentro de su ventana.

Para mantener esta información al día, cada vez que un *peer* recibe una nueva pieza, este debe enviar un mensaje de tipo HAVE (ver los detalles en la Tabla 5.6) a todo contacto suyo (llamémosle *peer B*) que cumpla con las siguientes condiciones:

- La pieza recibida pertenece a la ventana de *B*.

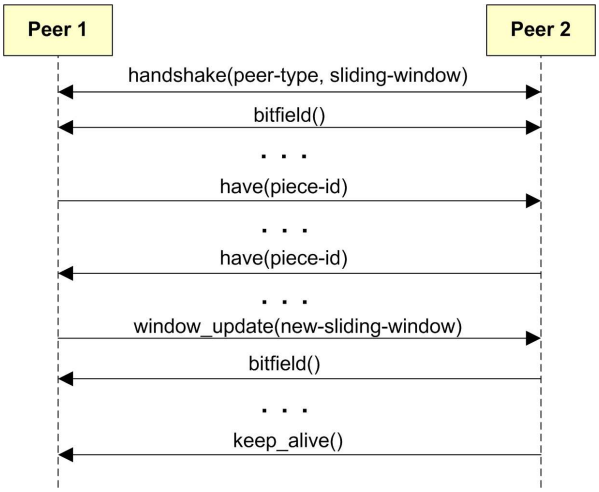


Figura 5.6: Resumen del intercambio de información contextual. Un *peer* intercambia mensajes de HANDSHAKE y BITFIELD con todos los *peers* de su lista para establecer la comunicación. Para mantener actualizada la información de las piezas que cada *peer* posee, se envían mensajes de HAVE, WINDOW UPDATE y BITFIELD. Ante una inactividad prolongada entre dos *peers*, es necesario enviarse mensajes de KEEP-ALIVE para mantener la comunicación.

Cuadro 5.4: Definición del mensaje HANDSHAKE. Este es el mensaje inicial en la comunicación entre 2 *peers*. Técnicamente, este debe ser enviado del uno al otro, cuando una nueva conexión es establecida. El largo de este mensaje es de 77 *bytes*.

Parámetro	Tamaño(*bytes*)	Descripción	Valor
pstrlen	1	Largo del parámetro *pstr*.	16
pstr	*pstrlen*	Identificación del protocolo (un *string* de caracteres)	GoalBit protocol
reserved	8	Espacio reservado por el protocolo.	0
info_hash	20	Identificador de la transmisión. Este es el identificador del contenido en cuestión (en formato 20-byte SHA1).	N/A
peer_id	20	Identificador del *peer*. Este es un *string* aleatorio de 20 *bytes*, generado por el *peer* al iniciar la sesión.	N/A
peer_type	4	Tipo de *peer*: *broadcaster-peer* = 1, *super-peer* = 2, *peer* = 3, *broadcaster-super-peer* = 4.	N/A
offset	4	Base de la ventana deslizante del *peer*.	N/A
length	4	Largo de la ventana deslizante del *peer*.	N/A

Cuadro 5.5: Definición del mensaje BITFIELD. Este mensaje informa acerca de las piezas que un *peer* tiene en una ventana especifica. Este tiene largo variable dependiendo del tamaño de la ventana. Es enviado inmediatamente luego del proceso de HANDSHAKE o como una respuesta al mensaje WINDOW UPDATE. Los *seeders* envían un *bitfield* nulo (*len*=0).

Parámetro	Tamaño(*bytes*)	Descripción	Valor
len	1	Largo del mensaje.	$5 + X$
id	1	Identificador del mensaje.	5
offset	4	Base del *bitfield*	N/A
bitfield	X	Secuencia de *bits* en donde cada *bit* representa si el *peer* tiene o no una pieza. Cada pieza se identifica relativo al *offset* (el *bit* N representa la pieza con *ID* = *offset* + *N*).	N/A

- El *peer B* aun no tiene la pieza a su disposición (un *peer* nunca va a pedir una pieza que ya posee, por lo que no tiene sentido enviar un mensaje de tipo HAVE a este).
- El ABI del *peer B* debe ser menor que el ID de la pieza recibida (un *peer* nunca va a pedir una pieza cuyo ID sea menor que su ABI).

Cuadro 5.6: Definición del mensaje HAVE. Este mensaje avisa la recepción de una nueva pieza. Es un mensaje de largo fijo de 10 *bytes*.

Parámetro	Tamaño(*bytes*)	Descripción	Valor
len	1	Largo del mensaje.	9
id	1	Identificador del mensaje.	4
piece_id	4	Identificador de la pieza recibida.	N/A
abi	4	ABI corriente del *peer* que envía este mensaje.	N/A

También ocurre que los *peers* desplazan sus ventanas deslizantes, obteniendo una nueva base. En estos casos, el cambio debe ser comunicado inmediatamente a todos los *peers* conectados, mediante el mensaje de tipo WINDOW UPDATE (ver los detalles en la Tabla 5.7). Los *peers* que reciben un mensaje de este tipo, deben responder con un mensaje de tipo BITFIELD, brindando la información acerca de las piezas que tienen en la nueva porción de la ventana del *peer* emisor del WINDOW UPDATE.

Cuadro 5.7: Definición del mensaje WINDOW UPDATE. Este mensaje es enviado por un *peer* cuando cambia su ventana deslizante. Este es un mensaje de largo fijo de 6 *bytes*, que debe ser contestado con un mensaje de BITFIELD.

Parámetro	Tamaño(*bytes*)	Descripción	Valor
len	1	Largo del mensaje.	5
id	1	Identificador del mensaje.	11
offset	4	Nueva base de la ventana.	N/A

Los *peers* deben periódicamente intercambiar mensajes entre ellos. Si luego de un determinado tiempo (normalmente cerca de 2 minutos) no ha habido ningún intercambio de mensajes

entre dos *peers*, estos deben intercambiar mensajes de tipo KEEP-ALIVE (ver detalles en la Tabla 5.8). De lo contrario, la conexión entre ellos se asumirá terminada.

Cuadro 5.8: Definición del mensaje KEEP-ALIVE. Este mensaje es usado para mantener activa una conexión y tiene un largo fijo de 1 *byte*.

Parámetro	Tamaño(bytes)	Descripción	Valor
len	1	Largo del mensaje.	0

5.2.4.2 Intercambio de Piezas

En la Figura 5.7 se presenta un resumen de los mensajes incluidos en el proceso de intercambio de piezas.

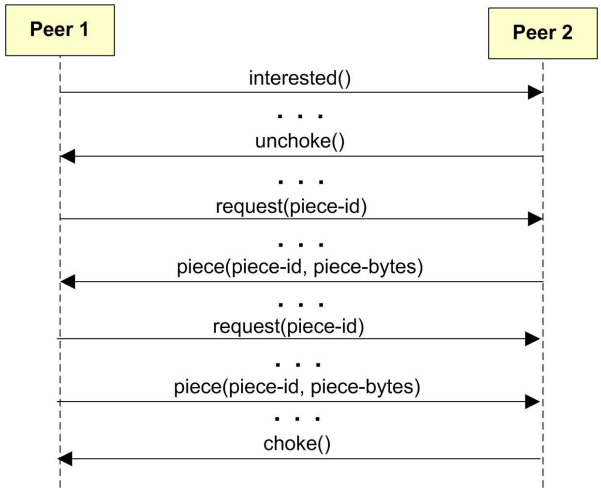


Figura 5.7: Resumen del proceso de intercambio de piezas. El *peer1* se encuentra informado de la disponibilidad en el *peer2* de piezas necesarias. Entonces el *peer1* informa su interés al *peer2* (envía un mensaje de INTERESTED). El *peer2* responde con un mensaje de UNCHOKE. Luego el *peer1* pide las piezas necesarias (mediante mensajes de REQUEST) y el *peer2* envía dichas piezas (mediante mensajes de PIECE). Finalmente cuando el *peer2* lo considera necesario, este inhabilita al *peer1*, mediante un mensaje de CHOKE.

Hasta ahora, se presentó la manera en que los *peers* se conectan entre sí e intercambian información acerca de sus respectivos contextos. A continuación, se presenta como es llevado a cabo el intercambio de piezas.

Como se comento anteriormente, los *peers* conocen las piezas tienen sus pares, por lo tanto se encuentran en condiciones de decidir si están o no interesados en realizar descargas de un cierto *peer*. En caso de estarlo, se debe enviar un mensaje de tipo INTERESTED (ver detalles del mensaje en la Tabla 5.9). En caso de haber estado interesado en un *peer* y de no estarlo

más, se debe enviar un mensaje de tipo NOT INTERESTED (ver detalles del mensaje en la Tabla 5.10).

Cuadro 5.9: Definición del mensaje INTERESTED. Este mensaje es usado por los *peers* con el fin de presentar interés en realizar descargas a otro *peer*. Su largo es fijo, de 2 *bytes*.

Parámetro	Tamaño(*bytes*)	Descripción	Valor
len	1	Largo del mensaje.	1
id	1	Identificador del mensaje.	2

Cuadro 5.10: Definición del mensaje NOT INTERESTED. Este mensaje es usado por los *peers* con el fin de presentar que no se encuentran más interesados en descargar piezas de otro *peer*. Su largo es fijo, de 2 *bytes*. Debe ser enviado únicamente cuando un mensaje de tipo INTERESTED fue previamente enviado del mismo emisor al mismo receptor.

Parámetro	Tamaño(*bytes*)	Descripción	Valor
len	1	Largo del mensaje.	1
id	1	Identificador del mensaje.	3

El protocolo establece que un *peer A* descarga una pieza de un *peer B*, solo si *A* está interesado en *B* y *B* habilita a *A* para realizar descargas. Los *peers* habilitan y deshabilitan a otros *peers* mediante los mensajes de UNCHOKE y CHOKE respectivamente (ver la definición de estos mensajes en la Tabla 5.11 y Tabla 5.12).

Cuadro 5.11: Definición del mensaje UNCHOKE. Si un *peer A* envía este mensaje al *peer B*, este último se encuentra habilitado para descargar piezas del primero. El mensaje tiene un largo fijo de de 2 *bytes*

Parámetro	Tamaño(*bytes*)	Descripción	Valor
len	1	Largo del mensaje.	1
id	1	Identificador del mensaje.	1

Cuando un *peer* es autorizado para descargar piezas de otro, porque envió un INTERESTED y recibió un UNCHOKE, este puede pedir partes de una pieza mediante un mensaje de REQUEST (ver detalles de este mensaje en la Tabla 5.13). Notar que el intercambio de piezas (al igual que en BitTorrent) es siempre realizado en base a pedidos de partes de estas (llamadas *slices*).

También es posible que un *peer* cancele un pedido de un *slice* mediante el mensaje CANCEL (ver la definición de este mensaje en la Tabla 5.14). Finalmente el *peer* que recibe un REQUEST envía la parte de la pieza solicitada mediante el mensaje PIECE (ver su descripción en la Tabla 5.15).

Si un *peer* es más rápido que un *seeder* (un *super-peer* o *broadcaster-peer*), es posible que este solicite piezas que aún no fueron generadas. En esta situación un mensaje de tipo DONT

Cuadro 5.12: Definición del mensaje CHOKE. Si un *peer A* envía este mensaje al *peer B*, este último no puede descargar piezas del primero. El mensaje tiene un largo fijo de de 2 *bytes*

Parámetro	Tamaño(*bytes*)	Descripción	Valor
len	1	Largo del mensaje.	1
id	1	Identificador del mensaje.	0

Cuadro 5.13: Definición del mensaje REQUEST. Este mensaje es usado por los *peers* para solicitar la descarga de un *slice* de una pieza. Su largo es fijo de 14 *bytes*.

Parámetro	Tamaño(*bytes*)	Descripción	Valor
len	1	Largo del mensaje.	13
id	1	Identificador del mensaje.	6
piece_id	4	Identificador de la pieza a solicitar.	N/A
begin	4	Comienzo del *slice* (en *bytes*, relativo al tamaño de la pieza).	N/A
length	4	Tamaño del *slice* (en *bytes*).	N/A

Cuadro 5.14: Definición del mensaje CANCEL. Este mensaje es usado por los *peers* con el fin de cancelar la solicitud de una pieza. Su largo es fijo de 14 *bytes*.

Parámetro	Tamaño(*bytes*)	Descripción	Valor
len	1	Largo del mensaje.	13
id	1	Identificador del mensaje.	8
piece_id	4	Identificador de la pieza a cancelar.	N/A
begin	4	Comienzo del *slice* (en *bytes*, relativo al tamaño de la pieza).	N/A
length	4	Tamaño del *slice* (en *bytes*).	N/A

Cuadro 5.15: Definición del mensaje PIECE. Este mensaje envía un *slice* de un pieza. Su largo es variable, dependiendo del tamaño del *slice* solicitado. Este es la respuesta al mensaje de REQUEST.

Parámetro	Tamaño(*bytes*)	Descripción	Valor
len	1	Largo del mensaje.	$9 + X$
id	1	Identificador del mensaje.	7
piece_id	4	Identificador de la pieza	N/A
begin	4	Comienzo del *slice* (en *bytes*, relativo al tamaño de la pieza).	N/A
block	X	Datos del *slice* (en binario).	N/A

HAVE debe ser enviado por parte del *seeder*. Ver la definición del mensaje DONT HAVE en la Tabla 5.16.

Cuadro 5.16: Definición del mensaje DONT HAVE. Este mensaje es usado por los *seeder*, como respuesta a solicitudes de piezas que aún no han sido generadas. Su largo es fijo de 6 *bytes*.

Parámetro	Tamaño(*bytes*)	Descripción	Valor
len	1	Largo del mensaje.	5
id	1	Identificador del mensaje.	10
piece_id	4	Identificador de la pieza.	N/A

5.2.5 Estrategias Aplicadas en el Protocolo

El protocolo GBTP, se especifica en base a la definición de los mensajes previamente detallados (significado y composición) y el comportamiento asociado a cada uno de ellos. Sin embargo, esto no es suficiente para desarrollar una aplicación que use dicho protocolo. En particular varias estrategias del algoritmo de comunicación son imprescindibles de detallar para lograr una buena *performance* de la aplicación.

A continuación, a modo de notas de implementación, se presentan diferentes estrategias que pueden ser aplicadas dentro del protocolo. La primera referente a la selección de *peers* para realizar transferencias de piezas, la segunda referente a la selección de piezas para descargar y la tercera referida a la construcción del *swarm* de un *peer* en el *tracker*.

5.2.5.1 Selección de *Peers*

La estrategia usada para seleccionar los *peers* a ser habilitados para realizar descargas, es la misma que en BitTorrent, llamada *tit-for-tat*. La misma consiste en el siguiente principio: un *peer* debe habilitar a los *peers* de los que realiza más descargas, un modo de recompensa para los contactos más generosos (los que comparten más piezas con el resto de la comunidad). También, con el fin de explorar la red y de integrar a los nuevos *peers*, se agrega una política de habilitación optimista, llamada *optimistic-unchocking*. Esta consiste en la selección de un *peer* al azar sin tener en cuenta que tan generoso ha sido en el pasado.

Más en detalle, el pseudo-código del algoritmo de selección de pares es el siguiente:

- El tiempo es dividido en ciclos de T segundos. Estos ciclos son numerados.
- Al finalizar el ciclo i se definen N *peers* a ser habilitados. Si $T * i$ es múltiplo de M, se aplica la política *tit-for-tat* para seleccionar los primeros $N - 1$ *peers* y el *peer* restante es seleccionado mediante *optimistic-unchocking*. Si $T * i$ no es múltiplo de M, se aplica *tit-for-tat* para seleccionar los N *peers*.

Los valores de T, N y M usuales son: $T = 10$ segundos, $N = 4$, $M = 3$. Por lo cual se evalúan los *peers* habilitados cada 10 segundos y cada 30 se realiza una habilitación optimista. Se espera entonces una justicia estadística, en donde los *peers* que más contribuyen sean los más favorecidos.

5.2.5.2 Selección de Piezas

Otra estrategia clave, refiere al proceso de selección de piezas: al momento de pedir una nueva pieza a un cierto *peer*, que pieza pedir.

En BitTorrent, la estrategia usada es llamada *rarest-first*, en donde los *peers* deben solicitar primero las piezas menos difundidas (las más "raras") en el sistema. El objetivo es lograr una distribución más uniforme de las piezas, reduciendo el riego de que el archivo quede incompleto. Para el caso de distribución de video en vivo, otra estrategia a considerar es la llamada *greedy* o *golosa*, en donde los *peers* siempre piden la siguiente pieza faltante a partir de la línea de reproducción.

Está estudiado que al aplicar *rarest-first* a la distribución de video en vivo, esta conlleva a grandes tiempos de *buffering* inicial [16](lo cual no es un comportamiento deseado). Por otro lado al aplicar la estrategia *greedy*, se obtienen muy buenos valores en el tiempo de *buffering* inicial pero una baja continuidad en la reproducción del video [16] (lo cual tampoco es deseado).

Más adelante, en el Capítulo 6 se ofrece un detallado análisis matemático sobre esta política. En él, se introduce un modelo de cooperación entre *peers*, sobre el que se define un concepto de estrategia de selección de piezas. En base a este concepto, y mediante la resolución de un problema de optimización combinatoria (aplicando ciertas metaheurísticas), se halla una estrategia de selección de piezas óptima para dicho modelo. Este estudio es un extracto del trabajo [109].

En esta sub-sección se presenta la estrategia implementada actualmente en el cliente GoalBit, llamada *exponencial*, que ofrece mejores propiedades que la estrategia óptima hallada en base al análisis matemático, tal como es presentado en la Sección 6.3.

La estrategia *exponencial*, intenta encontrar un compromiso entre el tiempo de *buffering* inicial y la continuidad en la reproducción. Por este motivo, se sigue un enfoque híbrido entre la estrategia *rarest-first* y la *greedy*.

Como se puede observar en la Figura 5.8, se definen 3 rangos dentro del *buffer* respecto a la línea de ejecución: el "urgente", el "próximo" y el "futuro". Se aplica el siguiente algoritmo para seleccionar una pieza:

- Si existe una pieza faltante en el rango urgente, se pide esa pieza. Si faltan más de una pieza en este rango, estas son pedidas según el orden de ejecución de las mismas.
- Si todas las piezas del rango urgente se encuentran descargadas, se piden piezas en base al resultado del muestreo de una variable aleatoria exponencial, la cual da mayor peso al rango próximo pero permite de vez en cuando, solicitar piezas pertenecientes al rango futuro.

5.2.5.3 Armado del *Swarm* de un *Peer*

La estrategia referente al armado de un *swarm* por parte del *tracker* resulta un factor muy relevante en la calidad percibida por un *peer*. El *tracker* puede simplemente seleccionar aleatoriamente los pares a ser retornados o puede aplicar alguna estrategia más inteligente. En

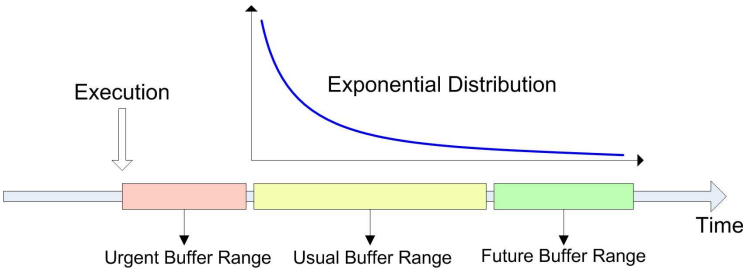


Figura 5.8: Estrategia de selección de piezas.

particular, el *tracker* se podría basar en la geografía de los *peers* y retornar pares que se encuentren relativamente "cerca" del par solicitante. De esta manera, no solo las conexiones entre *peers* tendrían una menor latencia, sino que se evitaría pasar por los cuellos de botella de la red.

Hoy en día existen varias técnicas a aplicar con el fin de implementar estas optimizaciones, algunas de ellas son presentadas a continuación.

P4P: *Proactive network Provider Participation for P2P*. Es una iniciativa del grupo *P4P Working Group*[4] (P4PWG), la cual intenta optimizar las conexiones P2P, mediante la cooperación entre los ISPs y las propias aplicaciones P2P. P4P funciona de la siguiente manera: el ISP dispone de un *iTracker*, el cual provee información acerca de la configuración y estado su red. Las aplicaciones P2P consultan al *iTracker* (usualmente mediante el *Tracker*), identificando que rutas los ISP prefieren y que conexiones deben ser evitadas. Luego, en base a esta información, el *Tracker* puede armar un *swarm*, enviando los *peers* que se encuentran más cercanos al solicitante (o desde el punto de vista del ISP, un *swarm* con los *peers* cuyas rutas son de bajo costo).

Este enfoque fue analizado en detalle por Darío Padula en su tesis de maestría, por más detalles ver [100]. Los principales resultados muestran que el ISP se beneficia con este enfoque, dado que la aplicación P2P hace un uso más limitado de los enlaces costosos, y al mismo tiempo el usuario es beneficiado con un mayor ancho de banda efectivo para la aplicación P2P.

ONO. La idea general de ONO, al igual que P4P, es sesgar la comunicación entre pares de redes P2P incentivando que se comuniquen aquellos que se encuentren geográficamente cerca. La diferencia entre ONO y P4P es que P4P utiliza a los ISPs para identificar las distancias entre pares, mientras que ONO lo realiza comparando el resultado de consultas DNS a CDNs bien conocidas, tales como *Google*[5] o *Akamai*[6] o *Limelight*[7]. En este método se asume que si dos clientes son enviados al mismo servidor de una CDN, entonces es altamente probable que

[4] http://www.openp4p.net/front/p4pwg
[5] http://www.google.com/
[6] http://www.akamai.com/
[7] http://uk.limelightnetworks.com/index.php

estos *peers* se encuentren muy próximos entre sí. Por más información acerca de esta técnica leer [30].

ALTO: Application-Layer Traffic Optimization. En Noviembre del 2008, la IETF [54] creó el grupo de trabajo ALTO[8], con el fin de optimizar el tráfico de Internet generado a nivel de aplicación. ALTO, plantea la creación de una completa especificación estándar, mediante la cual las aplicaciones (incluyendo las aplicaciones de tipo P2P) puedan encontrar la mejor vía para enviar y recibir datos (los *peers* más convenientes) según su red de base. La creación del protocolo ALTO se basa en contribuciones realizadas por el grupo P4P, por lo que ambas comparten el mismo enfoque. Actualmente el grupo de trabajo ALTO se encuentra activo.

Respecto al cliente GoalBit, en la actualidad se encuentra implementado el método ONO, pero no se encuentra activo, dado que aún no se encuentra integrado en el Tracker GoalBit. Con el fin de realizar pruebas se desarrolló un prototipo de *tracker* con soporte para ONO, el cual será próximamente migrado al Tracker GoalBit.

5.3 *GoalBit Packetized Stream*

La sección anterior presenta el protocolo GBTP. GBTP especifica como se deben distribuir las piezas de video, pero no especifica como estas deben ser generadas y encapsuladas. *GoalBit Packetized Stream* (GBPS), define el contenido y la estructura de cada una de las piezas a ser distribuidas mediante GBTP. Mantener separados GBTP y GBPS es una decisión de diseño, cuyo principal objetivo es el de proveer universalidad a los contenidos que pueden ser distribuidos mediante GBTP, incluyendo distintos *codecs* y *muxers*, pero también mensajes, aplicaciones u otro contenido necesario en el futuro.

Tal como se presentó en la Sección 2.1, usualmente las siguientes etapas se encuentran involucradas en la generación y codificación del contenido en vivo:

1. captura del contenido: el contenido (audio y video) es capturado desde una señal analógica.

2. codificación de las señales analógicas: en esta etapa, el audio y video son codificados independientemente, mediante alguna de las especificaciones conocidas (por ejemplo MPEG2/4-AVC, VC-1, ACC, VORBIS, MPGA, WMA, etc.). Como resultado se obtienen al menos 2 flujos elementales (uno para el video y otro para el audio).

3. multiplexación y sincronización de los flujos elementales: esta etapa consiste en la generación de un único flujo de *bytes* (llamado *stream* de transporte) que contenga tanto el audio, como el video y otra información necesaria para poder demultiplexar los flujos elementales. Algunas de las especificaciones (llamadas *muxers*) más conocidas son: MPEG-TS, MPEG-PS, MP4, ASF, OGG, etc.

4. Distribución del contenido: en esta etapa, el *stream* de transporte es encapsulado y distribuido sobre algún protocolo de transporte (RTP, HTTP, UDP, RTMP, GBTP, etc.).

[8] https://datatracker.ietf.org/wg/alto/charter/

La especificación GBPS se ubica entre la tercera y cuarta etapa: luego de que el contenido es multiplexado y antes de que este sea distribuido. Como se puede observar en la Figura 5.9, los paquetes GBPS son formados por una secuencia de paquetes del *muxer*.

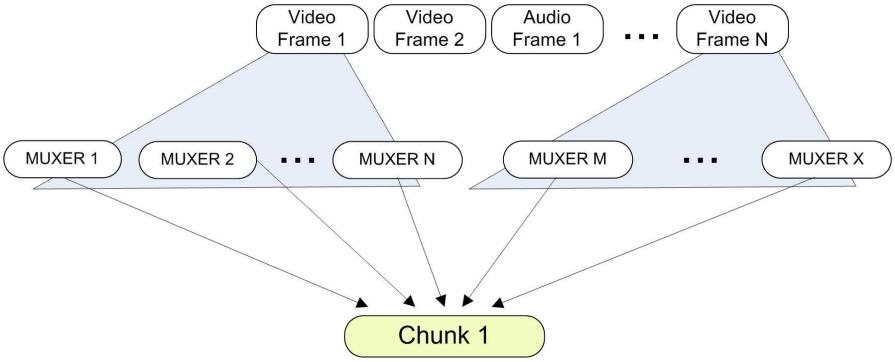


Figura 5.9: Contenido de las piezas en GBPS.

Los paquetes GBPS tienen un tamaño fijo, definido por el *broadcaster* al iniciar la transmisión de un contenido. Los paquetes del *muxer* pueden tener tamaño variable, dependiendo de la especificación de multiplexación usada. Por lo tanto, para llenar las piezas GBPS con paquetes del *muxer*, es necesario fragmentar ciertos paquetes del *muxer*. La Figura 5.10 muestra la estructura de las piezas GBPS, los campos i_data_start y i_data_end son usados para soportar la fragmentación de paquetes del *muxer*.

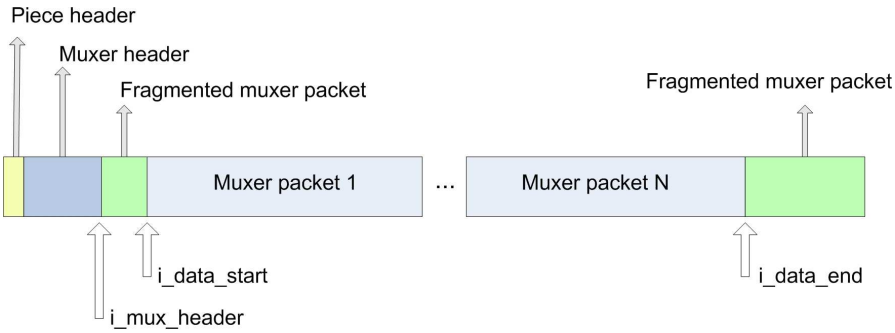


Figura 5.10: Estructura de una pieza GBPS.

Una característica importante de la especificación GBPS, es el hecho de que esta fue diseñada para ser independiente del *muxer* usado, lo que brinda la posibilidad de distribuir cualquier tipo de *stream*.

Como se presentó en la Subsección 2.1.2, si bien el objetivo de todas las especificaciones de multiplexación es la misma, existen grandes diferencias entre ellas. Mientras algunas son orientadas a la transmisión de video en video (MPEG-TS), otras son orientadas a escenarios de video bajo demanda (MPEG-PS, ASF, OGG, etc.). Las primeras embeben toda la información

necesaria para demultiplexar los flujos elementales dentro del propio *stream*, bajo intervalos regulares de tiempo. Las segundas ubican esta información en un paquete específico, llamado *header del muxer*, el cual debe ser el primer paquete enviado al reproductor de video (sin este paquete no es posible demultiplexar el *streaming*, siendo así imposible reproducir el contenido).

Con el fin de ser una especificación genérica y soportar cualquier tipo de *muxer*, GBPS debe considerar la existencia o no del *header del muxer*. Todas las piezas GBPS tienen su propio *header*. En la Tabla 5.17 se presenta el formato y contenido de este *header*. En el caso de especificaciones que requieran el envío de un *header del muxer*, este es insertado periódicamente a las piezas GBPS usando el campo i_mux_header (usualmente un *header del muxer* es insertado cada 10 piezas GBPS). Dado que el *header del muxer* puede cambiar durante la ejecución del *streaming*, existe la posibilidad de anunciarlo a toda la comunidad mediante el campo i_flag. Si el primer *bit* de este campo vale 1, el *header del muxer* debe ser enviado nuevamente al reproductor.

Cuadro 5.17: Formato del *header* de un paquete GBPS

Posición (*bytes*)	Parámetro	Descripción	
0..3	i_data_start	Inicio del primer paquete no fragmentado del *muxer*	
4..7	i_data_end	Final del último paquete no fragmentado del *muxer*	
8..11	i_mux_header	Tamaño del *header del muxer* (0 si no existe)	
12	i_flags	Banderas del protocolo:	
		Posición (*bits*)	Descripción
		0	1 significa que el *header* del *muxer* ha cambiado.
		1..7	Reservados (no en uso)

5.4 Implementación

La implementación de GoalBit se basa en 3 diferentes aplicaciones: Videolan Media Player (VLC) [131], Enhanced CTorrent [43] y OpenTracker [97]. VLC es un reproductor de video multiplataforma, que provee una gran cantidad de funcionalidades referidas a la captura, procesamiento y distribución de video. El licenciamiento de VLC es GPL [49]. El Enhanced CTorrent es un cliente BitTorrent eficiente de baja complejidad. El licenciamiento de Enhanced CTorrent es GPL [49]. Por su parte el OpenTracker, es una implementación de un *tracker* BitTorrent con licenciamiento Beerware [15]. Todas estas aplicaciones son gratuitas y escritas en C/C++.

Básicamente, sobre el VLC (del cual se usan la gran mayoría de sus funcionalidades), se desarrollaron una serie de módulos con el fin de implementar las especificaciones GBTP, GBPS y una interfaz de usuario propia. Como *software* de base para implementar el módulo de GBTP, se integró al VLC la aplicación Enhanced CTorrent. A su vez, con el fin de implementar el *tracker* definido por GBTP, se usó la aplicación OpenTracker.

A continuación, en la Subsección 5.4.1 se presentan detalles sobre los módulos desarrollados, en la Subsección 5.4.2 se presenta la política de *rebuffering* implementada y finalmente en

la Subsección 5.4.3 se presenta la política de control del *bitrate* del video.

5.4.1 Módulos Desarrollados en GoalBit

VLC es un *software* diseñado en base a una librería central, llamada *libvlc* y a un amplio conjunto de módulos. La *libvlc* brinda funcionalidades básicas y genéricas, tales como el control y la reproducción de un cierto contenido, el acceso a cierto tipo de módulos, etc. Por su parte, los módulos implementan funcionalidades específicas, tales como la codificación y decodificación de una cierta especificación de audio o de video (MPEG-2, MPEG-4, H.264, etc.), el acceso a un cierto tipo de datos (acceso a un video en disco, acceso a un streaming HTTP/UDP/MMS, lecturas desde un DVD), la multiplexación y demultiplexación de un cierto *muxer* (MPEG-TS, MPEG-PS, ASF, OGG), entre otras.

Con el fin de desarrollar GoalBit, se implementaron un conjunto de módulos, los cuales se integraron a los módulos ya existentes dentro del VLC. En la Figura 5.11, se presentan los principales módulos desarrollados en GoalBit.

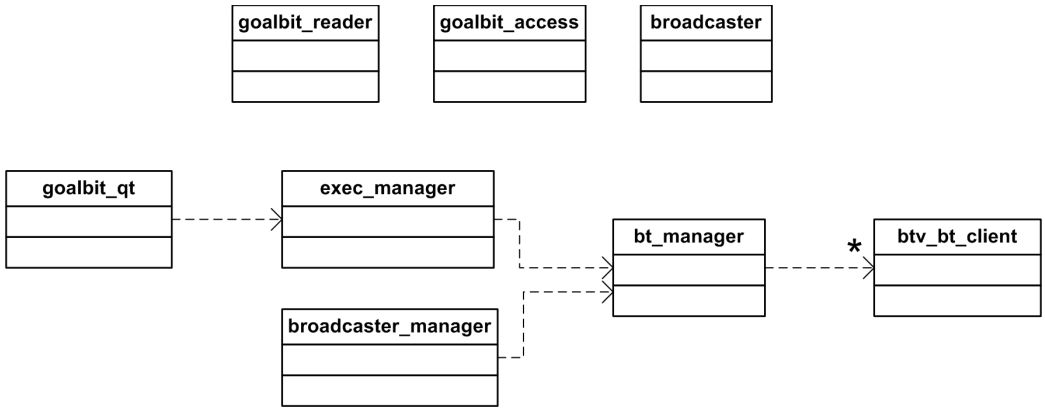


Figura 5.11: Principales módulos desarrollados en GoalBit sobre el VLC.

El módulo *goalbit_qt*, es la actual interfaz gráfica de usuario (GUI), esta fue implementada sobre el *framework* QT. La lógica de ejecución de un *streaming* GoalBit, se encuentra centralizada en el módulo *exec_manager*. La GUI interactúa muy a menudo con este módulo. El módulo *bt_manager*, provee funcionalidades sobre el manejo de los módulos de tipo *btv_bt_client* (por ejemplo funcionalidades como el alta, la baja y suspensión de estos módulos). El *btv_bt_client*, implementa las especificaciones GBTP y GBPS. Para su implementación se usó como base el Enhanced CTorrent. El módulo *broadcaster_manager*, es el responsable de iniciar y controlar la ejecución de un *broadcasting* (la distribución de contenido sobre GBTP). Este módulo es usado al ejecutar GoalBit desde la línea de comandos (no al ejecutar la funcionalidad *"Broadcast Yourself"* desde la GUI). El módulo *goalbit_reader*, es un módulo auxiliar, el cual provee el acceso e interpretación de los archivos GoalBit (con extensión .goalbit). Finalmente los módulos *goalbit_access* y *broadcaster*, tal como veremos a continuación, son módulos pertenecientes al flujo de ejecución del video. El primero es el encargado de generar las piezas GBPS y el segundo de pedir por estas a un módulo *btv_bt_client*.

Proceso de *Broadcasting* en GoalBit. A continuación presentamos la interacción entre módulos durante el proceso de *broadcasting*. Como se puede observar en la Figura 5.12, el principal controlador de la ejecución del proceso es el *broadcaster_manager*. En este caso suponemos que el *broadcasting* se está ejecutando desde línea de comandos y no desde la GUI. Se llevan a cabo las siguientes interacciones:

1. El *broadcaster_manager* crea una instancia del módulo *goalbit_reader*, con el fin de leer el archivo .goalbit a distribuir.

2. El *broadcaster_manager* crea una instancia del módulo *bt_manager*, quien a su vez, instancia uno o más módulos de tipo *btv_bt_client*, con el fin de distribuir el contenido.

3. El *broadcaster_manager* crea una instancia del *VLC::vlm_manager* (módulo propio del VLC), con el fin de iniciar el flujo de ejecución del video.

4. El *VLC::vlm_manager* mediante el uso de la *libvlc*, instancia todos los módulos necesarios para llevar a cabo el flujo de ejecución del video. Estos son: un módulo de acceso, un módulo de demultiplexación, módulos de decodificación y codificación (si se realiza algún tipo de transcodificación), un módulo de multiplexación y por último el *broadcaster*. A su vez, dentro del flujo de ejecución del video, se dan las siguientes interacciones:

   4.1 El módulo de acceso (alguno de los módulos de tipo “access” del VLC) obtiene el *streaming* de video desde alguna fuente (un archivo, un *streaming*, un DVD, etc.). Este le pasa dicho flujo al módulo de demultiplexación.

   4.2 El módulo de demultiplexación (alguno de los módulos de tipo “demux” del VLC) demultiplexa el *stream*, generando los flujos elementales (uno para el audio y otro para el video). En caso de transcodificar el *streaming*, estos son pasados a los módulos de decodificación y codificación (en otro caso, se pasan directo al módulo de multiplexación).

   4.3 Los módulos de decodificación (algunos de los módulos de tipo “decoder” del VLC) decodifican los flujos elementales por separado y se los pasan a los módulos de codificación (algunos de los módulos de tipo “encoder” del VLC), quienes a su vez vuelven a codificar el audio y video en otras especificaciones. El audio y video codificado es pasado al módulo de multiplexación.

   4.4 El módulo de multiplexación (alguno de los módulos de tipo “mux” del VLC), multiplexa los flujos elementales usando algún formato conocido (MPEG-TS, ASF, OGG, etc.). Finalmente este flujo multiplexado es pasado al módulo *broadcaster*.

   4.5 El módulo *broadcaster* toma el *streaming* multiplexado, genera piezas GBPS, las escribe en disco y le comunica a los módulos *btv_bt_client*, sobre la existencia de dichas piezas.

   4.6 Por último los módulos *btv_bt_client* distribuyen las piezas GBPS (obteniéndolas de disco) a la comunidad P2P mediante el protocolo GBTP.

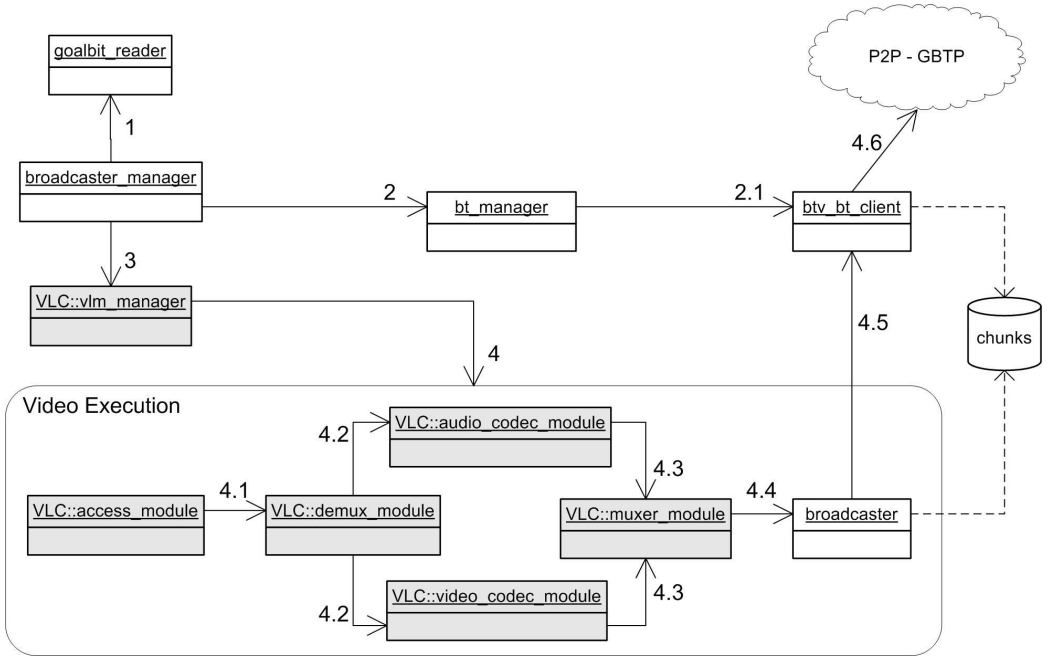


Figura 5.12: Interacción entre módulos llevada a cabo durante el proceso de *broadcasting*.

Proceso de Distribución de un *Streaming* GoalBit. En la Figura 5.13 se presenta la interacción entre módulos llevada a cabo durante el proceso de distribución, ejecutado por los *super-peers*. En este caso, el principal controlador de la ejecución es el módulo *bt_manager*. Dentro de este proceso se llevan a cabo las siguientes interacciones:

1. El *bt_manager* crea una instancia del módulo *goalbit_reader*, con el fin de leer el archivo .goalbit a distribuir.

2. El *bt_manager* crea una instancia del módulo *btv_bt_client*, quien lleva a cabo la distribución de las piezas GBPS sobre GBTP.

Notar que en este proceso, no se ejecuta ningún flujo de reproducción del video, dado que el único fin es la distribución del *streaming* GoalBit.

Proceso de Reproducción de un *Streaming* GoalBit. Como se puede observar en la Figura 5.14, en el caso de la reproducción de un *streaming* GoalBit, el principal responsable de la ejecución del proceso es el módulo de la interfaz gráfica: *goalbit_qt*. Dentro de este proceso, se llevan a cabo las siguientes interacciones:

1. La *goalbit_qt* crea una instancia del módulo *goalbit_reader*, con el fin de leer el archivo .goalbit a ejecutar.

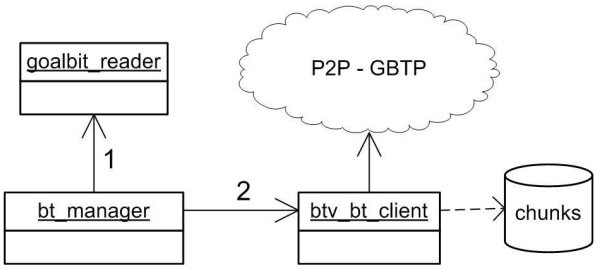


Figura 5.13: Interacción entre módulos llevada a cabo durante el proceso de distribución, ejecutado por los *super-peers*.

2. La *goalbit_qt* crea una instancia del módulo *exec_manager*, con el fin de comenzar la ejecución del *streaming*.

3. El *exec_manager* crea una instancia del módulo *bt_manager*, quien a su vez, instancia un módulo *btv_bt_client*, con el fin de obtener el contenido desde la red P2P. El módulo *btv_bt_client* además de llevar adelante el intercambio de piezas mediante GBTP, es el encargado de regular la ejecución del *streaming* (encargado de controlar el estado del *buffer*, determinando cuando se debe *bufferear* y cuando se debe comenzar con la reproducción del *streaming*).

4. El *exec_manager* crea una instancia del *VLC::playlist_manager* (módulo propio del VLC), con el fin de iniciar el flujo de ejecución del video.

5. El *VLC::playlist_manager* mediante el uso de la *libvlc*, instancia todos los módulos necesarios para llevar a cabo el flujo de ejecución del video. Estos son: el módulo *goalbit_access*, un módulo de demultiplexación, los módulos de decodificación y por último los módulos de *display*. Dentro del flujo de ejecución del video existen las siguientes interacciones:

   5.1 El *goalbit_access*, le pide piezas al *btv_bt_client*. Cuando el *btv_bt_client* se encuentra en estado de ejecución, este le retorna paquetes del *muxer* al *goalbit_access* (notar que el módulo *btv_bt_client* también es el responsable de desencapsular los paquetes del *muxer* desde dentro de las piezas GBPS).

   5.2 El *goalbit_access* le pasa el contenido adquirido al módulo de demultiplexación.

   5.3 El módulo de demultiplexación (alguno de los módulos de tipo “demux” del VLC) demultiplexa el *stream*, generando los flujos elementales (uno para el audio y otro para el video).

   5.4 Los módulos de decodificación (algunos de los módulos de tipo “decoder” del VLC) decodifican los flujos elementales por separado y se los pasan a los módulos de *display*.

   5.5 Los módulos de *display*, presentan el contenido (el audio y el video decodificado) al usuario.

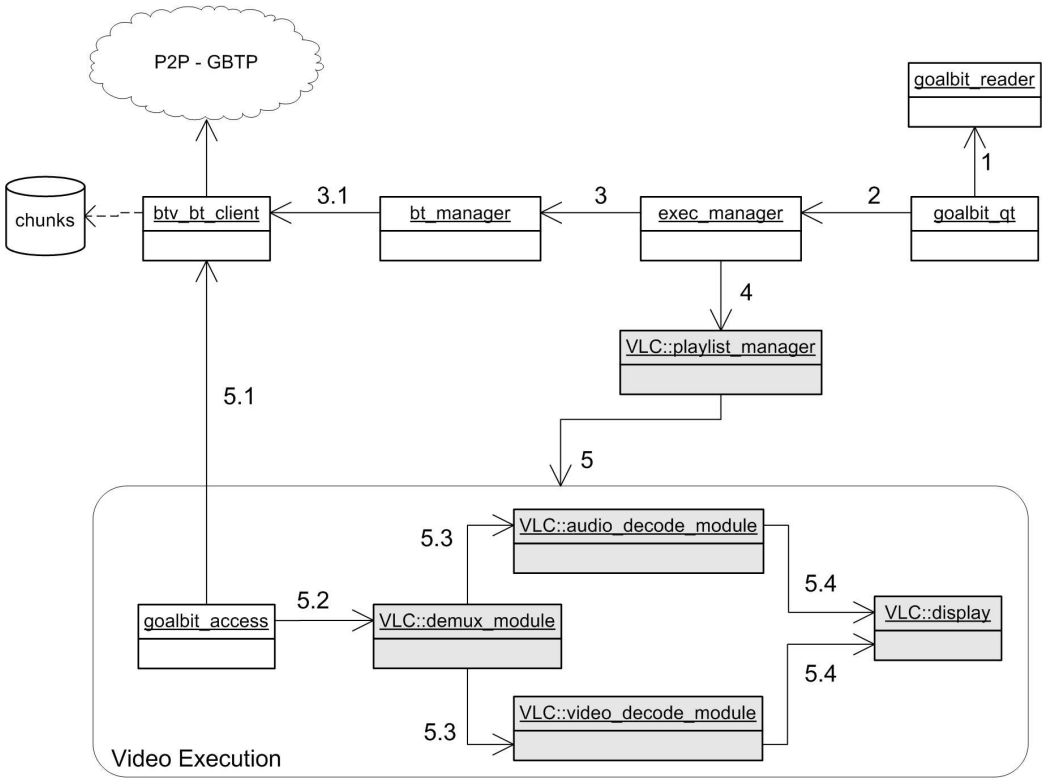


Figura 5.14: Interacción entre módulos llevada a cabo durante la reproducción de un *streaming* GoalBit.

A continuación se presentan algunas de las políticas aplicadas en la implementación del GoalBit Player.

5.4.2 Política de *Rebuffering*

Como vimos anteriormente, el módulo *btv_bt_client* es el encargado de controlar la ejecución de un *streaming* GoalBit. Este módulo se puede encontrar en dos estados diferentes: "ejecutando" o en *"buffering"*. Si el *btv_bt_client* se encuentra en estado *"buffering"* y el *goalbit_access* le solicita piezas, el primero le retorna piezas vacías (dejando la ejecución suspendida). Si el *btv_bt_client* se encuentra en estado "ejecutando", este le retornará todos los paquetes del *muxer* que se encuentren contenidos en la próxima pieza a ejecutar. Si al buscar la próxima pieza a consumir, esta no se encuentra en el *buffer* (lo cual llamamos una pérdida), el *btv_bt_client* retorna piezas vacías hasta que haya transcurrido el tiempo estimado de procesamiento y presentación de la pieza faltante (esto con el fin de no desincronizar el *streaming* de video).

Al comenzar la ejecución, el *btv_bt_client* se encuentra en estado *"buffering"*, cambiando a estado "ejecutando" al completar el *buffer* inicial (el cual normalmente suele ser de 16 piezas a partir de la línea de ejecución definida por el *tracker*). La permanencia del módulo *btv_bt_client* en estado "ejecutando", se encuentra definida en base al siguiente algoritmo:

- Se definen 3 secuencias: X_n, C_n y L_n en donde $X_0 = C_0 = L_0 = 0$. C_n representa el total de piezas consumidas desde la última pérdida. L_n representa el total de piezas perdidas desde la última pieza consumida. X_n es un balance entre C_n y L_n. A su vez se definen 3 constantes: α, β y γ. α es el coeficiente asociado al consumo de piezas, β es el coeficiente asociado a la pérdida de piezas y γ el límite para ejecutar el cambio de estado en el módulo *btv_bt_client*.
- Si al momento de consumir la pieza i, esta se encuentra en el *buffer* del cliente:
  - Si $X_{i-1} == 0$, entonces $X_i = C_i = L_i = 0$.
  - Si $X_{i-1} > 0$, entonces $X_i = X_{i-1} - (1 + \alpha C_{i-1})$, $C_i = C_{i-1} + 1$ y $L_i = 0$.
- Si al momento de consumir la pieza i, esta no se encuentra en el *buffer* del cliente, entonces $X_i = X_{i-1} + (1 + \beta L_{i-1})$, $L_i = L_{i-1} + 1$ y $C_i = 0$.
- Finalmente si se cumple que $X_n > \gamma$, entonces se cambia el módulo *btv_bt_client* a estado de *"buffering"*.

Notar que normalmente se debería cumplir que $\beta > \alpha$ con el fin de castigar más a las pérdidas de piezas que al consumo de las mismas. En nuestra implementación, los valores definidos para estas constantes son: $\alpha = 0{,}1$, $\beta = 0{,}3$ y $\gamma = 3{,}5$.

5.4.3 Política de Control del *Bitrate* del Video

Normalmente ocurre que el ancho de banda de descarga disponible en los usuarios es mayor que el *bitrate* del video. Motivo por el cual, los clientes GoalBit descargan piezas a una tasa mayor que la del video. En este caso, al alcanzar la línea de generación de piezas del *broadcaster*, los

clientes comienzan a recibir una gran cantidad de mensajes de tipo DONT HAVE, dado que estos solicitan piezas que aún no han sido generadas. Para no sobrecargar la red, y en particular a los *seeders*, con pedidos de piezas futuras, se define una política de control del *bitrate* del video, la cual se describe en el siguiente pseudocódigo:

- Se definen 2 constantes: $DONT\_HAVE\_MAX$ y $DONT\_HAVE\_TIMEOUT$. A cada *peer* durante la ejecución se le asocia un *buffer* de $DONT\_HAVE\_MAX$ posiciones, en donde se almacenan un ID de pieza y una fecha de última modificación.
- Al momento de recibir un mensaje de tipo DONT HAVE para una cierta pieza desde un *peer* A, se agrega el ID de pieza y la fecha de recibida en el *buffer* asociado con el *peer* A. Si dicha pieza ya existía dentro el *buffer*, entonces únicamente se actualiza la fecha de última modificación.
- Antes de solicitar una pieza a un cierto *peer* se actualiza el *buffer* asociado a este, eliminando las piezas cuya fecha de última modificación sea menor que $DONT\_HAVE\_TIMEOUT$ milisegundos atrás. Si luego de esta actualización, el *buffer* se encuentra lleno (lo que significa que al *peer* en cuestión le estamos pidiendo varias piezas aún no generadas) se suspende la solicitud de piezas a este *peer* por $DONT\_HAVE\_TIMEOUT$ milisegundos.

En nuestra implementación, los valores definidos para las constantes son: $DONT\_HAVE\_MAX = 5$ y $DONT\_HAVE\_TIMEOUT = 1500$.

Capítulo 6

Análisis de la Estrategia de Selección de Piezas

Este capítulo presenta un profundo análisis de posibles estrategias de selección de piezas a aplicar en el protocolo, la cual, tanto a nivel de pruebas empíricas, como a nivel de modelo matemático, se presenta como factor de gran relevancia en la continuidad y latencia del *streaming*, y por lo tanto en la calidad percibida por los usuarios finales.

En base a trabajos previos sobre PSQA [90], es bien conocido que la pérdida de información de video es el factor de mayor impacto en la calidad percibida por los usuarios finales. Esta pérdida puede ser contabilizada a diferentes niveles, como por ejemplo a nivel de *frames* de video [110], a nivel de paquetes de red [108] o como en nuestro caso a nivel de paquetes P2P (los cuales, como se presentó en 5.3, encapsulan audio y video multiplexado).

La latencia inicial o "tiempo de *buffering*" se define como el tiempo transcurrido desde que un *peer* se conecta a la red hasta que comienza a reproducir el video deseado. Claramente éste es un parámetro influyente en la calidad de experiencia: no es lo mismo comenzar a ver un contenido de forma inmediata, que tener que esperar unos minutos antes de comenzar a hacerlo. A su vez, la latencia impacta directamente en el atraso global de la señal: cuanto más tardamos en comenzar a ver una señal, más atrasados estamos respecto a la señal en vivo (por ejemplo, si la transmisión es de un partido de fútbol, nuestros vecinos pueden gritar el gol varios segundos antes de lo que nosotros lo vamos a ver).

En base a lo anterior, es razonable y frecuente estudiar el compromiso entre el tiempo de latencia inicial y la continuidad. En nuestro equipo [1] este fue el trabajo de tesis de maestría de Pablo Romero [109]. En este documento incluimos el modelo presentado en dicha tesis y extendemos los resultados mostrando el desempeño de otras políticas, como la exponencial (ver Subsección 5.2.5.2). Los resultados de este capítulo fueron parcialmente presentados en [col-latincom09] y [ru-icumt09] (ver Sección 1.4).

En la Sección 6.1 se presenta un modelo matemático de cooperación entre pares, junto con una definición formal de continuidad y latencia. En la Sección 6.2 se introduce formalmente un

[1] El equipo de investigación está compuesto por los siguientes miembros: María Elisa Bertinat, Daniel de Vera, Darío Padula, Franco Robledo, Pablo Rodríguez Bocca, Pablo Gabriel Romero y Gerardo Rubino

concepto de optimalidad de una estrategia de selección de piezas, sobre el cual se formaliza un problema de búsqueda de estrategias de permutación a partir de un problema de optimización combinatoria (COP), siendo resuelto en base a la aplicación de meteheurísticas como búsqueda local y colonia de hormigas [39]. Finalmente la Sección 6.3 contrasta el modelo matemático considerado con la realidad, mediante la realización de emulaciones con GoalBit, con el fin de evaluar las diferentes estrategias de selección de piezas.

6.1 Definición de un Modelo Matemático de Cooperación

El modelo a presentar es una generalización del modelo introducido en [149]. La escritura de esta sección se basa en [109].

6.1.1 El Modelo Original

Se considera una red en donde un servidor posee el contenido de video original (el *broadcaster*), y M pares que desean obtener y reproducir el mismo. A los efectos de utilizar eficientemente el ancho de banda y la cooperación de los pares, el *broadcaster* corta el video en piezas, y en cada *slot* de tiempo elige un *peer* X al azar dentro de la red, a quien le envía la última pieza generada. Únicamente el *peer* X va a ser el beneficiado de obtener esa pieza sin necesidad de realizar consultas a sus otros pares. Notar que la duración de los *time slot* es, y debe ser, equivalente al tiempo de generación de las piezas por parte del *broadcaster*, y al tiempo de reproducción de las mismas por parte de los *peers*. Todos los *peers* poseen igual capacidad de almacenamiento de piezas, denotado con N (tamaño del *buffer* de los *peers*). Se supone que todos los nodos de la red intercambian piezas con la misma estrategia de cooperación. A su vez, un par únicamente realiza una consulta por *time slot*, pudiendo o no obtener una pieza.

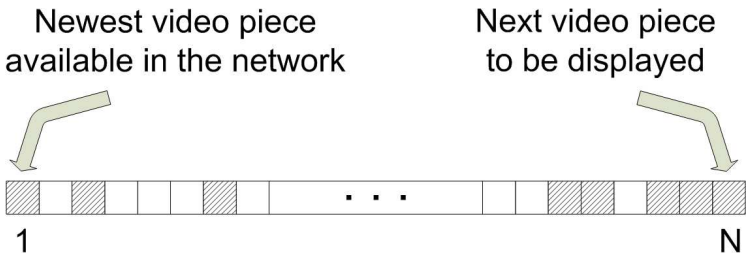


Figura 6.1: Modelo del *buffer* en cada *peer*. La posición 1 representa la pieza de video más nueva en la red y la posición N la siguiente pieza a ser consumida.

Todos los pares están sincronizados entre sí. Estos consumen al mismo tiempo la pieza más antigua dentro de sus *buffers* (ubicada en la posición N, ver Figura 6.1). En caso de no disponer dicha pieza, el usuario percibirá un defecto o error en la imagen.

El intercambio de piezas funciona de la siguiente manera : el par h escoge al azar a otro par k perteneciente a la red, y solicita una pieza que no dispone. Si k tampoco posee esta

pieza, h puede consultar por otra. Este proceso se repite hasta que h consigue una pieza y la consulta resulta exitosa, o bien no consigue ninguna pieza, refiriéndonos en este último caso a una consulta fracasada. Recordar que por *time slot*, únicamente es posible contactarse con un *peer*, por lo que en caso de consultas fracasadas, no es posible seleccionar otro par de la red y realizar una nueva solicitud. Llamemos p_i a la probabilidad de que un *peer* tenga la pieza correcta en la $i-$ésima posición del *buffer*, $B(i)$. Asumiendo estado estacionario el vector de p_i's es idéntico en todos los pares de la red por simetría.

Notar que se va a encontrar una pieza en $B(j)$ en el *slot* de tiempo t_i si se cumple alguna de las siguientes 2 situaciones: se tenía dicha pieza en el *time slot* anterior t_{i-1} (en la posición $B(j-1)$, dado que los índices de las piezas se incrementan junto con los *time slots*) o durante la consulta ocurrida en t_{i-1}, se obtuvo dicha pieza. Si definimos u_i como la probabilidad de adquisición de la pieza en la posición i del *buffer* mediante una consulta, podemos formalizar la situación anterior mediante la siguiente expresión (recordar que la pieza de la posición 1 del *buffer* únicamente puede ser obtenida desde el *broadcaster*):

$$p_1 = \frac{1}{M}$$
$$p_{i+1} = p_i + u_i, \forall i = 1 \ldots N-1.$$

A continuación se plantea una expresión analítica simple, la cual nos permitirá hallar el vector de probabilidades de ocupación p_i para cada estrategia de selección. Llamemos h al par solicitante y k al solicitado. Consideremos los siguientes 3 eventos:

(1) $Q(h,i)$: el par h quiere la pieza i, dado que esta no se encuentra en su *buffer* ($B(i)$ está vacío).

(2) $T(k,i)$: el par k tiene la pieza i.

(3) $S(k,h,i)$: el par k dispone de la pieza i, y aplicando una estrategia de selección, la pieza i es solicitada por el par h.

A partir de estos eventos es posible expresar u_i de la siguiente manera:

$$u_i = P(Q(h,i) \cap T(k,i) \cap S(k,h,i)) =$$
$$P(Q(h,i))P(T(k,i)/Q(h,i))P(S(k,h,i)/Q(h,i) \cap T(k,i)) \quad (6.1)$$

Analizando cada uno de estos factores:

- La probabilidad de que el par h quiera la pieza i del *buffer*, es sencillamente la probabilidad de que este no tenga dicha pieza en su poder: $P(Q(h,i)) = 1 - p_i$.

- Podemos asumir que la probabilidad de que el par k disponga de la pieza i no se ve influenciada por la necesidad del par solicitante h de esa misma pieza. Por lo que: $P(T(k,i)/Q(h,i)) \approx P(T(k,i)) = p_i$.

- Las piezas se distribuyen independientemente en la red, por lo que podemos asumir que la distribución de probabilidades de la pieza en la posición i no se ve fuertemente influenciada por el conocimiento del estado de otras posiciones. Entonces:

$$s_i = P(S(k,h,i)/Q(h,i) \cap T(k,i)) \approx P(S(k,h,i)) \tag{6.2}$$

A partir de las expresiones anteriormente definidas, y sustituyendo en (6.1), se obtiene una expresión, que vincula la probabilidad de ocupación con la función de estrategia de selección:

$$p_1 = 1/M; \tag{6.3}$$
$$p_{i+1} = p_i + (1-p_i)p_i s_i,\ i = 1,\ldots,N-1 \tag{6.4}$$

Notar que s_i es la probabilidad de que un par adquiera la pieza de la posición i, teniendo en cuenta que el par solicitante no posee la pieza i y el solicitado sí. En otras palabras, esta representa la estrategia de selección de piezas, en donde se establece a priori, el orden en que las piezas serán solicitadas.

A su vez, podemos definir formalmente la continuidad y la latencia inicial de la siguiente manera:

Definición 6.1.1 *La continuidad de reproducción es medida mediante $C = p_N$, y esta representa la probabilidad de tener la pieza inmediata a ser reproducida.*

Definición 6.1.2 *La latencia de inicio del video se mide con:*

$$L = \sum_{i=1}^{N} p_i.$$

Y representa el tiempo (medido en slots) que se requiere para alcanzar el estado estacionario, partiendo de un buffer vacío.

6.1.2 Estrategias Clásicas de Selección de Piezas

Antes de extender aún más este modelo, y con el fin de lograr una mejor comprensión del significado de s_i, se presentan dos clásicas estrategias de selección de piezas: *rarest-first* y *greedy*.

La estrategia *rarest-first*, muy popular en los sistemas P2P para la descarga de archivos y usaba por BitTorrent desde sus inicios, establece que se deben solicitar primero las piezas menos distribuidas en el conjunto de pares. En nuestro modelo, esta se corresponde con seleccionar las piezas más lejos de la línea de reproducción. Dichas piezas son las más raras, debido a la monotonía de p (ver la demostración de la monotonía de p en [109]).

$$s_i = (1 - \frac{1}{M}) \prod_{j=1}^{i-1} (p_j + (1-p_j)^2) \tag{6.5}$$

Esta expresión se puede interpretar de la siguiente manera: la pieza i será solicitada únicamente si se cumplen las siguientes 3 condiciones:

- El *broadcaster* no ha seleccionado al *peer* solicitante para enviarle la pieza 1 ($1 - \frac{1}{M}$).
- El *peer* solicitante no posee la pieza i y el consultado sí.
- Para cada una de las piezas previas a i ($\prod_{j=1}^{i-1}$), se cumple que el par consultante o bien ya tenía dicha pieza (p_j), o bien no la tenía pero el par consultado tampoco ($(1-p_j)^2$).

Por otra parte, la estrategia *greedy*, establece que se debe consultar primero por las piezas más cercanas a la reproducción (ver Figura 6.2). Ésta es representada en nuestro modelo por la siguiente expresión (muy similar a la anterior salvo el cambio en el orden de solicitud):

$$s_i = (1 - \frac{1}{M}) \prod_{j=i+1}^{N-1} (p_j + (1-p_j)^2) \tag{6.6}$$

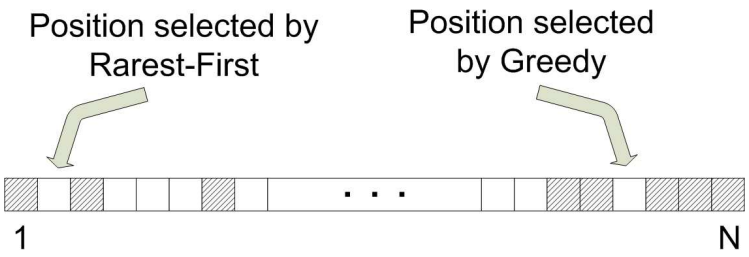


Figura 6.2: La estrategia *rarest-first* establece que se deben solicitar primero las piezas menos distribuidas en el conjunto de pares. Por otra parte, la estrategia *greedy*, establece que se debe consultar primero por las piezas más cercanas a la reproducción.

Es bien sabido que *greedy* logra bajas latencias, pero presenta mayores problemas de continuidad que *rarest-first* [139, 149].

6.1.3 Generalización del Modelo Original

Con el fin de extender y/o generalizar el modelo anteriormente presentado, en [109] se presenta una manera natural de lograr una diversidad de opciones de estrategias, considerando una permutación arbitraria de tamaño $N-1$, y rigiéndose por la misma para decidir el orden de consultas. Denotaremos una permutación π formalmente como una aplicación biyectiva de los primeros $N-1$ enteros positivos (índices del *buffer*):

$$\pi : \{1, \ldots, N-1\} \rightarrow \{1, \ldots, N-1\},$$
$$\forall i \neq j, \pi(i) \neq \pi(j),$$

donde N es el tamaño de *buffer* y $\pi(j)$ la evaluación de la permutación en j.

En base a este concepto se sucede la siguiente definición:

Definición 6.1.3 *La Familia de estrategias de permutación es el conjunto de estrategias cuyos órdenes de solicitud se rigen por una permutación π arbitraria. Para cada permutación π, la estrategia correspondiente admite la siguiente expresión:*

$$s_{\pi(i)} = (1 - \frac{1}{M}) \prod_{j=1}^{i-1} (p_{\pi(j)} + (1 - p_{\pi(j)})^2) \tag{6.7}$$

La interpretación se esta expresión es muy similar a la de las estrategias clásicas (6.5 y 6.6), exceptuando que el orden de consulta es distinto. En esta, primero se revisa la posición $\pi(1)$ del *buffer*. Si el *peer* posee la pieza en dicha posición (con probabilidad $p_{\pi(i)}$) o bien si este no la posee pero el par consultado tampoco (evento con probabilidad $(1 - p_{\pi(1)})^2$), se sigue mediante el chequeo de la posición $\pi(2)$ del *buffer*, y así sucesivamente.

Se puede observar que tanto la estrategia *greedy* como la *rarest-first*, tienen sus correspondientes permutaciones en las $(N-1)!$ estrategias posibles. Para el caso de *rarest-first*, simplemente basta con definir π como la permutación identidad ($\pi(i) = i, \quad \forall i = 1, \ldots, N-1$), mientras que para el caso de *greedy* $\pi(i) = N - i, \quad \forall i = 1, \ldots, N-1$.

6.2 Una Solución Óptima para la Estrategia de Seleeción de Piezas

En esta sección se introduce formalmente el concepto de optimalidad de una estrategia de selección de piezas. En base a dicha definición [109] formaliza un problema de búsqueda de estrategias de permutación a partir de un problema de optimización combinatoria (COP), el cual es resuelto mediante la aplicación de meteheurísticas como búsqueda local y colonia de hormigas. En esta sección se presenta un resumen de dicho trabajo.

6.2.1 Optimalidad de una Estrategia de Selección

A continuación se presenta una medida de optimalidad inspirada en el mejor vector de probabilidades de ocupación p_i^*, el cual debe tomar valores pequeños en índices de *buffer* menores que N (a los efectos de tener baja latencia) y al mismo tiempo el valor de p_N lo más alto posible (con el fin de presentar una alta continuidad en la reproducción).

Para ello, consideremos una permutación arbitraria π de los elementos $\{1, \ldots, N-1\}$. Toda consulta se realiza en una cantidad finita de pasos, donde cada paso consiste en el chequeo de una posición del *buffer*. En particular, toda consulta exitosa logra obtener una pieza en no más de $N-1$ pasos. Sea X_π la variable aleatoria que cuenta la extensión (cantidad de revisiones del *buffer*) de una consulta exitosa, utilizando la estrategia de permutación π. En base a estas definiciones, se expresa la siguiente proposición:

Proposición 6.2.1 *El número esperado de pasos necesarios en una consulta exitosa mediante la estrategia s_π es:*

$$E(X_\pi) = \frac{M}{M-1} \sum_{i=1}^{N-1} i(p_{\pi(i)+1} - p_{\pi(i)}). \tag{6.8}$$

Ver demostración en [109]. El valor medio de la extensión de la consulta es una combinación lineal de la sumatoria de saltos en todos los índices consecutivos de la probabilidad de ocupación.

Una pregunta pertinente aquí es la siguiente: ¿se desea que las consultas sean extensas o cortas? Para responder debemos recordar la hipótesis de que todo el proceso de solicitud de una pieza tiene una duración inferior al *time slot*, por lo que la extensión no preocupa en cuanto a tiempos refiere. Más aún, una consulta extensa significa que tanto el par consultado como el consultante poseen una buena cantidad de piezas en sus *buffers*. Lo que indica que es deseable un valor esperado de largo de consultas mayor. Observar que en 6.8, la esperanza se incrementa a medida que aumenta la continuidad p_N, pues el índice que hace que $\pi_i = N - 1$ es tal que $p_{\pi_i+1} = p_N$, y nunca es cancelado. Lo cual significa que, para toda estrategia, al aumentar la esperanza aumenta la continuidad. Sin embargo, la latencia puede aumentar o disminuir al aumentar la esperanza dependiendo de la estrategia.

En base a la Proposición 6.2.1 y haciendo cuentas (ver detalles en [109]), se tiene que la esperanza de la variable X_π en *rarest-first* se reduce a:

$$E(X_\pi) = \frac{M}{M-1}(Np_N - L) \tag{6.9}$$

O también:

$$E(X_\pi) = \frac{M}{M-1}\sum_{i=1}^{N}(p_N - p_i). \tag{6.10}$$

Para el caso de *greedy*, esta se corresponde con:

$$E(X_\pi) = \frac{M}{M-1}(L - Np_1) \tag{6.11}$$

O también:

$$E(X_\pi) = \frac{M}{M-1}\sum_{i=1}^{N}(p_i - p_1) \tag{6.12}$$

Es posible observar que en el caso de *rarest-first*, la igualdad (6.10) establece que el número esperado de pasos de una consulta es proporcional a la suma global de las distancias entre todos los índices del vector p y su continuidad. Por lo tanto, si deseamos incrementar la optimalidad de *rarest-first* (es decir, incrementar el valor esperado de X_π), debemos incrementar el valor del término $(p_N - p_i)$, lo que en otras palabras significa que se debe mejorar la latencia de *rarest-first* (el cual, es justamente el factor débil de dicha estrategia).

Por otra parte, en *greedy* la extensión esperada de una consulta es proporcional a la suma global de diferencias entre el vector de probabilidad de ocupación p_i y el valor base p_1 (6.12).

En este caso, al intentar mejorar la optimalidad de *greedy*, es necesario incrementar el valor del término $(p_i - p_1)$. Esto requiere incrementar el valor de p_i, lo cual, dada la monotonía de p, redunda en un aumento de su continuidad (notar que nuevamente nuestro concepto de optimalidad lleva a mejorar el factor débil la estrategia).

6.2.2 Formalización del Problema a Optimizar

En esta subsección, y basado en el concepto de optimalidad definido anteriormente, se formaliza el problema de la elección de una estrategia óptima, mediante un Problema de Optimización Combinatoria (COP, por sus siglas en ingles).

Un COP es un problema de maximización (o minimización) de una función sobre un dominio discreto y con ciertas restricciones. Más formalmente:

Definición 6.2.2 *Sea $\mathcal{D}$ es un espacio de soluciones discreto, $f : \mathcal{D} \to \mathcal{R}^+$ una función y $\{r_1, \ldots, r_n\}$ un conjunto de restricciones que deben cumplir las variables de $\mathcal{D}$, que determinan una región factible Ω. Un COP (Ω, f) consiste en determinar el máximo global (o bien el mínimo global) de la función f en el espacio de soluciones factibles Ω.*

Usualmente, es muy complejo (sino imposible) hallar una solución analítica o algoritmo polinomial que determine la solución exacta de un COP. Es frecuente entonces el diseño de algoritmos en base a la aproximación, buscando comúnmente un factor de garantía de proximidad de la solución hallada respecto de la óptima. Otro enfoque muy común, y no excluyente con el primero, es la aplicación de metaheurísticas [21]. Básicamente, una metaheurística es una herramienta de búsqueda aplicable a cualquier problema. La intensificación y diversificación son dos propiedades inherentes a toda metaheurística. La diversificación consiste en lograr un conocimiento del espacio de soluciones mediante la visita a las mismas, o aprendizaje de su estructura. Mediante la intensificación se explota este conocimiento. Esta consiste en concebir un subconjunto del subespacio destacado o propiedades a partir del proceso de diversificación, e intensificar la búsqueda sobre este subconjunto o explotar las propiedades del espacio de soluciones.

Tal como se presenta en la Subsección 6.2.1, sabemos que es deseable que la extensión de una consulta sea lo mayor posible. En base a este concepto, presentamos la siguiente definición:

Definición 6.2.3 *El siguiente Problema de Optimización Combinatoria tiene una permutación como variable de decisión, y procura maximizar la extensión esperada de una consulta con la permutación a elegir, sujeta a las restricciones del problema y definición de estrategia de selección. En términos matemáticos:*

$$max_{\pi} E(X_{\pi}) \tag{6.13}$$
$$s.a.$$
$$p_1 = \frac{1}{M} \tag{6.14}$$
$$s_{\pi(1)} = 1 - \frac{1}{M} \tag{6.15}$$
$$p_{i+1} = p_i + (1 - p_i)p_i s_i \tag{6.16}$$
$$s_{\pi(i+1)} = s_{\pi}(p_{\pi(i)} + (1 - p_{\pi(i)})^2) \tag{6.17}$$
$$S = \{1, \ldots, N-1\}$$
$$\pi : S \to S, \pi(i) \neq \pi(j), \quad \forall i \neq j$$

En la próxima subsección, se abordará este problema, buscando soluciones en base a metaheurísticas, logrando así un compromiso entre el esfuerzo computacional y la calidad de los resultados.

6.2.3 Resolución del Problema mediante Búsqueda Local

En primer instancia, se intentará resolver el problema anteriormente planteado aplicando la metaheurística de búsqueda local. A continuación se presentan algunos conceptos básicos sobre esta metaheurística, para luego presentar nuestra solución propuesta, junto con algunos resultados teóricos obtenidos.

6.2.3.1 Conceptos Básicos

La búsqueda local se caracteriza por el requisito de encontrar inicialmente una solución factible al problema de optimización, junto con el de comprender su estructura, mediante el concepto de estructura de vecindad de esta solución. Más formalmente podemos definir el concepto genérico de vencidad, de la siguiente manera:

Definición 6.2.4 *Estructura de Vecindad*
Sea Ω un conjunto de soluciones factibles de un problema de optimización combinatoria $\max_{S \in \Omega} \{f(s)\}$. *Una estructura de vecindad es una colección $\{\mathcal{N}(s)\}_{s \in \Omega}$ de subconjuntos $\mathcal{N}(s) \subseteq \Omega$ tales que:*

1) $\forall s, t \in \Omega, \exists s = s_0, \ldots, s_k = t : s_i \in \mathcal{N}(s_{i-1}) \quad \forall i = 1, \ldots, k$

2) Para cada $s \in \Omega$ es posible decidir en tiempo polinomial si existe $t \in \mathcal{N}(s) : f(t) > f(s)$, y hallar esa mejor solución factible, si existe.

A su vez, el Algoritmo 1 muestra la aplicación de una búsqueda local genérica a un COP.

Algoritmo 1 Metaheurística de Búsqueda Local

1: $s \leftarrow \mathsf{GenerateInitialSolution}()$
2: **repeat**
3: $\quad s \leftarrow \mathsf{Improve}(s, \mathcal{N}(s))$
4: **until** $\forall t \in \mathcal{N}(s), f(t) < f(s)$
5: **return** s

6.2.3.2 Solución Propuesta

Con el fin de resolver el problema de la definición 6.2.3 mediante un algoritmo de búsqueda local, se define a continuación un concepto de distancia junto con una noción de estructura de vecindad, para nuestro problema específico.

Definición 6.2.5 *Definimos la distancia $d(\pi, \pi^*)$ entre dos permutaciones π y π^* como el mínimo número de intercambios de elementos necesarios para transformar π en π^*.*

En base a esta definición se establecen las siguientes 2 proposiciones (ver sus respectivas pruebas en [109]):

Proposición 6.2.6 *Llamemos Π al espacio de permutaciones. Luego (Π, d) es un espacio métrico, es decir que cumple con las siguientes tres afirmaciones:*

1) $\forall \pi, \pi^* \in \Pi, d(\pi, \pi^*) = 0 \leftrightarrow \pi = \pi^*$ *(reflexividad)*

2) $\forall \pi, \pi^* \in \Pi, d(\pi, \pi^*) = d(\pi^*, \pi)$ *(simetría)*

2) $\forall \pi_a, \pi_b, \pi_c \in \Pi, d(\pi_a, \pi_b) + d(\pi_b, \pi_c) \geq d(\pi_a, \pi_c)$ *(desigualdad triangular)*

Proposición 6.2.7 *Consideremos el problema de maximizar $E(X_\pi)$. Dada una permutación π (solución factible) definamos $\mathcal{N}(\pi) = \{\pi^* : d(\pi, \pi^*) = 1\}$ como el conjunto vecindad de la permutación π. Luego $\{\mathcal{N}(\pi)\}$ es una estructura de vecindades para el problema de maximización* $\max_\pi E(X_\pi)$

De esta manera, aplicando este concepto de vecindad y siguiendo el Algoritmo 1, es posible mejorar una solución inicial dada. A su vez, en el Capítulo 4.2.2 de [109], se presenta un algoritmo mediante el cual es posible obtener una solución factible (una permutación π), en base a una probabilidad de ocupación deseada (un vector p). Llamémosle al mismo: *algoritmo inicial*. Observar que esta solución puede ser usada para generar la solución inicial de la busqueda local.

6.2.4 Resolución del Problema mediante Colonia de Hormigas

A continuación se presenta una propuesta más elaborada para la resolución del COP. Para esto se presentará una reconfiguración del COP en términos del conocido problema del vendedor ambulante, o TSP (*Travelling Salesman Problem* [77, 135]). Para la resolución del TSP se aplica la metaheurística inspirada en el comportamiento de las hormigas para hallar las rutas más

cortas, conocida por sus siglas en ingles como ACO (*Ant Colony Optimization* [39, 40]). Finalmente estos resultados serán optimizados usando el algoritmo de búsqueda local, anteriormente presentado.

A continuación se presentan algunos conceptos básicos sobre esta metaheurística, para luego presentar nuestra solución propuesta, junto con los resultados teóricos obtenidos.

6.2.4.1 Conceptos Básicos

ACO es una metaheurística inspirada en la naturaleza, basada en el aprendizaje cooperativo de poblaciones (hormigas artificiales). ACO fue introducida en 1992 por Marco Dorigo [38] y procura simular el comportamiento de las hormigas en su hábitat natural, para hallar el camino más corto entre su hormiguero y el alimento. En particular, ACO es una metaheurística aplicada a la optimización, que incorpora elementos de aprendizaje para identificar regiones de alta calidad en el espacio de soluciones, aprovechando la estructura distribuida de las hormigas. La hormiga a pesar de ser un insecto casi ciego, logra encontrar el camino más corto entre el hormiguero y su fuente de alimento [14]. La comunicación entre las hormigas es a base del químico feromona, que deja un rastro que sirve de referencia a otras hormigas.

La esencia de esta técnica se halla en la definición y actualización de una función de feromonas, dado que las hormigas tomarán con mayor probabilidad la dirección que posee mayor magnitud de la misma.

Se construye un grafo en el que los nodos representan soluciones factibles y los enlaces sus vínculos. Entonces cada hormiga realiza una exploración del espacio de soluciones recorriendo un camino (subconjunto conectado de soluciones del espacio) y depositando feromonas $\tau_{(i,j)}$ por las conexiones (i, j) y los nodos $\tau_n, n \in V$. Estas feromonas representan una medida de deseo del uso de ese enlace y sesgan la distribución de probabilidades a utilizar por las siguientes hormigas. La actualización de la función de feromonas se realiza en base a la calidad del trayecto realizado por la hormiga, como también en base a un parámetro de heurística η. A continuación se presenta un algoritmo genérico para la aplicación de ACO (Algoritmo 2).

Algoritmo 2 Algoritmo genérico para la aplicación de ACO.

```
1: InicializarFeromonas()
2: while No se cumpla una condición de parada do
3:    for hormigas = 1 a n do
4:       s_a ← ConstruirSolucion(τ, η)
5:    end for
6:    ActualizarFeromonas()
7: end while
```

6.2.4.2 Solución Propuesta

A continuación se presenta un mapeo entre el COP definido en la Subsección 6.2.2 y un problema de ATSP (TSP asimétrico). La novedad en este enfoque, se halla no solamente en el significado de este grafo, sino en el proceso de construcción del mismo, que difiere del clásico realizado en los sistemas de colonias de hormigas. Un grupo de hormigas artificiales van

a cooperar para construir este grafo, aprovechando la experiencia que ya disponemos del problema. Una vez construido este grafo, un segundo grupo de hormigas rastrean ciclos de alta calidad, de modo similar al aplicado en ACO. Este ciclo se puede interpretar como una solución del COP, mediante la definición de una biyección ciclo-permutación. La Definición 6.2.8 determina dicha función biyectiva.

Definición 6.2.8 *Sea K_N un $N-$clique, con nodos etiquetados $\{1,\ldots,N\}$, y sea N el nodo auxiliar, donde inician todos los ciclos dirigidos. La igualdad de cardinalidad entre una permutación sobre el conjunto de nodos $\{1,\ldots,N-1\}$ y el ciclo se halla en el orden de los nodos visitados, sin considerar el nodo N. Tomemos un ciclo con sentido que visita a todos los nodos e inicia en el auxiliar N: $C=\{N,v_1,v_2,\ldots,v_N-1,N\}$, donde los v_i representan distintos nodos del clique. Entonces $\pi(i)=v_i, \quad \forall i=1,\ldots,N-1$ es una función biyectiva entre el espacio de permutaciones y un $N-$clique. Recíprocamente, a cada permutación $\pi(i)$ le corresponde un único ciclo con sentido $C=\{N,\pi(1),\ldots,\pi(N-1),N\}$. La existencia de estos ciclos en el grafo se encuentra asegurada pues en el clique todos sus nodos están conectados directamente.*

En la Figura 6.3, se presenta un ejemplo de esta definición, aplicada a un clique de 6 nodos.

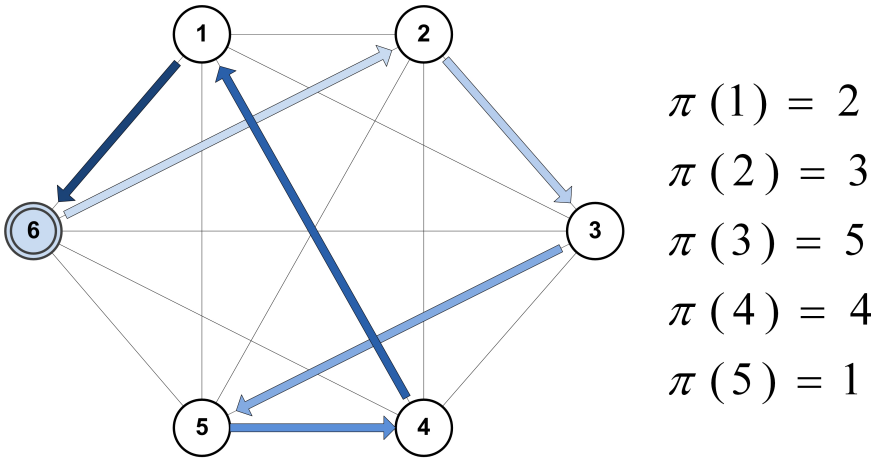


Figura 6.3: Sea K_N un $N-$clique, con nodos etiquetados $\{1,\ldots,N\}$, y sea N el nodo auxiliar, donde inician todos los ciclos dirigidos. Tomemos un ciclo con sentido que visita a todos los nodos e inicia en el auxiliar N: $C=\{N,v_1,v_2,\ldots,v_N-1,N\}$, donde los v_i representan distintos nodos del clique. Entonces $\pi(i)=v_i, \quad \forall i=1,\ldots,N-1$ es una función biyectiva entre el espacio de permutaciones y un $N-$clique.

Obsérvese que a efectos de traducir nuestro COP, el TSP debe ser asimétrico (ATSP), dado que en caso contrario, *greedy* y *rarest-first* definirían el mismo ciclo (pero son estrategias de consulta muy diferentes).

Respecto al algoritmo, en una etapa previa y con el fin de poder asignar distancias a las aristas e inicializar la feromona, se hace uso de un mecanismo de aprendizaje basado en la

exploración de hormigas artificiales. Se enumeran todos los nodos del clique $1, \ldots N$, y se colocan hormigas en el nodo auxiliar N (recordemos que las permutaciones son de tamaño $N-1$). Cada hormiga realizará un *tour* sesgado, en donde las distancias entre las aristas serán definidas en función de la calidad $E(X_\pi)$ de cada ciclo obtenido. Este proceso, se corresponde con el paso 1 del Algoritmo 2.

Luego, a partir del grafo asimétrico anteriormente construido, se aplica el algoritmo ACO. A su vez, en cada iteración, las permutaciones producidas por esta, son sustituidas por la mejor de sus vecinas (aplicando el enfoque de búsqueda local, presentado en la Subsección anterior 6.2.3).

Por más detalles acerca del diseño e implementación de esta solución, referirse a [109].

6.2.4.3 Resultados Teóricos Obtenidos

En la Figura 6.4, se presenta una comparación [2] entre estrategias previas y los resultados obtenidos mediante el algoritmo basado en ACO, para el caso en que $N = 30$ y $M = 100$.

Si bien los resultados finales no garantizan la optimalidad global, tal como se puede observar en la Tabla 6.1, logran una excelente continuidad y al mismo tiempo una latencia comparable con *greedy*.

Cuadro 6.1: Desempeño de distintas estrategias.

Estrategia	**Continuidad**	**Latencia**
Rarest First	0.9571	21.0011
Greedy	0.9020	4.1094
Mixta	0.9953	11.1253
Nueva Permutación	0.9998	7.9821

6.3 Resultados Obtenidos mediante Emulaciones con GoalBit

Las secciones anteriores de este capítulo son un extracto del trabajo [109], donde se presenta el trabajo original, aún no publicado. En esta sección se presentan los resultados obtenidos al aplicar una selección de piezas óptima en GoalBit (al menos en el modelo), y los resultados al realizar un conjunto de emulaciones con él. A su vez, se compara dicha estrategia con la *estrategia exponencial* implementada en el cliente GoalBit y descripta en la Subsección 5.2.5.2.

[2] La estrategia Mixta, es introducida en [149], y consiste en una mezcla de *rarest-first* con *greedy*, en donde cortando el *buffer* en un índice dado (m: $1 \leq m \leq N$), se aplica *rarest-first* en la primer partición y *greedy* en la segunda.

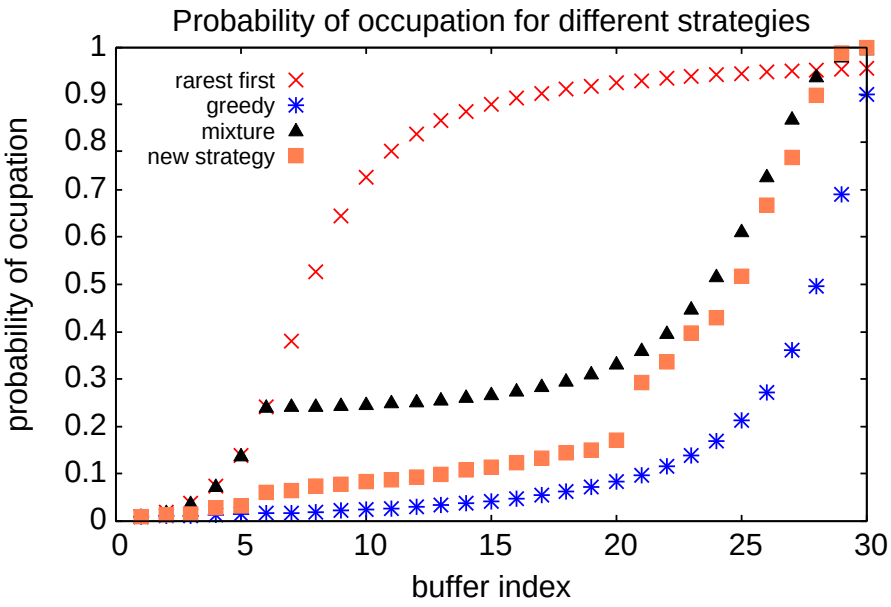


Figura 6.4: Comparación entre distintas estrategias

6.3.1 Emulaciones con GoalBit

6.3.1.1 Introducción

Con el fin de realizar pruebas de desempeño en GoalBit se construyó un simulador, desarrollado sobre el propio cliente GoalBit. Este trabajo fue realizado por A. Barrios, M. Barrios, J.J. Comas y P. Perdomo en su tesis de grado [8] y simula la capa de red, el reloj interno del sistema y la reproducción del video. Más en concreto, se simula la ejecución de N_s *peers* de la siguiente manera:

- se toman M *peers* (notar que $M << N_s$) y se ejecutan en paralelo durante t segundos. Durante la ejecución, los *peers* se comportan de igual manera que un *peer* no simulado (estos se pueden anunciar en el *tracker*, se pueden comunicar con otros *peers* en el *swarm*, etc.).
- una vez transcurrido los t segundos, los M *peers* son suspendidos (actualizando sus relojes internos), pasándose a ejecutar otros M *peers*.
- un componente central simula la línea de reproducción, consumiendo piezas a una cierta tasa dada, contabilizando las pérdidas y los tiempos de *buffering* de cada *peer*.
- a su vez, se simula la capa de red con el fin de permitir la comunicación entre *peers* que no son ejecutados en paralelo (es decir, *peers* que pertenecen a M disjuntos).

Siguiendo este enfoque, si bien es posible simular la ejecución de una gran cantidad de *peers*, los resultados obtenidos no reflejan en su totalidad lo ocurrido en un escenario real.

Debido a este motivo, se creó el llamado *Emulador GoalBit*, en donde se ejecutan N_e *peers* (notar que $N_e < N_s$) en tiempo real, usando la capa de red real, únicamente simulando la reproducción del video (registrando las pérdidas y los tiempos de *buffering* de cada *peer*). Con este emulador, si bien no es posible ejecutar una gran cantidad de *peers* (esta depende de las capacidades del computador en donde se realicen las pruebas), los resultados obtenidos son más cercanos a la realidad.

Este trabajo fue realizado por M. Barrios como complemento a su trabajo de tesis de grado.

6.3.1.2 Escenario de Emulación

Para realizar las emulaciones se siguió el patrón de conexiones y desconexiones de usuarios observado en AdinetTV (ver la Sección 2.3) durante la transmisión de un evento muy popular. En base a este patrón, se definen 2 diferentes casos de pruebas. En el *caso de prueba 1*, existen 45 *peers* involucrados, con una concurrencia media en la red de 33 *peers*. En la Figura 6.5 se presenta la evolución de la cantidad de *peers* en la red. El *caso de prueba 2*, está compuesto por la ejecución de 156 *peers*, existiendo una concurrencia media de 89 *peers*, tal como se puede observar en la Figura 6.6. En ambos casos se distribuye un video de 330Kbps, mediante la ejecución de 2 *broadcaster-super-peers* con una capacidad de 2Mbps de ancho de banda de subida cada uno de ellos (notar que para simplificar la emulación no existen *super-peers* en la red). En la Figura 6.7, se presenta un histograma de la cantidad de *peers* según su ancho de banda de subida disponible. Tal como se puede observar en esta, en ambos casos la distribución es muy similar, con un ancho de banda de subida promedio de 311 Kbps para el caso 1 y 330 Kbps para el caso 2.

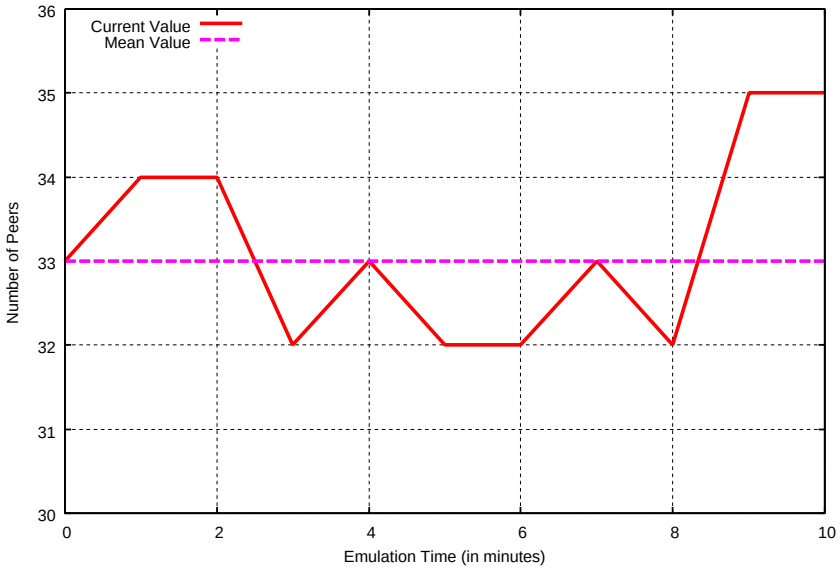


Figura 6.5: Cantidad de *peers* conectados según tiempo de emulación para el caso de prueba 1 (45 *peers* involucrados).

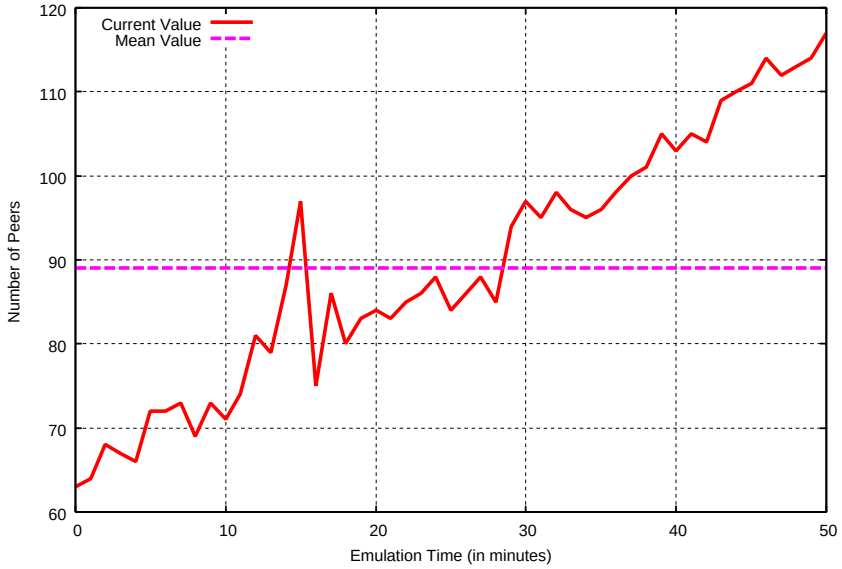


Figura 6.6: Cantidad de *peers* conectados según tiempo de emulación para el caso de prueba 2 (156 *peers* involucrados).

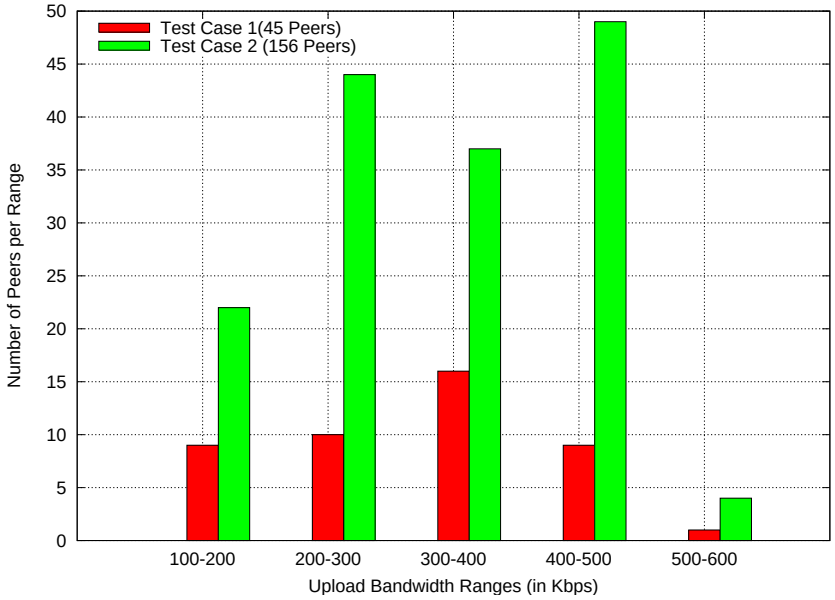


Figura 6.7: Cantidad de *peers* según su ancho de banda de subida disponible.

A su vez, cada uno de estos casos de prueba fue ejecutado 2 veces, uno usando la estrategia exponencial y otra usando una estrategia basada en la siguiente permutación:

$$\pi(i) = N - i, i = 1, \ldots, I, \tag{6.18}$$

$$\pi(I + j) = j, j = 1, \ldots, J \tag{6.19}$$

$$\pi(I + J + k) = \left\lfloor \frac{N + J - I}{2} \right\rfloor + \left\lceil \frac{k}{2} \right\rceil (-1)^{k+1}, \tag{6.20}$$

$$k = 1, \ldots, N - I - J - 1,$$

En particular, se usaron los siguientes valores: $N = 40, I = 16, J = 1$ Tal como se demuestra en [109], esta estrategia de permutación, presenta valores muy cercanos a la estrategia óptima, obtenida mediante la aplicación de ACO y búsqueda local.

Finalmente, en cada ejecución, y para cada *peer*, se mide el tiempo de *buffering* inicial (la latencia), la cantidad de *re-bufferings* y el tiempo transcurrido en dicho estado (estas 2 últimas representan una medida indirecta de la continuidad de la reproducción).

6.3.2 Resultados Obtenidos

6.3.2.1 Caso de Prueba 1: 45 Peers

En la Figura 6.8 se presenta el tiempo de *buffering* inicial de cada *peer* para el caso de prueba 1, según las diferentes estrategias de selección de piezas. Como es posible observar en dicha figura, la latencia medida al aplicar la estrategia exponencial es claramente superior a la obtenida mediante la aplicación de la estrategia de permutación.

A su vez, en las Figuras 6.9 y 6.10 se presenta la cantidad de *re-bufferings* y el tiempo incurrido en estos respectivamente, para cada *peer* en la emulación, diferenciando según la estrategia aplicada. En este caso, la continuidad es muy superior al aplicar la estrategia exponencial.

Como resultado de esta prueba, se observa que, si bien la estrategia exponencial impone una mayor latencia que la estrategia de permutaciones, esta provee de una continuidad de reproducción mucho mayor, lo cual es un factor muy importante a su favor (la estrategia exponencial presenta 0 *re-bufferings* y por lo tanto un tiempo nulo de *re-bufferings*).

6.3.2.2 Caso de Prueba 2: 156 Peers

Respecto al caso de prueba 2, la Figura 6.11 presenta el tiempo de *buffering* inicial de cada *peer*, nuevamente clasificado según la estrategia aplicada. A diferencia del caso anterior se puede observar que en esta ocasión, la aplicación de la selección de piezas exponencial logra considerables mejoras en la latencia respecto a la aplicación de la estrategia de permutaciones.

Por otro lado, tal como presentan las Figuras 6.12 y 6.13, en este caso la continuidad provista por la selección exponencial es considerablemente mayor que la provista por la estrategia

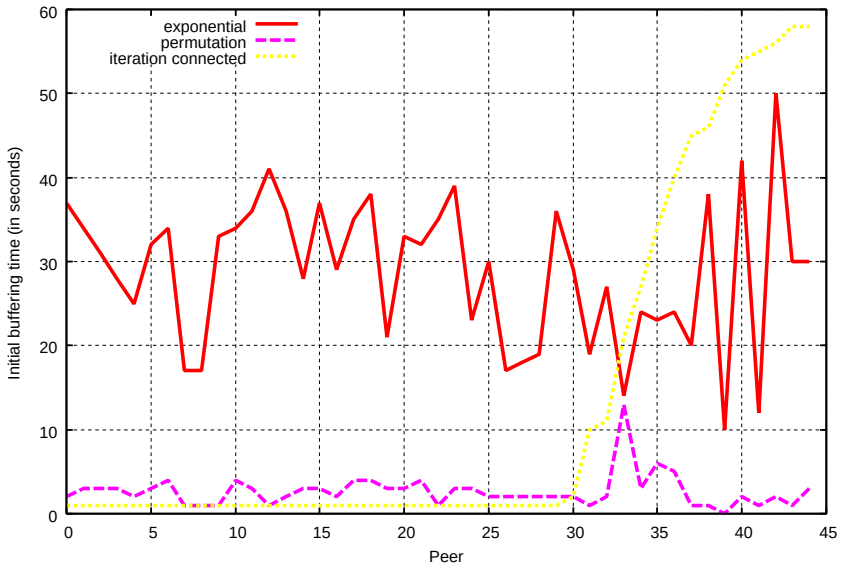


Figura 6.8: Tiempo de *buffering* inicial de cada *peer* para el caso de prueba 1, según la estrategia de selección de piezas. A su vez se presenta en que momento cada peer es ingresado en la red (parámetro *iteration connected*).

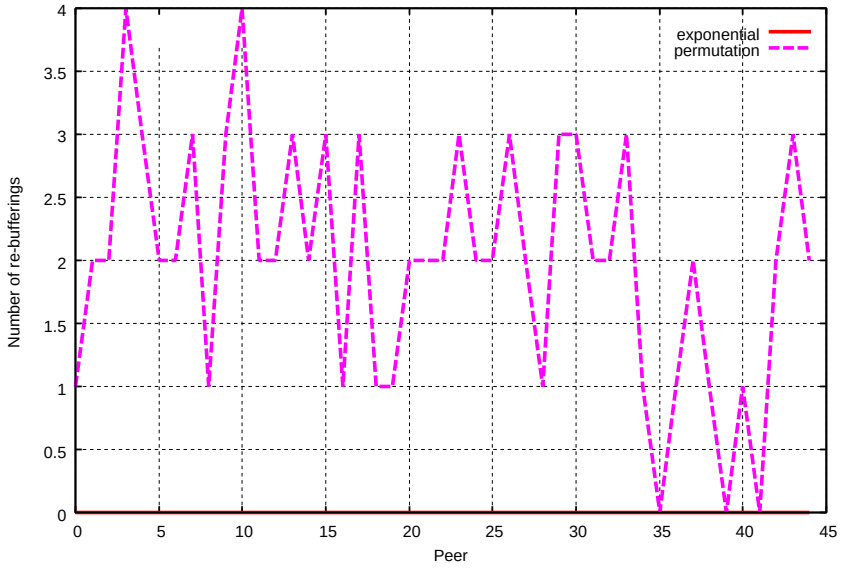


Figura 6.9: Cantidad de *re-bufferings* por *peer* para el caso de prueba 1, según la estrategia de selección de piezas. Notar que en el caso de la estrategia exponencial no existieron *re-bufferings*.

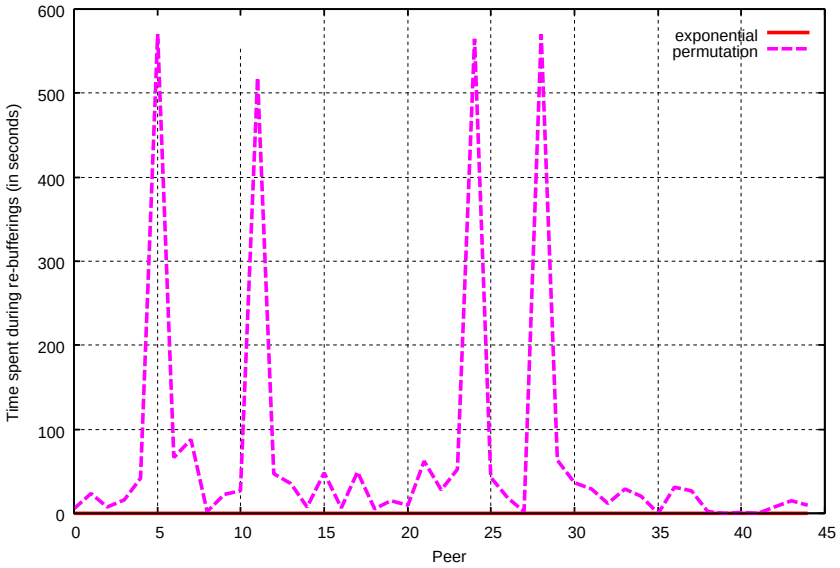


Figura 6.10: Tiempo transcurrido por cada *peer* en estado de *re-buffering* para el caso de prueba 1, según la estrategia de selección de piezas. Notar que la curva asociada con la estrategia exponencial es de la forma $f(x) = 0$.

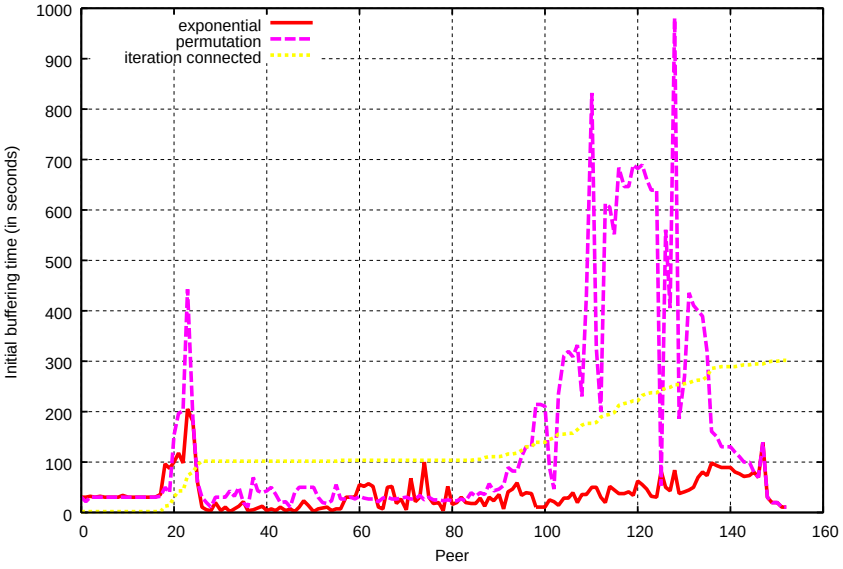


Figura 6.11: Tiempo de *buffering* inicial de cada *peer* para el caso de prueba 2, según la estrategia de selección de piezas. A su vez se presenta en que momento cada peer es ingresado en la red (parámetro *iteration connected*).

de permutaciones. Esto último señala que al aumentar la cantidad de *peers* en la red, la selección de piezas exponencial amplia sus virtudes respecto a la estrategia óptima hallada en nuestro modelo matemático.

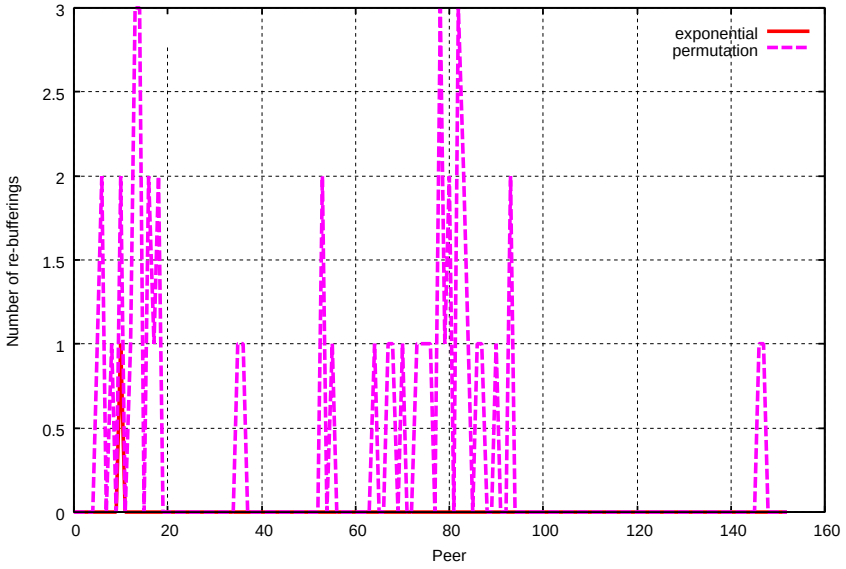


Figura 6.12: Cantidad de *re-bufferings* por *peer* para el caso de prueba 2, según la estrategia de selección de piezas. Observar que en el caso de la estrategia exponencial no existieron *re-bufferings*.

6.3.3 Análisis de los Resultados

Como es bien sabido, las redes P2P son de extrema complejidad, y sus modelos matemáticos suelen imponer considerables restricciones sobre las mismas. Mediante las emulaciones realizadas, contrastamos el modelo considerado con la realidad, observando que la correspondencia entre estos no es directa: una solución óptima sobre el modelo (la estrategia de permutaciones), no se corresponde con una solución óptima en la realidad. Según nuestra experiencia, esta "no correspondencia" puede ser explicada en base a las siguientes restricciones impuestas por el modelo.

Una primer restricción del modelo considerado es la concepción de la red como una topología cerrada. En la realidad los *peers* ingresan y egresan constantemente de la red, lo que impone complejidad en el manejo de recursos, que son dinámicos en el tiempo.

Una segunda restricción del modelo es el hecho de considerar que todos los pares son idénticos en la red. Es claro que en redes P2P, montadas sobre la infraestructura de Internet, los pares poseen anchos de banda muy variables, como también distinta capacidad de procesamiento.

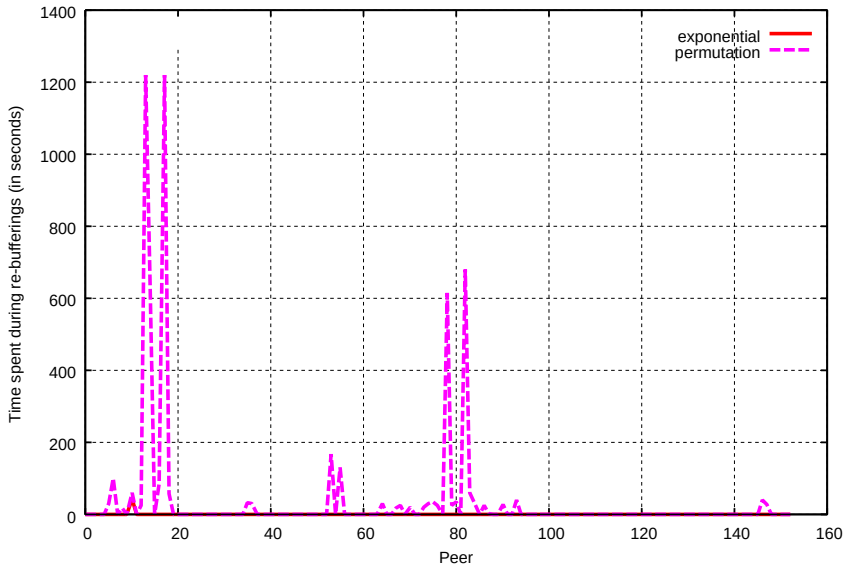


Figura 6.13: Tiempo transcurrido por cada *peer* en estado de *re-buffering* para el caso de prueba 2, según la estrategia de selección de piezas. Notar que la curva asociada con la estrategia exponencial es de la forma $f(x) = 0$.

Finalmente un tercer y muy importante elemento diferenciado entre el modelo y las redes reales de *streaming* de video es la hipótesis de sincronismo. La cooperación analizada en el modelo matemático se sostiene en el hecho de que todos los pares reproducen la misma pieza en cada instante (si es que la disponen). A su vez, la consulta se refiere a índices del *buffer*, entendidos estos globalmente, gracias al sincronismo. En las redes reales, si bien se intenta que exista una sincronización entre *peers*, la misma no puede ser considerada tan finamente como en el modelo, ni puede ser asegurada en la totalidad de los casos. El hecho que dos *peers* se encuentren sincronizados, significa que estos pueden intercambiar piezas entre sí, pero no que estos reproduzcan lo mismo, en el mismo instante de tiempo.

Capítulo 7

Resultados Empíricos

Este capítulo brinda algunos resultados empíricos obtenidos al usar GoalBit para distribuir contenidos. En la Sección 7.1 se presenta un análisis detallado del protocolo GBTP. En la Sección 7.2 se presentan un conjunto de mediciones realizadas sobre una transmisión de un usuario final.

7.1 Análisis del protocolo GBTP

Con el fin de obtener un análisis detallado del protocolo GBTP, se capturaron todos los mensajes de dicho protocolo recibidos y enviados durante la ejecución de un cliente GoalBit.

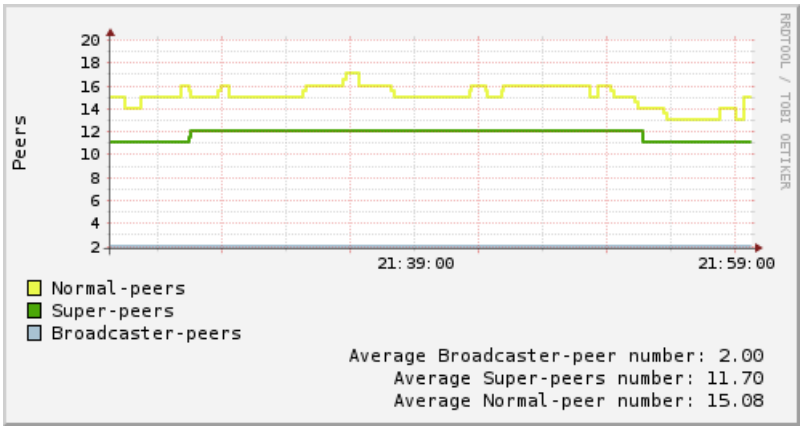
El cliente GoalBit se ejecutó recibiendo el *streaming* de uno de los canales de AdinetTV (ver la Sección 2.3), bajo dos diferentes escenarios: el primero durante la transmisión de un evento poco popular y el segundo durante la transmisión de un evento muy popular.

La Figura 7.1(a) muestra la cantidad de usuarios presentes durante la transmisión del evento poco popular, discriminado por tipo de *peer*. Como se puede observar en la misma, en promedio existieron 12 *super-peers* conectados y 15 *peers* normales. En este escenario, nuestro cliente descargó a una tasa promedio de 350 Kbps y realizó subidas a una tasa promedio de 16 Kbps (ver Figura 7.1(b)). Claramente en este caso el sistema se encontraba con poca carga y los *super-peers* fueron los responsables de absorber la gran mayoría de ésta.

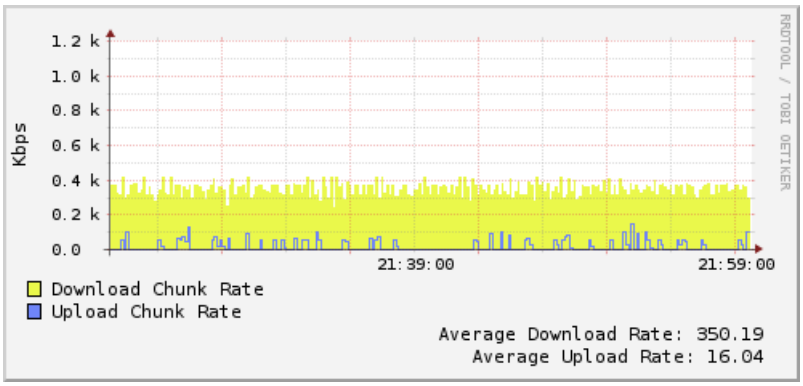
Por otro lado, la Figura 7.2(a) muestra la cantidad de usuarios presentes durante la transmisión de un evento muy popular, en donde se registraron un promedio de 69 *peers* y 14 *super-peers*. En este caso, nuestro cliente descargó a una tasa promedio de 347 Kbps y realizó subidas a una tasa promedio de 403 Kbps (ver Figura 7.2(b)). Como se puede observar, nuestro cliente aportó a otros *peers* a una tasa mayor que la tasa del video. Este hecho, principalmente se debe a que los *super-peers* no pudieron cubrir la totalidad de la carga, por lo que los *peers* colaboraron más entre sí.

Observando las Figuras 7.1(a) y 7.2(a) se puede observar el efecto del balanceo de carga aplicado por la Plataforma GoalBit (la cual será presentada en el Capítulo 8), en donde varía la cantidad de *super-peers* conectados respecto a la cantidad de usuarios en la red.

En la Tabla 7.1, se presentan todos los mensajes GBTP recibidos y enviados durante 2400 segundos de transmisión del evento poco popular y en la Tabla 7.2, se presenta dicha informa-



(a) Cantidad de usuarios en la red.

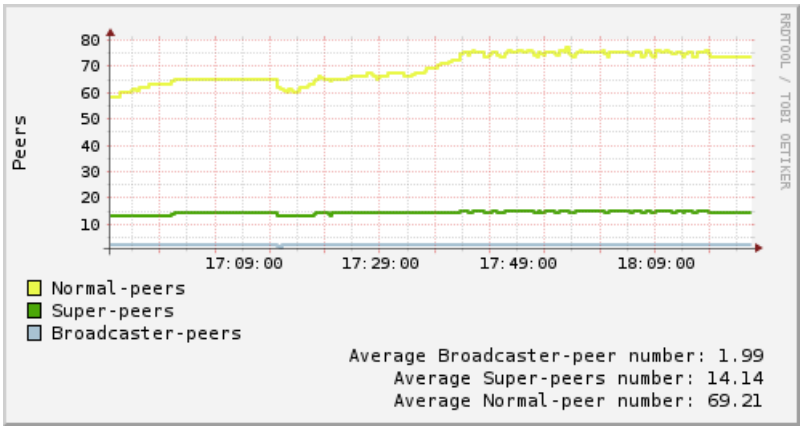


(b) Tasa de descarga vs tasa de subida.

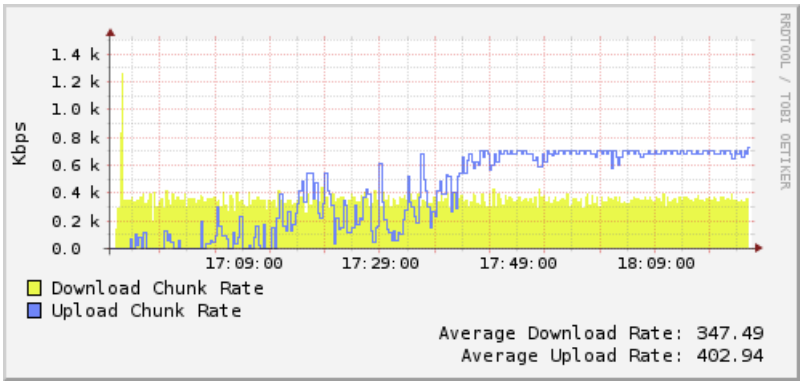
Figura 7.1: Datos obtenidos durante la transmisión de un evento poco popular.

Cuadro 7.1: Mensajes GBTP recibidos y enviados durante 2400 segundos de transmisión de un evento poco popular.

	Recibidos			Enviados		
Tipo de mensaje	Cantidad	*Bytes*	%	Cantidad	*Bytes*	%
HANDSHAKE	24	1848	0.00	365	28105	0.56
BITFIELD	42	2135	0.00	128	4792	0.10
HAVE	3323	33230	0.03	5469	54690	1.10
WIN UPDATE	120	720	0.00	65	390	0.01
KEEP-ALIVE	19	19	0.00	19	19	0.00
INTERESTED	2208	4416	0.00	2179	4358	0.09
NOT INTERESTED	2203	4406	0.00	2166	4332	0.09
CHOKE	636	1272	0.00	617	1234	0.02
UNCHOKE	653	1306	0.00	627	1254	0.03
REQUEST	63	882	0.00	54013	756182	15.17
CANCEL	4	56	0.00	7	98	0.00
PIECE	1707	111887022	99.68	63	4129398	82.84
DONT HAVE	52296	313776	0.28	0	0	0.00
Total		112251088	100.00		4984852	100.00



(a) Cantidad de usuarios en la red.



(b) Tasa de descarga vs tasa de subida.

Figura 7.2: Datos obtenidos durante la transmisión de un evento muy popular.

Cuadro 7.2: Mensajes GBTP enviados y recibidos durante 5600 segundos de transmisión de un evento muy popular.

	Recibidos			Enviados		
Tipo de mensaje	Cantidad	*Bytes*	%	Cantidad	*Bytes*	%
HANDSHAKE	66	5082	0.00	953	73381	0.03
BITFIELD	279	12956	0.01	387	18692	0.01
HAVE	16286	162860	0.06	46968	469680	0.16
WIN UPDATE	322	1932	0.00	321	1926	0.00
KEEP-ALIVE	0	0	0.00	14	14	0.00
INTERESTED	11031	22062	0.01	7253	14506	0.01
NOT INTERESTED	10989	21978	0.01	7237	14474	0.01
CHOKE	2664	5328	0.00	2537	5074	0.00
UNCHOKE	2689	5378	0.00	2584	5168	0.00
REQUEST	4741	66374	0.03	56159	786226	0.27
CANCEL	608	8512	0.00	173	2422	0.00
PIECE	3871	253728566	99.75	4358	285649468	99.52
DONT HAVE	52174	313044	0.12	0	0	0.00
Total		254354072	100.00		287041031	100.00

ción pero para 5600 segundos de transmisión del evento muy popular.

Comparando estas tablas respecto a los mensajes recibidos en cada uno de estos escenarios, se puede observar que en ambos casos la mayor cantidad de *bytes* recibidos se correspondieron con mensajes de tipo PIECE (en el escenario no popular un 99.68 % y en el popular un 99.75 %), los cuales, salvo excepciones, se suelen recibir a una tasa constante y muy cercana al *bitrate* del video. Las diferencias entre estos dos escenario se dieron principalmente en los mensajes de tipo HAVE (en el caso no popular un 0.03 % y en el popular un 0.06 %), en los de tipo REQUEST (en el escenario no popular un 0.00 % y en el popular un 0.03 %) y en los de tipo DONT HAVE (en el escenario no popular un 0.28 % y en el popular un 0.12 %). Estas tres diferencias se explican fácilmente en base a la cantidad de *peers* conectados como se presenta a continuación.

- Recordando el protocolo GBTP (ver sección 5.3), los *seeders* (*super-peers* o *broadcaster-peers*) no envían mensajes de tipo HAVE, únicamente los *peers*, por lo que al aumentar la cantidad de *peers* en la red, tiene sentido que aumente la cantidad de mensajes de tipo HAVE recibidos (existe una cota para este valor dado que un *peer* mantiene como máximo 55 conexiones simultaneas con otros *peers*).

- A su vez, el hecho de la existencia de una mayor cantidad de *peers* en la red, conlleva a la existencia de un mayor intercambio entre estos, aumentando así la cantidad de mensajes de tipo REQUEST recibidos (recordar que los *seeders* nunca solicitan piezas a *peers* normales).

- Respecto a las diferencias en los mensajes de tipo DONT HAVE, dado que estos mensajes únicamente son enviados por los *seeders*, tiene sentido que cuanto mayor sea la comunicación de los *peers* con los *seeders* (tal como ocurre en el caso del evento poco popular) mayor sea la cantidad de mensajes DONT HAVE recibidos.

El hecho de comparar los mensajes enviados en cada escenario no es sencillo, dado que la mayor cantidad de *bytes* enviados se corresponde con mensajes de tipo PIECE y a diferencia con la comparación de los mensajes recibidos, en este caso la tasa de envío de piezas es extremadamente variable (por lo que los porcentajes varían mucho entre ambos casos). De cualquier manera, se pueden observar grandes diferencias respecto al envío de mensajes de tipo PIECE (en el escenario poco popular un 82.84 % y en el popular un 99.52 %) y en el envió de mensajes de tipo REQUEST (un 15.17 % en el no popular y un 0.27 % en el escenario popular). Estas diferencias también se explican en base a la cantidad de *peers* conectados.

- Al existir una mayor cantidad de *peers* en la red, debería existir un mayor intercambio de piezas entre estos (sin considerar a los *seeders*), por lo que el envío de mensajes de tipo PIECE debería ser mayor entre los *peers* normales.

- Respecto a los mensajes enviados de tipo REQUEST, al aumentar la comunicación con los *seeders* (y suponiendo que el ancho de banda de descarga disponible de los *peers* es mayor que el *bitrate* del video), debería aumentar la cantidad de solicitudes de piezas, debido a la existencia de una mayor probabilidad de solicitar piezas aún no generadas

(teniendo que volver a solicitar las piezas más adelante). A su vez, esto último trae como consecuencia una mayor recepción de mensajes de tipo DONT HAVE (como comentamos anteriormente).

Con el fin de evaluar la eficiencia del protocolo GBTP, vamos a analizar el *overhead* agregado por éste en la transmisión del *streaming*. Definimos como el *overhead*, a los *bytes* de la transmisión no pertenecientes al contenido de video. En otras palabras definimos el *overhead* del protocolo GBTP como $O = TB - PB + 10PC$, en donde TB es el total de *bytes* transmitidos, PB es el total de *bytes* transmitidos en mensajes de tipo PIECE y PC la cantidad de mensajes de tipo PIECE transmitidos (recordar que el mensaje PIECE tiene un encabezado de 10 *bytes*). En la Tabla 7.3 se presenta el *overhead* calculado para el escenario de la transmisión poco popular y en la Tabla 7.4 el mismo para el escenario muy popular. Como se puede observar en éstas, el *overhead* es calculado según su tipo (entrante, saliente y global). El escenario no popular presenta un *overhead* global de 1.05 % mientras que en el escenario popular, éste es de un 0.38 % de los *bytes* transmitidos. Notar que el primer escenario presenta un alto *overhead* saliente (17.17 %), debido a la baja transferencia de piezas entre *peers* normales.

Teniendo en cuenta que las redes P2P usualmente son diseñadas para ser usadas con grandes cantidades de usuarios (dado que en este tipo de escenarios es en donde estas redes presentan sus principales virtudes, tal como su escalabilidad), podemos afirmar que el protocolo GBTP presenta un *overhead* muy bajo en la transmisión del video, de tan solo un 0.04 %.

Cuadro 7.3: *Overhead* del protocolo GBTP durante la transmisión de un evento poco popular.

Tipo de *Overhead*	*Bytes*	%
Entrante	381136	0.34
Saliente	856084	17.17
Global	1237220	1.05

Cuadro 7.4: *Overhead* del protocolo GBTP durante la transmisión de un evento muy popular.

Tipo de *Overhead*	*Bytes*	%
Entrante	664216	0.26
Saliente	1435143	0.50
Global	2099359	0.38

7.2 Resultados del *Broadcasting* de un Usuario Final

En esta sección se presentan un conjunto de resultados obtenidos sobre una transmisión de un usuario final.

La infraestructura del usuario para esta transmisión estaba compuesta por un único servidor con una capacidad de ancho de banda de subida de 2 Mbps. En este servidor se ejecutaron 6 *broadcaster-peers* y 6 *super-peers*. El *stream* fue codificado en H.264/ACC bajo un *bitrate* de 300 Kbps y fue multiplexado sobre ASF. El tamaño de las piezas GBTP fue de 65536 *bytes*.

La mayoría de los resultados fueron obtenidos a nivel de cliente, mediante el análisis de *logs* generados por un cliente (cliente GoalBit) conectado a la red durante la transmisión. A nivel de servidor, únicamente se midió el ancho de banda entrante y saliente consumido durante el *broadcasting*.

La información presentada a continuación, se corresponde con 2000 segundos (más de media ahora) de la ejecución del *streaming*.

Evolución del Número de Peers en la Red. La Figura 7.3 muestra la evolución del número de *peers* conectados a la red, durante la transmisión del *streaming*. Existen 6 *broadcaster-peers*, 6 *super-peers* (todos ellos pertenecientes al propietario del *streaming*) y un promedio de 62,3 *peers*.

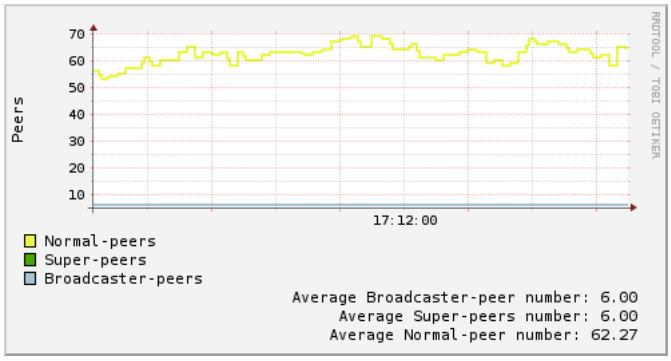


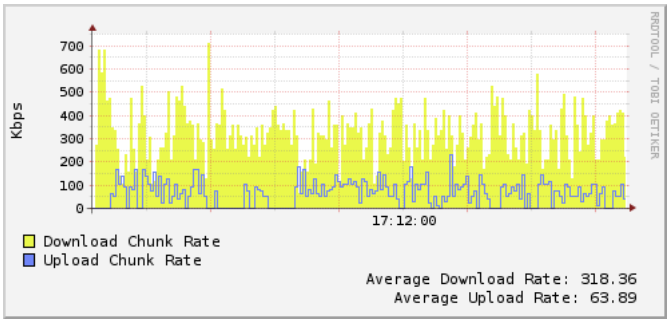
Figura 7.3: Evolución del número de *peers* en la red.

Esta información puede ser medida a nivel de cliente, debido al diseño de GBTP, en donde el *tracker* siempre informa acerca del número total de *peers* conectados en la red.

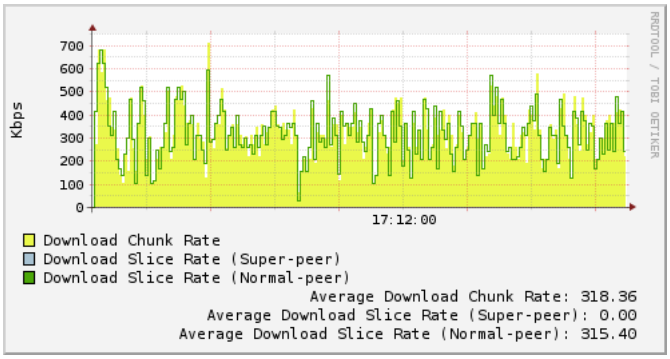
Tasa de Descarga vs Tasa de Subida. La Figura 7.4(a) presenta la evolución de las tasas de descarga y de subida de piezas (ambas en Kbps). Nuestro *peer* descargó y subió piezas a una tasa promedio de 318 Kbps y 64 Kbps respectivamente (este subió el 20 % de lo que se bajó). La diferencia en estas tasas se puede explicar debido a restricciones en la conectividad a Internet (en donde el máximo ancho de banda de subida disponible en nuestro cliente era próximo a 100 Kbps).

En la Figura 7.4(b), presentamos la tasa de descarga de piezas (en Kbps) discriminada por tipo de *peer*. Nuestro *peer*, descargó el 100 % del *streaming* de otros *peers* normales (notar que en ningún momento durante la transmisión del *streaming*, nuestro *peer* descargo una pieza de un *super-peer*).

Latencia vs Descargas Concurrentes. Definimos latencia como el tiempo transcurrido entre la solicitud de una pieza y la obtención de la misma. La Figura 7.5(a) presenta la latencia (en segundos) discriminada por tipo de *peer*. Se puede observar que nuestro cliente presentó una latencia promedio de 10 segundos en la descarga de piezas.



(a) Tasa de descarga vs tasa de subida.

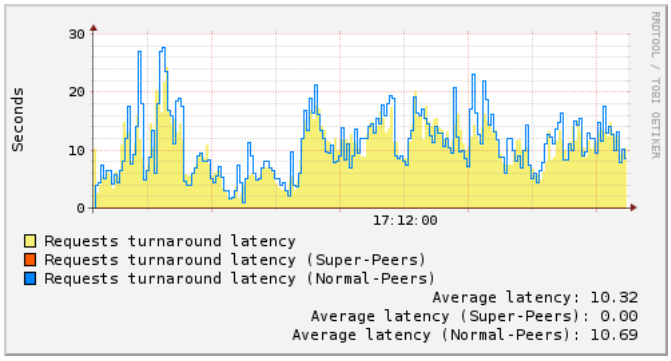


(b) Tasa de descarga de piezas (en Kbps) discriminada por tipo de *peer*.

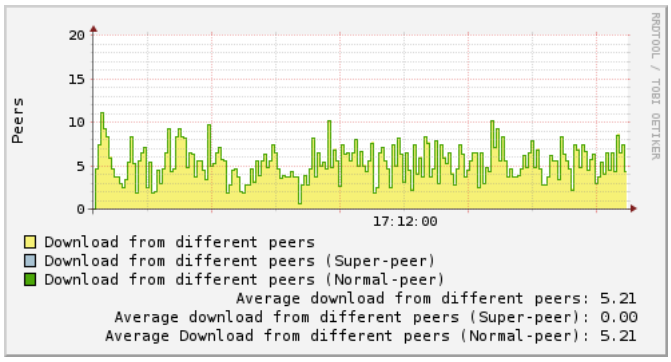
Figura 7.4: Tasas medidas durante la transmisión del *streaming*.

Dado que el *streaming* fue codificado a una tasa de 300 Kbps y el tamaño de las piezas GBTP fue de 65536 *bytes*, el reproductor de video debería consumir una pieza cada 1,7 segundos. Por lo tanto, si tenemos una latencia de 10 segundos, con el fin de cubrir la tasa de consumo del reproductor, nuestro cliente debe realizar solicitudes a diferentes *peers* en paralelo. En la Figura 7.5(b) se presentan el número de descargas concurrentes desde diferentes *peers*, agrupadas en períodos de 10 segundos. Ésta muestra que cada 10 segundos, nuestro *peer* realiza 5,2 descargas de piezas, con lo cual se satisface la tasa de consumo de piezas (obteniendo más de 5 piezas cada 10 segundos y considerando un *buffer* mínimo, es posible consumir 1 cada menos de 2 segundos).

Evolución de ABI y Pérdidas de Piezas. La Figura 7.6(a) muestra la cantidad de piezas promedio dentro del *active buffer* de nuestro cliente, esto es: la cantidad de piezas descargadas que se pueden reproducir ininterrumpidamente desde la línea de ejecución. Por otro lado, en la Figura 7.6(b), se presentan las pérdidas ocurridas durante la ejecución del *streaming*. Como se puede observar en estas gráficas, las mismas se encuentran correlacionadas: cada vez que la cantidad de piezas dentro del *buffer* activo es cercana a 0, se registran pérdidas. También podemos observar que el promedio de piezas en el *active buffer* es de casi 20 piezas. Este valor es mayor que el tamaño del *buffer* inicial (de 16 piezas) por lo cual el porcentaje de pérdidas es muy bajo, de tan solo un 0,03 %.



(a) Latencia (en segundos) discriminada por tipo de *peer*.



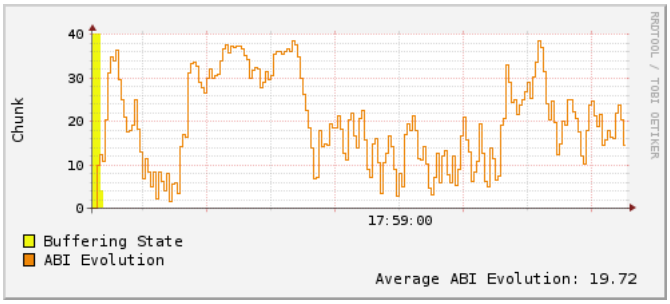
(b) Descargas concurrentes desde diferentes *peers* agrupadas en períodos de 10 segundos.

Figura 7.5: Latencia vs descargas concurrentes.

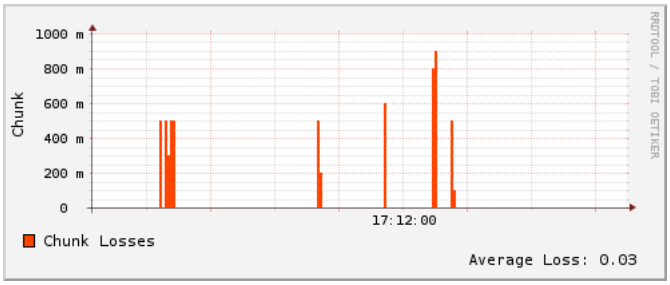
Ancho de Banda Consumido en el Servidor. La Figura 7.7 muestra el ancho de banda consumido por el servidor durante todo el día de la transmisión. En ésta se pueden observar dos grandes picos en el ancho de banda saliente, cada uno se corresponde con la transmisión de un evento popular. Los resultados anteriormente presentados, se corresponden con el segundo pico, en donde la tasa promedio de subida fue de 2 Mbps.

Conclusiones de la Evaluación Los resultados presentados anteriormente, muestran grandes ahorros en términos de ancho de banda para el *broadcaster* y una buena calidad de experiencia para los usuarios finales.

Comparando el *streaming* GoalBit con la clásica arquitectura cliente-servidor, durante la transmisión del *streaming* se miden ahorros de hasta un $90\,\%$ en el ancho de banda consumido por el servidor (con el fin de distribuir un *streaming* clásico de 300 Kbps a 62 usuarios, es necesario un ancho de banda de subida superior a 18 Mbps, usando GoalBit únicamente 2 Mbps fueron consumidos). Por otro lado, nuestro cliente GoalBit solo midió un $0{,}03\,\%$ de pérdidas durante la ejecución completa del *streaming* (generando alguna distorsión prácticamente imperceptible sobre el video), lo que hace a GoalBit una solución de *streaming* válida y muy interesante.



(a) Número de piezas dentro del *buffer* activo.



(b) Pérdidas de piezas ocurridas durante el período de transmisión.

Figura 7.6: Evolución del ABI y pérdidas de piezas.

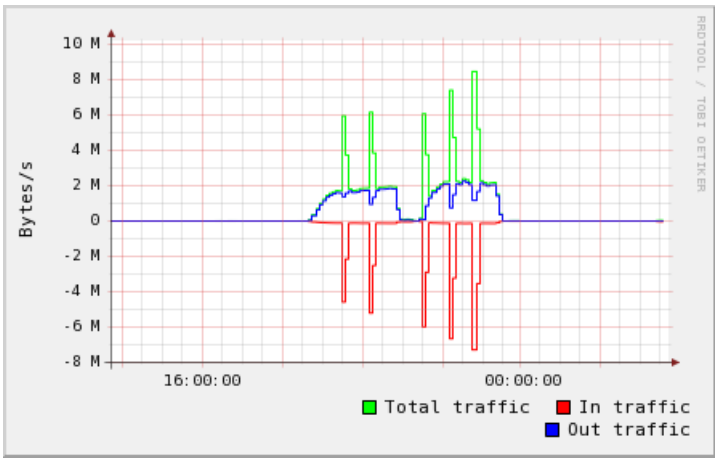


Figura 7.7: Ancho de banda consumido en el servidor.

Parte III

EXTENSIONES DEL STREAMING GOALBIT

Capítulo 8

La Plataforma GoalBit

Este capítulo presenta la *Plataforma GoalBit*, una extensión del *streaming* GoalBit con el fin de lograr una mejor utilización y administración de los recursos de la red.

Los *super-peers* fueron creados para asistir en la entrega de video a los usuarios finales. Típicamente el distribuidor del video dispone de una cantidad de *super-peers* para asignar entre todos sus contenidos. Muy a menudo suele ocurrir que contenidos de baja popularidad tienen asociados la misma cantidad de *super-peers* que contenidos muy populares, situación que conlleva a que unos pocos *super-peers* se encuentren muy cargados (en algunos casos hasta saturados) y otros con muy poca carga. Uno de los objetivos de este sistema es atacar esta problemática, distribuyendo los *super-peers* según la demanda (cantidad de usuarios conectados) en cada contenido. Por otro lado, la administración de los *broadcasters* y *super-peers* de una red, puede llegar a ser una tarea engorrosa. Por ejemplo, en un escenario en donde existen 5 servidores, 10 contenidos (y por lo tanto 10 *broadcasters*) y 20 *super-peers*, saber en qué nodo se encuentra un *super-peer* de un cierto contenido, no es un hecho trivial. La Plataforma GoalBit, presenta una visión centralizada de la red a nivel de *broadcasters* y *super-peers*, brindando la posibilidad de administrar estos últimos, asignando y desasignando *super-peers* sobre los contenidos, de una forma centralizada. Finalmente, este sistema también brinda la posibilidad de adaptar la señal GoalBit para ser distribuida por otros mecanismos, como por ejemplo usando servidores *Windows Media Services* o servidores de *streaming Flash*.

En la Sección 8.1 se presenta la arquitectura de la Plataforma GoalBit, en la Sección 8.2 se presenta la implementación de la misma y finalmente en la Sección 8.3 se presenta una implantación de este sistema en una red de distribución de contenido real.

8.1 Arquitectura

En esta sección se presenta la arquitectura de la Plataforma GoalBit: cuales son sus componentes y cuales son las funciones de cada uno de estos. Como se puede observar en la Figura 8.1, dentro del sistema participan cuatro diferentes componentes: los *broadcasters*, los controladores GoalBit, el *tracker* y el servidor de control.

A continuación se brindan detalles sobre las acciones desarrolladas por cada uno de estos.

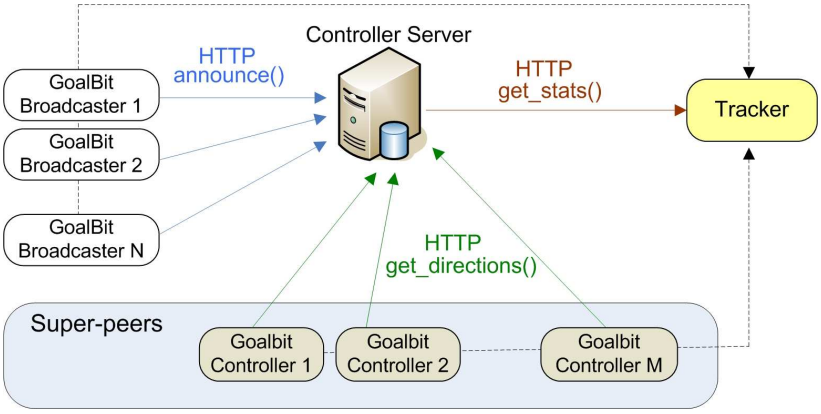


Figura 8.1: Arquitectura del sistema GoalBit Controller. Los 4 componentes de la red y sus relaciones son presentadas en esta imagen (los *broadcasters*, los controladores GoalBit, el *tracker* y el servidor de control).

8.1.1 El Broadcaster

Si bien las principales funcionalidades del *broadcaster* son las presentadas en la Subsección 5.1.1, al ser ejecutado dentro de la Plataforma GoalBit, este debe implementar las siguientes acciones:

- al comenzar la ejecución, el *broadcaster* debe registrar el archivo GoalBit que está transmitiendo en el servidor de control. De esta manera se dan de alta los contenidos a administrar en dicho servidor;
- a su vez el *broadcaster* debe anunciarse periódicamente al servidor de control con el fin de presentar actividad, si pasado más de una determinada cantidad de segundos, un *broadcaster* no se anuncia en el servidor, este último puede asumir que el primero ha suspendido su ejecución;
- finalmente, un *broadcaster* al terminar su ejecución debe reportar dicho evento al servidor de control, para que este libere los recursos previamente asignados a dicho *broadcaster*.

8.1.2 Los Controladores GoalBit

Los controladores GoalBit son procesos que se anuncian en el servidor de control y toman direcciones de este, como por ejemplo la de distribuir un cierto contenido o la de adaptar la señal para un cierto medio de distribución. Estos, al registrarse en el servidor, anuncian dos propiedades importantes: sus cualidades y su capacidad. Existen tres cualidades diferentes que un controlador puede implementar, estas son la de *pumper-peer* (llamada *PP*), la de *super-peer* (llamada *SP*) y la de *output-adaptor-peer* (llamada *OA*). Los *pumper-peers*, son *super-peers* que pueden ser asignados dinámicamente a los contenidos. Un controlador de tipo *super-peer*,

es asignado a un cierto contenido y este permanece asignado a dicho contenido hasta el final de la ejecución del controlador o hasta el final de la distribución del contenido (final de la ejecución del *broadcaster* del contenido en cuestión). Los *output-adaptor-peers*, son *super-peers* que tienen la posibilidad de transformar el *streaming* GoalBit a otro tipo de *streaming* (por ejemplo transcodificando la señal y distribuyéndola mediante HTTP). Al igual que los controladores de tipo *super-peers*, la asignación de estos a los contenidos es estática. Por otro lado la capacidad de un controlador se refiere a la cantidad de instancias de *super-peers* y/o *pumper-peers* y/o *output-adaptor-peers*, que el controlador puede ejecutar en paralelo. Los controladores GoalBit pueden poseer más de una cualidad, por ejemplo un controlador con cualidades "SP,PP,OA" y capacidad 5, puede ser asignado por el servidor de control a: 2 contenidos como *super-peer*, 2 contenidos como *pumper-peer* y a un contenido como *output-adaptor-peer*.

8.1.3 El *Tracker*

El *tracker* en lo referente a la administración de los *peers* conectados a cada contenido, lleva a cabo las mismas funcionalidades que fueron descriptas en la Subsección 5.1.4. Como funcionalidad extra, bajo la Plataforma GoalBit, este debe presentar un listado estadístico de todos los *peers* y *super-peers* conectados por cada contenido administrado. Este listado debe ser accesible desde la *Web*, mediante HTTP o HTTPS.

8.1.4 El Servidor de Control

Como comentamos anteriormente, el servidor de control mantiene el listado de todos los contenidos activos en un momento dado. A su vez, este lleva el registro de todos los controladores GoalBit existentes en el sistema, junto con sus cualidades y capacidades. Periódicamente el servidor de control consulta al o los *trackers* de los contenidos registrados, con el fin de obtener la cantidad de usuarios conectados a cada uno de estos. En base a esta información y a la cantidad de instancias y cualidades de los controladores, el servidor realiza una asignación de estos a cada uno de los contenidos. La asignación sigue los siguientes principios:

1. se asignan todos los *output-adaptor-peers* configurados por cada contenido (para esto se asignan controladores que incluyan la cualidad OA). Si quedan instancias de controladores libres se avanza al paso siguiente;

2. se asignan todos los *super-peers* configurados por cada contenido (para esto se asignan los controladores que incluyan la cualidad SP). Si quedan instancias de controladores libres se avanza al paso siguiente;

3. Finalmente en base a la información recogida de los *trackers* se asignan todos los *pumpers-peers* restantes a los contenidos (en este caso se asignan controladores que incluyan la cualidad PP).

Notar que al ejecutar este algoritmo pueden quedar instancias de controladores sin asignar, controladores que no implementen la cualidad de PP. Un diseño de este tipo puede ser usado para realizar asignaciones rápidas a los nuevos contenidos, sin tener la necesidad de esperar a desasignar *pumper-peers* para asignar los nuevos *super-peers* configurados.

8.2 Implementación

En esta sección se presentan los cambios realizados al cliente GoalBit con el fin de integrarlo a la Plataforma GoalBit, junto con algunos detalles acerca de la implementación del servidor de control.

8.2.1 Extensión del Cliente GoalBit

Extensión sobre el *Broadcaster*. En la Figura 8.2 se presenta el flujo de ejecución del proceso de *broadcasting* bajo la Plataforma GoalBit. Este es idéntico al presentado en la Subsección 5.4.1, salvo por la existencia del módulo *broadcaster_webinterface*. Al iniciar la ejecución, el módulo *broadcaster_manager* crea una instancia del *broadcaster_webinterface*, quien es el responsable de llevar a cabo toda la comunicación con el servidor de control. Inicialmente, el *broadcaster_webinterface* se registra en el servidor de control y le envía el archivo *.goalbit* a transmitir. Luego, durante todo el proceso de *broadcasting*, este mantiene una comunicación periódica con el servidor de control. Finalmente al terminar la distribución del contenido, el *broadcaster_webinterface* anuncia dicho evento al servidor de control, siendo luego cerrado.

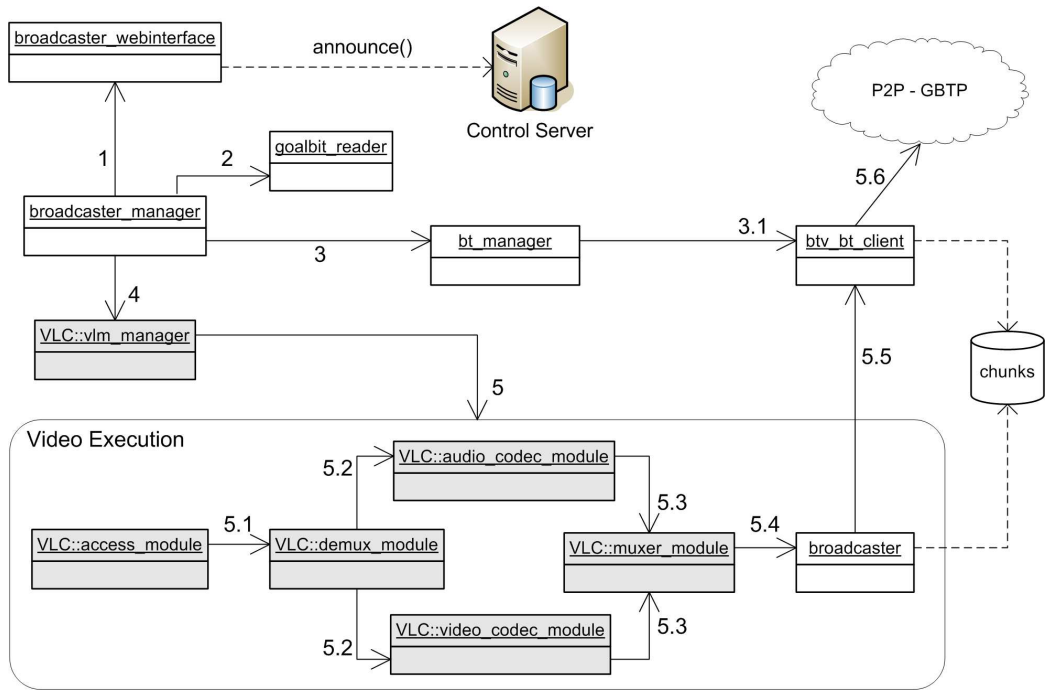


Figura 8.2: Extensión sobre el *broadcaster* al ser integrado a la Plataforma GoalBit.

Creación de los Controladores GoalBit. En la Figura 8.3 se presentan los módulos implicados en la ejecución de un controlador GoalBit. En este caso el módulo responsable del control

de la ejecución es el *controller_manager*. Muy resumidamente el proceso ejecutado por un controlador GoalBit es el siguiente:

1. el *controller_manager* se registra en el servidor de control y espera direcciones de este. Para esto, cada cierto intervalos de tiempo el módulo *controller_manager* consulta al servidor central;

2. una vez que el *controller_manager* ha recibido ordenes, este ejecuta dichas ordenes mediante el módulo *bt_manager*;

3. el *bt_manager* crea una instancia del módulo *goalbit_reader* con el fin de leer los archivos *.goalbit* a distribuir, los cuales son obtenidos por el *goalbit_reader* desde el servidor de control;

4. finalmente el *bt_manager* crea los módulos *btv_bt_client* correspondientes para llevar a cabo las ordenes del servidor de control. Notar que los módulos *btv_bt_client* son independientes entre sí (cada uno ejecuta un *streaming* GBTP diferente) y algunos de estos pueden tener un flujo de video asociado (este es el caso de los *output-adaptor-peers*).

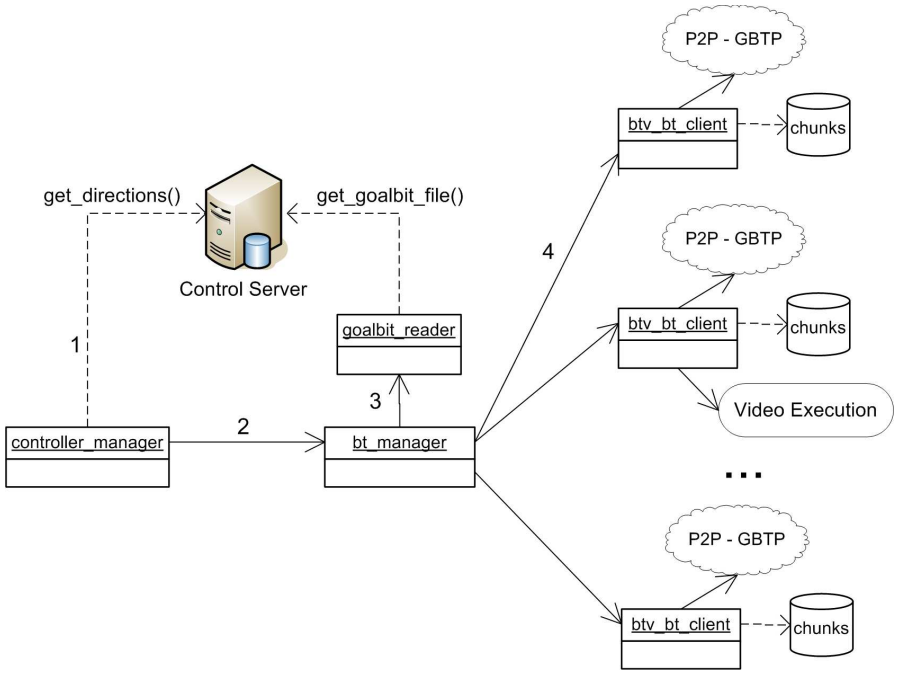


Figura 8.3: Diagrama de módulos de los controladores GoalBit.

8.2.2 Implementación del Servidor de Control

El servidor de control fue implementado en PHP [102] y usa bases de datos Berkeley [98] y MySQL [93]. El mismo se encuentra montado sobre un servidor *Web* Apache [10].

Interfaz Web. En la Figura 8.4 se presenta una vista de la interfaz *web* en donde se listan todos los controladores GoalBit registrados en el sistema, junto con sus capacidades (las anunciadas y las asignadas) y sus cualidades.

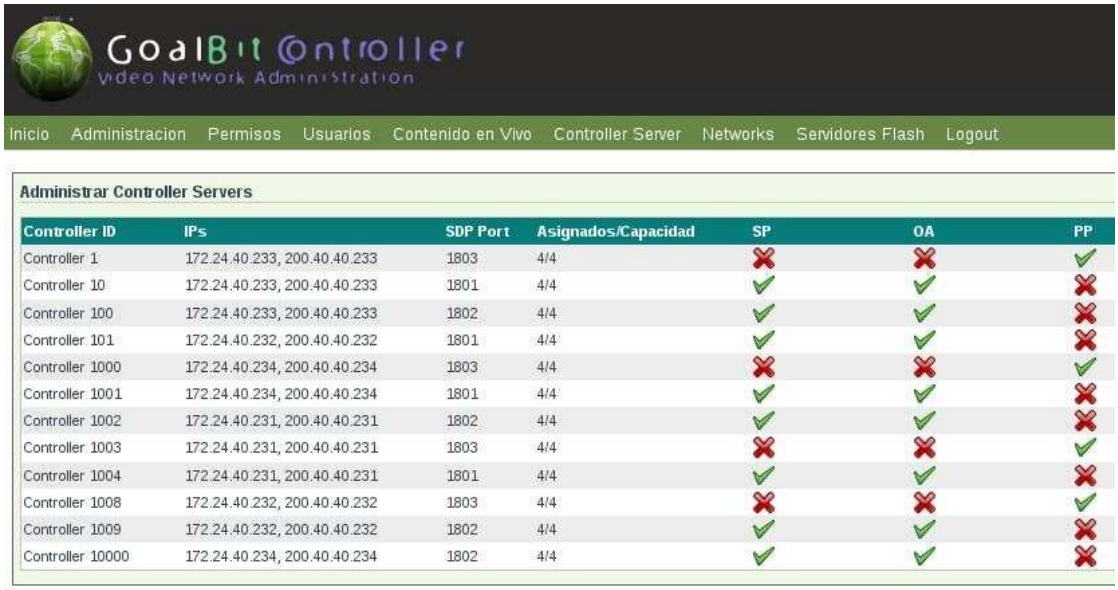


Figura 8.4: Interfaz *Web* del servidor de control: listado de controladores GoalBit registrados en el sistema.

La Figura 8.5 presenta el listado de los contenidos registrados en el sistema. Como se puede observar en ésta, el contenido "mimoviltv.com.uy" no se encuentra transmitiendo, por lo que no tiene ningún recurso asignado. También en este listado se pueden observar la cantidad de *peers* conectados a cada contenido y los *pumper-peers* asignados a estos.

En la Figura 8.6 se presenta el panel de administración para la configuración de un contenido. En este, se definen la cantidad de *super-peers* y de *output-adaptor-peers* a asignar a dicho contenido. Al asignar un *output-adaptor-peer*, se debe definir una línea con el tipo de transcodificación a aplicar (en caso de ser vacía no se aplica ninguna) y otra con el tipo de salida del *streaming* (por ejemplo un *streaming* HTTP/RTP/MMS, guardar en un archivo, etc.).

Detalles Técnicos. El tiempo transcurrido entre dos anuncios consecutivos de *broadcasters* o de controladores GoalBit es regulado por el servidor de control. Actualmente se deben reportar cada 30 segundos. Si pasado 120 segundos el servidor de control no tiene reportes de un controlador o de un *broadcaster*, este lo considera finalizado. Finalmente, el proceso de asignación

Broadcaster	XMLTV ID	Logo	Peers Conectados	Pumper-Peers (Asignados)	Super-peers (Asignados/Conf)	Output-adaptors (Asignados/Conf)	Editar Configuración
✗	mimoviltv.com.uy		0	0	0/2	0/2	
✓	tnu.com.uy		1	5	0/4	2/2	
✓	tvciudad.com.uy		0	0	3/3	2/2	
✓	vtv.com.uy		2	7	2/2	2/2	
✓	www.canal10.com.uy		1	4	4/4	2/2	

Figura 8.5: Interfaz *Web* del servidor de control: listado de contenidos (*broadcasters*) registrados en el sistema.

Administrar Configuración de vtv.com.uy

Agregar Super-peer

Super-peer ID	Redes	Asignado
Super-peer 30	Red Privada / Red Publica	
Super-peer 31	Red Privada / Red Publica	

Agregar Output Adaptor

Ouput-Adaptor ID	Redes	Opciones	Asignado
Output Adaptor 73	Red Privada / Red Publica	Output: rtp{dst=172.24.40.56,sdp=http://0.0.0.0:[SDP_PORT]/vtv.com.uy-73.sdp,port-audio=50012,port-video=50014} Transcoding:	
Output Adaptor 74	Red Privada / Red Publica	Output: rtp{dst=172.24.40.57,sdp=http://0.0.0.0:[SDP_PORT]/vtv.com.uy-74.sdp,port-audio=50012,port-video=50014} Transcoding:	

Figura 8.6: Interfaz *Web* del servidor de control: configuración de un contenido.

de recursos (asignación de controladores a los contenidos) es ejecutado cada 4 minutos.

8.3 Implantación en AdinetTV

A modo de prueba, el prototipo de la Plataforma GoalBit fue implantado en AdinetTV (ver Sección 2.3) con el fin de facilitar la administración de la granja de servidores GoalBit y a su vez con el fin de suministrar el contenido a una granja de servidores Flash (*Wowza Media Servers* [138]) para su posterior distribución a través del protocolo RTMP.

La Figura 8.7 presenta los diferentes componentes dentro del sistema. Los *broadcasters* toman la señal desde un *streaming* MMS en formato WMV/WMA/ASF y la transcodifican a H.264/AAC/ASF. Estos distribuyen dicha señal sobre una red interna (mediante GBTP) hacia la granja de *super-peers*. A su vez dentro de esta red interna se encuentra una granja de *output-adaptor-peers* quienes adaptan la señal GBPS/GBTP a una señal TS/RTP. El *streaming* TS/RTP es transmitido hacia la granja de servidores Flash. Finalmente, los usuarios al usar el *GoalBit Media Player* acceden a los *super-peers* de distribución, y al reproducir los contenidos desde un *Flash Player* (embebido en la *Web* de AdinetTV) acceden a la granja de servidores Flash.

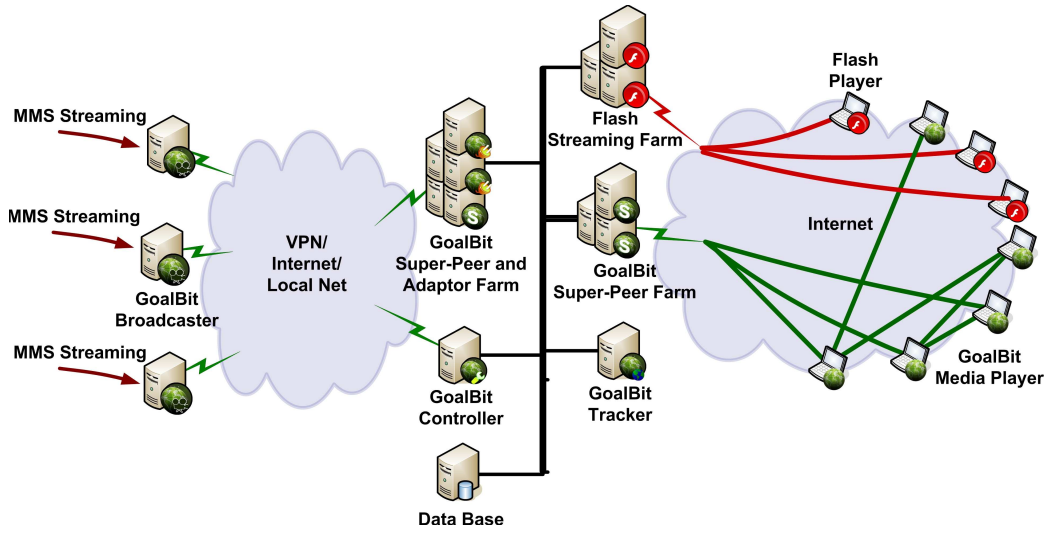


Figura 8.7: Implantación de la Plataforma GoalBit en AdinetTV.

Dentro de la Plataforma GoalBit de AdinetTV, cada contenido tiene configurado al menos dos *super-peers* (con el fin de asegurarse una distribución mínima) y dos *output-adaptor-peers*, quienes envían la señal a dos diferentes servidores Flash para su posterior distribución a los usuarios finales. Finalmente, existen 16 instancias de *pumper-peers* las cuales son distribuidas en los diferentes contenidos según su carga (cantidad de usuarios conectados).

8.4 Aplicación y Utilidad

En esta sección se presenta la necesidad existente en las redes de distribución de contenidos, y en particular, en las de distribución de contenidos en vivo, de implementar políticas de asignación dinámica de recursos según la demanda de sus usuarios. Mediante este hecho se destaca la aplicación y la utilidad de la Plataforma GoalBit, descripta en este capítulo.

El principal factor en contra de una asignación estática de recursos es el hecho de la gran variabilidad en las conexiones de usuarios a los diferentes contenidos. En la Figura 8.8, se presenta el total de conexiones de usuarios a los contenidos en vivo de AdinetTV, distribuidos desde la granja de servidores de *Windows Media Services* (*streaming* MMS), en el período desde el primero de Febrero al primero de Junio del 2010. Como se puede observar en dicha figura, la cantidad de conexiones varía mucho de un día a otro. En ciertas transmisiones existen una gran cantidad de usuarios conectados y en otras es muy baja.

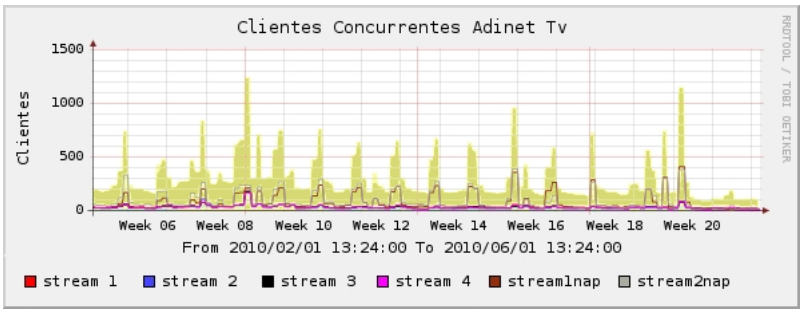


Figura 8.8: Total de conexiones de usuarios a los contenidos en vivo de AdinetTV, distribuidos desde la granja de servidores de *Windows Media Services* (*streaming* MMS), en el período desde el primero de Febrero al primero de Junio del 2010.

Ante este problema, y ante la falta de una asignación dinámica de los recursos, la solución a la que recurren las redes de distribución de contenido, y en particular AdinetTV, es la de disponer de servidores de *streaming* dedicados para los contenidos con mayor popularidad. De esta manera ante grandes ráfagas de conexiones a un cierto contenido, existen servidores cuya única función es la de servir el contenido en cuestión. En la Figura 8.9 se presentan las conexiones de usuarios a un servidor *Windows Media Services* dedicado a servir uno de los contenidos más populares de AdinetTV, en la semana del 12 al 18 de Mayo del 2010. A su vez, en la Figura 8.10 se presenta el ancho de banda consumido por dicho servidor para llevar a cabo la distribución del contenido a los usuarios. Observando estas figuras es sencillo ver uno de los problemas de la asignación estática de servidores a contenidos: por momentos estos se encuentran muy cargados, pero en otros momento, estos se encuentran sumamente osciosos, desperdiciando la capacidad de responder ante conexiones de usuarios a otros contenidos. Por otro lado, muchas veces ocurre que ante eventos de gran popularidad, la asignación es tal, que no permite cubrir con la demanda de los usuarios, más alla de que quizás considerando el total de recursos en la granja de servidores, éste sea suficiente. Por este motivo, una asignación

dinámica de los recursos es más que necesaria, la cual es brindada de forma muy eficiente por la Plataforma GoalBit, evitando estos problemas mediante la presentación de una asignación óptima de los recursos en todo momento.

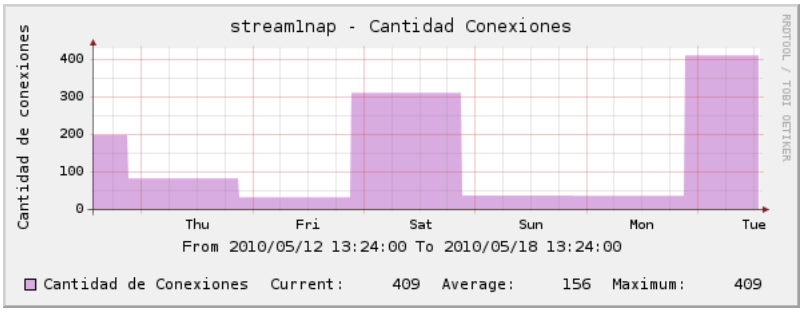


Figura 8.9: Conexiones de usuarios a un servidor *Windows Media Services* dedicado a servir uno de los contenidos más populares de AdinetTV, en la semana del 12 al 18 de Mayo del 2010

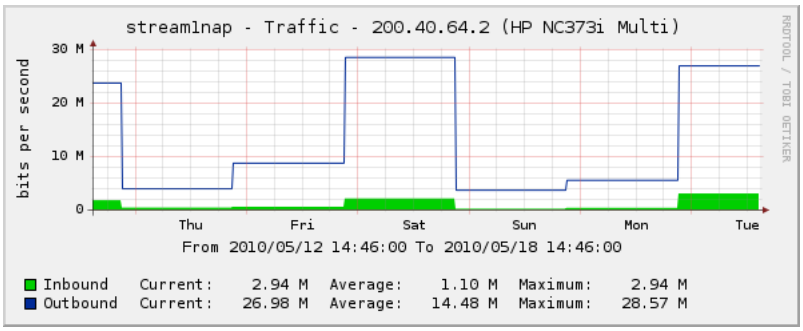


Figura 8.10: Ancho de banda consumido por un servidor *Windows Media Services* dedicado a servir uno de los contenidos más populares de AdinetTV, en la semana del 12 al 18 de Mayo del 2010

Capítulo 9

Monitoreo de la Calidad de Experiencia en GoalBit

En este capítulo se presenta el monitoreo de la calidad de experiencia realizado sobre los *streamings* GoalBit, llevado a cabo mediante la aplicación de la técnica PSQA.

Tal como presentamos en [usa-ipom07, acm-per08] y mostramos en [usa-sigm08], es posible realizar un monitoreo de la QoE por medio de PSQA, de forma automática y en tiempo real, sobre cualquier tipo de red de distribución de video (sin importar el protocolo de transporte, ni la codificación, ni la multiplexación). Para esto, es necesario insertar cierta información de usuario dentro del contenido a distribuir, lo cual es posible dado que usualmente las especificaciones de codificación de video soportan este tipo de acciones. Tanto en un escenario de video bajo demanda, como en uno de video en vivo, esta información puede ser insertada en un proceso de post-codificación previo a la distribución del contenido (notar que no existen restricciones sobre el *encoder*, ni sobre el servidor de distribución a usar).

A continuación se brindan detalles acerca de la integración del enfoque presentado en [usa-ipom07, acm-per08] en el *streaming* GoalBit, en donde se aplicó la función de PSQA descripta en la Subsección 4.3.2. En la Sección 9.1 se presenta la arquitectura y el diseño de la solución. La Sección 9.2 presenta detalles sobre la implementación de este monitoreo. Finalmente, en la Sección 9.3 se brindan algunos de los resultados obtenidos en el monitoreo de los contenidos GoalBit de AdinetTV (ver la Sección 2.3).

9.1 Arquitectura y Diseño del Monitoreo

Arquitectura. Como se puede observar en la Figura 9.1, los componentes involucrados en el monitoreo de la calidad de experiencia, son tres: el *broadcaster*, los *peers* y el *tracker*. El *broadcaster* es el encargado de insertar dentro del *streaming* la información necesaria para que los *peers* puedan calcular su PSQA. Los *peers* reciben el *streaming* GoalBit, capturan la información de usuario insertada por el *broadcaster*, calculan el PSQA y lo reportan al *tracker* (en el próximo anuncio a este). El *tracker* mantiene registro de la calidad de experiencia reportada por los *peers* en cada contenido administrado.

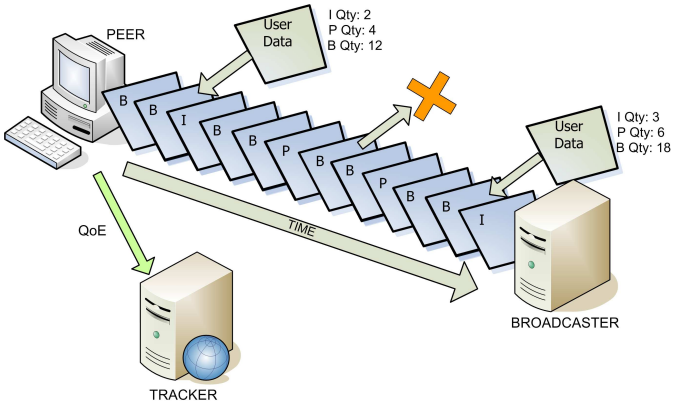


Figura 9.1: Arquitectura del monitoreo de la QoE sobre GoalBit. En este ejemplo se muestra como el *broadcaster* inserta los datos de usuario dentro del *streaming*. Cada dato de usuario contiene la cantidad absoluta de *frames* generados por el broadcaster desde el comienzo de la transmisión. El peer lleva la cuenta de los *frames* recibidos y la compara con la información incluida en los datos de usuario, detectando de esta manera pérdidas de *frames*. Estas pérdidas son usadas para calcular el PSQA, el cual es enviado al *tracker*.

Diseño. La Tabla 9.1 recuerda los parámetros definidos en la función PSQA presentada en la Subsección 4.3.2, la cual es usada para calcular la calidad de experiencia en el monitoreo de la red GoalBit.

Parámetros de fuente
Tamaño del GOP
Peso de los *P-Frames* por GOP
Parámetros de la transmisión
Tasa de pérdidas de *I-Frames*
Tasa de pérdidas de *P-Frames*
Tasa de pérdidas de *B-Frames*

Cuadro 9.1: Parámetros de la función PSQA aplicada en el monitoreo de los *streamings* GoalBit.

Los parámetros de fuente (tamaño del GOP y peso de los *P-Frames* por GOP) deben ser insertados por el *broadcaster* como información de usuario, dado que en caso de pérdidas los *peers* no tienen como conocer dicha información. A su vez, con el fin de poder calcular las pérdidas de los diferentes tipos de *frames*, el *broadcaster* inserta contadores de la cantidad absoluta de *frames* generados desde el comienzo de la transmisión. En base a esta información, el algoritmo que usan los *peers* para calcular las pérdidas de *frames* es muy sencillo y eficaz, este consiste en contabilizar la cantidad de *frames* recibidos según su tipo, y luego comparar estos contadores con los agregados por el *broadcaster*. Observando la Figura 9.1, el *peer* recibe la primer información de usuario en donde se indica que el *broadcaster* generó 2 *I-Frames*, 4

P-Frames y 12 *B-Frames*, al recibir la segunda información de usuario, comparándola con la anterior, el *peer* conoce que el *broadcaster* en ese período generó 1 *I-Frame*, 2 *P-Frames* y 6 *B-Frames*. Suponiendo que en la transmisión se perdió un *B-Frames*, los contadores de la cantidad de *frames* recibidos por el *peer* serían: 1 *I-Frame*, 2 *P-Frames* y 5 *B-Frames*. Finalmente, al comparar estos valores con los enviados por el *broadcaster*, detectamos la pérdida del *B-Frame*.

A modo de resumen, en la Tabla 9.2 se presenta la información insertada en el *streaming* por el *broadcaster*. Notar que el *broadcaster* inserta dicha información una única vez por cada GOP. En caso de que esta se pierda (por ejemplo debido a problemas en la red), los *peers* acumularán los resultados hasta que llegue el próximo dato de usuario, es decir, que no existe pérdida de información, inclusive si eventualmente se perdiera algún dato de usuario.

Tamaño del GOP
Peso de los *P-Frames* por GOP
I-Frames enviados desde el comienzo de la ejecución
P-Frames enviados desde el comienzo de la ejecución
B-Frames enviados desde el comienzo de la ejecución

Cuadro 9.2: Información insertada en el *streaming* por el *broadcaster*.

9.2 Implementación

Respecto a la implementación de este sistema dentro del cliente GoalBit, en la Figura 9.2 se presenta las interacciones entre módulos llevada a cabo durante el proceso de *broadcasting*. Este proceso es muy similar al presentado en la Subsección 5.4.1, salvo por el agregado del módulo *qoe_injector_duplicate* luego de la salida de los módulos de codificación y previo al módulo de multiplexación. Este módulo es el responsable de: mantener los contadores por tipo de *frame*, calcular las propiedades de la fuente (tamaño del GOP y peso de los *P-Frames* por GOP) e insertar dicha información dentro del *streaming* a distribuir.

A su vez la Figura 9.3 presenta el proceso de reproducción del *streaming* GoalBit. Al igual que en el caso anterior éste es muy similar al presentado en la Subsección 5.4.1. Dentro de este proceso, se agrega el módulo *qoe_extractor_duplicate* luego de los módulos de decodificación y previo a los módulos de presentación. El *qoe_extractor_duplicate* captura la información insertada por el *broadcaster*, contabiliza los *frames* recibidos y calcula el PSQA obtenido. El valor del PSQA es enviado al módulo *btv_bt_client*, quien a su vez lo reporta al *tracker*.

Notar que únicamente los *peers* que tienen un flujo de video asociado son quienes reportan la QoE (dado que en la reproducción del *streaming* es en donde se realiza dicho cálculo). Esto genera que los *super-peers* y los *pumper-peers* no reporten su calidad de experiencia al *tracker*. Si se desea mantener ciertos monitores en la red, diferentes de los propios *peers*, se pueden configurar *output-adaptor-peers* con una salida nula (es decir que únicamente ejecuten el proceso de video, descartando su salida).

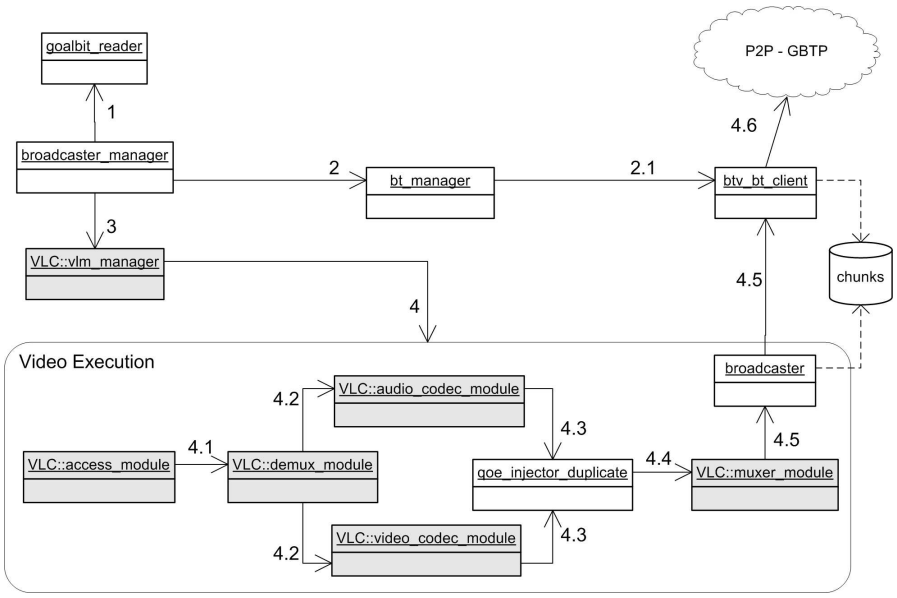


Figura 9.2: Interacción entre módulos llevada a cabo durante el proceso de *broadcasting* al medir la QoE.

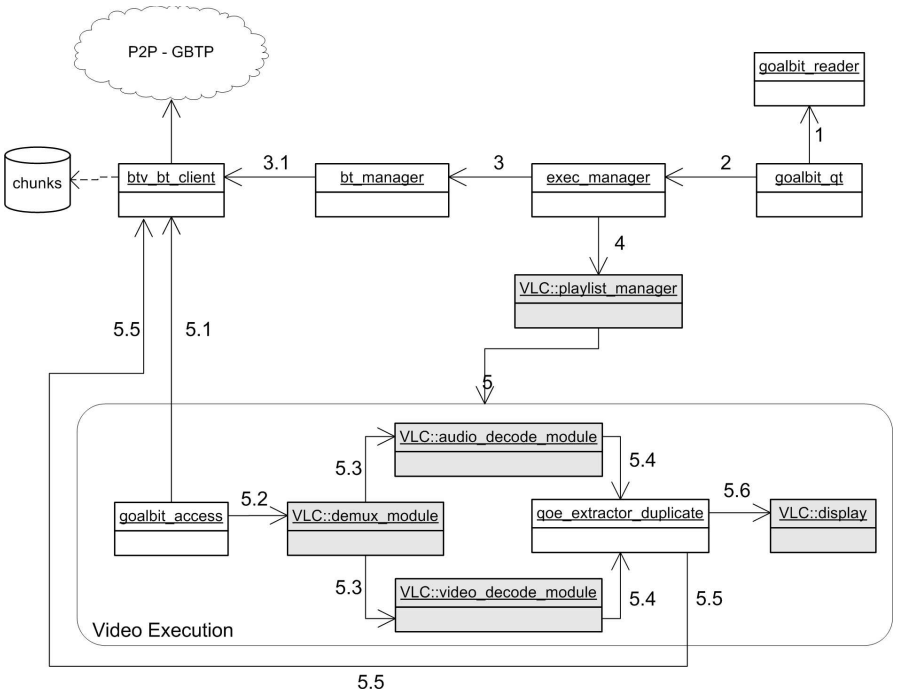


Figura 9.3: Interacción entre módulos llevada a cabo durante la reproducción de un *streaming* GoalBit al medir la QoE.

9.3 Resultados Obtenidos

El sistema de monitoreo se encuentra integrado en todos los clientes GoalBit, el hecho que los clientes reporten o no la calidad de experiencia al *tracker*, depende únicamente de la existencia dentro del *streaming* de la información necesaria para calcular dicho valor. En otras palabras, el *broadcaster* es quien define si se reportará o no la QoE del *streaming* a ser distribuido.

En AdinetTV, todos los *broadcasters* insertan la información de usuario necesaria para calcular la calidad de experiencia. A su vez, tal como se presentó en la Sección 8.3, cada contenido tiene dos *output-adaptor-peers* configurados, por lo que se tiene un monitoreo constante de la calidad de experiencia en cada uno de los contenidos de AdinetTV (más allá de la cantidad de *peers* conectados).

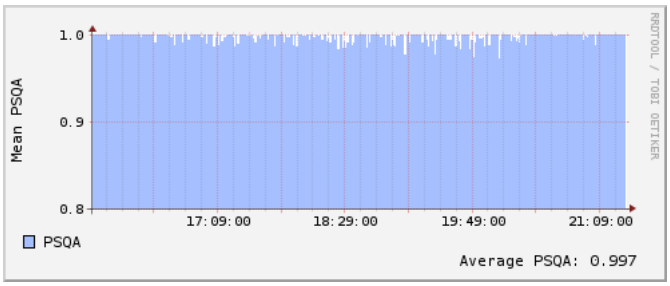


Figura 9.4: Evolución del PSQA promedio durante la transmisión de un evento popular.

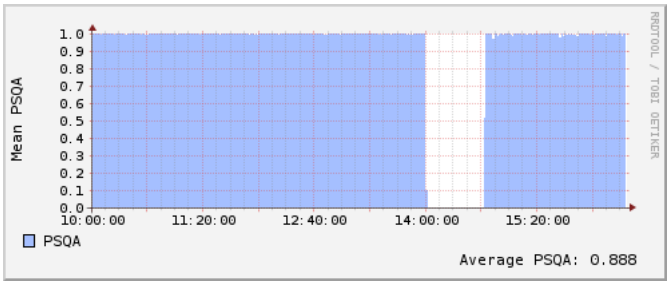


Figura 9.5: Evolución del PSQA promedio durante la transmisión de un *streaming* con problemas.

En la Figura 9.4 se presenta la evolución del PSQA promedio medido por el *tracker* durante más de 5 horas de la transmisión del evento popular presentado en la Sección 7.1 (2 horas y media previas al evento, más 2 horas del evento, más 1 hora posterior al evento). Como se puede observar en dicha figura, al comienzo el PSQA se mantiene muy estable cerca de 1. Aproximadamente 1 hora antes de comenzar el evento, el PSQA promedio comienza a tener pequeñas variaciones debido al aumento en la cantidad de usuarios. Claramente, luego de finalizado el evento, el PSQA promedio se vuelve a acercar a 1. Durante todo el período de captura el PSQA promedio valió 0,997, lo que significa que los usuarios vieron el video con

una calidad excelente.

Por otro lado, la Figura 9.5, presenta un caso en donde se puede observar problemas con la fuente (problemas con la señal de ingreso al *broadcaster* o problemas con el *broadcaster* en sí mismo). Esta figura presenta más de 7 horas de transmisión de un *streaming* no muy popular, en donde por más de media hora los *peers* reportaron un PSQA de 0. Finalmente la señal se recompuso y los *peers* comenzaron nuevamente a reportar un PSQA cercano a 1.

Con estos dos casos sencillos se puede observar la potencialidad de la herramienta PSQA aplicada en GoalBit. Gracias a su utilización es posible detectar problemas en la red de distribución de video, de forma automática e inmediata. Esto es una mejora sustancial a las opciones de monitoreo disponibles en la actualidad (ver Sección 4.4). En el futuro se podrán aplicar acciones correctivas en base a esta información.

Capítulo 10

Monitor Goalbit: un Monitor Genérico para la Distribución de Video

Con el fin de completar nuestro prototipo de red de distribución de contenidos, en este capítulo presentamos el diseño y desarrollo de un monitor de video genérico, llamado *Monitor GoalBit*, mediante el cual es posible monitorear el estado tanto de contenidos en vivo, como de contenidos bajo demanda, independientemente de sus protocolos de *streaming* y de sus codificaciones.

Este monitoreo se basa en la técnica de *probing*, en donde se ejecuta un conjunto de *procesos chequeadores* que representan a usuarios finales en ciertas ubicaciones de la red, con el fin de capturar y reportar el estado y las características de los diferentes *streamings* de video. Notar la similitud de nuestro enfoque con las grandes soluciones de monitoreo presentadas en la Sección 4.4, en donde sustituimos los dispositivos de *probing* (*hardware* específico) por procesos de chequeo, implementados en *software*, que pueden ser ejecutados en cualquier servidor genérico, logrando así una solución más simple y menos costosa en términos enconómicos.

Vale remarcar la diferencia de enfoque en el uso de esta herramienta de monitoreo respecto a la presentada en el capítulo anterior. Mientras el capítulo anterior detalla una herramienta de monitoreo activa, distribuida a todos los usuarios que utilizan el *streaming* GoalBit, en este capítulo se presenta un monitor pasivo, instalable en determinados puntos de la red, capaz de monitorear cualquier tipo de *streaming* (GoalBit, MMS, RTMP, etc.) sin la necesidad de disponer de los datos de usuario insertados en la etapa de post-codificación. Un monitor pasivo automático es imprescindible en redes de distribución de mediano y gran porte. Supóngase el caso de AdinetTV, donde se ofrecen 26 canales en vivo y 700 videos. Estos contenidos son distribuidos desde múltiples servidores, una decena aproximadamente, por lo tanto es necesario chequear miles de puntos de acceso al contenido ($(700+26)\times 10$), con el fin de revisar si alguno puede estar fallando.

En la Sección 10.1 se presenta la arquitectura del Monitor GoalBit. La Sección 10.2 presenta detalles sobre la implementación de esta solución. Finalmente, en la Sección 10.3, y a modo de ejemplo de la generalidad y potencialidad de este monitor, se presenta un completo análisis de los resultados obtenidos al realizar un amplio monitoreo de los videos presentados en YouTube [145].

10.1 Arquitectura

Como se puede observar en la Figura 10.1, se definen N *colas de prioridad*. Cada cola de prioridad almacena X_N contenidos a monitorear. Cada contenido es ingresado en una única cola. A su vez, cada cola de prioridad tiene asociado M_N *procesos chequeadores*. Los procesos chequeadores ejecutan continuamente el siguiente algoritmo:

1. el proceso chequeador toma el primer contenido de la cola a la que éste se encuentra asignado;
2. dicho contenido es procesado, obteniendo las características del *streaming* (*codec* de video, *codec* de audio, *muxer*, *bitrate*, etc.) y el estado del mismo (chequeando si el video y el audio de éste se encuentran activos);
3. finalmente el proceso chequeador registra el resultado del procesamiento en una base de datos e inserta nuevamente el contenido al final de la cola correspondiente.

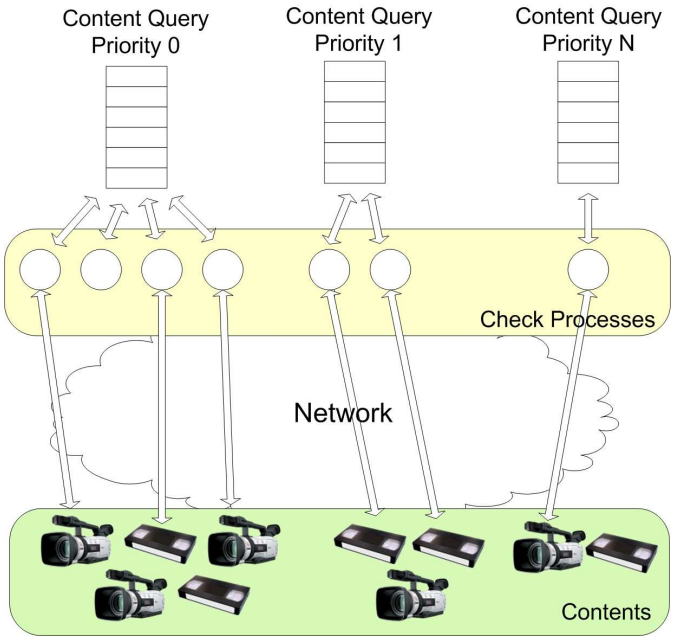


Figura 10.1: Arquitectura del Monitor GoalBit.

Esta arquitectura permite un alto nivel de flexibilidad a la hora de definir la frecuencia de chequeo de cada contenido (variando los parámetros N, X_N y M_N). Por ejemplo, se pueden crear dos diferentes colas, una con 10 procesos chequeados y otra con 5 ($N = 2, M_1 = 10, M_2 = 5$), ubicando los contenidos de mayor importancia en la primera y dejando a los contenidos menos relevantes en la segunda. De esta manera, los contenidos de la primer cola serán chequeados con una mayor frecuencia que los de la segunda.

10.2 Implementación

La implementación del Monitor GoalBit, consiste en 2 diferentes componentes: uno referente a la ejecución del monitoreo y otro referente a la administración y presentación de los resultados.

El diseño de base de estos componentes asume que los contenidos a monitorear pueden tener varias fuentes, o en otras palabras, que un video o un canal de televisión puede tener diferentes puntos de acceso al *streaming*. Con el fin de representar este concepto se extendió la definición de los archivos GoalBit, en donde para cada contenido dentro de estos archivos, se agregó un listado opcional, de otros puntos de acceso al contenido (por ejemplo *streamings* HTTP, RTP, MMS, etc.).

El ingreso de los contenidos al sistema puede ser realizado de dos formas diferentes: manualmente o mediante un módulo de *web-crawling*. Para agregar un conjunto de contenidos al sistema de forma manual, se debe generar un archivo GoalBit con todos los puntos de acceso a los contenidos a monitorear (notar que el archivo GoalBit por cada contenido puede contener: un único *streaming* GBTP, un único *streaming* no GBTP o un conjunto de estos) e insertar dicho archivo GoalBit en el sistema mediante la interfaz provista por el componente gráfico. También es posible desarrollar un módulo de *web-crawling*, el cual explore un cierto sitio *Web*, extraiga las URLs de los *streamings* contenidos en dicho sitio y los inserte en la base de datos de nuestro sistema. Este es un módulo de auto-suministro de contenidos, el cual puede ser ejecutado cada ciertos períodos de tiempo, actualizando la información provista al sistema.

A continuación se presentan detalles sobre el componente de ejecución del monitoreo y el componente de administración y presentación de los resultados.

10.2.1 Componente de Ejecución del Monitoreo

El componente de ejecución del monitoreo es el responsable de llevar a cabo los procesos de análisis y chequeo de los contenidos ingresados en el sistema. Como se puede observar en la Figura 10.2, este se encuentra compuesto por un proceso llamado *channel_checkers_manager* y múltiples instancias de procesos *channel_checker*. El proceso *channel_checkers_manager* es el responsable de instanciar y controlar los procesos *channel_checker*. A su vez, los procesos *channel_checker* son los que ejecutan el algoritmo presentado en la sección anterior (Sección 10.1), en donde estos toman un contenido de una cola, lo analizan, insertan los resultados en una base de datos y lo devuelven a dicha cola. Los procesos *channel_checker* instancian clientes GoalBit, con el fin de analizar y chequear los contenidos.

Tanto el proceso *channel_checkers_manager* como los procesos *channel_checker* fueron implementados en PHP [102] y usan bases de datos Berkeley [98] y MySQL [93].

La idea detrás del monitoreo de un cierto contenido es muy simple, y consiste en conectarse unos segundos a dicho contenido, mediante un cliente GoalBit, verificar la presencia de audio y video dentro del *streaming*, y capturar las propiedades del mismo (especificación de video usado, especificación de audio, especificación de multiplexación, *bitrate* del video y resolución del video).

El proceso ejecutado dentro del cliente GoalBit al ejecutar el monitoreo de un *streaming* GBTP es presentado en la Figura 10.3. Como se puede observar en esta, el proceso es muy similar al proceso de reproducción de un *streaming* GoalBit (presentado en la Subsección 5.4.1),

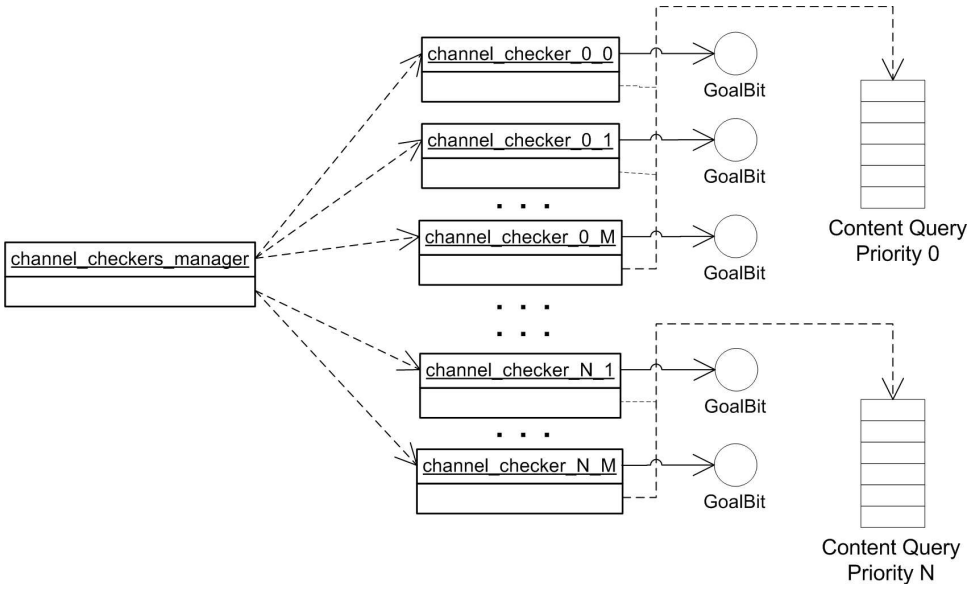


Figura 10.2: Procesos ejecutados dentro del componente de ejecución del monitoreo.

salvo por la presencia de algunos módulos nuevos. En este caso, el principal controlador de la ejecución es el *goalbit_monitor* y se llevan a cabo los siguientes pasos:

1. el *goalbit_monitor* crea una instancia del módulo *goalbit_reader*, con el fin de leer el archivo *.goalbit* a ejecutar;

2. el *goalbit_monitor* crea una instancia del módulo *bt_manager*, quien a su vez, instancia un módulo *btv_bt_client*, con el fin de obtener el contenido desde la red P2P;

3. el *goalbit_monitor* crea una instancia del *VLC::vlm_manager* (módulo propio del VLC), con el fin de iniciar el flujo de ejecución del video;

4. el *VLC::vlm_manager* mediante el uso de la *libvlc*, instancia todos los módulos necesarios para llevar a cabo el flujo de ejecución del video y el chequeo del mismo. Estos son: el módulo *goalbit_access*, un módulo de demultiplexación, los módulos de decodificación, el módulo *monitor_duplicate* y un módulo *dummy*. Dentro del flujo de ejecución del video existen las siguientes interacciones:

   4.1 el *goalbit_access*, le pide piezas al *btv_bt_client*;

   4.2 el *goalbit_access* le pasa el contenido adquirido al módulo de demultiplexación;

   4.3 el módulo de demultiplexación (alguno de los módulos de tipo “demux” del VLC) demultiplexa el *stream*, generando los flujos elementales (uno para el audio y otro para el video);

4.4 los módulos de decodificación (algunos de los módulos de tipo "decoder" del VLC) decodifican los flujos elementales por separado y se los pasan al módulo *monitor_duplicate*.

4.5 el *monitor_duplicate*, cuenta la cantidad de *bytes* de audio y de video recibidos y los escribe en un archivo en disco. Tanto el audio como el video es pasado a un módulo *dummy* del VLC, quien descarta dichos flujos;

5. finalmente el *goalbit_monitor*, a través de la *libvlc* obtiene las propiedades del *streaming* en cuestión y obtiene una captura del video recibido. Ambos elementos son escritos en disco.

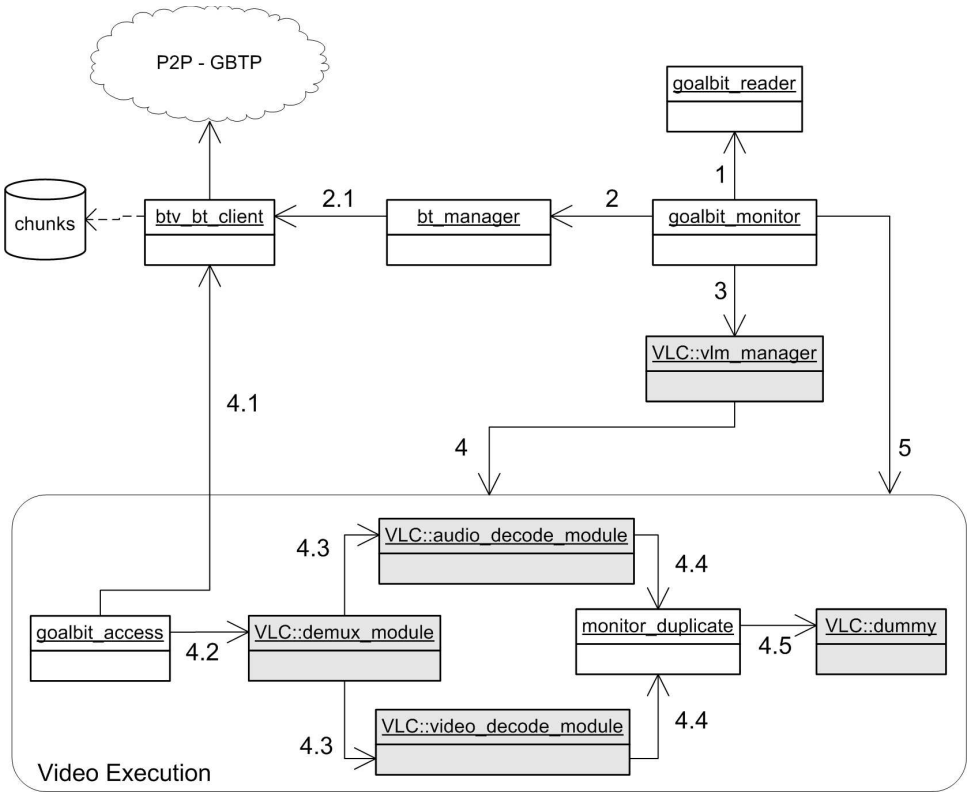


Figura 10.3: Proceso ejecutado dentro del cliente GoalBit al realizar el monitoreo de un *streaming* GBTP.

Como se puede observar en la Figura 10.4, el proceso llevado a cabo al realizar el monitoreo de un *streaming* genérico (no GBTP), es muy similar al presentado anteriormente, con la diferencia que en este caso se usa un módulo de acceso propio del VLC para obtener el *streaming* (en lugar del módulo *goalbit_access*, quien consume piezas del *btv_bt_client*).

Finalmente los procesos *channel_checker*, luego de terminada la ejecución de un cliente GoalBit, toman los resultados generados por este (un archivo con las características del *streaming*, un archivo con los *bytes* de audio y de video recibidos y una captura del video) y los

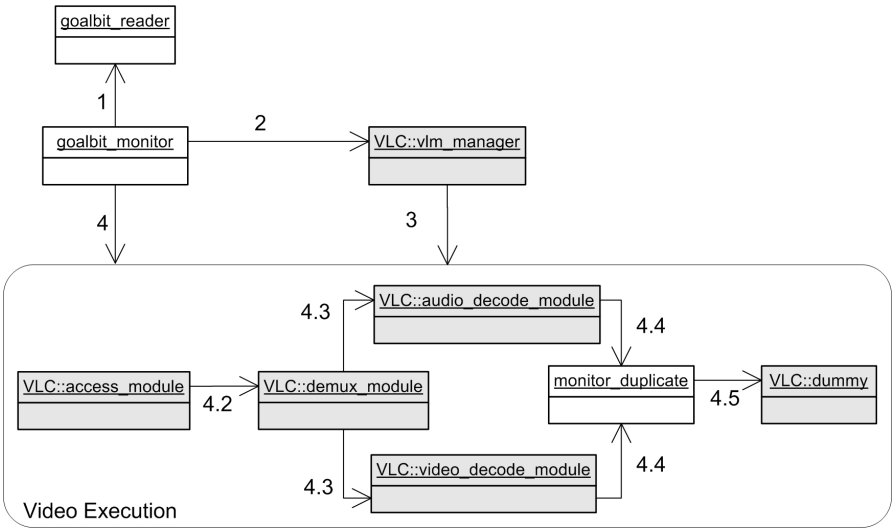


Figura 10.4: Proceso llevado a cabo al realizar el monitoreo de un *streaming* genérico.

almacenan en la base de datos, haciéndolos accesibles para su posterior presentación mediante el componente gráfico.

A continuación se presentan las propiedades capturadas por cada uno de los *streamings* a monitorear.

- Cantidad de *bytes* de video recibidos.
- Captura de una imagen del video.
- Codec de video.
- *Bitrate* de video.
- Resolución del video.
- Cantidad de *bytes* de audio recibidos.
- Codec de audio.
- *Bitrate* de audio.
- Muxer del *streaming*.

10.2.2 Componente de Administración y Presentación de los Resultados

La administración del sistema y la muestra de los resultados obtenidos en el monitoreo fue realizada en base a una interfaz *Web* desarrollada con PHP [102] y montada sobre un servidor Apache [10].

En la Figura 10.5 se presenta una vista de la interfaz cuando no se ha detectado ningún error en los contenidos monitoreados.

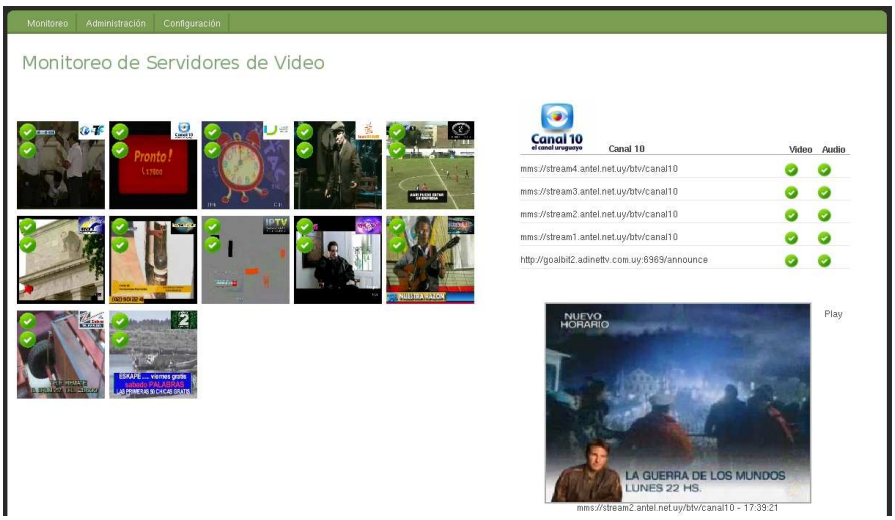


Figura 10.5: Interfaz del Monitor GoalBit al no detectar ningún error en los contenidos monitoreados.

La Figura 10.6 muestra la interfaz al detectar un error sobre un *streaming* (en este caso no se detectó ni audio ni video en un canal en vivo).

Finalmente la Figura 10.7 muestra la interfaz *Web*, al desplegar el estado de todas las fuentes de un cierto contenido. En esta, se presenta para cada fuente, la existencia o no del audio y del video, junto con la fecha en que se realizó la última evaluación.

10.3 Resultados Obtenidos: un Análisis sobre YouTube

Con el fin de presentar el potencial del Monitor GoalBit, en Octubre del 2009, llevamos a cabo un profundo y detallado monitoreo sobre YouTube [145], la mayor red de distribución de videos bajo demanda de la actualidad. Para esto, se implementó un módulo de *web-crawling*, el cual explora el portal *Web* capturando los videos existentes, junto con sus puntos de acceso. A su vez, este módulo captura como dato extra, la cantidad de vistas de los videos, dato que usaremos en el análisis presentado a continuación.

Con el fin de establecer una nomenclatura en común, vamos a llamar “contenido” al concepto que engloba una secuencia de video, junto con un nombre, una descripción, una cantidad de vistas, etc. (al concepto llamado “video” en YouTube). A su vez llamaremos “video” a una codificación concreta de la secuencia de video de un contenido (usando una especificación de audio, una de video y una de multiplexación, bajo un cierto *bitrate*). Notar que un contenido al menos debe tener un video que lo represente, pudiendo tener múltiples de estos.

Respecto al monitoreo, y como se puede observar en la Tabla 10.1, se capturaron 31159 contenidos diferentes. En promedio, cada contenido tiene 2,74 videos (normalmente existen 2

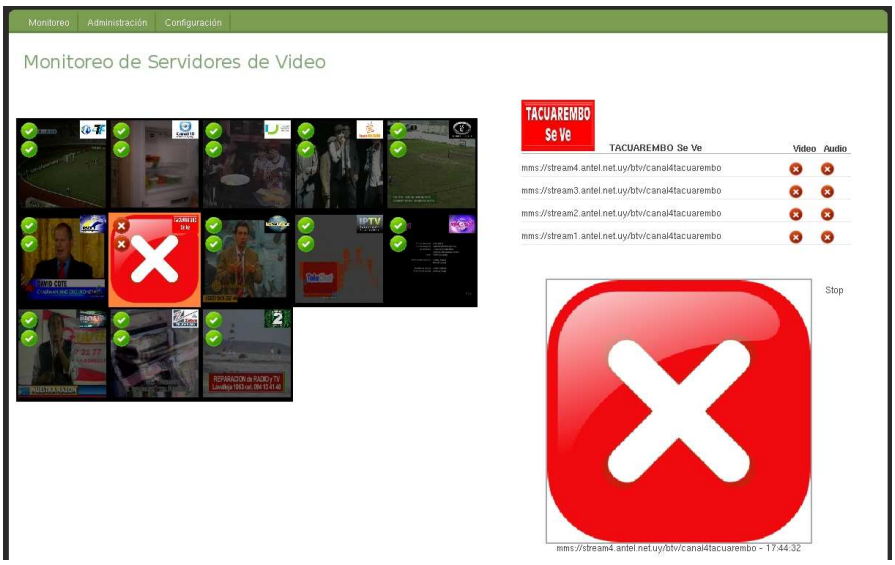


Figura 10.6: Interfaz del Monitor GoalBit al detectar un error sobre un *streaming*.

Figura 10.7: Despliegue del estado de todas las fuentes de un cierto contenido.

o 3 codificaciones diferentes por cada contenido). Por lo tanto, se analizaron un total de 85205 videos, de los cuales 861 (un 1,01 %) no fue posible reproducirlos (no se recibió ningún *byte* de audio ni de video).

Cuadro 10.1: Datos generales del monitoreo realizado sobre YouTube.

Cantidad de contenidos capturados:	31159
Cantidad de codificaciones por contenido:	2.74
Cantidad de videos analizados:	85205
Cantidad de videos con error:	861
Porcentaje de videos con error:	1.01 %

En la Tabla 10.2 se presentan las principales codificaciones de los contenidos usadas en YouTube. El 44,4 % de los videos analizados fueron de tipo H.264/MP4A/FLV, con un *bitrate* promedio de 590 Kbps. Dentro de este tipo de codificación, las resoluciones más usadas son: 320x240 (el 24.3 % de los videos de este tipo), 640x360 (el 16,1 %), 640x480 (el 15,4 %), 480x360 (el 10,8 %) y 854x480 (el 10,2 %), bajo una tasa de codificación promedio de 302 Kbps, 729 Kbps, 712 Kbps, 647 Kbps y 1040 Kbps respectivamente. El 35,9 % de los videos fueren de tipo FLV1/MP3/FLV, codificados a un *bitrate* promedio de 329 Kbps. Dentro de los videos de este tipo, las resoluciones más usadas son: 320x240 (el 56,5 %), 400x226 (el 20,3 %), 320x180 (el 6,0 %) y 320x214 (el 5,1 %), codificadas a un *bitrate* promedio de 316 Kbps, 370 Kbps, 333 Kbps y 329 Kbps respectivamente. Finalmente el 18,5 % de los videos analizados se corresponden con la codificación AVC1/MP4A/MP4, con un *bitrate* promedio de 856 Kbps. A su vez, dentro de este tipo de codificación, las resoluciones más usadas son: 480x360 (el 41,0 %), 1280x720 (el 23,2 %), 480x270 (el 15,3 %) y 480x320 (el 7,0 %), bajo una tasa promedio de 706 Kbps, 1211 Kbps, 610 Kbps y 680 Kbps respectivamente.

Como se pueden observar en estos resultados, los videos con mayores resoluciones son codificados en AVC1/MP4A/MP4, en el otro extremo, los videos de tipo FLV1/MP3/FLV, son videos con resoluciones pequeñas, codificados a bajos *bitrates*. Seguramente FLV1/MP3/FLV fue la codificación usada por YouTube en el pasado, hoy en día la mayoría de los videos se encuentran en formato H.264/MP4A/FLV y los nuevos videos (con altas resoluciones) se están codificando bajo AVC1/MP4A/MP4.

En la Tabla 10.3 se presentan las principales resoluciones usadas en YouTube, independientemente de las codificaciones aplicadas. Existen múltiples resoluciones usadas en YouTube (en los videos analizados se contabilizan 798 resoluciones diferentes), algunas de las de mayor presencia son 320x240 (en el 31,1 % de los videos analizados), 480x360 (en el 12,6 % de los videos), 400x226 (en el 7,8 %), 640x360 (en el 7,2 %) y 640x480 (en el 7,1 %). Notar que el 4,3 % de los videos monitoreados se encuentran en formato HD (*High Definition*), presentando una resolución de 1280x720.

En la Tabla 10.4, se presentan los contenidos agrupados según su cantidad de videos asociados (diferentes codificaciones). El 40,4 % de los contenidos poseen 3 diferentes codificaciones, el 28,2 % de estos poseen 4 videos, el 26,3 % 2 videos, el 2,9 % 5 videos y el 2,3 % restante únicamente 1 video. Notar que la cantidad de videos por contenido no depende de la cantidad de vistas que estos posean (tal como se podría llegar a suponer).

En la Tabla 10.5, se presentan los perfiles mayormente usados en los contenidos según

Cuadro 10.2: Principales codificaciones de los contenidos en YouTube.

Codec de video	*Codec* de audio	*Muxer*	Cantidad total de videos	Porcentaje sobre el total de videos	*Bitrate* promedio (en Kbps)
H.264	MP4A	FLV	37823	44.4 %	590
Resoluciones más populares:		320x240	9185	24.3 % (10.8 %)	302
		640x360	6103	16.1 % (7.2 %)	729
		640x480	5809	15.4 % (6.8 %)	712
		480x360	4074	10.8 % (4.8 %)	647
		854x480	3874	10.2 % (4.5 %)	1040
FLV1	MP3	FLV	30562	35.9 %	329
Resoluciones más populares:		320x240	17278	56.5 % (20.3 %)	316
		400x226	6191	20.3 % (7.7 %)	370
		320x180	1836	6.0 % (2.2 %)	333
		320x214	1571	5.1 % (1.8 %)	329
		300x240	500	1.6 % (0.6 %)	323
AVC1	MP4A	MP4	15742	18.5 %	856
Resoluciones más populares:		480x360	6450	41.0 % (7.6 %)	706
		1280x720	3648	23.2 % (4.3 %)	1211
		480x270	2408	15.3 % (2.8 %)	610
		480x320	1106	7.0 % (1.3 %)	680
		450x360	458	2.9 % (0.5 %)	592

Cuadro 10.3: Principales resoluciones de los videos en YouTube.

Resolución	Cantidad total de videos	Porcentaje sobre el total de videos
320x240	26539	31.1 %
480x360	10769	12.6 %
400x226	6661	7.8 %
640x360	6162	7.2 %
640x480	6078	7.1 %
854x480	3875	4.5 %
1280x720	3652	4.3 %
320x180	3416	4.0 %
480x270	2697	3.2 %
320x214	2350	2.8 %

Cuadro 10.4: Cantidad de contenidos agrupados por cantidad de codificaciones.

Cantidad de videos por contenido	Cantidad de contenidos	Porcentaje del total de contenidos	Promedio de vistas por contenido
1	1982	2.3 %	100620
2	11199	26.3 %	634110
3	11488	40.4 %	559185
4	5997	28.2 %	284001
5	493	2.9 %	196283

su cantidad de videos asociados. En los casos de contenidos con un único video asociado, normalmente, éste es de tipo FLV1/MP3/FLV bajo una resolución de 320x240. En los casos de contenidos con 2 videos asociados, el primer video suele corresponderse con un video de tipo H.264/MP4A/FLV bajo una resolución de 640x360 o 320x240 y el segundo con un video de tipo FLV1/MP3/FLV con una resolución de 400x226 o 320x240. Los contenidos con 3 videos asociados, suelen tener un video H.264/MP4A/FLV con una resolución de 640x480 o 640x360, otro video H.264/MP4A/FLV con una resolución de 480x360 o 640x360 y un video FLV1/MP3/FLV con una resolución de 320x240 o 400x226. Los contenidos con 4 videos asociados, tienen un video AVC1/MP4A/MP4 de 1280x720, un video H.264/MP4A/FLV de 854x480, otro video H.264/MP4A/FLV de 640x360 y un video FLV1/MP3/FLV de 400x226. Finalmente, los contenidos con 5 videos (no se contabilizaron contenidos con más de 5 videos asociados), tienen un video AVC1/MP4A/MP4 de 1280x720, un video H.264/MP4A/FLV de 854x480 o 640x360, un video AVC1/MP4A/MP4 de 480x270, un video H.264/MP4A/FLV de 640x360 o 320x180 y un video FLV1/MP3/FLV de 400x226 o 320x180. Notar que todos los diferentes perfiles tienen asociado un video de tipo FLV1/MP3/FLV de baja resolución, seguramente por motivos de compatibilidad con reproductores Flash viejos (dado que YouTube usa esta tecnología).

Cuadro 10.5: Perfiles de las diferentes codificaciones de los contenidos.

Cantidad de videos por contenido	Número de video	*Codec* de video	*Codec* de audio	*Muxer*	Posibles resoluciones
1	1	FLV1	MP3	FLV	320x240
2	1	H.264	MP4A	FLV	640x360 o 320x240
	2	FLV1	MP3	FLV	400x226 o 320x240
3	1	H.264	MP4A	FLV	640x480 o 640x360
	2	H.264	MP4A	FLV	480x360 o 640x360
	3	FLV1	MP3	FLV	320x240 o 400x226
4	1	AVC1	MP4A	MP4	1280x720
	2	H.264	MP4A	FLV	854x480
	3	H.264	MP4A	FLV	640x360
	4	FLV1	MP3	FLV	400x226
5	1	AVC1	MP4A	MP4	1280x720
	2	H.264	MP4A	FLV	854x480 o 640x360
	3	AVC1	MP4A	MP4	480x270
	4	H.264	MP4A	FLV	640x360 o 320x180
	5	FLV1	MP3	FLV	400x226 o 320x180

El detalle técnico de los resultados muestra el potencial de la herramienta. Los resultados son útiles para empresas competidoras de YouTube o para público en general que desee entender más sobre el servicio brindado. A su vez, los resultados son útiles también para el propio YouTube, de forma de detectar vínculos rotos o problemas en las codificaciones de video (en este caso no fue posible visualizar el $1{,}01\,\%$ de los videos chequeados).

Capítulo 11

Conclusiones Generales y Perspectivas

11.1 Conclusiones Generales

En esta tesis se presenta el diseño e implementación de GoalBit, la primera red P2P de distribución de video en vivo gratuita y de código abierto.

Siguiendo el exitoso enfoque de BitTorrent [19], se desarrolló el *GoalBit Transport Protocol (GBTP)*. Un protocolo mediante el cual es posible distribuir un *streaming* de video en vivo sobre una red de pares. En la arquitectura definida por GBTP existen 4 tipos diferentes de componentes en la red: el *broadcaster*, los *super-peers*, los *peers* normales y el *tracker*. El *broadcaster* es el responsable de obtener el contenido y de volcarlo a la red. Los *super-peers* son *peers* con buena capacidad (principalmente de ancho de banda), los cuales cumplen principalmente la función de la distribución inicial del contenido. Los *peers* normales son los usuarios finales, quienes se conectan a la red con el único fin de reproducir un cierto contenido. El *tracker* cumple el mismo rol que en BitTorrent, siendo el encargado de administrar los *peers* en la red. Al ingresar un *peer* a la red, éste se comunica con el *tracker*, quien le retorna un conjunto de referencias a otros *peers* que se encuentran reproduciendo el mismo contenido. Luego, dicho *peer* se contactará con un subconjunto de los *peers* retornados por el *tracker*, comenzando de esta manera el intercambio de piezas (y la consiguiente reproducción del video). En GBTP la comunicación entre los *peers* y el *tracker* es realizada en base a HTTP/HTTPS, mientras que la comunicación entre *peers*, se lleva a cabo sobre TCP en base al intercambio de 13 diferentes tipos de mensajes binarios.

Definiendo *overhead* como los *bytes* de la transmisión no pertenecientes al contenido de video, y según el análisis realizado, el protocolo GBTP introduce un *overhead* muy pequeño en la distribución del *streaming*, de entre un $0{,}3\,\%$ y un $1{,}10\,\%$.

Se definió el *GoalBit Packetized Stream (GBPS)* con el fin de especificar la forma de paquetizar un *streaming* de audio y video, encapsulándolo dentro de las piezas GBTP. GBPS no impone ninguna restricción sobre las codificaciones (tanto de audio como de video) y formatos contenedores a encapsular, por tanto GBPS puede extenderse para distribuir cualquier contenido en forma de *broadcast*. Respecto al *overhead* agregado por GBPS, éste es muy pequeño, simplemente 10 *bytes* por cada pieza GBTP generada (usualmente cada pieza en GBTP se en-

cuentra compuesta por $65536\ bytes$, por lo que el *overhead* agregado por GBPS sería de un $0{,}015\,\%$).

Se apoyó a la comunidad *open-source* en la implementación de un primer cliente de referencia de las especificaciones GBTP/GBPS, el cual llamamos *GoalBit*. Dicha implementación se basa en 3 diferentes aplicaciones: Videolan Media Player (VLC) [131], Enhanced CTorrent [43] y OpenTracker [97]. VLC es un reproductor de video multiplataforma, el cual provee una gran cantidad de funcionalidades referidas a la captura, procesamiento y distribución de video. El Enhanced CTorrent es un cliente BitTorrent eficiente de baja complejidad. Por su parte el OpenTracker, es una implementación de un *tracker* BitTorrent. Todas estas aplicaciones son gratuitas, de tipo *open-source* y escritas en C/C++. Mediante el uso del cliente GoalBit es posible tanto reproducir, como emitir *streamings* GBTP/GBPS. Actualmente, el cliente se encuentra hosteado en SourceForge[1], de donde lleva más de 88.000 descargas de usuarios (observando en el último año un promedio de más de 50 descargas diarias).

Se presentaron los resultados de un *streaming* realizado por un usuario final, en donde se pueden observar grandes ahorros en términos de ancho de banda para el *broadcaster*, junto con una muy buena calidad de experiencia percibida por los usuarios finales. Si comparamos el *streaming* GoalBit con la clásica arquitectura cliente-servidor, durante la transmisión de dicho *streaming* es posible medir ahorros de hasta un $90\,\%$ en el ancho de banda consumido por el servidor. A su vez, sólo se midió un $0{,}03\,\%$ de pérdidas durante la completa ejecución del *streaming*, lo que hace de GoalBit una solución de *streaming* muy válida e interesante.

Sin dudas, tanto el diseño de GBTP/GBPS, como su implementación, ha sido un gran aporte a la comunidad *open-source*, pero además de esto, representan un importante aporte a la comunidad académica, y en particular a la generación de nuevos proyectos de investigación sobre las redes de tipo P2PTV. Con GoalBit se dispone de un protocolo y de una herramienta accesible, en donde es posible desarrollar y probar nuevas políticas, nuevas estrategias, entre otras. Desde la creación de GoalBit, han habido 2 tesis de grado y 4 tesis de maestría inspiradas en él: la de Pablo Romero [109], la de Darío Padula [100], la de Nicolás De León [74], y la de Claudia Rostagnol (aún en curso).

Se contribuyó con el grupo PPSP de la IETF en la definición de un protocolo estándar de *streaming* sobre redes P2P. En base a nuestra experiencia con el diseño e implementación de GoalBit, en la IETF Meeting 78[2], se presentó un conjunto de comentarios sobre dicho protocolo, proponiendo una nueva versión del *draft*.

Se presentó un modelo matemático de cooperación entre pares, sobre el cual se realizó un detallado análisis de las diferentes estrategias de selección de piezas y sus propiedades, calculando para cada una de éstas, sus valores de latencia (tiempo de *buffering* inicial) y de continuidad en la reproducción del *streaming*. En base a estos conceptos se planteó un problema de optimización combinatorio (COP) con el fin de encontrar una estrategia de selección de piezas óptima. Dicho COP fue resuelto en base a la aplicación de metaheurísticas como

[1] http://goalbit.sourceforge.net/

[2] La IETF Meeting 78, se llevo a cabo en la ciudad de Maastricht, Holanda, entre el 25 y el 30 de Julio del 2010.

búsqueda local y colonia de hormigas, obteniendo una estrategia de selección de piezas con considerables mejoras respecto a las existentes en la literatura. En este trabajo, se contrastó dicho modelo matemático con la realidad, en base a la realización de emulaciones con GoalBit.

Con el fin de validar nuestras ideas, se desarrolló un prototipo de una red de distribución de contenido basado en GoalBit, llamado la *Plataforma GoalBit*. Mediante esta plataforma es posible administrar los contenidos a ser distribuidos sobre GBTP/GBPS junto con sus recursos asociados (los *super-peers* configurados para cada contenido). A su vez, ésta provee una asignación dinámica de recursos según la demanda de cada contenido (asignando más *super-peers* a los contenidos con mayor demanda de usuarios) y la posibilidad de adaptar el *streaming* GBTP/GBPS a diferentes infraestructuras como el *streaming* Flash o el *streaming* Windows (entre otras). La Plataforma GoalBit fue implantada en AdinetTV a modo de prueba, obteniendo buenos resultados.

Continuando con el trabajo realizado en el marco de la tesis de doctorado de Pablo Rodriguez-Bocca [110] sobre la medición de la calidad de experiencia mediante la tecnología PSQA, se implementó un prototipo para medir la QoE en tiempo real. Dicho prototipo fue presentado en [usa-sigm08], obteniendo el premio de *"Best Student Demonstration Award"*, siendo posteriormente integrado en la implementación del cliente GoalBit. En la red GoalBit, cada *peer* reporta su calidad de experiencia (obtenida mediante el calculo de PSQA) al *tracker*. De esta manera, se puede obtener una visión global y completa del estado de la red.

Finalmente, como otro aporte al área del monitoreo de redes de video, se implementó un prototipo de un monitor pasivo genérico de redes de distribución de video (basado en el cliente GoalBit), con el cual es posible visualizar tanto el estado de señales en vivo, como de videos a ser servidos bajo demanda, sin importar su codificación y su protocolo de distribución. Una de las principales características de este prototipo radica en su simpleza, siendo una solución de monitoreo que no requiere de ningún *hardware* específico, representando una solución genérica de bajo costo. Con el fin de presentar el potencial de este monitor, se realizó un profundo y detallado monitoreo sobre los videos distribuidos en YouTube [145], la mayor red de distribución de videos bajo demanda de la actualidad, analizando de forma automática más de 85.000 videos.

11.2 Perspectivas a Futuro

A partir de este trabajo se presenta un amplio conjunto de actividades a realizar en el futuro cercano. A continuación presentamos algunas de las perspectivas a futuro de este trabajo.

Con el fin de asegurar el éxito del protocolo GBTP, es necesario realizar un profundo análisis sobre su escalabilidad, ya sea evaluando y optimizando las actuales políticas y estrategias aplicadas (tanto a nivel de protocolo, como a nivel de implementación), como creando otras nuevas. Dicho análisis puede ser llevado a cabo, usando el emulador GoalBit [8].

Existen múltiples mejoras a realizar en la implementación del cliente de referencia. Desde la implementación de nuevas técnicas de apertura de puertos en la red del usuario (problema conocido como *NAT Transversal* [24, 60, 114]), hasta la propia optimización del cliente, como por ejemplo mediante la aplicación de alguna técnica de *multi-threading*. A su vez, también existen mejoras a realizar en la implementación del *tracker*, como por ejemplo integrando ONO [30].

Actualmente mediante el protocolo GoalBit únicamente es posible distribuir video en vivo, sería necesario también contemplar la distribución de contenidos bajo demanda. La tesis de Claudia Rostagnol, consiste en la integración de las funcionalidades de VoD, tanto a nuestra implementación de referencia, como a la plataforma GoalBit, definiendo un modelo matemático sobre el cual optimizar la replicación de videos según su correspondiente demanda. Aquí el problema de asignación de *super-peers* a los contenidos deja de ser trivial, dado que se introduce la restricción de capacidad de alojamiento en el servidor, convirtiendo el problema presumiblemente en un problema NP-dificil.

A su vez, se plantea como objetivo del grupo de investigación, el seguir contribuyendo con el grupo PPSP de la IETF, volcando nuestra experiencia adquirida en el campo de las redes P2P.

Finalmente se propone seguir extendiendo el prototipo de la red de distribución de contenidos ya implementado (llamado *Plataforma GoalBit*), junto con el del monitor genérico (llamado *Monitor GoalBit*), con el fin de crear la primera *CDN-P2P open-source*, brindando de esta manera nuevos desafíos para la creación de proyectos académicos de investigación, contribuyendo a una mayor difusión de contenidos en Internet.

Bibliografía

[1] Bittorrent protocol specification v1.0. http://wiki.theory.org/BitTorrentSpecification, 2010.

[2] Abilene Internet2 Network home page. http://abilene.internet2.edu/, 2007.

[3] AdinetTV home page. AdinetTV home page. http://www.adinettv.com.uy/, 2010.

[4] Adobe Systems Incorporated. Real Time Messaging Chunk Stream Protocol, April 2009.

[5] Alcatel-Lucent Home. http://www.alcatel-lucent.com/, 2007.

[6] Shahzad Ali, Anket Mathur, and Hui Zhang. Measurement of commercial peer-to-peer live video streaming. In *In Proc. of ICST Workshop on Recent Advances in Peer-to-Peer Streaming*, Weaterloo, Canadda, 2006.

[7] Stephen Alstrup and Theis Rauhe. Introducing octoshape - a new technology for large-scale streaming over the internet. Technical report, European Broadcasting Union (EBU), 2005.

[8] Juan José Comas y Pablo Perdomo Andrés Barrios, Matías Barrios. Análisis y evaluación del rendimiento de la plataforma P2P Goalbit.

[9] ANSI home page. ANSI home page. http://www.ansi.org/, 2009.

[10] Apache HTTP Server home site. http://httpd.apache.org/, 2010.

[11] W. Ashmawi, R. Guerin, S. Wolf, and M. Pinson. On the impact of policing and rate guarantees in diff-serv networks: A video streaming application perspective. *Proceedings of ACM SIGCOMM'2001*, pages 83–95, 2001.

[12] A. Basso, I. Dalgic, F. Tobagi, and C. Lambrecht. Study of mpeg-2 coding performance based on a perceptual quality metric. In *Proceedings of PCS'96*, pages 263–268, March 1996.

[13] BBC Multicast initiative. http://support.bbc.co.uk/multicast/, 2007.

[14] R. Beckers, J.L. Deneubourg, and S. Goss. Trails and U-turns in the selection of the shortest path by the ant lasius niger. *Journal of Theoretical Biology*, 159:397–415, 1992.

[15] Beerware License home site. http://people.freebsd.org/ phk/, 2010.

[16] María Elisa Bertinat, Daniel De Vera, Darío Padula, Franco Robledo, Pablo Rodríguez-Bocca, and Pablo Romero. Systematic procedure for improving continuity and latency on a p2p streaming protocol. In *Proceedings of the 1th IEEE Latin-American Conference on Communications (LatinCom'09)*, Washington, DC, USA, 8-11 september 2009. IEEE Computer Society.

[17] Gilles Bertrand. The IP Multimedia Subsystem in Next Generation Networks. http://www.rennes.enst-bretagne.fr/ gbertran/files/IMS_an_overview.pdf, May 2007.

[18] BitComet home page. http://www.bitcomet.com/, 2009.

[19] Bittorrent home page. http://www.bittorrent.org, 2007.

[20] Bittorrent movie site. http://www.bittorrent.com, 2007.

[21] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[22] A. Bouch and M. A. Sasse. Why Value is Everything: a User-Centered Approach to Internet Quality of Service and Pricing. *Lecture Notes in Computer Science*, 2092:59–72, 2001.

[23] A. Bouch, M. A. Sasse, and H. DeMeer. Of Packets and People: a User-centered Approach to Quality of Service. *Proceedings of IWQoS2000*, pages 189–197, 2000.

[24] Bryan Ford, and Pyda Srisuresh, and Dan Kegel. Peer-to-Peer Communication Across Network Address Translators. http://www.brynosaurus.com/pub/net/p2pnat/, 2005.

[25] BTSharp home page. http://www.btsharp.com/, 2009.

[26] Bengt Carlsson and Rune Gustavsson. The rise and fall of napster - an evolutionary approach. In *AMT '01: Proceedings of the 6th International Computer Science Conference on Active Media Technology*, pages 347–354, London, UK, 2001. Springer-Verlag.

[27] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications (JSAC)*, 20(8):1489–1499, 2002.

[28] Miguel Castro, Manuel Costa, and Antony Rowstron. Debunking some myths about structured and unstructured overlays. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 85–98, Berkeley, CA, USA, 2005. USENIX Association.

[29] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: High-bandwidth content distribution in a cooperative environment. In *IPTPS '03: Proceedings of the Second International Workshop on Peer-to-Peer Systems*, 2003.

[30] David R. Choffnes and Fabián E. Bustamante. Taming the torrent: A practical approach to reducing cross-isp traffic in peer-to-peer systems. *ACM SIGCOMM Computer Communication Review*, 38(4):363–374, 2008.

[31] Cisco Systems, Inc. http://www.cisco.com/, 2007.

[32] Cisco Systems, Inc. Hyperconnectivity and the Approaching Zettabyte Era., June 2010.

[33] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *International workshop on Designing privacy enhancing technologies*, pages 46–66, New York, NY, USA, 2001. Springer-Verlag New York, Inc.

[34] CoolStreaming home page. http://www.coolstreaming.us, 2007.

[35] Charles D. Cranor, Matthew Green, Chuck Kalmanek, David Shur, Sandeep Sibal, Jacobus E. Van der Merwe, and Cormac J. Sreenan. Enhanced streaming services in a content distribution network. *IEEE Internet Computing*, 5(4):66–75, 2001.

[36] C. Dana, Danjue Li, D. Harrison, and Chen-Nee Chuah. Bass: Bittorrent assisted streaming system for video-on-demand. In *Multimedia Signal Processing, 2005 IEEE 7th Workshop on*, pages 1–4, 2005.

[37] Hrishikesh Deshpande, Mayank Bawa, and Hector Garcia-Molina. Streaming live media over a peer-to-peer network. Technical Report 2001-30, Stanford InfoLab, April 2001.

[38] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, DEI Politecnico de Milano, Italia, 1992.

[39] Marco Dorigo and Luca Maria Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[40] Marco Dorigo and Thomas Stutzle. *Ant Colony Optimization.* MIT Press, 2004.

[41] DSL Forum Technical Work WT-126. Video services quality of experience (qoe) requirements and mechansims, Apr 2007.

[42] eMule home page. http://www.emule-project.net, 2007.

[43] Enhanced CTorrent home page. http://www.rahul.net/dholmes/ctorrent/, 2009.

[44] Hans Eriksson. Mbone: the multicast backbone. *Commun. ACM*, 37(8):54–60, 1994.

[45] Evertz Microsystems Ltd. - Dual Channel Video and Analog Audio Monitoring. http://www.evertz.com/products/7761AVM2-DC, 2010.

[46] Evertz Microsystems Ltd. - MVP. http://www.evertz.com/products/MVP, 2010.

[47] Evertz Microsystems Ltd. - VistaLINK PRO PLUS. . http://www.evertz.com/products/VistaLINK_PRO, 2010.

[48] E. Gelenbe. Random Neural Networks with Negative and Positive Signals and Product Form Solution. *Neural Computation*, 1(4):502–511, 1989.

[49] GNU General Public License home site. http://www.gnu.org/copyleft/gpl.html, 2010.

[50] Gnutella home page. http://gnutella.wego.com/, 2005.

[51] GoalBit - The First Free and Open Source Peer-to-Peer Streaming Network. http://goalbit.sf.net/, 2008.

[52] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross. Insights into pplive: A measurement study of a large-scale p2p iptv system. In *In Proc. of IPTV Workshop, International World Wide Web Conference*, 2006.

[53] Scott Hull. *Content Delivery Networks*. McGraw-Hill., New York, February 2002.

[54] IETF Home Page. http://www.ietf.org/, 2010.

[55] IETF Network Working Group. User Datagram Protocol (RFC 768), August 1980.

[56] IETF Network Working Group. A Transport Protocol for Real-Time Applications (RFC 1889), January 1996.

[57] IETF Network Working Group. Hypertext Transfer Protocol – HTTP/1.1 (RFC 2616), June 1999.

[58] IETF Network Working Group. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks (RFC 3411), December 2002.

[59] IETF Network Working Group. TCP Friendly Rate Control (TFRC): Protocol Specification (RFC 3448), 2003.

[60] IETF Network Working Group. State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs) (RFC 5128). http://tools.ietf.org/html/rfc5128, March 2008.

[61] IETF Network Working Group. Draft Problem Statement of P2P Streaming Protocol (PPSP). http://tools.ietf.org/html/draft-zhang-ppsp-problem-statement-05, October 2009.

[62] IETF Network Working Group. Draft Survey of P2P Streaming Applications. http://tools.ietf.org/html/draft-gu-ppsp-survey-01, October 2009.

[63] IneoQuest. - Content Monitoring System. . http://www.ineoquest.com/icms-content-monitoring-system, 2010.

[64] IneoQuest. - Cricket FrameGrabber. http://www.ineoquest.com/cricket-family, 2010.

[65] ISO/IEC 13818 Standard - MPEG-2. Information technology: generic coding of moving pictures and associated audio information, January 2005.

[66] ISO/IEC 14496-10 Standard - MPEG-4 Part 10. Advanced video coding, 2003.

[67] ISO/IEC 14496-2 Standard - MPEG-4 Part 2. Coding of audio-visual objects - part 2: Visual, 2001.

[68] ITU-R Recommendation BT.500-11. Methodology for the subjective assessment of the quality of television pictures, June 2002.

[69] ITU-T Standard H.263. Video coding for low bit rate communication, January 1998.

[70] jumpTV Home page. http://www.jumptv.com/, 2007.

[71] justinTV Home page. http://www.justin.tv/, 2007.

[72] KaZaA home page. http://www.kazaa.com, 2007.

[73] L. Kontothanassis, R. Sitaraman, J. Wein, D. Hong, R. Kleinberg, B. Mancuso, D. Shaw, and D. Stodolsky. A Transport Layer for Live Streaming in a Content Delivery Network. In *Proceedings of the IEEE, Special Issue on Evolution of Internet Technologies*, pages 92(9):1408–1419, sep 2004.

[74] Nicolás De León. Evaluación de calidad de video en una aplicación P2P: GoalBit. Universidad de la República, Facultad de Ingeniería, Centro de Postgrados y Actualización Profesional. Montevideo, Uruguay, August 2010.

[75] LibTorrent Rakshasa home page. http://libtorrent.rakshasa.no/, 2009.

[76] Libtorrent Rasterbar home page. http://www.rasterbar.com/products/libtorrent/, 2009.

[77] Dongmei Lin, Xiangbin Wu, and Dong Wang. Exact heuristic algorithm for traveling salesman problem. In *The 9th International Conference for Young Computer Scientists*, pages 9–13, Washington, DC, USA, 2008. IEEE Computer Society.

[78] Lugdunum home page. http://lugdunum2k.free.fr, 2007.

[79] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS '02: Proceedings of the 16th annual ACM international conference on Supercomputing*, pages 84–95, New York, NY, USA, 2002. ACM.

[80] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65, London, UK, 2002. Springer-Verlag.

[81] Microsoft Corporation. Microsoft Media Server (MMS) Protocol Specification, July 2010.

[82] Milan Milenkovic, Scott H. Robinson, Rob C. Knauerhase, David Barkai, Sharad Garg, Vijay Tewari, Todd A. Anderson, and Mic Bowman. Toward internet distributed computing. *IEEE Computer*, 36(5):38–46, 2003.

[83] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-peer computing. Technical Report HPL-2002-57, HP Labs, 2002.

[84] Miranda Technologies Inc. - iControl Edge. http://www.miranda.com/146, 2010.

[85] Miranda Technologies Inc. - Kaleido-X. http://www.miranda.com/139, 2010.

[86] Miro home page. http://www.getmiro.com/, 2009.

[87] S. Mohamed. *Automatic Evaluation of Real-Time Multimedia Quality: a Neural Network Approach*. PhD thesis, INRIA/IRISA, Univ. Rennes I, Rennes, France, jan 2003.

[88] S. Mohamed, F. Cervantes, and H. Afifi. Audio Quality Assessment in Packet Networks: an Inter–Subjective Neural Network Model. In *Proc. of IEEE International Conference on Information Networking (ICOIN-15)*, pages 579 –586, Beppu City, Oita, Japan, January 2001.

[89] S. Mohamed, F. Cervantes, and H. Afifi. Integrating Networks Measurements and Speech Quality Subjective Scores for Control Purposes. In *Proceedings of IEEE INFOCOM'01*, pages 641–649, Anchorage, AK, USA, April 2001.

[90] S. Mohamed and G. Rubino. A Study of Real–time Packet Video Quality Using Random Neural Networks. *IEEE Transactions On Circuits and Systems for Video Technology*, 12(12):1071–1083, December 2002.

[91] S. Mohamed, G. Rubino, H. Afifi, and F. Cervantes. Real–time Video Quality Assessment in Packet Networks: A Neural Network Model. In *Proceedings of International Conference on Parallel and Distibuted Processing Techniques and Applications (PDPTA'01)*, Las Vegas, Nevada, USA, June 2001.

[92] msnTV website. http://www.msntv.com, 2007.

[93] MySQL home site. http://www.mysql.com/, 2010.

[94] Napster home page. http://www.napster.com/, 2005.

[95] Octoshape Home page. http://www.octoshape.com/, 2007.

[96] On2 home page. http://www.on2.com/, 2009.

[97] OpenTracker home page. http://erdgeist.org/arts/software/opentracker/, 2009.

[98] Oracle Berkeley DB home site. http://www.oracle.com/technetwork/database/berkeleydb/overview/index.html, 2010.

[99] Venkata N. Padmanabhan, Helen J. Wang, Philip A. Chou, and Kunwadee Sripanidkulchai. Distributing streaming media content using cooperative networking. In *NOSSDAV '02*, pages 177–186, New York, NY, USA, 2002. ACM Press.

[100] Darío Padula. Compromiso entre Pares e ISPs: Un Modelo de Optimización Multiobjetivo. Universidad de la República, Facultad de Ingeniería, Instituto de Investigación : LPE - IMERL. Montevideo, Uruguay, July 2010.

[101] Peercast home page. http://www.peercast.org/, 2007.

[102] PHP home site. http://php.net/, 2010.

[103] PPLive Home page. http://www.pplive.com, 2007.

[104] PPStream home page. http://www.ppstream.com/, 2007.

[105] Michael Rabinovich and Oliver Spatschek. *Web caching and replication*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[106] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[107] RealNetworks home page. http://www.realnetworks.com/, 2007.

[108] Martin Varela Rico. *Pseudo-subjective Quality Assessment of Multimedia Streams and its Applications in Control*. PhD thesis, INRIA/IRISA, Univ. Rennes I, Rennes, France, nov 2005.

[109] Pablo Romero Rodríguez. Optimización de la Estrategia de Selección de Piezas de Video en Redes P2P. Universidad de la República, Facultad de Ingeniería, Instituto de Investigación : LPE - IMERL. Montevideo, Uruguay, November 2009.

[110] Pablo Rodríguez-Bocca. *Quality-centric design of Peer-to-Peer systems for live-video broadcasting*. PhD thesis, INRIA/IRISA, Université de Rennes I, Rennes, France, April 2008.

[111] Pablo Rodríguez-Bocca, Gerardo Rubino, and Luis Stábile. Multi-Source Video Streaming Suite. In *7th IEEE International Workshop on IP Operations and Management (IPOM'07)*, San José, California, United States, October 31 - November 2 2007.

[112] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–347, 2001.

[113] Antony Rowstron and Peter Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 188–201, New York, NY, USA, 2001. ACM Press.

[114] Saikat Guha, and Paul Francis. Characterization and measurement of tcp traversal through nats and firewalls, 2005.

[115] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Multimedia Computing and Networking*, San Jose, USA, 2002.

[116] A. Sentinelli, G. Marfia, M. Gerla, L. Kleinrock, and S. Tewari. Will iptv ride the peer-to-peer stream? *Communications Magazine, IEEE*, 45:86–92, 2007.

[117] Seti@home home page. http://setiathome.ssl.berkeley.edu., 2005.

[118] Clay Shirky. What is p2p...and what isn't? Technical report, O'Reilly openP2P Network, 2000.

[119] Siemens AG. http://www.siemens.com/, 2007.

[120] Sling Media. - Slingbox SOLO. http://www.slingmedia.com/go/solo, 2010.

[121] SopCast - Free P2P internet TV. http://www.sopcast.org, 2007.

[122] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM Press.

[123] S. Tewari and L. Kleinrock. Analytical model for bittorrent-based live video streaming. In *4th IEEE Consumer Communications and Networking Conference (CCNC'07)*, pages 976–980, Las Vegas, NV, January 2007.

[124] The Freenet Project home page. http://freenetproject.org, 2007.

[125] Julio Orozco Torrentera. *Quality of Service management of multimedia flows over DiffServ IP networks*. PhD thesis, INRIA/IRISA, Univ. Rennes I, Rennes, France, apr 2005.

[126] TVAnts home page. http://cache.tvants.com/, 2007.

[127] TVUnetworks home page. http://tvunetworks.com/, 2007.

[128] C.J. van den Branden Lambrecht. Color Moving Picture Quality Metric. In *Proceedings of the IEEE International Conference on Image Processing*, September 1996.

[129] C.J. van den Branden Lambrecht. *Perceptual Models and Architectures for Video Coding Applications*. PhD thesis, EPFL, Lausanne, Swiss, 1996.

[130] Dinesh C. Verma. *Content Distribution Networks*. John Wiley & Sons, Inc., New York, February 2002.

[131] VideoLan home page. http://www.videolan.org, 2007.

[132] A. Vlavianos, M. Iliofotou, and M. Faloutsos. Bitos: Enhancing bittorrent for supporting streaming applications. In *9th IEEE Global Internet Symposium 2006*, April 2006.

[133] S. Voran. The Development of Objective Video Quality Measures that Emulate Human Perception. In *IEEE GLOBECOM*, pages 1776–1781, 1991.

[134] Vuze home page. http://azureus.sourceforge.net/, 2009.

[135] Li-Ying Wang, Jie Zhang, and Hua Li. An improved genetic algorithm for tsp. In *Proceedings of the Sixth International Conference on Machine Learning and Cybernetics*, pages 925–928, Washington, DC, USA, 2007. IEEE Computer Society.

[136] S. Wee, W. Tan, J. Apostolopoulos, and S. Roy. System design and architecture of a mobile streaming media content delivery network (msdcdn). Technical report, Streaming Media Systems Group, HP-Labs, 2003.

[137] Dan Werthimer, Jeff Cobb, Matt Lebofsky, David Anderson, and Eric Korpela. Seti@homemassively distributed computing for seti. *Comput. Sci. Eng.*, 3(1):78–83, 2001.

[138] Wowza Media Systems home page. http://www.wowzamedia.com/, 2009.

[139] Chi-Jen Wu, Cheng-Ying Li, and Jan-Ming Ho. Improving the download time of bittorrent-like systems. In *IEEE International Conference on Communications 2007 (ICC 2007)*, pages 1125–1129, Glasgow, Scotland, June 2007.

[140] Gang Wu and Tzi cker Chiueh. How efficient is bittorrent? In *Thirteenth Annual Multimedia Computing and Networking (MMCN'06)*, San Jose, CA., Jaunary 2006.

[141] Susu Xie, Gabriel Y. Keung, and Bo Li. A measurement of a large-scale peer-to-peer live video streaming system. In *ICPPW '07: Proceedings of the 2007 International Conference on Parallel Processing Workshops (ICPPW 2007)*, page 57, Washington, DC, USA, 2007. IEEE Computer Society.

[142] Xvid Codec home page. http://www.xvid.org/, 2007.

[143] Beverly Yang and Hector Garcia-Molina. Comparing hybrid peer-to-peer systems. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 561–570, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[144] Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. In *19th International Conference on Data Engineering (ICDE'03)*, pages 49–60, Bangalore, India, March 2003.

[145] youTube website. http://www.youtube.com/, 2007.

[146] X. Zhang, J. Liu, and B. Li. On large scale peer-to-peer live video distribution: Coolstreaming and its preliminary experimental results. In *IEEE International Workshop on Multimedia Signal Processing (MMSP'05)*, Washington, DC, USA, 2005. IEEE Computer Society.

[147] Xinyan Zhang, Jiangchuan Liu, Bo Li, and Tak-Shing Peter Yum. Coolstreaming/donet: A data-driven overlay network for peer-to-peer live media streaming. In *IEEE Conference on Computer Communications (INFOCOM'05)*, Washington, DC, USA, 2005. IEEE Computer Society.

[148] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

[149] Yipeng Zhou, Dah Ming Chiu, and J.C.S. Lui. A Simple Model for Analyzing P2P Streaming Protocols. In *Proceeding of the IEEE International Conference on Network Protocols (ICNP'07)*, pages 226–235, Beijing, China, October 2007.

Índice de figuras