UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

I.T.T. SONIDO E IMAGEN

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE VÍDEO BAJO DEMANDA P2P

Autor: José Luis Aldovera Escamilla

Director: Jaime José García Reinoso

Leganés, Septiembre de 2015

Título: DISEÑO E IMPLEMENTACION DE UN SISTEMA DE VÍDEO BAJO DEMANDA P2P

Autor: José Luis Aldovera Escamilla

Director: Jaime José García Reinoso

EL TRIBUNAL

Presidente: Iván Vidal Fernández

Vocal: Jerónimo Arenas García

Secretario: María del Carmen Fernández Panadero

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 30 de septiembre del 2015 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Resumen

La distribución de vídeo bajo demanda (VoD), ha alcanzado una gran popularidad entre la comunidad online, convirtiéndose en nuestros días en una "*killer application*" dentro de Internet. El principal motivo, ha sido la flexibilidad y comodidad que ofrece este tipo de servicio, permitiendo al usuario interactuar con el contenido, pudiendo ver cualquier punto de un vídeo en cualquier momento.

Pero las aplicaciones de VoD son costosas para los proveedores de servicios debido a la carga que se genera en los servidores de vídeo, que obliga a proveer de una importante infraestructura en arquitecturas centralizadas cliente-servidor. Como una solución a este problema, en los últimos años se han realizado multitud de estudios que proponen el uso de arquitecturas entre iguales (peer to peer o P2P) en la distribución de VoD. Con el uso de estas redes P2P lo que se pretende, es aprovechar toda la capacidad de proceso, almacenamiento y ancho de banda sobrante de los ordenadores cliente interconectados a través de la red.

En este proyecto se verán diferentes arquitecturas de video bajo demanda (VoD) y se explicarán las arquitecturas P2P, para terminar exponiendo los principales problemas que se encuentran en los actuales sistemas de VoD P2P. Diseñaremos e implementaremos una aplicación que proporcione video bajo demanda, dentro de una arquitectura P2P. Esta aplicación contará con mecanismos que permitan verificar la autenticidad de los contenidos, se diseñará un proceso de selección del peer fuente, y se implementará un protocolo de comunicación entre los diferentes elementos del sistema

Palabras Clave: Vídeo bajo Demanda, Peer to Peer, Segmento, Vídeo Streaming, Round Trip Time (RTT).

Abstract

The distribution of Video on Demand (VoD), has gained such popularity within the online community that now days, it has become a "*killer application*" in the Internet. The main reason for VoD success is the flexibility and comfort the service offers to the users allowing them to interact with the content which it can be watched at any point and at any time of the recorded video.

But VoD applications are costly to service providers that are forced to invest on centralized infrastructure client-server architectures to hold the enormous streaming loads to the video servers. During the last years, various studies propose to mitigate this problem with the use of peer to peer architectures (P2P) distribution of VoD. With the use of these P2P networks the intent is to utilize the processing capacity, storage and excess bandwidth, of the client's computers that are interconnected through the network.

This project will cover different architectures of Video on Demand (VoD) and will explain P2P architecture, as well as exposing the main issues VoD P2P systems face these days. It will design and implement an application that provides video on demand by using P2P architecture. This application will include mechanisms to verify the authenticity of the contents, and the peer source selection process design. For intercommunicating elements among the system, a communications protocol will be implemented.

Keywords: Video on Demand, Peer to Peer, Chunk, Video Streaming, Round Trip Time (RTT).

ÍNDICE GENERAL

1 Introducción 11

1.1 Introducción al proyecto 11

1.2 Motivación 13

1.3 Objetivos 14

1.4 Medios empleados 14

1.5 Estructura de la memoria 16

2 Estado del Arte 17

2.1 Introducción al VoD 17

2.1.1 Componentes básicos de un sistema de VoD 19

2.1.2 Requisitos de un sistema de VoD 20

2.1.3 Tipos de servicios de VoD 21

2.2 Redes P2P 22

2.2.1 Características de la redes P2P 22

2.2.2 Clasificación de las redes P2P 23

2.3 VoD P2P 27

2.3.1 Tipos de Redes en Sistemas VoD P2P 27

2.3.2 Principales Problemas de los Sistemas de VoD P2P 28

2.3.3 Sistemas de VoD P2P 32

3 Diseño 37

3.1 Arquitectura básica 37

3.1.1 Nodo Servidor (BootStrapServer) 39

3.1.2 Nodo Peer (Usuario) 40

3.2 Aspectos relevantes del diseño 41

3.2.1 Elección del tamaño de los segmentos (chunks) y su identificador (ID). 41

3.2.2 Estructura de directorios 43

3.2.3 Protocolo de control “P2P Streaming Protocol” 44
3.2.4 Sockets 47
3.2.5 Interfaz gráfica 48
3.2.6 Selección del Peer Fuente 48
4 Implementación 49
4.1 Autenticidad de contenidos. 50
4.2 Directorios 51
4.2.1 Directorio del Servidor 52
4.2.2 Directorio del Peer 55
4.3 Interfaces 56
4.4 Sockets 58
4.5 Protocolo P2PSP 59
4.5.1 Descripción del Tracker Protocol 60
4.6 Proceso descarga vídeos (Selección del Peer-Fuente) 64
4.7 Librerías Importadas 66
4.7.1 Xuggle 66
4.7.2 JDOM 67
4.7.3 VLCJ 68
5 Validación 69
5.1 Configuración de equipos y puesta en servicio 69
5.2 Fase I (Servidor y usuarios en el mismo PC) 70
5.3 Fase II (Servidor y Usuarios en la misma VLAN) 74
5.4 Fase III (Servidor y Usuario en distintas VLAN) 76
5.5 Resultados 79
6 Herramientas de Trabajo 81
6.1 Entorno de Desarrollo: Eclipse 81

6.2 Lenguaje de Programación: Java 81
6.3 XML 82
6.4 VLC y VLCJ 83
6.5 Creación de jar 84
7 Conclusiones 85
8 Trabajos Futuros 87
9 Presupuesto 89
9.1 Costes de personal 90
9.2 Costes de equipamiento 91
9.3 Otros costes directos del proyecto 92
9.4 Resumen de costes 92
Anexos 101
Anexo I. Diagrama de Clases 101
Anexo II. Secuencia de mensajes XML 105

ÍNDICE DE FIGURAS

Figura 1. Funcionamiento de Imagenio [14] 18
Figura 2. Sistema de VoD 19
Figura 3. Arquitectura Básica de GridCast [43] 34
Figura 4. Diseño propuesto 38
Figura 5. Secuencia de mensajes entre nodos 45
Figura 6. Ejemplo petición de un vídeo 46
Figura 7. Carpetas creadas en el Servidor 52
Figura 8. Contenido carpeta InfoVídeos 52
Figura 9. Contenido de los .txt de la carpeta InfoVideos 53
Figura 10. Archivos creados en la carpeta PeersConectados 53
Figura 11. Archivos creados en la carpeta VideoPeers 54
Figura 12. Contenido de los archivos .txt de la carpeta VideoPeers 54
Figura 13. Estructura de carpetas en el Usuario 55
Figura 14. Archivo de la carpeta BaseDatos 55
Figura 15. Contenido del txt Lista-Id-name 55
Figura 16. Carpetas contenidas en la carpeta Descargas 56
Figura 17. Segmentos de vídeo contenidos en cada carpeta 56
Figura 18. Interfaz del Servidor 57
Figura 19. Interfaz del Cliente 57
Figura 20. Esquema de conexión en entorno LAN 74
Figura 21. Esquema de conexión en entorno WAN 76
Figura 22. Configuración del router para permitir el tráfico al servidor 77

ÍNDICE DE TABLAS

Tabla 1. Velocidad Líneas ADSL 16

Tabla 2. Funcionalidades implementadas en sistemas VoD P2P [32] 30

Tabla 3. Mensajes entre Peer y Servidor (Tracker Protocol)..... 44

Tabla 4. Mensajes entre Peers (Peer to Peer Protocol)..... 44

Tabla 5. Mensajes de Respuesta..... 45

Tabla 6. Vídeos reproducidos..... 71

Tabla 7. Respuesta de la aplicación en función del bitrate-retardo-buffer..... 73

Tabla 8. Valores de RTTQueryChunk para diferentes tamaños de segmento 75

Tabla 9. Respuesta del sistema con diferentes tamaños de buffer..... 75

Tabla 10. Valores de RTT con una conexión a red a través de UMTS..... 77

Tabla 11. Valores de RTT para diferentes características del sistema 78

Tabla 12. Valores de RTT compartiendo recursos del servidor..... 78

Tabla 13. Resultado de la prueba objeto del proyecto..... 79

Tabla 14. Coste en personal..... 91

Tabla 15. Coste en equipos..... 91

Tabla 16. Otros costes 92

Tabla 17. Resumen de costes..... 92

1 Introducción

1.1 Introducción al proyecto

En los últimos años, la visualización de contenidos en la red de manera simultánea a su descarga (streaming de vídeo), se ha convertido en una de las actividades más populares en Internet. Esto ha llevado a que la industria del entretenimiento y los medios de comunicación pasen a ser un sector económicamente relevante en la vida digital, produciéndose un gran número de estudios y propuestas sobre el streaming de vídeo, tanto por parte de la industria como en el ámbito universitario.

Estos estudios han definido dos tipo de streaming; streaming en vivo (Live Video) y vídeo bajo demanda (VoD o Video on Demand). En el streaming en vivo, los servidores transmiten contenidos en vivo, y los usuarios visualizan los contenidos secuencialmente desde el momento en que acceden a estos. A diferencia del streaming en vivo, el VoD permite al usuario seleccionar en cualquier momento un vídeo de entre un grupo de vídeos previamente grabados y almacenados (funcionamiento de YouTube), e interactuar con éste a través de comandos típicos de un reproductor DVD. Estos sistemas permitirán que varios usuarios accedan simultáneamente al mismo contenido, permitiendo la visualización de cada uno en un instante diferente.

Estos sistemas de VoD, han estado soportados tradicionalmente por arquitecturas centralizadas cliente/servidor, en las cuales, un servidor (arquitectura centralizada) o varios servidores (arquitectura distribuida) distribuyen los vídeos a todos los usuarios. En las arquitecturas distribuidas los proveedores del servicio de VoD, contratan plataformas de distribución de contenidos "*Content Distribution Networks*" (CDNs) como Akamai[1], o Limelight Networks[2]; estas plataformas están compuestas por un conjunto de servidores distribuidos en diferentes áreas geográficas, los cuales se encargan de almacenar el contenido y distribuirlo bajo demanda hacia los clientes en unicast. Aunque este tipo de arquitecturas presentan un elevado coste en su infraestructura, el principal inconveniente que presentan es la elevada carga que aparece en los servidores, especialmente en la distribución de vídeo en alta definición (HD) y con un elevado número de usuarios, lo que provoca un cuello de botella en los servidores del sistema.

[1] http://www.akamai.com
[2] http://www.limelight.com

Para solventar estos problemas, se han propuesto diferentes estrategias, como el uso de la tecnología IP Multicast, y arquitecturas de redes P2P, que nos brindan soluciones bastante interesantes, tanto de escalabilidad como de eficiencia de costes.

A pesar de que multicast sea probablemente la solución más eficiente, su desarrollo está bastante limitado debido a diversas cuestiones de índole política, económica o práctica. Los actuales sistemas multicast, presentan numerosos inconvenientes para su aplicación a gran escala en Internet [1][2], inconvenientes como; número de direcciones multicast IPv4 muy bajo, algoritmos y protocolos de encaminamiento multicast complejos, y la necesidad de una correcta interpretación de estos protocolos por parte de los switches y routers que forman la red IP, que obligaría a cambiar a día de hoy buena parte de estos equipos; ha provocado que su uso se enfoque más hacia redes de área local (LAN) o redes intra-campus (interconexión de un número limitado de redes LAN). Aun así, encontramos trabajos donde se propone el uso de multicast para la distribución de streaming en vivo [3][4][5], e incluso los que como Peer-to-Peer Batching (PPB) pretenden explotar las capacidades de multicast junto al paradigma peer to peer (P2P) [6].

Desde hace unos años se ha visto como una solución al problema de congestión en los servidores el uso de redes “overlay” P2P, las cuales se han mostrado muy eficientes para la distribución de contenido a gran escala con pocos recursos. Datos recogidos en diferentes estudios basados en aplicaciones como Gridcast [7], muestran cómo con técnicas P2P la carga en los servidores decrece hasta un 36% comparado con modelos cliente/servidor.

Actualmente las prestaciones de los modernos ordenadores, son cada vez más similares a los servidores y con precios más asequibles, por lo que estas redes pretenden aprovechar las capacidades de procesamiento, almacenamiento y ancho de banda sobrante de los equipos interconectados, para repartir la carga de trabajo entre los diferentes nodos de la red, disminuyendo los problemas de cuello de botella que se originaba en los tradicionales modelos cliente/servidor.

Sin embargo, mientras este tipo de técnicas se presentan muy eficientes en aplicaciones de compartición de archivos como Kazaa, eDonkey (aplicaciones actualmente fuera de servicio) o

BitTorrent[3], no podemos aplicarlas de manera similar cuando lo que se trata es de realizar streaming multimedia. En las aplicaciones de compartición de archivos, estos son divididos en bloques de datos, por lo que los clientes solicitan dichos bloques siguiendo la regla "*el más raro primero*" y los usuarios intercambian bloques entre ellos en cualquier orden. El orden en que llegan los bloques al receptor no es importante, ya que no será hasta que el receptor tenga todo el archivo, cuando podrá abrir el fichero descargado. Por otro lado, las aplicaciones de streaming tanto en vivo como bajo demanda, requieren que el orden en que llegan los bloques sea el adecuado para permitir la reproducción instantánea del contenido, imponiéndose unas limitaciones en el retraso de la llegada de estos bloques (delay), esto supondrá una necesaria modificación de los diferentes protocolos P2P para que puedan soportar las características multimedia.

El éxito que han tenido los sistemas P2P en la distribución de contenido a gran escala, hace que merezca la pena investigar su aplicación en la distribución de streaming multimedia. Sistemas como CoolStreaming [8] [9], han tenido bastante éxito en la distribución de streaming en vivo a un gran número de usuarios, pero… ¿ podríamos usar esas mismas técnicas para la distribución de VoD? .

En este proyecto se van a exponer los problemas actuales de las arquitecturas de VoD P2P y los aspectos que son necesarios abordar para buscar una solución a dichos problemas. Se implementará una arquitectura básica para la distribución de VoD P2P, que nos proporcione la base sobre la que poder realizar pruebas y proponer soluciones a los problemas planteados en este tipo de arquitecturas.

1.2 Motivación

A pesar del elevado número de investigaciones y estudios desarrollos en los últimos años sobre arquitecturas de VoD P2P, todavía no está claro cómo aprovechar de la mejor forma el paradigma P2P en la distribución de VoD, planteándose numerosos problemas para los que no se tiene una solución.

[3] http://www.bittorrent.com/intl/es/

En este escenario las oportunidades de investigación y desarrollo que se nos presentan son muy numerosas. Con este proyecto ofertado por el Departamento de Telemática de la Universidad Carlos III de Madrid, se pretende establecer la base de un diseño propio para la distribución de VoD P2P, que nos permita realizar pruebas e investigar en este campo.

Para esto se desarrollará un programa que de manera sencilla simule una red de compartición de archivo de vídeo bajo demanda peer to peer (VoD P2P). Este programa nos permitirá enfrentarnos a las cuestiones básicas en el diseño de este tipo de arquitecturas y nos brindará la base para seguir trabajando en este campo en sucesivos proyectos.

1.3 Objetivos

El objetivo del proyecto es diseñar e implementar una arquitectura básica para la distribución de VoD P2P. Se implementará un protocolo de comunicación Peer-Tracker y Peer-Peer, la arquitectura propuesta deberá implementar un sistema de Multi Video Caching (MVC), de manera que los peers realizarán un almacenamiento de los vídeos solicitados, compartiéndolos posteriormente con el resto de peers.

El objetivo final será hacer una prueba muy sencilla donde:

i. Un cliente se descargue parte del vídeo (como ningún otro equipo lo tiene aún, se lo descargará del servidor).
ii. Otro cliente se descarga el vídeo entero (se descarga la primera parte del primer usuario y el resto del servidor).
iii. Otro cliente se descarga el vídeo entero (se puede descargar parte del primer cliente y la otra parte del segundo).

1.4 Medios empleados

Durante toda la fase de desarrollo se ha usado un ordenador portátil de la marca Toshiba Tecra A11, y una conexión ADSL. Posteriormente en la fase de validación ha sido necesario emplear varios equipos y diferentes conexiones a la red Internet.

A continuación se describen e identifican los equipos y conexiones a red empleados durante las pruebas de validación realizadas en el capítulo 5.

Servidor*:* Toshiba Tecra A11, Procesador Intel Core i5, CPU 2.67 GHz, Memoria RAM 2,86 GB, S.O. W7 de 32 bits. Este es el pc en el cual se ha realizado el proyecto y en el que se han realizado las pruebas de la fase I.

Usuario-1: General Dynamics, Procesador Intel Pentium CPU 2 GHz, Memoria RAM 1 GB, S.O. W XP SP3 de 32 bits.

Usuario-2: Samsung, Procesador AMD A6-34020M APU, Memoria RAM 5,48 GB, S.O. W7 de 64 bits.

Usuario-3: Lenovo ThinkPad, Procesador Intel Core i3-2310, CPU 2.100Ghz Memoria RAM 2 GB, S.O. W 8.1 de 64 bits.

Las redes usadas las identificaremos como:

ADSL-1: Línea ADSL cuyo proveedor de servicios es ONO, y a la cual permanecerá conectado el Servidor. El Modem que da servicio a esta ADSL es de la marca Thomson TCW710.

ADSL-2: Línea ADSL cuyo proveedor de servicios es Movistar, el modem es de la marca "Observa Telecom".

ADSL-3: Línea ADSL cuyo proveedor es Movistar ubicada en el mismo edificio que la ADSL-2, el modem usado es de la marca "ADB Broadband".

UMTS-1: Conexión UMTS proporcionada por una tarjeta de datos del operador Movistar, el modem usado es un ZTE MF823 4G.

En la siguiente tabla se muestran los valores obtenidos al realizar un test de subida/bajada de las líneas ADSL. Hay que tener en cuenta que las líneas ADSL-2, y 3, son compartidas entre varios usuarios, por lo que la velocidad que se obtiene es menor de la esperada si estuvieran dedicadas en exclusividad al equipo de pruebas. El pc conectado a la tarjeta UMTS se encuentra en una zona de cobertura 3G.

	Test de Subida (Mbps)	**Test de Bajada (Mbps)**
ADSL-1	1	5.54
ADSL-2	0.26	2.54
ADSL-3	0.43	1.35
UMTS-1	3.77	10.05

Tabla 1. Velocidad Líneas ADSL

1.5 Estructura de la memoria

El proyecto se encuentra organizado de la siguiente manera: en el Capítulo 2 veremos un estudio del estado del arte, comenzando por una introducción al VoD (2.1), para a continuación ver las características de las redes P2P (2.2), y terminar viendo las principales características y problemas que encontramos en los actuales sistemas de VoD P2P (2.3). En el Capítulo 3 se verá la arquitectura planteada para el diseño del programa, así como los aspectos más relevantes abordados durante el diseño. En el Capítulo 4 se explicará la implementación llevada a cabo. En el Capítulo 5 se presentarán los resultados de las pruebas realizadas y el resultado de estas. En el Capítulo 6 veremos las herramientas de trabajo usadas en la fase de implementación y pruebas. En el Capítulo 7 hablaremos de las líneas de trabajo futuras que se nos plantean, para terminar en el Capítulo 8 con las conclusiones finales.

2 Estado del Arte

2.1 Introducción al VoD

VoD (Vídeo on Demand) es un tipo de streaming de vídeo que permite al usuario acceder a diferentes contenidos multimedia de una manera personalizada. Las plataformas de televisión convencionales no ofrecen al usuario ningún tipo de control sobre los contenidos, más allá de poder seleccionar una película de entre unas horas de emisión ya predefinidas. Sin embargo los usuarios están habituados a usar opciones de un DVD doméstico, como avanzar, rebobinar, detener la imagen, y esto conlleva a que exista una creciente demanda de servicios de ocio más interactivos.

Hoy en día, casi todos los espectadores quieren ser interactivos, en este sentido y favorecido por el elevado aumento en el ancho de banda de las actuales líneas ADSL, se ha propiciado un incremento en la comercialización de este tipo de servicios. Un ejemplo es el ofertado por la compañía Telefónica S.A. y denominado “Imagenio”, cuyo esquema de funcionamiento se muestra en la Figura 1, el cual, además de proporcionar servicios de vídeo en vivo, permite poder acceder a ciertos programas y películas cuando el usuario lo desee. Servicios similares son ofertados por compañías en países de nuestro entorno como son Fastweb en Italia, T-Online Vision en Alemania, MaLigne TV en Francia y Kingston Interactive TV en Reino Unido. Todas estas plataformas proporcionan un servicio conocido como *Tripleplay*, que ofrece televisión, VoD y acceso a Internet. Otros sistemas de VoD los podemos encontrar en la red de forma gratuita como es el caso de “YouTube”.

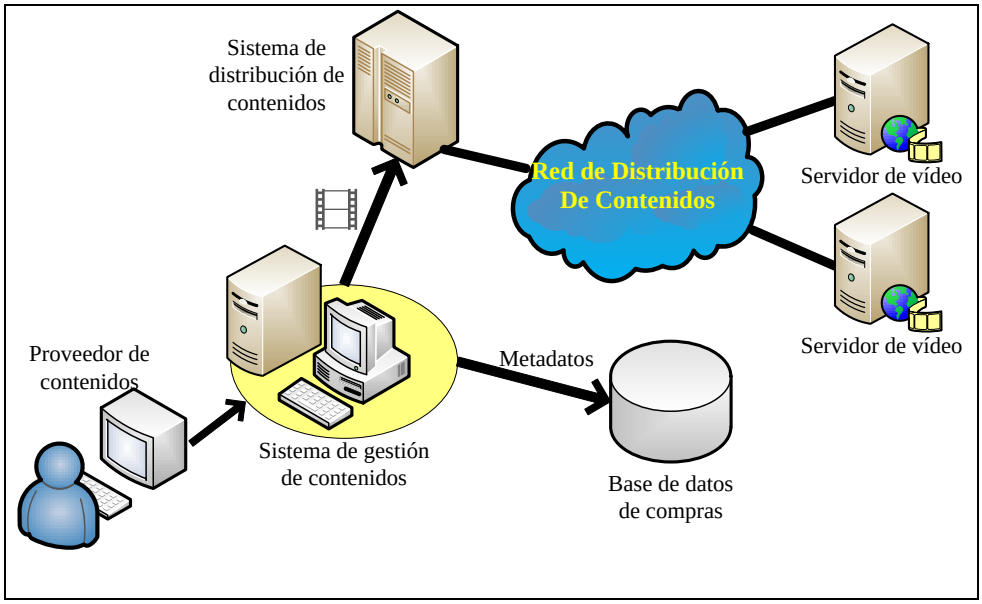


Figura 1. Funcionamiento de Imagenio [11]

Originalmente los sistemas de VoD basados en arquitecturas cliente-servidor, se podían clasificar en centralizados y distribuidos. Inicialmente se usaron arquitecturas centralizadas con un único servidor central, lo cual hacía el sistema muy susceptible a fallos, ya que había un único elemento que era el servidor del que dependía todo el sistema, además, cuando el número de usuarios aumentaba, el coste del sistema se disparaba si se querían evitar cuellos de botella en el servidor.

Una solución económicamente viable a estos problemas, fue el uso de sistemas distribuidos, CDN, donde los vídeos se almacenaban en diferentes servidores; obviamente no se replicaba todos los contenidos en todos los servidores, sino que solo los contenidos más famosos o actuales están replicados en todos los servidores, y los más antiguos o menos populares, se encuentran en unos servidores determinados, de manera que si a un servidor se le solicita un vídeo que no posee, este re-direcciona la petición hacia un servidor que sí disponga de este vídeo.

A continuación realizaremos una breve descripción de los componentes básicos de un sistema de VoD, de los requisitos básicos de estos sistemas y de los tipos de servicio de VoD que podemos encontrar.

2.1.1 Componentes básicos de un sistema de VoD

La estructura básica de un sistema de VoD consiste, en clientes, servidores de vídeo y una red que los conecte, como se puede ver en la Figura 2.

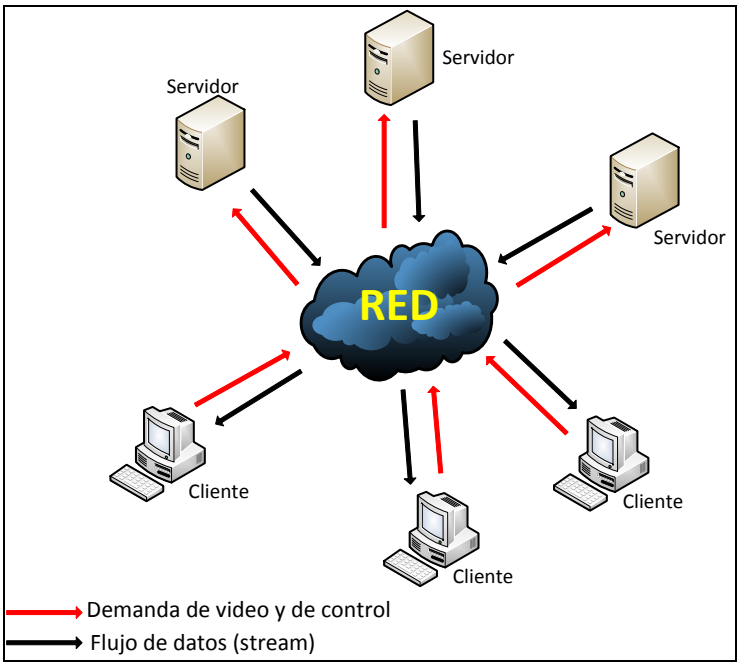


Figura 2. Sistema de VoD

El S*ervidor* será el encargado de almacenar los diferentes contenidos (vídeos), que serán distribuidos a los usuarios. El *Cliente* será quien solicite el contenido al servidor, y debe ser capaz de soportar la recepción y reproducción de los contenidos sin cortes, así como la opción de los comandos DVD. Finalmente la *Red de Comunicación* será el medio físico que ponga en contacto servidores y clientes.

Para que el cliente experimente un grado de satisfacción alto, las peticiones de los usuarios de la red, deben ser servidas con una calidad de servicio adecuada. En este aspecto, el servicio de VoD es un sistema exigente con la red, ya que requiere de una gran cantidad de ancho de banda para transmitir volúmenes de información de gran tamaño a una tasa elevada sobre todo en el lado del servidor.

2.1.2 Requisitos de un sistema de VoD

Los sistemas de vídeo bajo demanda deben satisfacer una serie de requisitos derivados de su funcionalidad [10][11], entre los cuales destacan:

- **Gran capacidad de almacenamiento:**

Los contenidos multimedia debido a su calidad de imagen y audio son cada vez de un tamaño mayor. Por ejemplo un vídeo de 2 horas en calidad media (MPEG-2) requiere aproximadamente unos 7.2 Gigabytes. Esto junto a que un servidor debería de gestionar una gran cantidad de contenidos, hace que nos encontremos ante almacenamientos del orden de TeraBytes.

- **Servicio en tiempo real:**

El servidor debe tratar en todo momento que el tiempo de espera sea el mínimo posible para todos los usuarios. Se debe garantizar una reproducción continuada de contenidos, por lo que el sistema debe tener una noción temporal de los hechos. El envío y recepción de información debe realizarse dentro de un tiempo determinado, para lo cual la aplicación de VoD debe realizar mediante estimaciones estadísticas del sistema en ese momento, una aproximación del tiempo de espera.

Una de las exigencias de red más importantes son las relativas al retardo. Hay un tipo de retardo que afecta mucho a la calidad de los servicios que se realizan mediante streams, este retardo se conoce como jitter, y se trata de la diferencia del retardo de unos paquetes respecto a otros. Para minimizar este efecto el servidor enviará datos de forma adelantada a los usuarios, que dispondrán de un buffer donde almacenarán los datos que reciben, de manera que dispondrá de un margen de tiempo para corregir los posibles retardos introducidos por la red a los siguientes paquetes de datos.

- **Gran ancho de banda:**

El hecho de que los contenidos multimedia requieran de un elevado ancho de banda, unido a la necesidad de dar servicio a un número potencialmente grande de usuarios, conlleva que estos sistemas tengan que transmitir una gran cantidad de información a través de la red de una manera continuada durante un largo periodo de tiempo, lo que hace necesario que el sistema sea diseñado contando con una red capaz de soportar el tráfico generado, y que el servidor sea capaz de

aprovechar el gran ancho de banda disponible. Si no se tiene en cuenta este parámetro en el diseño, puede llevar a que un incremento en el número de peticiones al sistema, puede aumentar tanto los requisitos de ancho de banda que llegue a saturar el sistema.

- **Calidad de Servicio (QoS):**

La calidad de servicio en estos sistemas, va a estar íntimamente relacionada con el G*rado de Satisfacción del Cliente* (Quality of Experience). Para conseguir este grado de satisfacción, el tiempo de espera desde que se solicita un video hasta que comienza la reproducción debe ser mínimo (unos pocos segundos), el video debe poder reproducirse con una buena calidad de imagen y sonido, y la reproducción tiene que ser continua (sin paradas ni cortes), lo cual requerirá de una buena sincronización entre cliente y servidor.

2.1.3 Tipos de servicios de VoD

Los servicios de VoD pueden llegar a ser muy complicados de implementar, por esto podemos encontrar diferentes tipos de servicios de VoD [12], cuya diferencia quedará determinada en función del grado de interacción permitido a los usuarios y su complejidad en la implementación del sistema de VoD estará directamente relacionada con este grado de interacción.

- **Pago por visión (Pay-per-View, PPV)**

Aunque aparece en la literatura como un tipo de servicio VoD, este no permite ningún tipo de interacción, simplemente acceder a un programa a una hora determinada por el proveedor de servicios.

- **Cuasi vídeo bajo demanda (Quasi-VoD, Q-VoD)**

En este caso el usuario no tiene el control sobre el canal, cuando solicita un vídeo, es agrupado por el proveedor de servicios, dentro de un grupo de usuarios que desea ver el mismo vídeo al mismo tiempo y lo transmite a todos ellos. La única opción de interacción que se le permite al usuario que por ejemplo quiere rebobinar a un momento del vídeo pasado, sería cambiándose a un grupo que comenzó más tarde.

- **Vídeo bajo demanda aproximado (Near-VoD, N-VoD)**

Aquí el proveedor transmite un mismo contenido a intervalos regulares (cada 5 ó 15 minutos por ejemplo), cuando un usuario realiza una petición, ésta es atendida por el siguiente canal que vaya a transmitir el contenido deseado, esto puede conllevar la espera por parte del usuario de un tiempo, en función del intervalo establecido por el proveedor. Cambiando al canal adecuado se ofrece una posibilidad de función de avance y rebobinado del programa.

- **Vídeo bajo demanda verdadero (True-VoD, T-VoD)**

Es el servicio más completo, donde el usuario decide qué quiere ver y cuándo quiere verlo, y dispone del control total sobre la sesión, soportando de manera general todos los comandos disponibles de un reproductor DVD.

2.2 Redes P2P

Básicamente una red P2P, es una red que no tiene clientes ni servidores fijos, sino una serie de nodos, que se comportan a la vez como clientes y como servidores de los demás nodos de la red.

Las redes P2P son un tipo de red overlay [13], y surgen como una alternativa al tradicional modelo cliente-servidor, con el objeto de aprovechar toda la capacidad de procesamiento, almacenamiento y ancho de banda sobrante de los nodos conectados a la red. Una red P2P, se refiere a una red que no tiene clientes y servidores fijos, sino una serie de nodos finales que se comportan a la vez como clientes y servidores de los demás nodos de la red.

2.2.1 Características de la redes P2P

Las redes P2P presentan una serie de características, que las diferencian de otro tipo de redes y que son las que precisamente le dan sentido para su aplicación en este tipo de servicios de VoD.

- **Escalabilidad:**

La escalabilidad de este tipo de redes va estar ligada con la cantidad de usuarios que se conecten a ella, ya que son estos mismos con sus recursos, los que contribuyen a aumentar los recursos disponibles en la red P2P.

- **Robustez:**

Dada por su naturaleza distribuida, y el dinamismo de ellas a la hora de rehacer las estructuras.

- **Descentralización:**

Formada por nodos iguales, lo que implica que ninguno de ellos es imprescindible.

- **Costes repartidos:**

Los recursos en forma de ancho de banda, almacenamiento, etc. son aportados por los usuarios, basándose toda la estructura P2P en esta colaboración entre usuarios.

- **Anonimato:**

Estas redes proporcionan cierto anonimato a los usuarios, lo cual en algunas ocasiones entra en conflicto con los derechos de autor, y para el caso de algunos servicios de VoD no parece una característica importante el proporcionar anonimato a los usuarios.

- **Seguridad:**

Este es uno de los puntos débiles de estos sistemas, siendo bastante difícil identificar aquellos usuarios que tienen unas intenciones no colaborativas o incluso dañinas con la red.

2.2.2 Clasificación de las redes P2P

Respecto a los mecanismos y protocolos en que se basan las redes P2P, podemos diferenciar dos componentes necesarios:

I. Mecanismos para la localización de recursos (como usuarios, servicios o contenidos).
II. Mecanismos de acceso o comunicación con los recursos (como puede ser el establecimiento de una llamada o la descarga de un fichero).

Comúnmente las redes P2P se clasifican en función de los Mecanismos para la localización de recursos en; redes *estructuradas*, redes *no-estructuradas* y redes basadas en *superpeers* [14].

- **Redes Estructuradas**

En este tipo de estructuras la topología de red es controlada y la información sobre los recursos (contenidos, localización de estos contenidos, etc.) se mantiene actualizada. Las solicitudes de datos son eficientemente dirigidas hacia los peers que contienen estos datos, lo que implica que el papel fundamental de estos sistemas sea construir y mantener una relación eficiente entre los datos y su localización y proveer una rápida respuesta a los datos solicitados. En este tipo de red los nodos mantienen información de routing sobre cómo llegar a todos los nodos de la red, cada peer tiene una tabla de routing local, la cual es inicializada cuando el peer se une a la red overlay, usando un procedimiento de inicio (bootstrap) específico.

Estas redes ofrecen un servicio de encaminamiento fiable capaz de enviar un mensaje con una clave asociada a un solo nodo, responsable de esa clave. A este servicio se le conoce como *Key-Based-Routing* (KBR), en el cual, un conjunto de claves se asocia con unas direcciones, de tal manera que el peer más cercano a una dirección almacena los valores de las claves asociadas, y el algoritmo de encaminamiento trata las claves como direcciones.

Una de las técnicas usadas para gestionar las tablas de routing, y que es implementada sobre la interfaz KBR, son las denominadas DHT (Distributed Hash Table). Se trata de un sistema de almacenamiento de datos fiable y escalable, que almacena bloques de datos en cientos de máquinas conectadas a Internet, que replican los datos para aumentar la fiabilidad, y además son capaces de localizar datos rápidamente, aunque sean ejecutados sobre enlaces de baja calidad. Algunos de los sistemas basados en DHT son Chord [15]., Pastry [16], Tapestry [17].

- **Redes No-estructuradas**

A diferencia de las redes estructuradas, no hay una topología fija en las redes no-estructuradas. Estas redes presentan una arquitectura topológica plana, donde todos los nodos pertenecen al mismo nivel. Encontramos diferentes diseños en función del grado de centralización: *centralizados*, *híbridos* y *descentralizados puros*.

- **Redes P2P centralizadas**. En este tipo de redes, todas las gestiones se realizan a través de un único servidor central que sirve de punto de enlace entre los diferentes usuarios. Esto implica, que aunque la transferencia de datos no sea centralizada, el nivel de

escalabilidad se verá disminuido, debido a los recursos limitados que posee un servidor. Por otra parte la privacidad también se ve afectada dado que el servidor debe mantener dinámicamente un control de los nodos presentes.

Sus principales características son:

- Hay un único servidor el cual es el encargado de ser el punto de enlace entre los diferentes nodos y distribuir el contenido a petición de los nodos.
- Todas las comunicaciones dependen del servidor.

Estas características convierten al servidor en una entidad central del que depende todo el diseño del sistema, siendo este el principal inconveniente de este tipo de redes, ya que esta entidad central se convierte en un único punto de fallo. Si esta entidad central deja de funcionar, todo el sistema se verá seriamente degradado.

Como se comentó anteriormente otro de sus principales inconvenientes es la escalabilidad, debido a que cuando el número de usuarios aumenta y el tamaño de la red crece, el servidor puede ser un cuello de botella para todo el sistema.

- **Redes P2P híbridas**. En esta versión de las redes P2P el servidor central actuaría a modo de hub, administrando el ancho de banda, realizando funciones de enrutamiento, pero siempre sin conocer las características de los nodos ni mantener información sobre las diferentes conexiones. Una clara ventaja de este sistema la encontramos en el hecho de que si el servidor principal cae, las conexiones existentes entre los diferentes nodos posibilitarían continuar con las transferencias de datos de una manera normal.
  Sus principales características son:
  - Posee un servidor central que mantiene información en espera y responde a peticiones sobre esa información.
  - Los nodos son los responsables de hospedar toda la información.
  - Los terminales de enrutamiento son direcciones usadas por el servidor, administradas por un sistema de índices para obtener una dirección absoluta.

- **Redes P2P totalmente descentralizadas**. No requieren de ningún equipo central que las gestione. Consiste en una red overlay plana, donde todos los nodos son iguales. Aunque tiene un diseño sencillo y requiere para su mantenimiento mucha menos información sobre los contenidos, presenta problemas de escalabilidad. Esto es debido a causa del tráfico generado por el mecanismo de distribución de solicitudes, el cual crece de una manera lineal con el tamaño de la red.

  Sus principales características son:

  - Los nodos realizan funciones de cliente y de servidor.
  - Inexistencia de un servidor central.
  - Carencia de un enrutador central que sirva como nodo y administre direcciones.

  Uno ejemplo es Gnutella [18], donde si un peer quiere unirse a la red, este se conectará a otro peer que ya pertenezca a esta red, y los peers conectados proporcionarán al nuevo peer, información (direcciones IP y puertos) sobre otros peers que ya se encuentran en la red. Las solicitudes son distribuidas en la red, basadas en mecanismos de inundación (flooding).

- **Redes basadas en Superpeers**

En alguna literatura podemos leer clasificaciones donde a este tipo de redes se le denomina, “redes sin estructurar híbridas” [19]. Este tipo de topologías intentan mejorar algunas de las características de las redes no-estructuradas, como el retardo y el rendimiento en el re-encaminamiento. Presentan una arquitectura topológica jerárquica, donde los clientes se conectan a un super-peer que a su vez se conecta a un conjunto de superpeers vecinos. Cada super-peer, mantiene un registro de los datos de sus clientes. Cuando un cliente desea enviar una solicitud a la red, la enviará a su super-peer, si el super-peer encuentra algún resultado lo devolverá al cliente, en caso contrario, enviará la solicitud a sus super-peers vecinos y devolverá las posibles respuestas al cliente.

Estos super-peers son diseñados con un gran ancho de banda, mayor capacidad de almacenamiento y mayor potencia de procesamiento. Pueden desempeñar los papeles de servidores centrales, facilitando la búsqueda, asignando contenidos etc. Incluso pueden crear estructuras overlay entre ellos con el objeto de mejorar la eficacia en la búsqueda.

Debido a estas características este tipo de redes no depende de un único servidor, al distribuirse la base de datos entre los super-peers, estas base de datos pueden reducirse en tamaño, al necesitar mantener cada super-peers, únicamente los contenidos de los peers asignados. Pero también presentan varios inconvenientes, como el requerir de un diseño complejo y un mantenimiento de la red overlay entre supe-peers.

2.3 VoD P2P

Uno de los principales objetivos de los servicios de VoD, es diseñarlos de tal manera, que soporten un gran número de solicitudes simultáneas generadas por clientes ampliamente distribuidos geográficamente; estos sistemas deben conseguir un servicio viable a gran escala, y para que estos sistemas de VoD alcancen una gran expansión, es necesario dotarlos con mecanismos que los hagan escalables, capaces de dar servicio a un elevado número de usuarios. Estos sistemas se denominan "Sistemas de Vídeo bajo Demanda a Gran Escala" (LVoD), y deben ser capaces de cumplir con los siguientes tres requisitos; elevada escalabilidad, una arquitectura tolerante a fallos y una redistribución de la carga de trabajo de forma equitativa entre los distintos componentes del servicio.

Como ya se ha mencionado, las aplicaciones P2P han tenido un gran éxito en la distribución de contenidos a gran escala, particularmente en la compartición de archivos y en Live Streaming. Por este motivo se ha pensado que la tecnología P2P puede ser usada para proporcionar servicios de VoD a gran escala, siendo muy numerosos los estudios que se han venido haciendo al respecto en los últimos años [20].

2.3.1 Tipos de Redes en Sistemas VoD P2P

Generalmente los sistemas P2P que nos encontramos presentan una arquitectura no-estructurada; dentro de esta, los protocolos más utilizados han mantenido un enfoque centralizado, bien impuesto por el propio protocolo o por la propia comunidad a la hora de recomendar y optimizar la compartición de contenidos, un ejemplo es BitTorrent el cual necesita de un servidor (tracker) que coordine la descarga, y de unos archivos (.torrent) necesarios para empezar esta descarga. Pero cada vez se está apostando más por la descentralización de estas redes, basándose en el uso de sistemas de recomendación e indicadores de reputación.

Los actuales sistemas VoD P2P, se basan principalmente en dos categorías relacionadas *"con la organización de los nodos participantes"*, redes VoD P2P basadas en malla y en árbol.

Las redes basadas en *árbol*, presentan un mecanismo de transporte bien organizado, que permite una latencia relativamente baja a la hora de enviar los contenidos entre el servidor y el cliente. El principal problema es la dificultad de preservar la arquitectura de árbol, debido a que los peers entran y salen del sistema con mucha frecuencia, una posible solución propuesta, es el uso de arquitecturas múltiples de árbol, donde se mantienen varias estructuras de árbol, y si un peer deja el sistema, sus hijos pueden continuar recibiendo contenido de otros árboles. Podemos encontrar diferentes propuestas de redes en árbol como BBTU [21], VMesh [22], rVoD [23], P2Cast, oStre[24], DirectStream [25]. Sin embargo es conveniente mencionar que no existen aplicaciones comerciales P2P basadas en soluciones de árbol.

Debido principalmente a este dinamismo entre nodos, es la topología de *malla*, la que ha demostrado ser más eficiente para la provisión de VoD P2P. Este tipo de topología presenta una alta eficiencia, permitiendo a los peers intercambiar bloques entre ellos simultáneamente, la relación entre los nodos son establecidas en función de los contenidos y el ancho de banda de estos. Estas redes se basan normalmente en la auto organización de los propios peers. Este tipo de arquitecturas suelen ser las más adecuadas en aplicaciones sensibles al retardo, intentando conseguir una rápida descarga de contenidos mediante swarming. Con este sistema, el servidor divide el fichero y lo distribuye a los usuarios que se lo solicitan. Los usuarios pueden descargar de sus vecinos los segmentos que no tienen, de esta manera en todo momento se mantienen múltiples vecinos, lo que hace a estos sistemas altamente robustos a los fallos de los nodos. BitTorrent es probablemente la aplicación más conocida que usa redes en malla, otros servicios de VoD P2P que usen redes en malla son BASS [26], Give-to-Get [27], PROMISE [28], pcVoD [29] .

2.3.2 Principales Problemas de los Sistemas de VoD P2P

Los sistemas de VoD requieren para su correcto funcionamiento de unas características más exigentes que los tradicionales sistemas de compartición de archivos. Será necesario diseñar unas políticas adecuadas de transmisión de datos, así como mecanismos de recuperación de fallos, que son esenciales si queremos ofrecer una calidad de servicio (QoS) adecuada dentro de una red con una alta probabilidad de fallos, como es Internet.

Las características intrínsecas de las redes P2P, principalmente su dinamismo, unido a los requisitos que impone el streaming de vídeo y las limitaciones de la red Internet, hacen que surjan una serie de problemas en el diseño de estos sistemas que es necesario abordar. Estos problemas pueden variar en función de la literatura que consultemos [30] [31], apareciendo como principales problemas:

1) *Manejo adecuado de las peticiones asíncronas de los clientes*: Las peticiones de los peer llegan al sistema en momentos diferentes. Se espera que el sistema entregue el vídeo completo a cada nodo.
2) *Recuperación robusta de fallos*: Se espera que ocurra un mayor número de fallos de conexión en los sistemas P2P (debido a que muchos nodos son terminales hogareños), el protocolo de recuperación de fallos debe reorganizar la red rápidamente con los nodos disponibles.
3) *Sobrecarga de control pequeña*: las tareas de gestión deben permitir que la sobrecarga de control se mantenga pequeña para hacer el sistema escalable.
4) *Usuarios Maliciosos (Trust)*: que intentan modificar el contenido de los streams de vídeo. Diferentes estudios [32] han encontrado que más del 50% de algunos archivos muy populares en Internet estaban infectados. En distribuciones P2P el contenido es descargado tanto del servidor como de otros usuarios, estos sistemas deben de ser capaces de identificar al proveedor de contenido y por otro lado detectar una posible modificación.
5) *Tráfico en la red*: elevado coste en la distribución de datos a través de los ISPs.
6) *Firewalls:* que limitan la conectividad en redes P2P, las cuales confían en disponer de conexiones directas entre usuarios finales, y los firewalls pueden impedir estas conexiones. Esta limitación afecta de manera especial a los sistemas de VoD, debido a la naturaleza lineal del streaming.
7) *Freeloaders*: que reducen la eficiencia del sistema, no contribuyendo a este. Se ha comprobado como en algunos sistemas los usuarios contribuyen poco o nada. Esto es un problema en los sistemas P2P pero aún mayor en aplicaciones de VoD donde un cliente solo puede contribuir a nuevos clientes que han llegado después de él.

En la Tabla 2, se muestra el resultado del análisis de diferentes sistemas y las funcionalidades que estos implementan. Un paréntesis indica que el sistema no parece que haya sido diseñado con esta funcionalidad pero que hay aspectos del diseño de este que ofrecen alguna funcionalidad al respecto. Muchos sistemas disponen de un cierto nivel de veracidad de los contenidos (Trust) respecto a que el contenido es recibido de un servidor bien definido, pero menos de la mitad disponen de mecanismos para protegerse de contenido malicioso. Respecto a los otros tres aspectos (caching, firewall, incentivos) son ignorados por la mayoría de los sistemas.

Sistema	Trust	Caching	Firewall	Incentivos
Π-Stream	X/	(X)	---	---
BASS	X/X	---	(X)	(X)
CDN-P2P hybrid	---	(X)	(X)	---
COPACC	X/X	X	(X)	---
CoopNet	X/X	---	(X)	---
DirectStream	X/--	---	---	---
GridCast	X/--	---	(X)	---
P^2VoD	X/--	---	---	---
P2Cast	X/--	---	---	(X)
PALS	---	---	---	---
PBA	X/--	---	X	---
PROMISE	---	---	---	X
PROP	X/--	X	(X)	---
PeerStreaming	X/X	---	---	---
Push-to-peer	X/(X)	---	---	---

Tabla 2. Funcionalidades implementadas en sistemas VoD P2P [32]

Hay una serie de aspectos que se considera necesario abordar si se pretenden buscar una solución eficiente a muchos de los problemas que aparecen en los sistemas de streaming P2P, algunos de estos son [33]:

a) *Mecanismos apropiados de codificación de vídeo*: Debido a la naturaleza de los contenidos multimedia, estos resultan ser muy sensibles a través de redes que no ofrecen garantías de transmisión. Por lo que se hace necesario implementar mecanismos de codificación eficientes que nos disminuyan tanto el tamaño como el bitrate de los vídeos a distribuir.
b) *Manejo dinámico de los peers*: Debido al comportamiento impredecible de los peers, (pueden unirse y dejar el sistema sin notificarlo a otros peers), se hace necesario proveer al sistema de mecanismos robustos y adaptables que manejen estos cambios en la estructura. De manera que cuando un emisor deja el sistema, este cambio sea detectado

lo más rápidamente posible y este peer sea reemplazado por otro peer emisor, al objeto de prevenir una interrupción del servicio.

c) *Capacidades heterogéneas de los peers*: Las capacidades que ofrecen los peers, como ancho de banda y capacidad de almacenamiento, son muy heterogéneas. Un mecanismo de selección del peer debe ser capaz de abordar estos problemas.

d) *Construcción de redes overlay eficientes*: Una topología de red no adecuada, puede generar un incremento de la sobrecarga de control que reduzca la eficacia del sistema.

e) *Selección de los mejores peers*: Se debe introducir estrategias eficientes que permitan la selección de los mejores peers emisores, con el objeto de minimizar el retardo extremo a extremo (E2E delay), teniendo en cuenta el número de nodos intermedios que se atravesarán y las políticas de routing.

f) *Monitorización de las condiciones de red*: Las condiciones de la red pueden cambiar drásticamente, debido a las características dinámicas de las redes P2P. El monitorizar las condiciones de la red es necesario para maximizar la utilización de las fuentes disponibles y minimizar la perdida de paquetes en ciertos enlaces.

g) *Incentivos para los peers participantes*: Muchos estudios muestran como el comportamiento de algunos peers en redes P2P se limita a descargarse contenidos, sin compartir sus propios recursos. Un estudio del año 2000 [34] presenta como el 70% de los usuarios de Gnutella solo se descargan contenidos, no compartiendo los propios. Con este tipo de comportamiento donde nadie quiere compartir su ancho de banda pero quiere usar el de otros, las redes P2P se comportan como sistemas cliente-servidor y fallan cuando el número de clientes aumenta.

Los numerosos aspectos que se deben abordar en las futuras implementaciones de sistemas de VoD P2P dejan a la vista el estado incipiente de estos sistemas. A continuación hablaremos de diferentes implementaciones de sistemas VoD P2P, y de cómo tratan estos sistemas, de minimizar los problemas descritos hasta ahora.

2.3.3 Sistemas de VoD P2P

A pesar de que la mayoría de los sistemas propuestos a día de hoy presentan unos principios básicos de funcionamiento muy similares, podemos encontrar variaciones sustanciales entre ellos, ya que es muy difícil que un sistema pueda abordar todos los problemas que pueden surgir en este tipo de redes. Así de manera general, los estudios más recientes se centran principalmente en abordar aspectos relacionados con; *mecanismos robustos de recuperación de fallos* (tanto los propios de la red Internet, como los debidos a la naturaleza dinámica de los usuarios), *búsqueda de la arquitectura de red más eficiente y escalable* (centrándose principalmente en el lado del cliente mediante técnicas de selección de peers), y el uso de *técnicas de caching* (los peers almacenan segmentos en memoria para compartirlos con otros peers) *y prefeching* (los peers replican segmentos de forma proactiva).

Algunos de los sistemas propuestos para este tipo de aplicaciones son: Π-Stream [35], el cual usa un proxy en un host local para permitir la reproducción estándar. BASS [36] combina BitTorrent y un sistema VoD basado en servidor para la descarga de los bloques a reproducir, a menos que estos hayan sido descargados con anterioridad vía BitTorren. DirectStream [37] usa un servidor como director para almacenar contenido y mantener información sobre clientes. En P2VoD [38][39] los contenidos recientemente recibidos por los clientes y almacenados en cache, son enviados a nuevos clientes si el ancho de banda de salida es suficiente, aquí los nodos se agrupan en generaciones (basándose en los datos que reciben), de esta manera cuando un peer cae, otro miembro de su generación puede reemplazarlo. GnuStream [40] construido sobre Gnutella, tiene en consideración el dinamismo y la heterogeneidad de las redes P2P, proporciona un control adaptativo del buffer, mantenimiento de la calidad del streaming y detección de la degradación o fallo de los peers, la recuperación ante estos fallos es manejada a través de la selección del mejor peer emisor. CollectCast [41] monitoriza la red y el estatus de los peers. PROMISE [42] es un sistema de streaming construido sobre CollectCast, el cual implementa las siguientes funcionalidades: (1) Selección del mejor peer emisor, (2) monitoriza las características de la red asignando segmentos de datos y ratios a los peers emisores, y (3) switching dinámico entre los peers emisores.

- **GridCast**

Mención aparte merece GridCast [43] considerado realmente, como el primer sistema de VoD P2P que ofrece un conjunto de operaciones VCR, este sistema ha sido desarrollado en China, y es usado en CERNET (China Education and Research Network) desde mayo del 2006. CERNET es una red de Internet de propósito general y es la principal red de acceso para las universidades chinas, los estudiantes usan esta red para muchas aplicaciones como: web, email, chat y noticias. En diciembre del 2006, aproximadamente 23.000 usuarios veían vídeo a través de GridCast.

El diseño básico de nuestro proyecto es bastante similar al de GridCast, por lo que analizaremos este sistema con algo más de detalle. GridCast se construye básicamente sobre la idea de una arquitectura P2P, que nos permita que los peers compartan datos unos con otros aliviando de este modo la carga en los servidores.

Los vídeos serán divididos en segmentos (chunks), estos segmentos tienen una duración fija de 1sg, la razón de que los segmentos tengan una duración fija y no un tamaño fijo, viene determinada por la facilidad que ofrece este modelo, a la hora de permitir al usuario interactuar con el vídeo mientras se reproduce. La aplicación comienza la reproducción tras recibir los 10 primeros sg, de igual manera sucede cuando se produce un salto en el vídeo.

Para este diseño, GridCast usa una arquitectura híbrida, que se muestra a continuación, formada por un portal web, tracker, sources y peers:

- *Portal Web*: Presenta un catálogo de películas disponibles a los usuarios. El usuario, llega al portal, busca entre el catálogo de vídeos y selecciona uno.

- *Tracker Server* (tracker): El tracker es el punto de encuentro de los peers. El tracker mantiene una lista de peers que se han unido a la red. Esta información es usada para facilitar la compartición de datos entre los peers, y se actualiza cada cierto tiempo.

- *Sources Serves* (sources): Estas fuentes proporcionan la base de contenidos disponibles, almacenando de forma permanente una copia completa de cada vídeo, y poniéndolos a disposición de los peers cuando estos datos no pueden ser servidos por otros peers.

- *Peers*: Los peers reciben segmentos de sources (servidores) u otros peers, y los almacena en

una memoria local. Con estos datos almacenados, realiza dos funciones, reproducirlos en local y compartirlos con el resto de peers del enjambre. Los datos más recientes son almacenados en la memoria local, cuando los datos almacenados en esta superan los 300 sg (aprox. 18 MB para un bitrate de 60 Kbyte), un algoritmo pasa parte de los datos de la memoria local al disco duro. Un peer de GridCast tiene tres componentes; el cliente peer (recibe los segmentos), un media server (almacena los datos recibidos) y un media player (decodifica los datos y los reproduce).

-

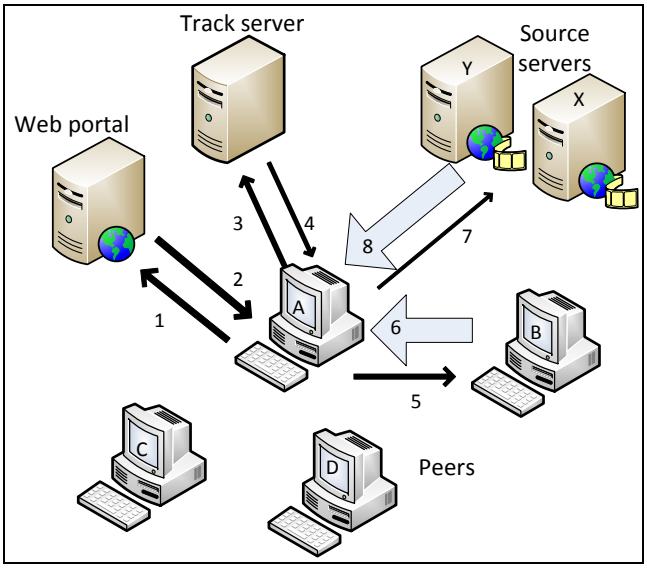


Figura 3. Arquitectura Básica de GridCast [43]

El diagrama muestra la interacción típica entre los componentes de GridCast. Las flechas finas representan metadatos o mensajes de control, las flechas gruesas transferencias de datos de vídeo. En la flecha 1, el Peer A contacta con el portal web para consultar el catálogo y seleccionar un vídeo. El web portal devuelve el ID del vídeo en la flecha 2. En la flecha 3, el Peer A contacta con el tracker enviándole el ID del vídeo y su estado actual. El tracker usa esta información y elabora una lista de peers candidatos, que es enviada en la flecha 4. A partir de este momento el Peer A solicitará segmentos del vídeo seleccionado, a otros peers o al source server, una solicitud al Peer B se muestra con las flechas 5 y 6, y con 7 y 8 se muestra una solicitud al server Y.

GridCast organiza a los peers que están visualizando el mismo vídeo en una red overlay, esta red tiene dos propósitos, el primero es disponer de unos peers con un elevado potencial de compartición, y el segundo difundir entre los peer una serie de metadatos de manera que estos puedan tomar decisiones localmente con una información amplia y oportuna.

El sistema realiza una serie de operaciones básicas, algunas de ellas necesarias para establecer la comunicación e iniciar el proceso de descarga, y otras le permiten interactuar con el vídeo.

- *Join*: Para unirse al sistema, el peer notifica mediante un mensaje UDP al tracker su intención de unirse al sistema, el peer debe enviar periódicamente un mensaje keep-alive para que el tracker lo mantenga en su lista.

- *Leave*: Cuando un usuario deja el sistema, el peer informa al tracker y al resto de usuarios cerrando las conexiones establecidas.

- *Start*: Cuando el usuario selecciona un vídeo comienza una sesión. El peer envía un identificador del archivo de vídeo al tracker, el cual devuelve al peer un listado de posibles candidatos a los cuales solicitar segmentos de este vídeo.

- *Play*: Para obtener una reproducción continuada, el peer realiza dos tareas, solicitar y recibir segmentos de la red y alimentar el reproductor multimedia. Cada 10 sg. el sistema evalúa los peers candidatos basándose en la proximidad de los contenidos, y selecciona el peer apropiado y los datos a solicitar.

- *Seek*: Con un salto el peer selecciona una nueva posición de reproducción del vídeo, en este instante se realiza una nueva sincronización con el tracker, para obtener una nueva lista de candidatos.

- *Stop*: Cuando se detiene la reproducción, el peer sigue recibiendo datos, almacenando estos y poniéndolos a disposición de otros peers. El peer solo cesa de almacenar y servir datos si el usuario cambia de vídeo o deja el sistema.

Gridcast ha propuesto el uso de técnicas de caching y replication, para mitigar tanto los problemas que produce la salida aleatoria y sin previo aviso de peers que se encontraban compartiendo archivos (comportamiento que llega a ocasionar el 40% de la carga en los servidores),

como los efectos producidos cuando se realizan saltos aleatorios mientras se visualiza un vídeo, produciéndose una cierta latencia que afecta a la calidad en la experiencia (QoE) del usuario.

Con caching el peer guardará los chunk recibidos en memoria o en el disco duro para compartirlos en un futuro con otros peers; mientras que con replication, un peer envía chunks a otros peers de una forma proactiva, es decir envía chunks a otros peers sin que lo hayan solicitado antes de que éste deje la red.

3 Diseño

3.1 Arquitectura básica

El diseño propuesto en este proyecto se ha basado en gran medida en la aplicación GridCast, y al igual que esta aplicación, la arquitectura que se pretende implementar será una arquitectura híbrida dentro del tipo de redes P2P No-estructurada. En nuestra arquitectura, el servidor central pondrá a disposición de los usuarios un catálogo de vídeos (a través de un Web Portal o de forma similar), mantendrá actualizada la información relativa al estado del enjambre (Track Server) y dispondrá de una copia íntegra (Source Server) de todos los vídeos disponibles en el catálogo. El peer se unirá a la red P2P al realizar su conexión con el servidor central y tras solicitar un vídeo del catálogo, recibirá del servidor información relativa al vídeo y a los potenciales peer-fuentes conectados. El peer-cliente realizará la selección del peer-fuente y establecerá las conexiones con las fuentes sin necesidad del servidor central.

El sistema propuesto pretende seguir una arquitectura y esquema de funcionamiento similar al de la Figura 3. Arquitectura básica de GridCast. La diferencia más notoria entre este esquema de GridCast y nuestro diseño Figura 4, lo encontramos en que para el diseño de nuestra aplicación, un único servidor “BootStrapServer” realizará las funciones de “Web Portal, Track Server y Source Server”.

Al igual que GridCast, usaremos la técnica de Caching en los peers, para almacenar (en memoria o disco) los segmentos del vídeo descargado y poder compartirlos con otros peers. Una de las diferencias que podremos encontrar en nuestro diseño en comparación con GridCast, será la segmentación de los chunks, que en nuestro caso tendrá un tamaño fijo independiente de la duración de este, al contrario que GridCast que segmenta el vídeo en chunks de 1 sg de duración.

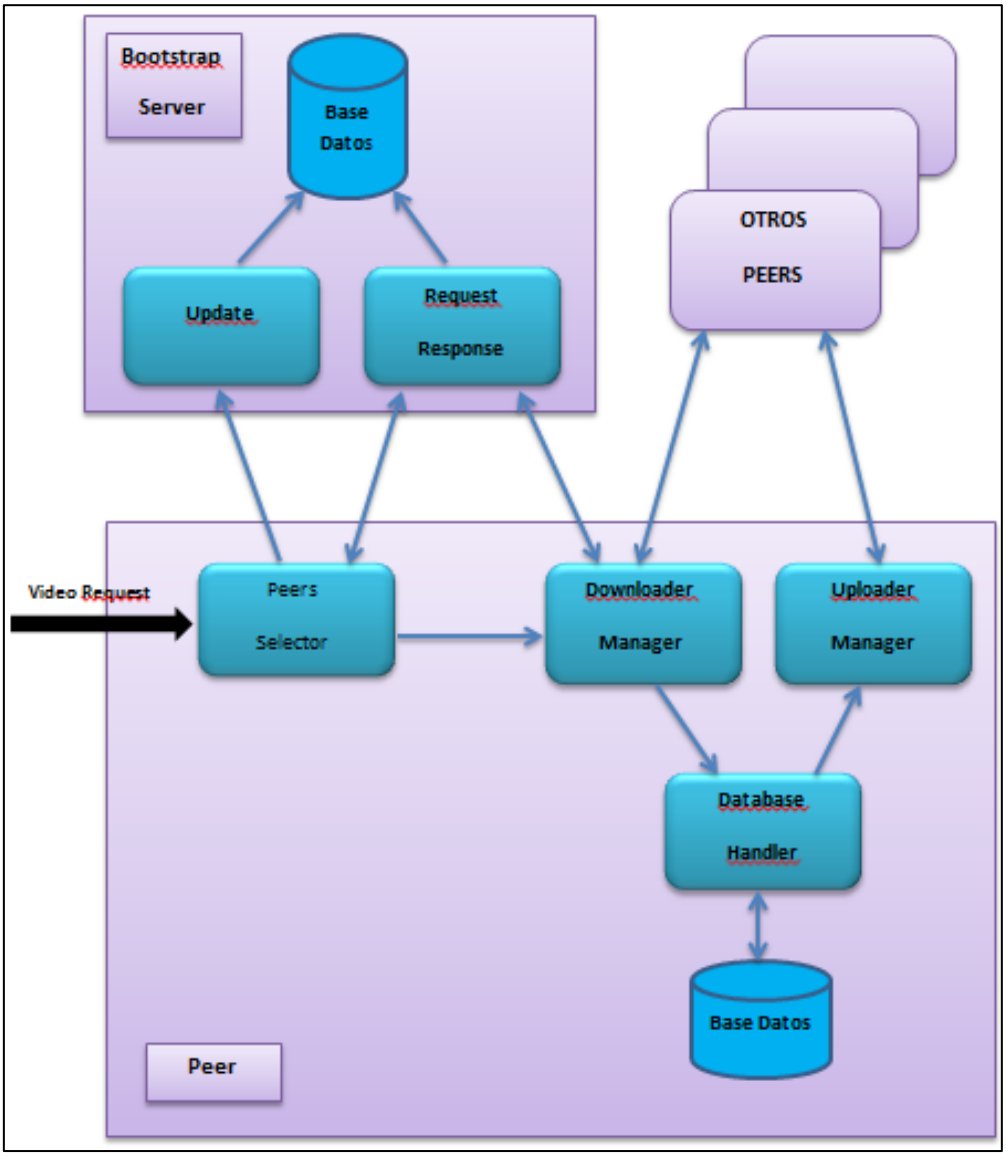


Figura 4. Diseño propuesto

- **Servidor (BootStrapServer)**: Este elemento realiza las funciones de Tracker Server y Source Server, solo existirá uno en el sistema, y sus funciones principales serán las de mantener una copia completa de los vídeos y coordinar el funcionamiento del enjambre. Para esto, dispondrá de una serie de metadatos correspondientes a estos vídeos (tamaño del vídeo, bitrate, nº segmentos del vídeo, md5 de cada segmento, identificador de cada vídeo), mantendrá actualizada la información necesaria para el funcionamiento del sistema (IPs de usuarios conectados al enjambre y vídeos que cada uno de estos tiene disponibles) y gestionará las peticiones de vídeos que recibirá de los peers.

- **Peer (Usuario)**: Al inicio de cada conexión el peer contacta con el servidor para unirse al enjambre, e informar de los vídeos que dispone en memoria (vídeos que previamente han sido descargados a través del sistema) y que pone a disposición del enjambre.
  El peer podrá actuar como *cliente:* peer que solicita un vídeo, y se lo descarga (del servidor o de otros peers), como *fuente:* peer que posee el vídeo (completo o parte de este) y lo comparte con otros peers que se lo solicitan, o simultáneamente como cliente y fuente, en el caso que durante la descarga de un vídeo por parte de un peer, este recibe la solicitud (de otro peer) de servir un vídeo que tiene en memoria.

3.1.1 Nodo Servidor (BootStrapServer)

El nodo Servidor (BootStrapServer), se ha diseñado pensado en una arquitectura híbrida, en este tipo de arquitecturas el servidor central aunque no interviene en la descarga de datos multimedia entre peers, sigue siendo un elemento imprescindible para el sistema, presentado los problemas conocidos en las arquitecturas centralizadas (cuello de botella ante múltiples solicitudes de usuarios, punto común de fallo).

Para el diseño del servidor se determinarán una serie de objetivos que expondremos a continuación y que posteriormente se irán materializando en acciones concretas durante la etapa de implementación.

El servidor deberá disponer de una base de datos donde almacenará: una copia de todos los vídeos disponibles, información de estos vídeos (metadatos), e información de la red P2P (enjambre) referente a peers conectados y vídeos disponibles en estos peers. Cada vez que un peer se conecta al servidor, el servidor guarda su dirección IP y el listado de vídeos que tiene disponible este peer, enviando a continuación al peer, el catálogo de los vídeos que el servidor pone a disposición de los usuarios. El peer seleccionará un vídeo del catálogo, esta petición del vídeo llegará al servidor quien responderá con la información disponible tanto del vídeo como del enjambre, para que el peer-cliente pueda iniciar el proceso de solicitud del vídeo.

Los dos bloques principales que conforman el servidor para su diseño serán; la estructura de *Base de Datos*, y el bloque de *Request/Response*.

- **Base de Datos**

La Base de Datos del servidor es el bloque fundamental de este elemento, convirtiéndolo único en la red P2P. Esta base de datos permitirá al servidor el poder realizar sus dos cometidos principales; el primero, de *Tracker Server*, para poder realizar esta función la base de datos del servidor guardará toda la información relativa al enjambre, manteniendo actualizado los peers conectados y los vídeos que estos ponen a disposición de la red, también almacenará determinados metadatos de los vídeos, estos metadatos serán puestos a disposición de los peers durante el proceso de solicitud y descarga de un vídeo. El segundo cometido principal que realizará el servidor, será actuar como *Source Server* (Servidor Multimedia) poniendo a disposición de los clientes los vídeos requeridos, siempre que no haya otro peer conectado al enjambre que lo tenga disponible.

- **Request/Response**

Este bloque alberga la estructura que nos permite atender las solicitudes de los peers, tanto de conexión a la red como de descarga de vídeos. De esta manera mantendremos actualizada nuestra base de datos, en lo que al estado del enjambre se refiere (peers disponibles y vídeos que ponen estos a disposición de la red), y pondremos a disposición de los peers-clientes los segmentos de vídeo de los cuales no haya disponibilidad por parte de ningún otro peer en la red.

Para establecer las comunicaciones entre el servidor y el peers, se crearán dos ServerSocket en el servidor, uno para el tráfico de mensajes de control y otro para el envío de datos multimedia (vídeo).

3.1.2 Nodo Peer (Usuario)

El Nodo Peer, pueden tomar cualquiera de los dos roles definidos para este nodo, como cliente, en el caso que este nodo solicite la descarga de un vídeo (peer-cliente), o como fuente (peer-fuente) en el caso que le sea solicitado un vídeo, como dijimos anteriormente se podrá dar el caso de que un nodo Peer actúe simultáneamente como cliente y como fuente.

El diseño propuesto para el nodo peer, será el siguiente:

i. El peer dispondrá de una interfaz gráfica donde el usuario introducirá los datos del servidor al que se quiere conectar, esta interfaz mostrará un catálogo de vídeos disponibles en el servidor, y tendrá un campo donde irán apareciendo determinados logs sobre el funcionamiento de la aplicación.

ii. Debe disponer de una BBDD, donde almacenará los vídeos que se va descargando en sucesivas solicitudes.

iii. Con la información que reciba del servidor referente al vídeo solicitado, tienen que poder, en primer lugar solicitar el bitmap a todas las posibles fuentes, y a continuación en base a estos bitmaps recibidos de los peers-fuentes, iniciar un procedimiento para la descarga del vídeo.

iv. Cuando el nodo peer actué como fuente, deberá recibir las solicitudes de otro peer-cliente donde se le solicita un segmento de un vídeo, esta solicitud le llegará a través de un server socket que dispondrá todo cliente.

v. El vídeo se reproducirá en un interfaz que permita al menos detener y reanudar la reproducción.

3.2 Aspectos relevantes del diseño

A continuación se tratarán algunos puntos del diseño que se consideran más relevantes. Estos aspectos han sido estudiados y definidos de manera previa a comenzar la implementación del sistema.

3.2.1 Elección del tamaño de los segmentos (chunks) y su identificador (ID).

El aspecto principal cuando hablamos de compartición de archivos, es conseguir que un peer se descargue un archivo desde varias fuentes, y para esto es necesario que este archivo esté segmentado en piezas que permitan una descarga ordenada y estructurada, de manera que con piezas de diferentes fuentes se consiga completar la totalidad del archivo.

El elegir piezas de tamaño pequeño produciría un tamaño de archivo de metadatos enorme generando mucho tráfico en la red, por el contrario piezas muy grandes reducirían la eficiencia del protocolo. Aunque en diseños como Gridcast la segmentación de los archivos de vídeo se realiza en función del tiempo, se ha decidido para este proyecto, realizar la segmentación en función del tamaño, eligiendo 512 Kbyte como tamaño de los chunks.

Por otro lado, la existencia de usuarios maliciosos que corrompen los archivos, hace necesario proveer de mecanismos que aseguren en la medida de lo posible que el vídeo que queremos ver es el que estamos descargando. Para esto implementamos una función hash criptográfica, que nos permita asignar un identificador único tanto al archivo de vídeo como a los diferentes segmentos que conforman este vídeo, de manera que permita al receptor verificar que el archivo que se ha descargado es realmente lo que quería descargarse.

Una función resumen (Hash) es una función fácilmente computable, que se aplica a un mensaje *m*, de tamaño variable, y proporciona un resumen de tamaño fijo. Este resumen es una función compleja dependiente de todos los bits del mensaje *m*; de modo que si se modifica un único bit del mensaje, su resumen debería cambiar aproximadamente, la mitad de sus bits.

Las condiciones que deben cumplir este tipo de funciones son:

- Resistencia a la pre imagen: Dado y, debe ser computacionalmente difícil obtener un x, tal que $h(x) = y$.
- Resistencia a la segunda pre imagen: Dado m_1, debe ser computacionalmente difícil encontrar m_2, con $m_2 \neq m_1$, tal que $h(m_1) = h(m_2)$.
- Resistencia a colisiones: Debe ser computacionalmente difícil encontrar dos mensajes distintos cualesquiera m_1, m_2, $m_1 \neq m_2$, de modo que $h(m_1) = h(m_2)$.

Cumpliendo estas condiciones, la función hash nos permitirá autentificar un mensaje, esto es, que el mensaje recibido desde la fuente no tiene en principio ninguna alteración.

Se han valorado como posibles funciones generadoras de los IDs de los chunks las funciones hash MD5 y SHA-1.

Las principales características de estas dos funciones son:

- SHA-1 genera una salida de 160 bits de longitud, mientras que MD5 lo genera de 128 bits. Esta diferencia de 16 bits a favor de SHA-1 lo convierte en más seguro y resistente a ataques por fuerza bruta que el algoritmo MD5.
- La dificultad de generar un mensaje que tenga un resumen dado es del orden de 2128 operaciones para MD5 y 2160 para SHA-1.
- La dificultad de generar dos mensajes aleatorios distintos y que tengan el mismo resumen (ataques basados en paradoja del cumpleaños) es del orden de 264 operaciones para MD5 y 280 para SHA-1.

Por otro lado:

- SHA-1 realiza un mayor número de pasos que MD5: 80 frente a los 64 que realiza MD5.
- SHA-1 debe procesar 160 bits de buffer en comparación con los 128 bits de MD5.

Por estos motivos la ejecución del algoritmo SHA-1 es más lenta que la de MD5 usando el mismo hardware.

Teniendo en cuenta que las aplicaciones de VoD son muy sensibles al retardo, se ha optado por usar el algoritmo MD5 para generar los identificadores necesarios.

3.2.2 Estructura de directorios

El sistema requiere que tanto el servidor como los usuarios mantengan almacenada y actualizada cierta información del sistema, por esto se requiere dotar tanto al cliente como al servidor de una base de datos (BBDD), donde almacenarán la información necesaria para el correcto funcionamiento del nodo dentro del sistema.

Para el diseño de esta BBDD en un principio se pensó usar SQL, pero rápidamente se ha desechado optando por una solución más sencilla basada en un directorio de carpetas con archivos .txt donde se almacenarán los datos necesarios para el correcto funcionamiento de la aplicación.

3.2.3 Protocolo de control “P2P Streaming Protocol”

Uno de los aspectos más importantes en la fase de diseño, fue determinar un protocolo de control para el intercambio de mensajes entre los diferentes elementos. La base para el protocolo descrito en este trabajo parte del documento “*Tracker Protocoldraft-gu-ppsp-tracker-protocol-02* “ [44], en el cual se presenta una propuesta para el protocolo “P2P Streaming Protocol (P2PSP)” que proporciona comunicación entre Trackers y Peers, este se divide a su vez en otros dos protocolos: el “P2PSP Tracker Protocol” y el “P2PSP Peer Protocol.

Este protocolo presenta un enfoque request/response, en el que los mensajes son o un request, solicitando una acción, o un response, en respuesta a un request. Los peers realizan solicitudes de vídeos al servidor (Tracker), y el servidor responde con la información que tiene almacenada en su BBDD, referente al vídeo solicitado y al estado del enjambre (Swarm). Los peers también realizarán solicitudes a otros peers, solicitando el Bitmap de un vídeo o segmentos concretos de este.

Los mensajes determinados para la comunicación de control entre peers y el Servidor (BootStrapServer) son los siguientes:

PEER	BOOTSTRAP_SERVER
CONNECT	
DISCONNECT	REPORT_INFOVIDEO
STAT_REPORT	VÍDEOLIST
QUERY_VIDEO	

Tabla 3. Mensajes entre Peer y Servidor (Tracker Protocol)

PEER
QUERY_BITMAP
REPORT_BITMAP
QUERY_CHUNK

Tabla 4. Mensajes entre Peers (Peer to Peer Protocol)

RESPUESTAS
SUCCESFUL (OK)
INVALID_SINTAX
INTERNAL_ERROR

Tabla 5. Mensajes de Respuesta

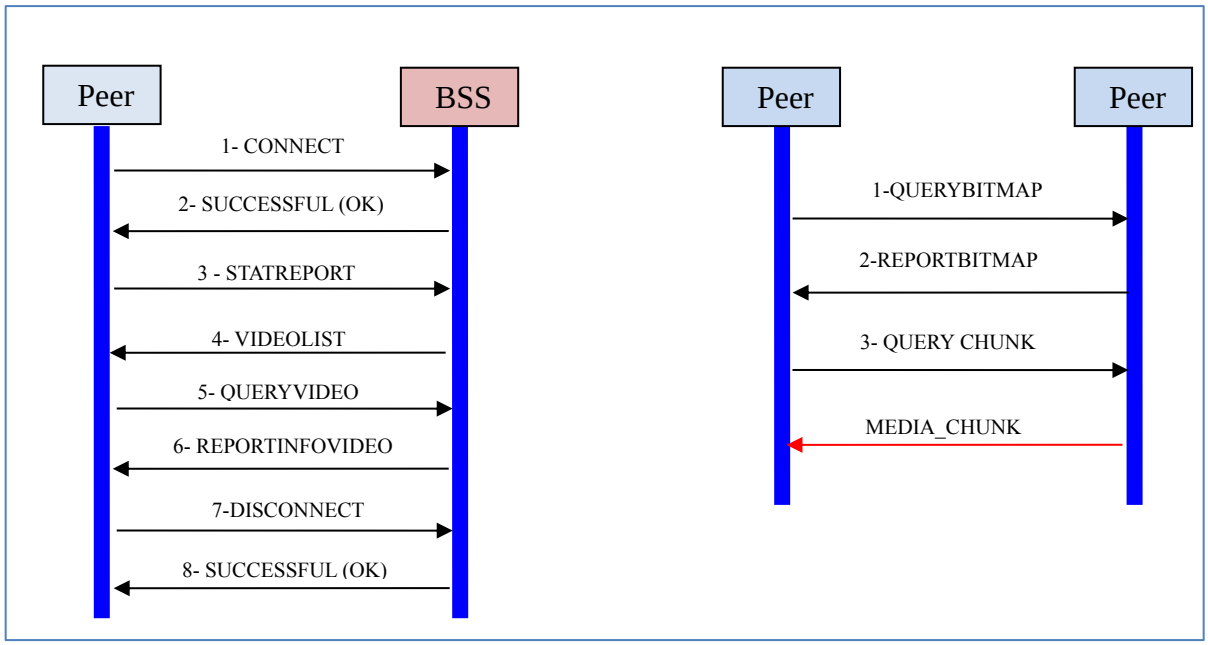


Figura 5. Secuencia de mensajes entre nodos

En la Fig.6, se muestra un ejemplo de lo que sería el intercambio de mensajes para un ejemplo concreto en el que tras la conexión inicial de los tres peers intervinientes el Peer-3, solicita un vídeo al servidor, este vídeo está disponible en el Peer-2 y Peer-1, pero estos dos peers no disponen del vídeo completo, por lo que el Peer-3 iniciará la descarga del Peer-2, cuando éste entregue el último segmento disponible de este vídeo, el Peer-3 comenzará la descarga del Peer-1, al entregar el Peer-1 su último segmento disponible, el Peer-3 iniciará la descarga del Servidor, finalizando el ejemplo con la desconexión de los tres Peers.

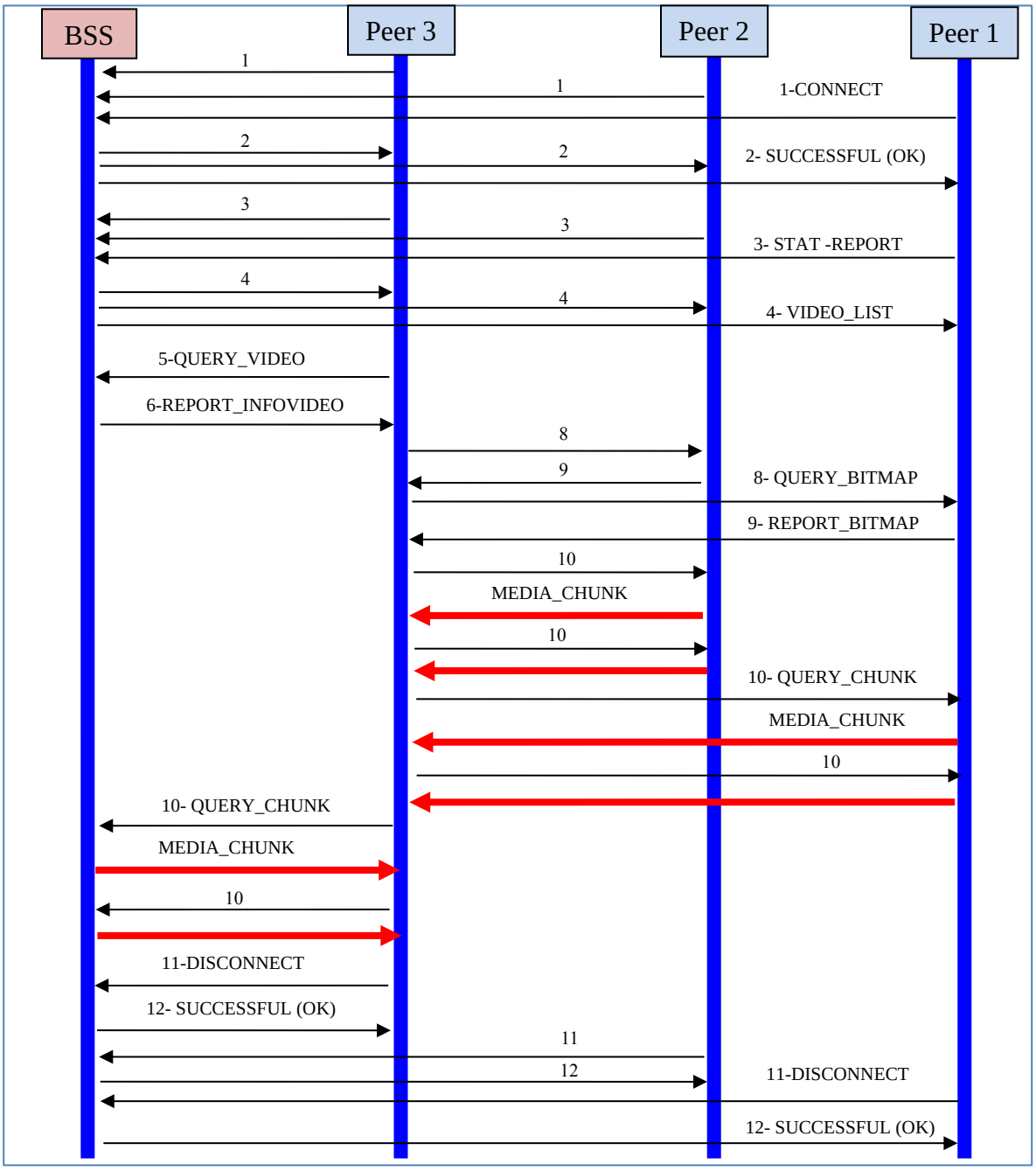


Figura 6. Ejemplo petición de un vídeo

3.2.4 Sockets

Para establecer la comunicación a través del protocolo "P2P Streaming Protocol" (P2PSP) entre los diferentes elementos del sistema (Peer-to-Peer y Peer-to-BSS), usaremos una herramienta que nos proporciona Java denominada "Socket".

Java nos ofrece dos tipos de sockets, en función del tipo de conexión que deseemos establecer: TCP o UDP:

- *Sockets Stream (TCP):* Ofrece un servicio orientado a conexión, propio del protocolo de transporte TCP donde los datos se transfieren sin encuadrarlos en registros o bloques. Si se rompe la conexión entre los procesos, éstos serán informados de tal suceso para que tomen las medidas oportunas.
- *Sockets Datagrama (UDP):* Son un servicio de transporte sin conexión. Son más eficientes que TCP, pero en su utilización no está garantizada la fiabilidad. Los datos se envían y reciben en paquetes, cuya entrega no está garantizada. Los paquetes pueden ser duplicados, perdidos o llegar en un orden diferente al que se envió.

La decisión del tipo de socket a utilizar dependerá de la aplicación cliente-servidor que se esté usando. A continuación describiremos las características más importantes de cada servicio.

En UDP hay un límite de tamaño de los datagramas, establecido en 64 Kbyte, es un protocolo desordenado, y no garantiza que los datagramas que se hayan enviado sean recibidos en el mismo orden por el socket de recepción.

Como el protocolo TCP está orientado a conexión, hay que establecer esta conexión entre los dos sockets antes de nada, lo que implica un cierto tiempo empleado en el establecimiento de la conexión, que no es necesario emplear en UDP. TCP no tiene límite en el tamaño de los datagramas; una vez que se ha establecido la conexión, el par de sockets funciona como los streams: todos los datos se leen inmediatamente, en el mismo orden en que se van recibiendo. TCP es un protocolo ordenado, garantiza que todos los paquetes que se envíen serán recibidos en el socket destino en el mismo orden en que se han enviado.

Aunque el socket UDP parece el más adecuado para el tipo de servicio que queremos proporcionar, en nuestro diseño se propone usar sockets TCP, con el fin de garantizar la fiabilidad en la transmisión sin necesidad de protocolos adicionales en capas superiores.

3.2.5 Interfaz gráfica

Tanto el cliente como el servidor deberán disponer de una interfaz gráfica. Las interfaces deberán permitir introducir datos de conexión (direcciones IP del servidor y cliente), la interfaz del cliente tiene que tener una cartelera donde el cliente seleccionará el video que desea descargar, y ambas interfaces dispondrán de un panel donde aparecerán determinados mensajes (logs) de funcionamiento.

3.2.6 Selección del Peer Fuente

Una de las líneas de trabajo con la que se pretende mejorar el proceso de intercambio de segmentos entre peers, se basa en la *selección del peer fuente* y aunque no es un objetivo de este proyecto, sería conveniente implementar un sencillo sistema, que permita una mínima selección del mejor peer-fuente.

A la hora de seleccionar el peer-fuente es importante usar criterios de selección inteligentes, que nos permitan minimizar el retardo extremo a externo (E2E delay). Estos criterios deben tener en cuenta las diferentes *capacidades de los peer fuentes* (ancho de banda de los usuarios, capacidades de sus equipos), y la *monitorización de la red* (para identificar las características de los diferentes segmentos de red y los posibles cuellos de botella).

Por esto, implementaremos un sencillo método de selección del peer-fuente, que valore de forma global las capacidades de los peers y el estado de la red, con objeto de minimizar las variaciones en el retardo que se producen en redes públicas como es el caso de Internet.

4 Implementación

La implementación del sistema se ha llevado a cabo sobre el entorno de desarrollo proporcionado por Eclipse, usando Java como lenguaje de programación.

La implementación ha comenzado por el nodo servidor sirviendo este de base para la implementación nodo usuario donde se ha reutilizando mucho código de programación para funcionalidades similares.

A continuación se muestra la secuencia seguida en la programación de la aplicación del nodo servidor, relacionando el hito alcanzado con la clase que lo implementa:

i. Procedimiento de segmentación de los vídeos, (*Segmentacion, VídeoInfo*).
ii. Creación de identificadores (*CheckSum*).
iii. Creando la estructura de carpetas y proceso de escritura/lectura de los datos (*CarpetasServidor, FileScaner*).
iv. Implementando los sockets de comunicación, (*ServerSocketMedia, AtiendeConexionMedia, ServerSocketControl, Atiendeconexion Control*).
v. Implementando los mensajes del protocolo de control (*Mensaje*).
vi. Creación de interfaz gráfica (*InterfazServidor, CreaServidor*)

Siguiendo el esquema anterior, se muestra el orden en el que se fueron alcanzando los diferentes hitos en el desarrollo del nodo usuario.

i. Se crea la clase que establece el socket con el servidor e intercambia mensajes de control y solicitud de vídeo (*SocketClienteServidor*).
ii. Implementando del protocolo de control (*MensajeP2P, MensajeP2S*).
iii. Estructura de directorios (*CarpetasCliente, FileScaner*).
iv. Implementación del Downloader Manager, para la gestión de la descargas (*DownLoaderManager, ProcesoDescarga, ConstructorVídeo, SocketSolicitudBitmap, SocketSolicitudChunk, CheckSum*).
v. Implementación del reproductor de vídeo (*InterfazReproductorVídeo*).
vi. Implementación del Uploader Manager para la gestión de las solicitudes de vídeos que lleguen de otros usuarios (*UpLoaderManager, AtiendeConexionUploaderManager*).

vii. Implementación de interfaz gráfica de usuario (*InterfazUsuairo, CreaUsuario*).

A continuación se va a explicar la implementación de ciertos puntos tratados en la parte de diseño.

4.1 Autenticidad de contenidos.

Para proporcionar cierta capacidad de autenticidad a los contenidos, y según lo expuesto en el apartado 3.2.1, se ha definido dos tipos de identificadores en la aplicación, uno será el identificador del vídeo y nos permitirá identificar un vídeo de manera única, y el otro será el identificador de cada segmento (chunk) del vídeo. Para crear estos dos tipos de identificadores usaremos la función hash MD5, implementada por la clase "CheckSum" (tanto en el servidor como en el cliente).

- Identificador MD5 del Chunk: Para generar el identificador de ese chunk, se le pasa a la clase *CheckSum*, el array de bytes que forman ese segmento (512 Kbyte).
- Identificador MD5 del Vídeo: Para generar el identificador del vídeo, se le proporciona a la clase *CheckSum*, un array de bytes formado por los identificadores MD5 de los dos primeros segmentos del vídeo.

El disponer de una copia de los vídeos completos en el servidor, nos permitirá poder proporcionar autenticidad del contenido a los usuarios. Para proporcionar esta autenticidad, el servidor obtendrá por cada segmento que forma parte de un video, una función hash, esta información que estará almacenada en el servidor dentro de los metadatos asociados a cada vídeo, es enviada al cliente cuando el servidor recibe la solicitud de un vídeo, de manera que el cliente, cuando recibe un segmento de un peer-fuente puede obtener la función hash de este chunk y compararlo con la función hash proporcionada por el servidor, comprobando si la información recibida es la que esperaba

4.2 Directorios

En cada nodo (servidor y cliente) se crea una estructura de directorios que nos permitirá gestionar los datos que cada elemento necesita manejar para su funcionamiento, esta estructura se crea a través de las clases *CarpetasServidor y CarpetasCliente.* Para la gestión (lectura-escritura-borrado) de la información contenida en estas carpetas, se crea la clase *FileScaner*.

A continuación se describen los directorios que se han creado tanto en el servidor como en el cliente, y los datos que en cada una se almacenan.

Estructura del directorio en el servidor.

- *CahedeVideos*: Esta carpeta contiene los vídeos íntegros almacenados en diferentes formatos, (.avi., .wmv., .mpg).

- *InfoVídeos:* Guardará información concerniente a los vídeos almacenados (tamaño, bitrate, MD5 de cada segmento, etc), y que será enviada a cada peer que solicite este vídeo, como información necesaria para que el peer realice la descarga.

- *PeerConectados*: Esta carpeta almacenará y mantendrá actualizada, información sobre los peers que se encuentran conectados al sistema, (dirección IP del peer, vídeos que este dispone).

- *VideoPeers*: Esta carpeta almacenará un listado, en el que relaciona el ID de un vídeo con los peers conectados en la red que disponen de este vídeo. Esta información es enviada al peer que solicita un vídeo para que realice la descarga.

El peer no tendrá necesidad de almacenar información del enjambre y únicamente deberá, almacenar los vídeos descargados por él y cierta información de esos vídeos, por lo que su estructura del directorio será más sencilla.

Estructura del directorio en el cliente.

- *BaseDatos*: Contendrá un listado donde se relacionará el ID de cada vídeo descargado con el nombre de la película a la que corresponde.

- *Descargas*: En esta carpeta se irán almacenando los segmentos descargados.

- *Vídeos*: En esta carpeta se almacenará el vídeo reconstruido, a partir de los segmentos descargados.

4.2.1 Directorio del Servidor

En la ruta C:\PROYECTOVoDP2P\SERVIDOR, se crea al inicializar el servidor la estructura de carpetas que podemos ver en la Fig. 8.

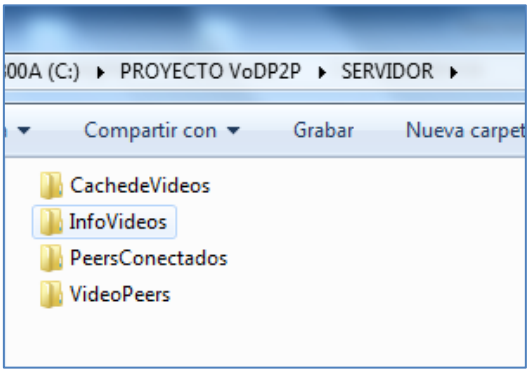


Figura 7. Carpetas creadas en el Servidor

- Carpeta "**CahedeVideos**": Contiene los vídeos íntegros almacenados en diferentes formatos, (.avi., .wmv., .mpg).

- Carpeta "**InfoVideos**": contiene archivos .txt nombrados con el ID del vídeo, en estos .txt se guardará información relativa al vídeo.

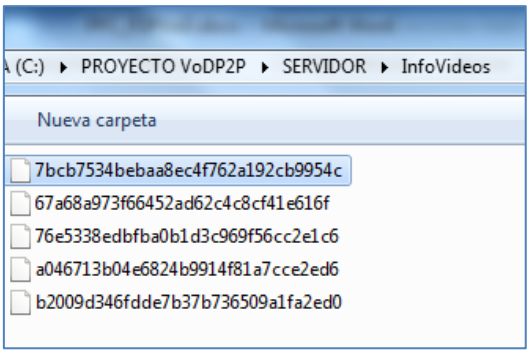


Figura 8. Contenido carpeta InfoVídeos

- Archivo .txt “7bcb7534bebaa8ec4f762a192cb9954c”. Este archivo almacenará:

  ¬ El nombre original del vídeo.

  ¬ Su tamaño en bytes.

  ¬ El bitrate en bits/seg

  ¬ Los IDs de todos sus chunks (en este caso el vídeo está dividido de 14 chunks).

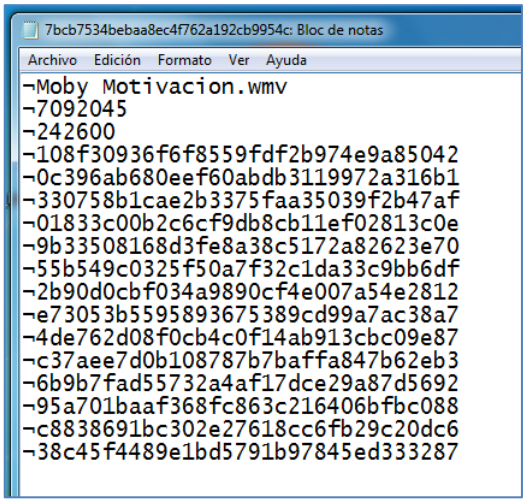


Figura 9. Contenido de los .txt de la carpeta InfoVideos

- Carpeta “**PeerConectados**”: Contiene un archivo .txt, por cada peer que actualmente se encuentra conectado al sistema, cada archivo se nombrará con la IP del peer en cuestión. Estos archivos son creados con la llegada de un mensaje “Connect” desde un peer, y eliminados al recibir un mensaje “Disconnect”.

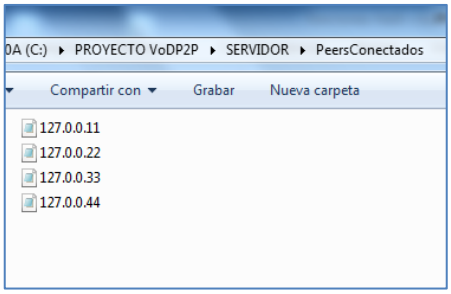


Figura 10. Archivos creados en la carpeta PeersConectados

- Archivo .txt “127.0.0.11”: Corresponde a cada peer conectado al sistema, guarda un listado con los identificadores de los vídeos que dispone el peer para compartir.

- Carpetas “**VideoPeers**”: Contiene archivos .txt nombrados con el ID de un vídeo que está disponible en la red, estos archivos guardan las direcciones IP de los usuarios que disponen de este vídeo, la primera dirección IP, siempre corresponderá al servidor.

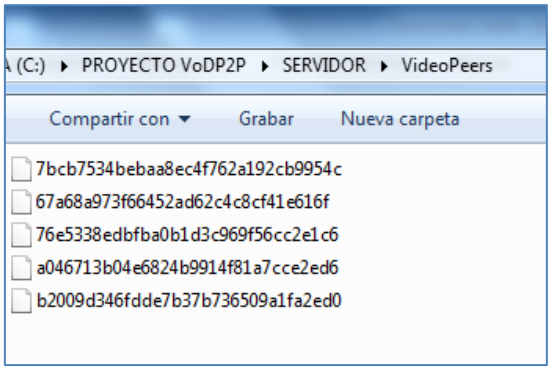


Figura 11. Archivos creados en la carpeta VideoPeers

- Archivo .txt “7bcb7534bebaa8ec4f762a192cb9954c”. Este archivo guarda las direcciones IP, de los peers que tienen disponible el archivo de vídeo asociado al identificador que da nombre al txt..

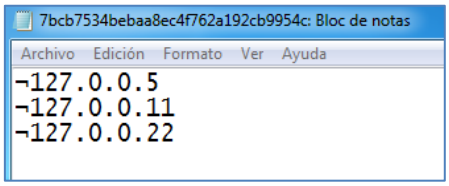


Figura 12. Contenido de los archivos .txt de la carpeta VideoPeers

Con esta estructura de directorios, el servidor dispone de la información suficiente para gestionar el enjambre y proporcionar a los peers solicitantes, la información necesaria para la descarga de un vídeo solicitado.

4.2.2 Directorio del Peer

La estructura del directorio en el cliente es más sencilla al no requerir más que un espacio de almacenamiento para los vídeos descargados y un archivo donde guardará la información de los vídeos que tiene disponibles.

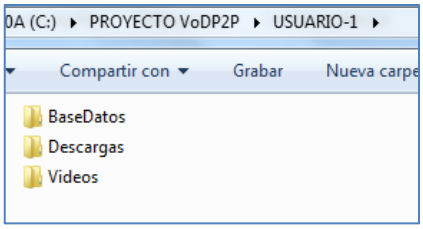


Figura 13. Estructura de carpetas en el Usuario

- Carpeta "**BaseDatos**": Contiene un archivo .txt nombrado "Lista-Id-name"

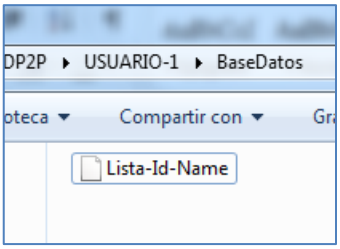


Figura 14. Archivo de la carpeta BaseDatos

- El archivo Lista-Id-name, contiene una relación de:

  ¬ID del vídeo ¬Nombre del vídeo.

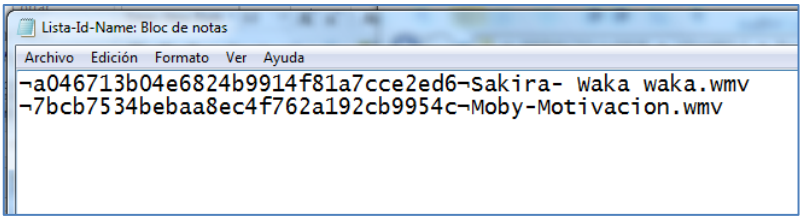


Figura 15. Contenido del txt Lista-Id-name

- Carpeta "**Descargas**": Contiene un carpeta por cada vídeo descargado, nombrada con el identificador de cada vídeo, en estas carpetas se van almacenando los segmentos que se descargan desde otras fuentes.

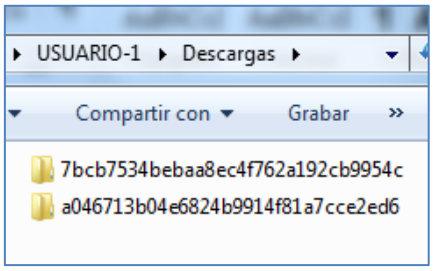


Figura 16. Carpetas contenidas en la carpeta Descargas

- Archivo .txt "7bcb7534bebaa8ec4f762a192cb9954c". En este archivo se almacenan los segmentos del vídeo descargados.

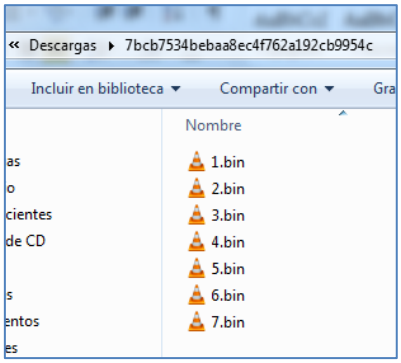


Figura 17. Segmentos de vídeo contenidos en cada carpeta

- Carpeta **"Videos"**: En esta carpeta se almacenarán los vídeos completos una vez se haya finalizado la descarga de todos los segmentos que lo componen.

4.3 Interfaces

Para la creación de las interfaces del servidor y del cliente, se ha usado el JMF de Java (JMF2.1.1e). Las interfaces son muy sencillas, buscado únicamente que cumplan con la funcionalidad necesaria para el correcto funcionamiento de la aplicación. La interfaz del servidor, mostrará dos áreas de texto una donde introducir la dirección IP pública de conexión a la red Internet y en el caso de que el equipo estuviera detrás de un firewall o de un NAT la dirección IP privada asignada a la máquina, también aparecerá un cuadro de texto, en el que irán apareciendo mensajes

referentes a la conexión del sistema.

Figura 18. Interfaz del Servidor

La interfaz del cliente, sigue el mismo criterio de funcionalidad que la del servidor, tenemos un campo de texto donde debemos introducir la dirección IP del servidor al cual nos queremos conectar y la dirección IP del equipo del nodo cliente. Dispone de una pestaña (cartelera) donde se despliega todas las películas que tiene en catálogo el servidor, tras seleccionar una de las películas y pulsar el botón “Solicitar”, comenzará el proceso de descarga.

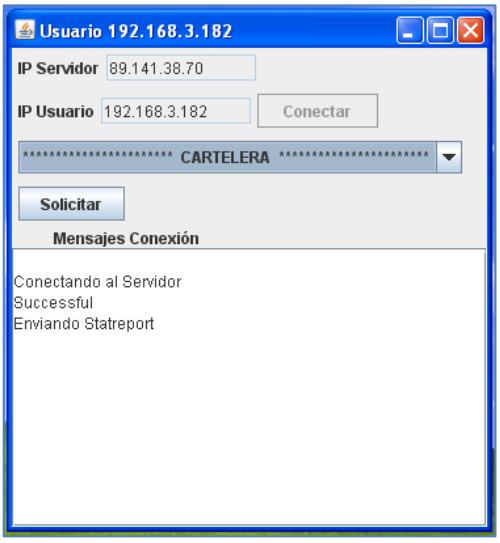


Figura 19. Interfaz del Cliente

4.4 Sockets

Para establecer una conexión a través de un socket, tenemos que programar por un lado el servidor y por otro los clientes. En el servidor, creamos un objeto de la clase ServerSocket este tiene un socket que responde en un puerto específico, el servidor espera escuchando a través del socket a que un cliente haga una petición. Si el servidor acepta la conexión crea un nuevo socket sobre un puerto diferente, para poder atender al cliente que se conectó mientras atiende en el socket original nuevas peticiones. Si la conexión es aceptada el cliente creará un socket estableciéndose la conexión a través de la cual podrán comunicarse cliente-servidor.

La implementación de estos Sockets tanto en el nodo cliente como servidor ha sido la siguiente:

- Servidor

En el servidor se han creado dos ServerSocket, uno para mensajes de "control" y otro para tráfico "multimedia", estos socket estarán permanentemente escuchando posibles conexiones por parte de clientes.

Para los mensajes de control se implementa el socket a través de dos clases, *ServerSocketControl* y *AtiendeConexionControl*, el puerto definido para este socket es el 4001. A través de este socket los clientes se conectarán y desconectarán del enjambre a través de mensajes de control (Connect, StatReport, Disconnect) y solicitarán los vídeos deseados (QueryVideo).

Para el tráfico multimedia, se implementa el socket a través de las clases *ServerSocketMedia* y *AtiendeConexionMedia*, el puerto definido para este socket es el 5001. A través de este socket se realizará el tráfico de datos multimedia.

- Cliente

En el nodo cliente solo se ha creado un ServerSocket que estará escuchando en el puerto 5001 y a través del cual se realizarán tanto las conexiones para mensajes de control (solicitudes de bitmap y chunks por parte de otros peers) como el envío de datos multimedia entre peers. Las clases implementadas han sido *UpLoaderManager* y *AtiendeConexionUploader*, este ServerSocket se mantendrá a la escucha desde el momento inicial de creación del cliente, y será a través del cual un peer-fuente envíe segmentos de vídeo al peer-cliente que se lo solicite.

También encontraremos tres sockets en el cliente para la conexión tanto al ServerSocket de un peer-fuente, como a los dos ServerSocket que dispone el Servidor. Estas conexiones se realizan a través de las clases; *SocketSolicitudBitmap* y *SocketSolicitudChunk* para la conexión a los ServerSocket de descarga Multimedia; y la clase *SocketClienteServidor* para la conexión al ServerSocket de control del servidor.

Para poder gestionar varias solicitudes simultáneas tanto por parte del servidor, como de los clientes cuando actúan como peer-fuente, se ha hecho uso de los hilos (thread). Crearemos una clase principal para cada socket (ServerSocketControl y ServerSocketMedia en el Servidor y UpLoaderManager en el Cliente), estas clases están escuchando las peticiones de conexión y cada vez que llegue una conexión se creará un hilo a través de las clases *AtiendeConexionControl, AtiendeConexionMedia y AtiendeConexionUploader,* que establece un socket con el cliente.

4.5 Protocolo P2PSP

Para realizar la codificación de los mensajes definidos en el diseño del protocolo P2PSP, tenemos dos opciones: una codificación binaria o algún tipo de codificación ASCII, tal como XML en HTTP. Ambos tienen pros y contras. ASCII es más fácil de entender para la gente, pero requiere más espacio y ancho de banda. Los protocolos binarios son más ligeros en cuanto al ancho de banda y procesamiento, aunque a veces son considerados más difíciles de entender.

Para la implementación de nuestros mensajes se ha decidido utilizar XML, basándonos en el documento *Tracker Protocoldraft-gu-ppsp-tracker-protocol-02* [44] y usando los mensajes que mejor se adaptan a nuestras necesidades, se ha simplificado al máximo la estructura de los mensajes, sin entrar en códigos de error de HTTP.

Para llevar a cabo la implementación de este protocolo se ha revisado diferentes librerías disponibles en Java que facilitan la manipulación de datos XML, en nuestro caso usaremos JDOM (Java Document Object Model). JDOM al contrario que otras librerías como DOM o SAX, se ha pensado para poder manipular documentos XML en un lenguaje concreto, JAVA, esto lo hace que sea un API más eficiente y natural para un programador de JAVA y por tanto requiere un menor coste de aprendizaje.

4.5.1 Descripción del Tracker Protocol

Para la creación del documento XML la primera línea obligatoria del documento será la "declaración del XML" donde se define la versión de XML usada y especificaciones sobre la codificación del documento (US-ASCII, UTF-8, UCS-2 etc), además de incluir una declaración del documento autónomo (standalone) para controlar los componentes del DTD, en nuestro caso esta declaración no aparecerá, al prescindir de DTD.

El elemento raíz de nuestro documento XML es "**method**" cuyo valor determinará el tipo de mensaje que es, en todos los tipos de mensajes el elemento "method" cuenta con un atributo "transationID". Este atributo contendrá un identificador de 64 bits, generado aleatoriamente y que identifica a este mensaje. Un mensaje de respuesta en este protocolo "response" deberá llevar el mismo "transationID" que el mensaje "request" al cual responde. De esta manera se identifica una respuesta con su solicitud.

Ejemplo de correspondencia entre dos mensajes request-response.

```
<?xml version="1.0" encoding="UTF-8"?>
<method transationID="-2993202499420986072">CONNECT</method>
<?xml version="1.0" encoding="UTF-8"?>
<method transationID="-2993202499420986072">SUCCESSFUL</method>
```

I. Métodos Específicos de cada mensaje

- **Entre el Peer y el BSS**

CONNECT: Primer mensaje que envía un peer al BSS para conectarse al sistema. El servidor guarda la IP del Peer en su carpeta “PeersConectados”.

```
<method transationID=”###”>CONNECT
    <IPHost>###</IPHost>
</method>
```

STAT_REPORT: Mensaje que envía el peer al BSS, indicándole que este peer dispone de los siguientes vídeos almacenados. Con esta información el servidor actualiza la carpeta “VideoPeers”:

```
<method transationID=”###”>STATREPORT
        <IPHost>###</IPHost>
        <videosUser>
                <videoID>###</videoID>
                <videoID>###</videoID>
        </videoUser>
 </method>
```

VIDEO_LIST: Mensaje que envía el BSS al peer, informándole de los vídeos que tiene disponible el servidor (Cartelera).

```
<method transationID###">VIDEOLIST
        <cartelera>
                <video>### </video>
                <video>### </video>
                <video>### </video>
        </cartelera>
</method>
```

QUERY_VIDEO: Mensaje que envía el peer al BSS, solicitándole información de un vídeo.

```
<method transationID="###">QUERYVIDEO
        <IPHost>###</IPHost>
        <videoName>###</videoName>
 </method>
```

REPORT_INFOVIDEO: Mensaje que envía el BSS al peer como respuesta a un "query_video". Le informa del identificador del vídeo, del nombre de este vídeo, de su bitrate, del número de segmentos de este vídeo así como del MD5 de cada uno de ellos, y de las direcciones IP de los peers que previsiblemente disponen de este vídeo.

```
<method transationID="###">REPORTINFOVIDEO
        <videoID>###</videoID>
        <videoName>###</videoName>
        <bitRate>###</bitRate>
         <segmentos>Nºsegmentos
                <IDsegmento1>###</IDsegmento1>
                 <IDsegmento2>###</IDsegmento2>
         ……..
         </segmentos>
         <peersdisponibles>
                <peerID>IP del peer</peerID>
         …….
         </peerdisponibles>
 </method>
```

DISCONNECT : Mensaje que envía el peer al BSS cuando quiere abandonar el sistema. El servidor borra la dirección IP del peer de "PeersConectados" y busca en aquellos archivos de la carpeta "VideoPeers" donde aparezca esta IP borrándola también.

```
<method transactionID="###">DISCONNECT
        <IPHost>###</IPHost>
 </method>
```

- **Entre dos Peer**

QUERY_BITMAP: Mensaje que envía un Peer a otro Peer solicitándole información del bitmap disponible de un vídeo determinado.

```
<method transactionID=”##”>QUERYBITMAP
        <videoID>##</videoID>
</method>
```

REPORT_BITMAP: El Peer interrogado con el mensaje anterior, responde con el identificador del vídeo requerido y el bitmap disponible de este vídeo, este bitmap es un nº que indica la cantidad de segmentos que dispone, siempre empezando a contar desde el primer segmento del vídeo.

```
<method transactionID=”##”>REPORTBITMAP
        <videoID>##</videoID>
        <bitmap>##</bitmap>
 </method>
```

QUERY_CHUNK: El Peer interesado en el vídeo, envía este mensaje a otro Peer solicitándole un segmento determinado de este vídeo.

```
<method transactionID=”##”>QUERYCHUNK
         <videoID>##</videoID>
        <chunk>##</chunk>
 </method>
```

4.6 Proceso descarga vídeos (Selección del Peer-Fuente)

El proceso de descarga conlleva las acciones que se ejecutarán en el peer-fuente, desde que llega a este la respuesta (mensaje ReportInfoVideo) a la solicitud de un vídeo, hasta que se completa la descarga del vídeo o se detienen la misma cerrando la ventana de reproducción del vídeo. Durante el proceso de descarga, se ejecuta un mecanismo de selección del peer-fuente, este mecanismo tendrá en cuenta principalmente dos datos, el primero será el bitmap disponible por cada peer-fuente y el segundo el tiempo de respuesta ante un mensaje enviado desde el peer-cliente al peer-fuente, este tiempo de respuesta se pretende que nos permita determinar las características de la red, y las capacidades del peer-fuente.

El mecanismo que utilizaremos para cuantificar de forma global tanto las características de la red como las capacidades de los peer-fuente, se basa en medir el tiempo que transcurre desde que un peer-cliente solicita el bitmap de un vídeo a un peer-fuente, hasta que se recibe en el cliente la respuesta con el bitmap del vídeo solicitado. Este valor en su concepto, es similar al Round Trip Time (RTT), y en nuestro caso como está asociado directamente a un mensaje concreto "QueryBitmap", lo denominaremos $RTT_{QueryBitmap}$. El tiempo que contabilizará, corresponderá al tiempo que tarda la petición QueryBitmap en llegar desde el peer-cliente al peer-fuente, el que lleva al peer-fuente a procesar este mensaje y generar la respuesta, el de transmisión de la respuesta desde la fuente al cliente y el procesamiento final en el cliente de esta respuesta.

$$RTT_{QueryBitmap} = t^{peticion}_{transmisión} + t^{peticion}_{procesamiento\ en\ peer\ fuente} + t^{respuesta}_{transmisión} + t^{respuesta}_{procesamiento\ en\ peer\ cliente}$$

A continuación se explicará con más detalle las acciones que se llevan a cabo durante el proceso de descarga. Este se inicia con la respuesta por parte del servidor a un mensaje "QueryVideo" del cliente. En el mensaje de respuesta del servidor "ReportInfoVideo", el servidor envía al peer-cliente una serie de metadatos necesarios para la descarga, alguno de estos datos son referentes al vídeo solicitado (tamaño del vídeo, bitrate, ID y nombre del vídeo y el MD5 de cada segmento) y otros propios de la red P2P (direcciones IP de posibles peer-fuente), al recibir el cliente el mensaje "ReportInfoVideo" llama a la clase "DownLoaderManager" la cual con la información remitida por el servidor, será la encargada de iniciar el proceso de descarga.

La clase DownLoaderManager, crea tres hilos simultáneamente:

1) *ProcesoDescarga*: Esta clase inicia el proceso de descarga, para lo cual lo primero que realiza es una consulta del bitmap disponible a todos los posibles peer-fuente que aparecen en el mensaje "ReporInfoVideo", guardando en un array bidimensional, el bitmap de cada peer asociado al retardo ($RTT_{QueryBitmap}$) que se ha generado por cada solicitud de bitmap a ese peer. Para asignar un valor de $RTT_{QueryBitmap}$ al servidory dado que al servidor no se le envía mensaje de "QueryBitmap", tomaremos el tiempo empleado en la solicitud y respuesta del mensaje "QueryVideo", que envía el cliente al servidor al solicitar un vídeo, asociando a este tiempo un valor de bitmap igual al número total de segmentos del vídeo, asumiendo que el servidor siempre dispondrá de los vídeos completos.

   Con este valor de $RTT_{QueryBitmap}$ pretendemos cuantificar el estado de la red, la proximidad de la fuente, las características de su conexión y capacidad de procesamiento del cliente. Para obtener este dato mediremos el tiempo de envío de un mensaje desde el peer-cliente al peer-fuente y su respuesta, estos mensajes tienen un tamaño reducido y similar, por lo que el tiempo obtenido podrá ser un valor global del estado de la red y la capacidad de procesamiento del peer-fuente. La secuencia seguida en la implementación es la siguiente; justo antes de enviar el mensaje "QueryBitmap" en la clase *SolicitaBitmap* se toma un valor con la variable "tiempoInicial" (System.*currentTimeMillis*());, y volveremos a tomar un segundo valor "tiempoFinal" , cuando se recibe el mensaje de respuesta "ReportBitmap", la diferencia entre estos dos tiempos será el valor de RTT que se tendrá en cuenta para la selección del peer candidato. A continuación se ordenará el array en función del $RTT_{QueryBitmap}$ obtenido (de menor a mayor) y se inicia el proceso de solicitud del vídeo (QueryChunk) empezando por los peer-fuentes con menor $RTT_{QueryBitmap}$. Los primeros 20 segmentos son descargados del peer con menor $RTT_{QueryBitmap}$ incluyendo en esta selección al servidor (BootStrapServer), en las sucesivas descargas la IP del servidor, se irá omitiendo, dejando el servidor solo para el caso que no hubiera más peer-fuente disponibles. Con esto se pretende aliviar la carga en el servidor. Por supuesto en cada selección del peer, se comprueba que éste disponga de los segmentos de interés para el peticionario.

2) *ConstructorVideo*: Esta clase será la encargada de reconstruir el vídeo a partir de los segmentos que se van almacenando en la carpeta “Descargas”, la reconstrucción del vídeo se guarda en la carpeta “Videos” y se identifica el archivo con el nombre original del vídeo.

3) *InterfazReproductorvideo*: Esta clase será la encargada de crear la interfaz de reproducción del vídeo. Para su creación se ha usado las librerías VLCJ, las cuales nos permiten a través de Java, integrar VLC Media Player dentro de una ventana AWT o JFrame. La interfaz nos permitirá realizar una serie de acciones sobre el vídeo (pausa, stop, play), en un apartado posterior se hablarán más detenidamente de VLC y VLCJ.

4.7 Librerías Importadas

Para el correcto funcionamiento de las aplicaciones es necesario importar usa serie de librerías que no contiene el JRE de Java SE-1.7.

4.7.1 Xuggle

La clase “VideoInfo” implementada en el servidor tiene como objeto obtener información multimedia del vídeo a tratar. Para su funcionamiento esta clase implementa las librerías que proporciona Xuggle, estas librerías nos permiten modificar y obtener información de cualquier fichero multimedia desde Java. En nuestro caso solo nos interesa conocer el tamaño del vídeo en bytes y el bitrate en bits/seg, pero nos podría proporcionar mucha más información, como la relativa a la duración del vídeo o codecs usados en el vídeo.

En el mensaje ReportInfoVideo, se envía información relativa al bitrate del vídeo y aunque esta información no es usada durante el proceso de descarga del vídeo, podría ser interesante junto a la duración del vídeo, a la hora de determinar el tamaño del buffer de inicio para la reproducción del vídeo.

Para el correcto funcionamiento de esta clase es necesario el añadir además de la librería de Xuggle

- xuggle-xuggler-5.4.jar

las correspondientes al sl4j-1.7.5 .

- slf4j-api-1.7.5.jar

Aunque Java proporciona en su JDK un librería java.util.logging para la generación de trazas y logging, usando la librería slf4j esta nos permite abstraernos de la librería que decidamos usar para la generación de estos logging simplemente cambiando un jar por otro. De entre las diferentes combinaciones posibles:

- slf4j+ log4j
- slf4j+ logback
- slf4j+ java.util.logging

La opción que parece ser más recomendable, ya que nos permitirá poder pasar a cualquiera de las otras de forma sencilla y sin cambio de código y por ser la sucesora de log4j (siendo los mismos creadores) es usar la combinación sl4j+logback; por lo que se importarán los jar siguientes.

- logback-classic-1.1.0.jar
- logback-core-1.1.0.jar

4.7.2 JDOM

JDOM es una librería de java que nos permite manipular de forma sencilla e intuitiva documentos XML. Este API está formado por 6 paquetes (org.jdom, org.jdom.input, org,jdom.output, org.jdom.transform, org.jom.xpath y org.jdom.adapters) Para poder empezar a usar JDOM es necesario descargarse la versión JDOM 1.1 de la página www.jdom.org , descomprimirla (en nuestro caso en C:\) e incluir las siguientes librerías en el proyecto.

- Jdom.jar
- jaxem.jar
- xalam.jar
- xml.jar
- ant.jar

Para una comprensión rápida se recomienda el documento de Oracle Magazine [45].

4.7.3 VLCJ

VLCJ[4] es un proyecto “open source”, que permite la integración de Java con el reproductor VLC, aunque en el apartado 6.3 se describe con más detalle estas librerías y el proceso de instalación, comentar que la versión de VLCJ utilizada en el proyecto ha sido la 3.4.0 y las librerías que se han requerido importar de la versión de VLCJ son:

- vlc-2.1.0.jar
- platform-3.4.0.jar
- jna-3.4.0.jar,

[4] https://github.com/caprica/vlcj

5 Validación

La validación del proyecto se ha realizado en 3 fases:

I) La primera fase ha correspondido a las pruebas realizadas durante la programación de la aplicación; en esta fase tanto el servidor como los clientes se ejecutan en el mismo pc, para lo cual se asignará a los diferentes nodos un direccionamiento local (127.0.0.xx).

II) En la segunda fase se han realizado una serie de pruebas, con el objetivo de valorar el comportamiento de la aplicación dentro de una red IP, validando el protocolo de comunicación y el proceso de descarga. Estas pruebas se ha realizado dentro de una LAN creada con el direccionamiento privado (198.168.0.xx) que nos proporciona un modem doméstico de una red ADSL y usando un pc para cada nodo.

III) En la tercera fase se pretende probar la validez de este sistema dentro de la red Internet, se observará la respuesta de la aplicación ante los retardos propios de esta red, y se realizará la prueba correspondiente al objetivo final propuesto en este proyecto, en esta fase se han usado diferentes líneas ADSL y una conexión a través de una tarjeta de datos UMTS

5.1 Configuración de equipos y puesta en servicio

La configuración que requieren los equipos es muy elemental. Los nodos clientes deben tener instalado la versión de VLC 2.0.1 o superior, se debe tener actualizado Java en servidor y clientes, e instalado la última versión del jdk (jre7 y jdk1.7.0).

Para poner en servicio el servidor, la primera vez que se ejecute, se creará una estructura de directorios en *C:\ProyectoVoDP2P\SERVIDOR*, a continuación debemos cerrar la aplicación y colocar en la carpeta *C:\ProyectoVoDP2P\SERVIDOR\CachedeVídeos*, todos los vídeos que el servidor vaya a poner a disposición de los clientes, seguidamente volvemos a ejecutar la aplicación servidor y ya estará en condiciones de recibir solicitudes y enviar vídeos.

Para poner en servicio el usuario, simplemente debemos ejecutar la aplicación del usuario y rellenar los campos correspondientes a la dirección IP del servidor.

5.2 Fase I (Servidor y usuarios en el mismo PC)

Este ha sido el entorno de pruebas usado durante el proceso de implementación de la aplicación. Se ha asignado un direccionamiento local para la creación de los ServerSocket tanto del servidor como de los usuarios.

Servidor: ServerSocketControl: 127.0.0.1:4001 y ServerSocketMedia: 127.0.0.1:5001

Usuario1: ServerSocket: 127.0.0.11:5001

Usuario2: ServerSocket: 127.0.0.22:5001

Las pruebas finales en este entorno con la aplicación finalizada, han estado orientadas a comprobar la capacidad de reproducir diferentes formatos de vídeo, validar el funcionamiento del programa simulando diferentes retardos de propagación y comprobar la robustez del sistema frente a fallos muy comunes en redes P2P, como son, pérdidas de conexión y salidas no controladas de peers cuando estos actúan como clientes o fuentes.

1. Capacidad de reproducir diferentes formatos de encapsulación y codificación de vídeos.

   Se ha seleccionado un conjunto de vídeos Tabla 6, con diferentes formatos tanto de contenedor como de codificador (audio-vídeo), con objeto de validar la elección del reproductor VLJ.

VÍDEOS	CODECS	TAMAÑO EN Kbyte	DURACIÓN	BITRATE Kbps	RESULTADO REPRODUCCIÓN
.avi					
Video_1	Vídeo: MPEG4 Audio: MP3	389.254	49min 22sg	1051	OK
Video_2	Vídeo: MPEG4 Audio: MP3	428.556	45min 39sg	1251	OK
.mkv					
Video_3	Vídeo: H264 Audio: AAC	282.712	43min 08sg	873	OK
Video_4	Vídeo: H264 Audio: AC3	1.847.431	57min 05sg	4314	OK
.mp4					
Video_5	Vídeo: H264 Audio:AAC	381.329	43min 15seg	1175	OK
Video_6	Vídeo: H264 Audio: AAC	57.578	25min 42sg	298	OK
.wmv					
Video_7	Vídeo: H264 Audio: AAC	96.620	3min 31sg	3646	OK
Video_8	Vídeo: H264 Audio: AAC	6.926	3min 53sg	237	OK
.mpg					
Video_9	Vídeo: MPEG1VÍDEO Audio: MP2	818.985	1h 40min 25sg	1087	OK
.asf					
Video_10	Vídeo: WMV2 Audio: WMAV2	784	31 sg	196	OK

Tabla 6. Vídeos reproducidos

2. Determinar el impacto del retardo en la red.

Observaremos como afecta el retardo de la red en el funcionamiento de la aplicación, e intentaremos mitigar este impacto dotando a la aplicación cliente de un buffer, donde almacenar una determinada cantidad de datos antes de comenzar la reproducción del vídeo. El número de chunks que formarán el buffer, no podrá ser ni muy pequeño, lo que produciría cortes en la reproducción del vídeo, ni muy grande, lo que supondría que el usuario estuviera esperando varios minutos a comenzar la reproducción.

Los parámetros que van a condicionar en mayor medida el tamaño de este buffer será; el bitrate del vídeo, y la característica de la red IP (velocidad de línea, retardos etc.). Es por esto, que en esta prueba se ha propuesto usar diferentes vídeos (con diferentes bitrate) e ir modificando valores de buffer a la vez que se variaba un parámetro definido como $RTT_{QueryChunk}$. Este valor $RTT_{QueryChunk}$, corresponderá al tiempo empleado desde que un cliente envía la solicitud de un chunk hasta que este cliente recibe el chunk completo. Es un concepto similar al $RTT_{QueryBitmap}$, visto en el apartado 4.6.

Para simular el valor $RTT_{QueryChunk}$, dado que estamos trabajando en la misma máquina (cliente y fuente), y el tiempo de transmisión de los paquetes es nulo, se va a introducir en la clase *SocketSolicitudChunk* un *Thread.sleep()* al inicio del método “procesarConexion”, de manera que el cliente cuando recibe el mensaje con los datos multimedia, espera un tiempo definido por nosotros para procesar este mensaje.

A continuación se muestra en la Tabla 7, el resultado de las pruebas realizadas con 3 vídeos. Podemos observar como para un tamaño de buffer dado, el éxito de la reproducción va a depender tanto del bitrate del vídeo como del retardo simulado ($RTT_{QueryChunk}$).

VÍDEOS REPRODUCIDOS	CARACTERÍSTICAS DEL VÍDEO	$RTT_{QueryChunk}$	SEGMENTOS EN BUFFER	RESULTADO DE LA REPRODUCCIÓN
Video_1	Tamaño: 389.254 Kbyte Nº Segmentos: 761 **BitRate: 1051 Kbps** Duración: 49´ 22´´ Codecs: Video: MPEG4 Audio: MP3	2000 msg	10	OK
		3000 msg	10	OK
		3500 msg	10	Se detiene en el segmento 424
		3500 msg	20	OK
		4000 msg	10	Se detiene en el segmento 184
		4000 msg	20	Se detiene en el segmento 222
Video_3	Tamaño: 282.712 Kbyte Nº Segmentos: 553 **BitRate: 873 Kbps** Duración: 43´ 08´´ Codecs: Video: H264 Audio: AAC	3000 msg	10	OK
		3500 msg	10	OK
		4000 msg	10	Se detiene en el segmento 38
		4000 msg	20	Se detiene en el segmento 48
Video_6	Tamaño: 57. 578 Kbyte Nº Segmentos: 113 **BitRate: 298 Kbps** Duración: 25´ 42´´ Codecs: Video: H264 Audio: AAC	4000 msg	10	OK
		5000 msg	10	OK
		6000 msg	10	OK
		10000 msg	10	OK

Tabla 7. Respuesta de la aplicación en función del bitrate-retardo-buffer

3. Desconexión de un peer de la red de forma no controlada.

Se proponen dos supuestos referentes a salidas de la red de usuarios de forma no controlada

a) Un peer-cliente solicita el bitmap a un peer-fuente que no se encuentra ya disponible.

b) Durante la descarga de chunks desde un peer-fuente, este abandona la red.

En los dos supuesto, se ha comprobado que la aplicación reacciona adecuadamente a cada una de las situaciones planteadas, actualizando el array de posibles peer-fuentes (remitidos por el servidor) si a la solicitud del bitmap alguno no responde, y reorganizando la descarga de chunks, si durante la descarga desde un peer-fuente este se desconecta,

5.3 Fase II (Servidor y Usuarios en la misma VLAN)

Para este entorno de trabajo se ha usado el direccionamiento privado que proporciona un router de la marca Thomson Mod: TCW710, este router nos proporciona direcciones de host para la red 192.168.0.0/24, teniendo el servidor DHCP activado y con un número de CPEs (máximo de direcciones asignar) de 245, comenzando desde la 192.168.0.10.

Se ha asignado al equipo del servidor una dirección IP fija (192.168.0.5), es conveniente que el servidor tenga una IP fija para configurar un forwarding en el router (configuración necesaria para la Fase III) y que esta dirección, este fuera del rango que el router asigna a través del DHCP. El PC usuario obtendrá una dirección IP dentro del mismo rango que el servidor, que le será asignada dinámicamente a través del DHCP.

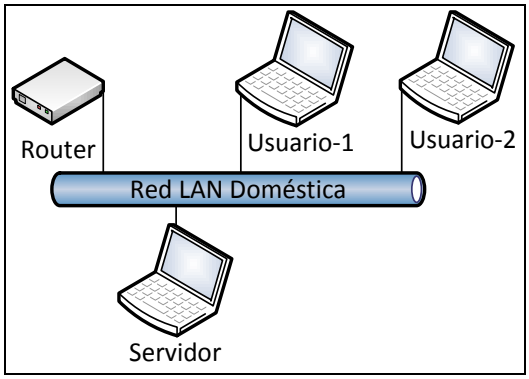


Figura 20. Esquema de conexión en entorno LAN

En este escenario se han realizado principalmente dos pruebas.

1. Ver las diferencias del valor $RTT_{QueryChunk}$ cuando usamos segmentos de 512 Kbyte y 128 Kbyte.

   El objetivo de esta prueba, es determinar cómo afecta el tamaño del segmento en el valor del $RTT_{QueryChunk}$, para valorar el usar un tipo u otro de tamaño en la segmentación. La prueba se realiza desde el PC del usuario-1, se utilizan dos vídeos y se ha configurado la aplicación para que en un caso se realice la segmentación en chunks de 512 Kbyte y en una segunda prueba en chunks de 128 Kbyte.

USUARIO	VÍDEO SOLICITADO	TAMAÑO DEL SEGMENTO	RTTQueryChunk (en msg)					
Usuario-1	video_1	512 Kbyte	1079	953	1047	1078	1094	1047
		128 Kbyte	390	610	625	391	421	562
Usuario-1	video_3	512 Kbyte	907	875	891	1344	1048	980
		128 Kbyte	406	375	375	380	412	450

Tabla 8. Valores de RTTQueryChunk para diferentes tamaños de segmento

2. Observar como afectan las características del pc del usuario y de vídeo en la aplicación.

En el resto de pruebas presentadas se va a mantener el tamaño de los segmentos en 512 Kbyte. Lo que pretendemos en esta prueba es ver la respuesta de la aplicación en función de:

- Las características del PC usado por el usuario
- Del vídeo solicitado
- Del tamaño del buffer definido en el cliente.

Los valores que observaremos serán; el del $RTT_{QueryChunk}$, el tiempo que tarda en iniciarse la reproducción, y el resultado de la descarga.

USUARIO	VÍDEO SOLICITADO	Nº SEGMENTOS EN BUFFER	RTTQueryChunk (en msg)				INICIO REPRODUCCIÓN	RESULTADO DE LA DESCARGA
usuario-1	video_1 (bitrate 1051 kbps)	10	889	945	984	1015	11 sg	Descarga incompleta hasta segmento 305
		20	875	906	875	922	23 sg	Descarga incompleta hasta segmento 473
usuario-2	video_1	10	962	854	876	860	11 sg	Descarga completa
usuario-1	video_3 (bitrate 873 kbps)	10	1063	896	986	932	10 sg	Descarga incompleta hasta segmento 196
		20	891	875	875	984	21 sg	Descarga completa
usuario-2	video_3	10	970	855	976	960	11 sg	Descarga completa

Tabla 9. Respuesta del sistema con diferentes tamaños de buffer

5.4 Fase III (Servidor y Usuario en distintas VLAN)

En esta última fase, se probará la aplicación dentro de la red pública Internet, validando el proyecto con la prueba propuesta como objetivo final del mismo. El servidor estará conectado a la red ADSL-1, los usuarios se conectarán a diferentes conexiones ADSL con distintas características y una conexión a través de una tarjeta UMTS con cobertura 3G.

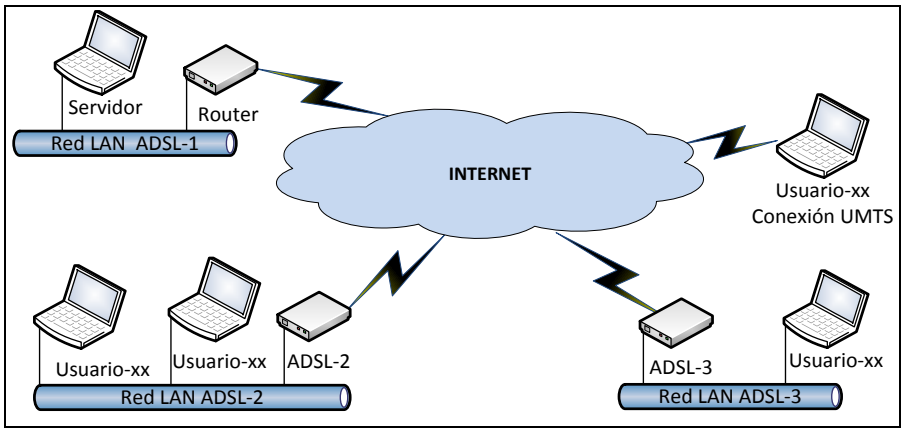


Figura 21. Esquema de conexión en entorno WAN

Para estas pruebas el servidor mantiene la dirección IP fija 192.168.0.5, y se configura el router (el cual tiene una dirección IP pública 89.141.38.70), con un forwarding, Fig. 22, de manera, que todo lo que llegue a los puertos 4001 y 5001 de router (puertos usados en la aplicación), se direcciona a la IP fija asignada a la máquina del servidor.

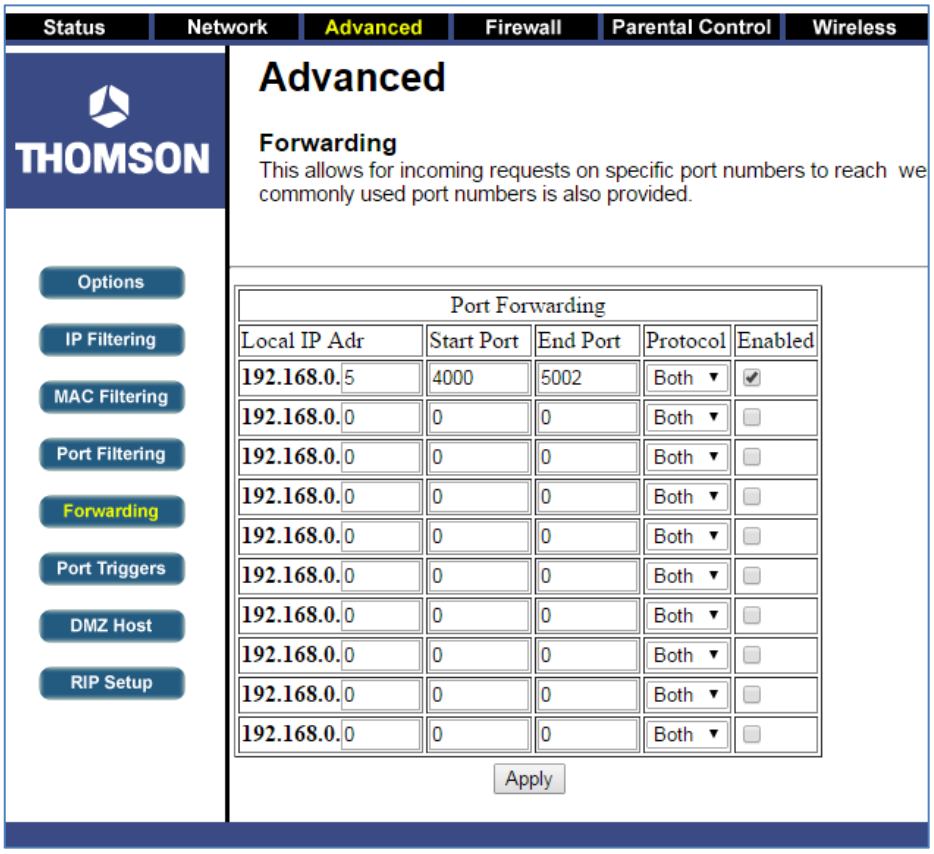


Figura 22. Configuración del router para permitir el tráfico al servidor

1. Servidor conectado en la línea ADSL-1 y cliente a tarjeta UMTS.

   i. El objetivo de esta prueba es comprobar el valor del RTT para una conexión a través de una tarjeta UMTS con cobertura 3G.

			RTT (msg)		
USUARIO	**CONEXIÓN**	**VÍDEO SOLICITADO**	**QueryVideo**	**QueryChunk-1**	**QueryChunk-21**
Usuario-1	UMTS	video_1	1890	4578	4703
Usuario-1	UMTS	video_3	1672	4468	4500

Tabla 10. Valores de RTT con una conexión a red a través de UMTS

2. Servidor conectado a línea ADSL-1 y clientes conectados a diferentes líneas ADSL.

   i. En esta prueba observaremos cómo afecta a los valores de RTT (QueryVideo y QueryChunk), las características del pc (usuario), la conexión a red utilizada y las características del vídeo solicitado. Para esto realizamos distintas solicitudes de vídeos desde dos usuarios, intercalando entre ellos sus conexiones a la red ADSL. Durante esta prueba nunca estarán los dos usuarios conectados simultáneamente. La conexión a las ADSL se realiza es mediante cable UTP.

			RTT (msg)		
USUARIO	**CONEXIÓN**	**VÍDEO SOLICITADO**	**QueryVideo**	**QueryChunk-1**	**QueryChunk-21**
Usuario-1	ADSL-2	video_1	1297	4500	4516
Usuario-3	ADSL-3	video_1	1800	5612	5621
Usuario-1	ADSL-2	video_3	1000	4500	4520
Usuario-3	ADSL-3	video_3	1309	5588	5587
Usuario-1	ADSL-3	video_1	1579	5531	8829
Usuario-3	ADSL-2	video_1	1201	4453	4456

Tabla 11. Valores de RTT para diferentes características del sistema

   ii. En esta prueba queremos comprobar cómo se ve afectado el valor del RTT, cuando hay varios usuarios descargándose segmentos de diferentes vídeos del servidor, de manera simultánea. Para esto, simultáneamente a la descarga por parte del usuario-1 del video_3, el usuario-3 solicita el video_1.

			RTT (msg)		
USUARIO	**CONEXIÓN**	**VÍDEO SOLICITADO**	**QueryVideo**	**QueryChunk-1**	**QueryChunk-21**
Usuario-3	ADSL-2	video_1	1420	7763	8769

Tabla 12. Valores de RTT compartiendo recursos del servidor

   iii. Para finalizar se realizará la prueba propuesta como objetivo final del proyecto. Durante la realización de la prueba se documentarán los diferentes valores de RTT obtenidos (solo se mostrarán los 3 primeros QueryChunk). En esta prueba el Usuario-1 realizará la descarga de parte de un vídeo del servidor (hasta el

segmento 31), a continuación el Usuario-2 solicitará ese mismo vídeo, descargándose del Usuario-1 hasta el segmento 31 y el resto del vídeo del servidor, por último el Usuario-3 solicita al servidor el mismo vídeo, descargándose del Usuario-1 hasta el segmento 31 y el resto del vídeo del Usuario-2. Para realizar esta prueba el servidor estará conectado a la línea ADSL-1 y los tres usuarios a la ADSL-2.

		RTT (msg)				
USUARIO	**VÍDEO SOLICITADO**	**QueryVideo (al servidor)**	**QueryBitmap**	**QueryChunk-1**	**QueryChunk-21**	**QueryChunk-32**
Usuario-1	video_1	1812		7015 (al servidor)	4656 (al servidor)	
Usuario-2	video_1	1328	266 (al usuario-1)	2953 (al usuario-1)	1344 (al usuario-1)	4581 (al servidor)
Usuario-3	video_1	1062	250 (al usuario-1) 262 (al usuario-2)	1329 (al usuario-1)	844 (al usuario-1)	960 (al usuario-2)

Tabla 13. Resultado de la prueba objeto del proyecto

5.5 Resultados

Las pruebas realizadas en la fase I, han demostrado la validez de VLC como reproductor de vídeo, al reproducir la totalidad de los vídeos probados. Por otro lado se ha comprobado que con bitrates superiores a 1000 Kbps y valores de $RTT_{QueryChunk}$ por encima de 3500 milisegundos el sistema no completa la descarga del vídeo si queremos mantener un tamaño de buffer no mayor de 20 segmentos, por lo que tendremos que limitar la codificación de los archivos a transferir a un bitrate por debajo de los 400 Kbps y trabajar en redes donde el valor del $RTT_{QueryChunk}$ no exceda por encima de los 7 segundos.

En la fase II observamos que trabajando dentro de la misma VLAN los valores de $RTT_{QueryChunk}$ están en torno a los 1000 msg para tamaños de chunks de 512 Kbyte y de 400 msg si usamos segmentos de 128 Kbyte. Se comprueba que la reducción de dos tercios en el tamaño del segmento, no disminuye de manera proporcional el valor del $RTT_{QueryChunk}$, por lo que se determina establecer en 512 Kbyte el tamaño de la segmentación del vídeo. En la segunda prueba, se observa como las

características del pc influyen en el éxito de la aplicación, vemos como el usuario-1 usando un equipo con características inferiores al usuario-2, no consigue reproducir la totalidad del vídeo solicitado, al quedarse sin datos en memoria para reproducir, mientras que el usuario-2 consigue completar la descarga del vídeo incluso con un buffer de menor tamaño, lo que pone de manifiesto la importancia de contar con equipos con una capacidad de procesamiento adecuado que minimice los tiempos de procesamiento para agilizar la solicitud de chunks a la fuente.

En la fase III, comprobamos que con las características de las líneas ADSL, los tiempos para el $RTT_{QueryChunk}$ son similares a la conexión UMTS (entre 4000 y 6000 msg de media), y que cuando dos usuarios solicitan simultáneamente dos vídeos distintos al servidor los valores para el $RTT_{QueryChunk}$, aumentan encontrándose entre 7500 y 9000 msg. Se observa como el tiempo en la solicitud del mensaje "QueryVideo" al servidor, está determinado por el tamaño del vídeo, ya que a mayor tamaño del vídeo, el mensaje de respuesta "ReportInfoVideo" contiene mayor información al tener un mayor número de segmentos.

Por último se realiza la prueba objeto final del proyecto, siendo satisfactoria y validando la aplicación según los requisitos marcados. Al realizar esta prueba, podemos comprobar que los valores del "QueryBitmap" están en torno a los 250 msg y los valores de $RTT_{QueryChunk}$ para peticiones entre usuarios están entre los l000 y los 2500 msg, valores por debajo de los obtenidos para un $RTT_{QueryChunk}$ cuando este segmento se solicita al servidor y que está en torno a los 5000 msg. Esto es debido a que en este entorno de prueba donde los usuarios están en la misma vlan, el tiempo de propagación de los datos en la red disminuye considerablemente.

6 Herramientas de Trabajo

6.1 Entorno de Desarrollo: Eclipse

Eclipse es un entorno de desarrollo integrado (IDE, Integrated Development Environemnt), que facilita enormemente las tareas de edición, compilación y ejecución de programas durante sus fases de desarrollo. Eclipse permite incorporar librerías con componentes ya desarrollados, los cuales se añaden al proyecto de manera muy sencilla. Aunque soporta varios lenguajes de programación, es con java con el que mejor se integra y el elegido para el desarrollo de este proyecto.

Eclipse fue desarrollado inicialmente por IBM, pero en 2003 paso a ser la fundación Eclipse (una organización independiente y sin ánimo) su desarrollador. La versión de Eclipse con la que se ha trabajado es “Indigo” (SDK 3.7.2 win32).

6.2 Lenguaje de Programación: Java

La elección de Java como lenguaje de programación ha sido principalmente por su característica de lenguaje multiplataforma. Java es un lenguaje de programación orientado a objetos, que fue diseñado inicialmente en 1995 por Sun Microsystems y en la actualidad pertenece a Oracle. Hoy en día podemos encontrar en todas partes aplicaciones y sitios web que necesiten Java.

La plataforma Java consta de varios elementos:

- El lenguaje de programación Java propiamente dicho.
- Un conjunto de bibliotecas estándar que debe existir en cualquier plataforma (Java API).
- Un conjunto de herramientas para el desarrollo de programas:
    - Compilador a bytecode (javac).
    - Generador de documentación (javadoc).
    - Ejecución de programa (intérprete de bytecode (java).
- Un entorno de ejecución cuyo principal elemento es la máquina virtual para ejecutar el bytecode.

En la página web de java[5], podremos descargarnos tanto el JRE (Java Runtime Environment) que es un conjunto de utilidades de Java (clases y bibliotecas) como la JVM (Java Virtual Machine) que será donde correrán los programas hechos en Java.

Para desarrollar programas en Java, es necesario descargarse el JDK (Java Development Kit) que es el kit para desarrolladores y que entre otras cosas contiene el JRE y la JVM. En el desarrollo de nuestro proyecto se ha usado el JDK 1.7.0

6.3 XML

XML (eXtensible Markup Language), es un lenguaje muy adecuado para describir datos, además es libre y extensible (si se cambia la estructura del documento XML para que contenga más información, las aplicaciones que usen este documento no deben verse afectadas) y cualquier persona puede crear un nuevo tipo de documento para definir un tipo de datos con sus propias reglas y tags.

XML aunque parece similar a HTML, no es lo mismo. No es ni un HTML mejorado ni ampliado. Es un meta-lenguaje que nos permite definir lenguajes de marcado adecuados para usos determinados. Sus fundamentos son muy sencillos, los documentos deben estar bien formados y ser válidos.

Para estar bien formados deben cumplir una serie de reglas; todos deben empezar por un encabezado estándar, tener un solo elemento raíz, todos los tags deben cerrarse y estar correctamente alineados, los atributos deben estar entre comillas etc. Para que el documento sea válido la estructura del documento debe estar en base a una Definición de Tipo de Documento (DTD), definir un DTD es crear nuestro propio lenguaje de marcado, en este documento definimos como queremos que sean nuestros mensajes, definiendo los elementos, atributos y entidades permitidas en el archivo XML. Cuando un documento XML se ajusta a un DTD decimos que es válido, este concepto no tiene nada que ver con el de bien-formado, ya que un documento XML puede estar bien-formado (por cumplir con la estructura y sintaxis definida en XML), pero no ser válido si no cumple con su DTD asociada. Puede ser el caso que un documento XML no tenga DTD asociada, en este caso el documento no será ni válido ni inválido. Simplemente, estará bien o mal formado. Estos DTD pueden ser un

[5] https://www.java.com/

fichero externo compartido por varios documentos o estar contenido en el propio documento XML, para validar el documento con el DTD se utiliza un parser. El uso de DTD está cayendo en desuso, y no se aplicará en la definición de nuestros mensajes.

6.4 VLC y VLCJ

El reproductor elegido para este proyecto ha sido VLC, las principales razones que han llevado a elegir este reproductor frente a otros han sido; ser un sistema de código abierto, multiplataforma, con versiones disponibles para muchos S.O. y capaz de reproducir casi cualquier formato de vídeo sin tener instalado apenas codecs, teniendo además la capacidad de streaming.

Para poder usar VLC desde Java, se ha usado las librerías correspondientes a VLCJ. El proyecto VLCJ es un proyecto Open Source, que nos permite la integración de Java con una instancia nativa del VLC Media Player para ser incrustado en una ventana de Java AWT o JFRAME.

Para usar correctamente VLCJ con Eclipse, es necesario tener correctamente creado el entorno de trabajo. El software y la documentación necesaria puede descargase en la página de Caprica[6].

La versión utilizada de VLCJ en este proyecto es VLCJ 2.1.0 (la última en el momento de realizar esta parte de la programación), esta versión es compatible con JDK 1.6 aunque hemos usado y se recomienda el JDK 1.7. Esta versión de VLCJ usa las libvlc 2.0, por lo cual deberemos tener instalado versiones de VLC a partir de 2.0, también será necesario la versión de JNA 3.4.0, para el uso de las librerías nativas de VLC (libvlc.dll" y "libvlccore.dll) que se encontraran en el directorio del VLC (C:\Program Files\VideoLAN\VLC\).

A la hora de configurar nuestro entorno, hay que incluir en nuestra librería del proyecto, los siguientes archivos vlc-2.1.0.jar, platform-3.4.0.jar y el jna-3.4.0.jar.

Si las librerías no son cargadas correctamente nos aparecerá un mensaje de error, indicándonos diferentes formas de cargar estas librerías:

1. Include NativeLibrary.addSearchPath("libvlc", "<libvlc-path>"); at the start of your application code.

[6] http://www.capricasoftware.co.uk/vlcj/downloads.php.

2. Include System.setProperty("jna.library.path", "<libvlc-path>"); at the start of your application code.
3. Specify -Djna.library.path=<libvlc-path> on the command-line when starting your application.
4. Add <libvlc-path> to the system search path (and reboot).

En este desarrollo se ha usado la opción 4. Añadiendo en el path < C:\Program Files\VideoLAN\VLC\>

Si nos siguiera dando problemas debemos revisar la arquitectura de nuestra máquina, si como en este caso el S.O. es de 32 bits debemos comprobar que tanto el JVM como el VLC sean de 32 bits.

6.5 Creación de jar

Eclipse ofrece diferentes opciones a la hora de crear archivos ejecutables (.jar). A continuación se explica el procedimiento y las opciones que se presentan.

Para crear el archivo jar desde Eclipse, seleccionamos el proyecto, botón derecho y la opción export. Se nos presentarán 3 opciones:

1. JAR file: Esta opción permite crear un JAR en forma de librería, lo cual permite que se pueda importar cualquier clase dentro del mismo y hacer uso de sus métodos y utilidades.

2. Javadoc: Esta opción es un auxiliar de la anterior, ya que permitirá crear un JAR que brindará la documentación necesaria de las clases y métodos del JAR anterior.

3. Runnable JAR file: Esta opción nos permite crear un JAR que se puede ejecutar con el comando "java -jar <archivo.jar>". Si deseamos poder ejecutar el programa esta es la opción a utilizar. A continuación podremos seleccionar entre 3 posibilidades:

    I) *Extract required libraries into generated JAR*. Opción recomendable si nuestra aplicación incluye otras librerías Java (otros JAR). Nuestro caso
    II) *Package required libraries into generated JAR.*
    III) *Copy requiredlibraries into a sub-folder next to a generated JAR*

7 Conclusiones

El objetivo principal planteado en este proyecto, proponía desarrollar una arquitectura básica para distribución de VoD P2P, esta arquitectura debía:

- Disponer de un protocolo de comunicación Peer to Tracker y Peer to Peer.
- Proveer a la red P2P de capacidad en los clientes para almacenar en su memoria los segmentos de vídeos descargados, de manera que posteriormente, estos segmentos descargados en la memoria de cada cliente, se compartieran con otros peers.

Tras las pruebas realizadas en el capítulo 5, se puede afirmar que la aplicación cumple con los objetivos planteados en el proyecto. Se ha diseñado un protocolo de comunicación a través de mensajes XML, y se ha establecido un sistema de almacenamiento y compartición de archivos que permite a un cliente poner a disposición de otros usuarios los segmentos de vídeos que previamente este ha solicitado a la red P2P.

Sin estar contemplado en los objetivos del proyecto, se ha implementado un procedimiento que permite verificar el contenido descargado, identificando individualmente cada segmento de cada vídeo, a través de la función hash (MD5).

También se ha pretendido establecer un procedimiento básico para la selección del mejor peer, mediante la cuantificación del Round Trip Time (RTT) que se produce en el envío de un mensaje (QueryBitmap) desde el cliente a cada uno de los posibles peer-fuente. Con este procedimiento para realizar la selección del mejor peer, se pretende mostrar la importancia que tiene el realizar una selección adecuada del peer-fuente, selección que debe valorar aspectos como, las capacidades de los peer-fuente y las condiciones de red.

Se ha probado el correcto funcionamiento dentro de la red Internet, siempre que los clientes se encuentren dentro de la misma vlan, y que las condiciones tanto de la red como de los vídeos distribuidos (RTT y bitrate) se mantengan por debajo de los valores determinados en el proceso de validación. También se ha verificado la robustez del sistema, recuperándose con cierta rapidez ante fallos de la red o desconexiones de peers-fuentes durante la descarga de un vídeo, fallos muy comunes en los sistemas P2P.

En resumen, la aplicación teniendo una arquitectura básica, se presenta como una herramienta válida sobre la que poder implementar desarrollos y realizar pruebas dentro del campo del Video bajo Demanda Peer to Peer.

8 Trabajos Futuros

Como se ha comentado a lo largo del proyecto, los servicios de streaming y en especial el VoD P2P son un campo de trabajo donde todavía queda mucho por hacer. Por este motivo son muy numerosas las líneas de trabajo que podemos plantear.

Se van a plantear 5 líneas de trabajo a seguir para mejorar la aplicación desarrollada.

I) Modificar el proceso de descarga.

   La solicitud de chunks por parte de un cliente se realiza a una fuente de forma secuencial. Para maximizar el uso del ancho de banda disponible en los clientes para la descarga, que como norma general será mayor que el de subida de los peers-fuente, sería conveniente realizar solicitudes de descarga simultáneas a diferentes peers-fuentes, requiriéndoles diferentes partes del vídeo.

II) Modificar la segmentación del vídeo.

   En nuestro diseño se ha realizado la segmentación del vídeo en bloques de tamaño fijo (512 Kbytes). Facilitaría la interacción con el contenido el realizar la segmentación por duración del vídeo que se reproduce en ese chunk (1 seg), esto facilitaría al sistema el modificar el proceso de descarga reiniciando las solicitudes de segmentos desde el bloque adecuado, cuando el usuario selecciona un instante del vídeo que todavía no se ha producido la descarga.

III) Implementar un protocolo UDP para el tráfico multimedia.

   Dado que TCP es un protocolo orientado a conexión se deben usar protocolos más adecuados para aplicaciones en tiempo real como es el caso de UDP y puesto que UDP no ofrece control sobre el orden en que se reciben los paquetes ni asegura la entrega, se deben implementar protocolos en la capa superior (capa de aplicación), como Real Time Protocol (RTP) y Real Time control Protocol (RTCP) los cuales nos proporcionarán capacidad para detectar pérdidas de paquetes y control en la transmisión.

IV) Selección del mejor peer-fuente y monitorización de la red.

El número de peers que disponen el vídeo solicitado por el cliente puede ser muy elevado, y el cliente no necesita conocer todos los nodos que disponen de este vídeo, sino tener una lista de los más próximos a él y que le pueden servir el vídeo en menor tiempo. El servidor debe realizar una selección de los peers, previa a la que posteriormente realizará el cliente, de manera que solo le envíe a este, una lista con los más idóneos para este cliente.

Otro aspecto a mejorar en nuestro sistema es el control del estado de la red, solo se realiza una medición del estado de la misma cuando el peer-cliente solicita el bitmap a los posibles fuentes, durante la descarga las condiciones de la red pueden variar y sería conveniente detectar estas variaciones para poder modificar el proceso de descarga, por lo que se cree muy interesante mantener una monitorización constante de la red.

V) Códec eficiente y bitrate adaptativo (DASH).

Otra línea de investigación se basa en obtener codificadores que reduzcan considerablemente la tasa de datos necesaria para codificar vídeo con una calidad óptima. H264 es a día de hoy el códec más usado en la transmisión de contenidos en streaming, pero cada vez suenan con más fuerza codecs como H.265 desarrollado por HEVC (High Efficiency Video Coding) y que se proclama como el sucesor del H.264 y VP9 desarrollado por Google y que ya ha integrado en su navegador Chrome y en You Tube alegando que reduce un 35% el ancho de banda de media por vídeo y que estos comienzan entre un 15-80% más rápido.

Pero casi más importante que el codec, es el bitrate que usemos, y en este sentido es muy interesante el poder usar streaming con bitrate adaptativo, codificando el vídeo a múltiples bitrate y ofreciendo al usuario el más adecuado, en función del ancho de banda disponible y las características de su dispositivo. En esta línea tenemos DASH (Dynamic Adaptative Streaming over HTPP), el cual usa la arquitectura de servidores web HTTP para distribuir vídeo en streaming codificado a diferentes valores de bitrate.

9 Presupuesto

A continuación se va a presentar los costes (materiales y humanos) asociados al proyecto. Durante la elaboración del proyecto no se ha tenido dedicación exclusiva al mismo, por lo que no se puede computar el tiempo completo transcurrido desde que se asignó por el tutor hasta su finalización.

Para el cálculo de presupuesto, tomaremos los datos que aparecen en el excell, *Formulario_PresupuestoPFC-TFG*, documento disponible en el Campus Global de la Universidad Carlos III, dentro de la documentación que la universidad pone a disposición de los alumnos para la realización de PFC.

Este formulario, toma como referencia la jornada de 1 hombre mes en 131,25 horas, si dividimos los meses en 4 semanas tendremos una jornada semanal de 32,81 horas, el coste del hombre mes en 4.289,54 para un ingeniero senior y en 2.694,39 para un ingeniero.

Para incluir en el presupuesto el material adquirido se usará una fórmula para el cálculo de la amortización del material.

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado

B = periodo de depreciación (60 meses)

C = coste del equipo (sin IVA)

D = % del uso que se dedica al proyecto (habitualmente 100%)

Se podrán incluir en el proyecto otros gastos imputables al proyecto, como fungibles, viajes, contratación de líneas ADSL etc.

9.1 Costes de personal

Para determinar con mayor exactitud el tiempo dedicado al proyecto, se dividirá este en siete fases, indicando el tiempo dedicado en cada fase.

I) Investigación y documentación

Las primeras acciones que se realizaron fueron el documentarse sobre la tecnología de VoD P2P actual, sus problemas y líneas de investigación actuales.

Duración: *2 semanas*.

II) Determinación de requisitos

En esta fase, se estudia los objetivos a lograr con el proyecto y se determinan las tareas que se deberán realizar para conseguirlos.

Duración: *1 semana*.

III) Diseño del sistema

Se realiza una propuesta de diseño, que permita completar los objetivos marcados, la fase de diseño también abarcará el estudio e investigación de herramientas de programación (VLCJ, XML, etc) necesarias para posteriormente realizar el desarrollo. Este diseño sufrirá modificaciones durante las fases de desarrollo.

Duración: *1 semanas*.

IV) Desarrollo del software

Programación de la aplicación, tanto el nodo cliente como el nodo servidor.

Duración: *12 semanas*.

V) Validación de la aplicación

En esta fase se realizan una serie de pruebas propuestas, con objeto de valorar el funcionamiento de la aplicación, y validar el objetivo final propuesto en el proyecto.

Duración: *1 semana*.

VI) Obtención de resultados y análisis

Análisis de los resultados obtenidos en la fase de validación y obtención de conclusiones.

Duración: *1 semana*.

VII) Desarrollo de la memoria

Redacción de la memoria, donde se recogen todas las fases seguidas durante la elaboración del proyecto y se exponen las conclusiones finales del mismo.

Duración: *8 semanas*.

Total duración 26 semanas.

El tiempo dedicado por el tutor, en tutorías presenciales, orientación y revisión del proyecto ha sido de 1 semana.

Categoría	Dedicación (hombres mes)	Coste hombre mes	Coste (Euro)
Ingeniero Senior	0,25	4.289,54	1.072,39
Ingeniero	6,5	2.694,39	17.513,54
		Total	18.585,92

Tabla 14. Coste en personal

9.2 Costes de equipamiento

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable
Ordenador portátil Toshiba Tecra A11	980,00	100	9	60	147,00
				Total	147,00

Tabla 15. Coste en equipos

9.3 Otros costes directos del proyecto

Línea ADSL contratada durante la realización del proyecto (40 € × 9 meses) y una tarjeta UMTS (30 € × 2 meses).

Descripción	Empresa	Costes imputables
Conexión línea ADSL	ONO	360,00
Tarjeta UMTS	Movistar	60,00
	Total	420,00

Tabla 16. Otros costes

9.4 Resumen de costes

Presupuesto Costes Totales	Presupuesto Costes Totales
Personal	18.586
Amortización	147
Subcontratación de tareas	0
Costes de funcionamiento	420
Costes Indirectos	0
Total	19.153

Tabla 17. Resumen de costes

El presupuesto total de este proyecto asciende a la cantidad total de **19.153 €**

Glosario

P2P	Per to Per
VoD	Video on Demand
RTT	Round Trip time
QoS	Quality of Service
QoE	Quality of Experience
CDN	Content Distribution Networks

Referencias

[1] X. Li, S. Pauly, and M. Ammar. *Video multicast over the Internet*. IEEE Network, (1999). Disponible [internet]:< http://inst.eecs.berkeley.edu/~ee290t/sp04/lectures/rdr_paper36.pdf>

[2] J. C. Pasquale, G. C. Polyzos, and G. Xylomenos. *The multimedia multicasting problem*. ACM Multimedia Systems, 6:43-59, (1998). Disponible [internet]: <http://cseweb.ucsd.edu/~pasquale/Papers/ms98.pdf>

[3] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron, and A. Sing. Splitstream: *High-b and width multicast in a cooperative environment*. In ACM olton, New York, USA, (2003). Disponible [internet]: <http://www.mpi-sws.org/~animesh/papers/splitstream-sosp.pdf>

[4] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. *Promise: Peer-to-peer media streaming using Collectcast*. In Multimedia. ACM Press, (2003). Disponible [internet]: < http://friends.cs.purdue.edu/pubs/ACMMM03.pdf>

[5] Y. hua Chu, A. Ganjam, T.E. Ng, S.G. Rao, K. SrIPanidkulchai, J. Zhan, and H. Zhang. *Early experience with and Internet broadcast system based on overlay multicast*. In USENIX Annual Technical Conference. USENIX, (2004). Disponible [internet]: < http://repository.cmu.edu/cgi/viewcontent.cgi?article=3221&context=compsci>

[6] King-Man Ho, Wing-Fai Poon, Kwok-Tung Lo. *Video-on-Demand Systems with cooperative clients in multicast environment*. IEEE Transactions on circuits and systems for video technology (2009). ISSN: 1051-8215 Vol.19 (3): 381-373

[7] Xiaofei Liao, Hai Jin, Linchen Yu. *A novel data replication mechanism in VoD P2P system*. Future Generation Computer Systems. Vol.28 (6): 930-939 (Junio 2012).

[8] Susu Xie, Bo Li, Gabriel Y. Keung, Xinyan Zhang. *Coolstreaming: Design, Theory, and Practice*. IEEE Transactions on multimedia. ISSN: 1520-9210 Vol.9 (8): 1661-1671. (Abril 2007).

[9] X.Zhan J. Liu, B. Li, and T.-S. P. Yum. *CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming*. In IEEE Infocom (2005). ISSN :0743-166 Vol.3: 2102-2111.

[10] ARTICULO M. Barreiro, V.M. Gulías, J. Mosquera, J. J. Sánchez. *Utilización de programación funcional distribuida y clusters Linux en el desarrollo de servidores de vídeo bajo demanda*. Departamento de Computación Universidad de La Coruña.

[11] PFC Javier Aguirre Arnal. *Implementación de la política DynaPeerUnicast para sistemas VoD bajo el paradigma P2P*. Universitat de Lleida. 2008. Disponible [internet]: <http://repositori.udl.cat/handle/10459.1/45735>

[12] TESIS Fernando Cores Prado. *Arquitecturas distribuidas para sistemas de Vídeo-bajo-Demanda a gran escala*. Departamento de Informática, Universidad Autónoma de Barcelona, 2003. Disponible en [internet]: < http://ddd.uab.cat/record/38440/>

[13] ARTICULO Juan Pedro Muñoz Gea y José María Malgosa Sanahuja. *Desarrollo de servicios y sistemas de comunicación sobre redes overlay peer-to-peer.* Departamento de Tecnologías de la Información y las Comunicaciones, Universidad Politécnica de Cartagena. 2010.Disponible en [internet]:< http://repositorio.bib.upct.es:8080/jspui/handle/10317/2472>

[14] TRABAJO, Juan Pedro Muñoz Gea. *Desarrollo de sistemas y servicios de información sobre redes overlay peer-to-peer.* Departamento de Tecnologías de la Información y las Comunicaciones, Universidad Politécnica de Cartagena. Disponible en [internet]:

< http://repositorio.bib.upct.es:8080/jspui/handle/10317/2940>

[15] I. Stoica, R. I. Stoica, R. Morris, D. Karger, M. FransKaashoek, M. Dabek H. Balakrishnan, Chord. *A Scalable Peer-To-Peer Lookup Service for Internet Applications*. Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, (New York, USA. 2001) ISBN: 1-58113-411-8 pp: 149-160.

[16] A. Rowstron, P. Druschel. Pastry. *Scalable, distributed object location and routing for large scale peer to peer systems*. Proceedings of IFIP/ACM International Conference on Distributed Systems Plat- forms (Middleware 2001), Heidelberg, Alemania.

[17] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, J.D. Kubiatowicz. *Tapestry: a resilient global scale overlay for service deployment*. IEEE Journal on Selected Areas in Communications (Enero 2004). ISSN: 0733-8716 Vol.22 (1):41-53.

[18] Antonio Rodriguez de la Torre. *Intercambio de ficheros en la red Gnutella*. Seminario Interdisciplinar sobre la Sociedad de la Información, Programa de Doctorado de la Universidad Ouberta de Cataluña (UOC) 2002.

[19] Naeem Ramzan, Hyunggon Park, Ebroul Izquierdo; *Video streaming over P2P networks: Challenges and opportunities*. Elsevier. Signal Processing: Image Comunication (Mayo 2012). Vol. 27 (5): 401-411.

[20] Guo Y, Suh K, Kurose J, Towsley. *P2Cast: peer-to-peer patching scheme for VoD service*. In Proceedings of WWW'03, (Budapest, Hungary. Marzo 2003). ISBN: 1-58113-680-3 pp:301-309.

[21] C. Xu, G.M. Muntean, E. Fallon, A. Hanle. *A Balanced Treebased Strategy for Unstructured Media Distribution in P2P Networks*. Proceedings of IEEE ICC '08,(Junio 2008). ISBN: 978-1-4244-2075-9 pp: 1797-1801.

[22] W.-P. Yiu, X. Jin, S.-H. Chan, *VMesh: distributed segment storage for Peer-to-Peer interactive video streaming*. IEEE Journal of Select. Areas Communications (Diciembre 2007). ISSN: 0733-8716 Vol.25 (9) 1717–1731

[23] B. Cheng, H. Jin, X. Liao. *Supporting VCR Functions in VoD P2P Services Using Ring-Assisted Overlays*. Proceedings of IEEE ICC '07 (junio 2007). ISBN: 1-4244-0353-7 pp:1698-1703

[24] L. Guo, S. Chen, S. Ren, X. Chen, S. Jiang, *PROP: a scalable and reliable P2P assisted proxy streaming system*. Proceedings of ICDCS'04 (2004). ISSN-1063-6927 pp. 778–786

[25] G. Dan. *Cooperative caching and relaying strategies for peer-to-peer content delivery*. Proceedings of IPTPS'08, (2008).

[26] C. Dana, D. Li, D. Harrison, C.-N. Chuah. *BASS: BitTorrent Assisted Streaming System for Vídeo-on-Demand*. Proceedings of IEEE 7th Workshop on Multimedia Signal Processing, (Noviembre 2005) (Singhuai, China) ISBN: 0-7803-9289-2 pp: 1-4

[27] J.J.D. Mol, J.A. Pouwelse, M. Meulpolder, D.H.J. Epema, H.J. SIPs. *Give-to-get: free-riding resilient video-on-demand in P2P systems.* Proceedings of SPIE, Multimedia Computing and Networking Conference (MMCN), 2008

[28] M. Hefeeda, A. Habib, B. Botev, D. Xu, D.B. Bhargava. *PROMISE: A peer-to-peer media streaming using collectcast*. Proceedings of the ACM Multimedia (2003). ISBN:1-58113-722-2 pp: 45-54

[29] L.H. Ying, A. Basu. *pcVOD: Internet peer-to-peer video-on-demand with storage caching on peers*. Proceedings of the Eleventh International Conference on Distributed Multimedia Systems, Canada, 2005. Disponible en [internet]:

< http://webdocs.cs.ualberta.ca/~lihang/Research/DMS05-LihangYing.pdf>

[30] Tai T. Do, Kien A. Hua, Mounir A. Tantaoui. *Robust video-on-demand streaming in pee-to-peer environments*. Elsevier. Computer communications (2007). Vol.31 (3):506-519

[31] Karl-André Skevik, Vera Goebel, Thomas Plagemann. *Evaluation of a comprehensive P2P video-on-demand streaming system.* Elsevier. Computer Networks (marzo 2009). Vol. 53 (4): 434-455.

[32] Jian Liang, Rakesh Kumar, Yongjian Xi, Keith W. Ross, *Pollution in p2p file sharing systems*. INFOCOM'05: Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (Marzo 2005). ISSN: 0743-166X Vol. 2

[33] Artículo. Djamal-Eddine Meddour, Mubasher Mushtaq, Toufik Ahmed. *Open Issues in P2P Multimedia Streaming.* Multicomm 2006 pp:43-48. Disponible en [internet]: <http://www.researchgate.net/profile/DjamalEddine_Meddour/publication/254683356_Open_Issues_in_P2P_Multimedia_Streaming/links/53f1ca930cf23733e815e0da.pdf>

[34] Eytan Adar and Bernardo Huberman. "*Free Riding on Gnutella, First Monday*". (Octubre 2000). Disponible en [internet]:

< http://www.firstmonday.org/ojs/index.php/fm/article/view/792/701&lt%3B/Hu96>

[35] Tadeusz Piotrowski, Suman Banerjee, Sudeept Bhatnagar, Samrat Ganguly, Rauf Izmailov. *Peer-to-peer streaming of stored media: the indirect approach*. SIGMETRICS'06/Performance'06: Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems, ACM Press, 2006

[36] Chris Dana, Danjue Li, David Harrison, Chen-Nee Chuah. *Bass: Bittorrent assisted streaming system for video-on-demand*. MMSP'05: Proceedings of the Seventh IEEE Workshop on Multimedia Signal Processing (Octubre 2005). ISSN: 0-7803-9289-2 pp:1-4

[37] Yang Guo, Kyoungwon Suh, Jim Kurose, Don Towsle. *A peer-to-peer on-demand streaming service and its performance evaluation*. ICME'03: Proceedings of the 2003 International Conference on Multimedia and Expo, 2003

[38] Tai T. Do, Kien A. Hua, Mounir A. Tantaou. *P2vod: Providing fault tolerant video-on-demand streaming in peer-to-peer environment*. Technical Report, School of Electrical Engineering and Computer Science, University of Central Florida (2004). ISBN: 0-7803-8533-0 Vol.3 pp: 1467-1472

[39] Tai T. Do, Kien A. Hua, Mounir A. Tantaoui. *Robust vídeo-on-demand streaming in peer-to-peer environments*. Elsevier. Computer Communications (Febrero 2008). Vol. 31 (3):506-519

[40] X. Jiang, Y. Dong, D. Xu, B. Bhargava. *GnuStream: a P2P Media Streaming System Prototype*. IEEE International Conference on Multimedia and Expo Baltimore, MD. Julio 2003. ISBN: 0-7803-7965-9 Vol.2

[41] M. Hefeeda, A. abib, D. Xu, B. Bhargava, B. Botev. *Collectcast: a peer-to-peer service for media streaming*. ACM/Springer Multimedia Systems Journal (2005).

[42] M. Hefeeda, A. Habib, B. Botev, D. Xu, B. Bhargava. *PROMISE: Peer-to-Peer Media Streaming Using CollectCast*. MULTIMEDIA '03 Proceedings of the eleventh ACM international conference on Multimedia (noviembre 2003) ISBN:1-58113-722-2 pp: 45-54

[43] B. Cheng, L. Stein, H. Jin, X.F. Liao, Z. Zhang. *GridCast: Improving Peer Sharing for P2P VoD*. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP) Vol.4 (4): 1–31 Art. 26 (2008).

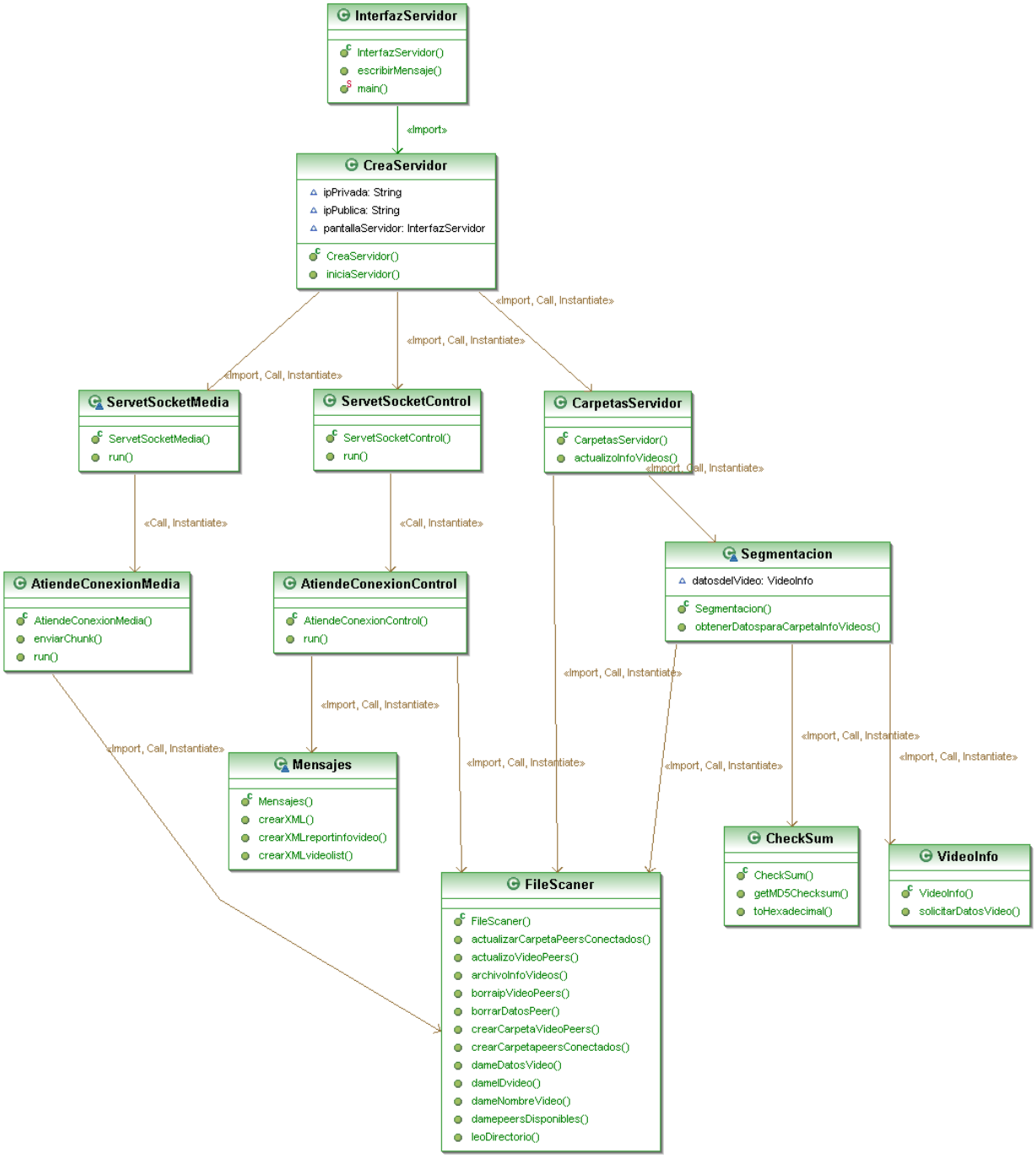
[44] Y. Zhang H. Liao, *Tracker Protocoldraft-gu-ppsp-tracker-protocol-02*, Internet-Draft Huawei,

[45] Jason Junter. JDOM y XML Parsing (Part 1,2,3). Oracle Magazine (2002-2003).

Anexos

Anexo I. Diagrama de Clases

Diagrama de clases del Servidor



Descripción de las clases del Servidor

InterfazServidor: Crea la interfaz gráfica donde se introducen las direcciones IP del equipo y del servidor a conectar y donde irán apareciendo mensajes de funcionamiento. En esta clase se encuentra el método main.

CreaServidor: Inicia la creación de la BBDD, y los dos hilos para la gestión de los Servetsocket de Control y Media.

CarpetasServidor: Crea las diferentes carpetas donde se almacenará toda la información necesaria para el servidor.

Segmentación: A esta clase se le pasa el nombre del archivo de video almacenado en el servidor, y obtiene de este, los id (md5) de cada segmento (512 Kb) y el id del video, quedando guardados estos datos en la BBDD del servidor.

FileScaner: Clase que implementa los métodos necesarios para crear archivos txt, y modificarlos.

ServetSocketControl y AtiendeConexionControl: Estas clases crean el Servetsocket para los mensajes de control entre el servidor y los peers.

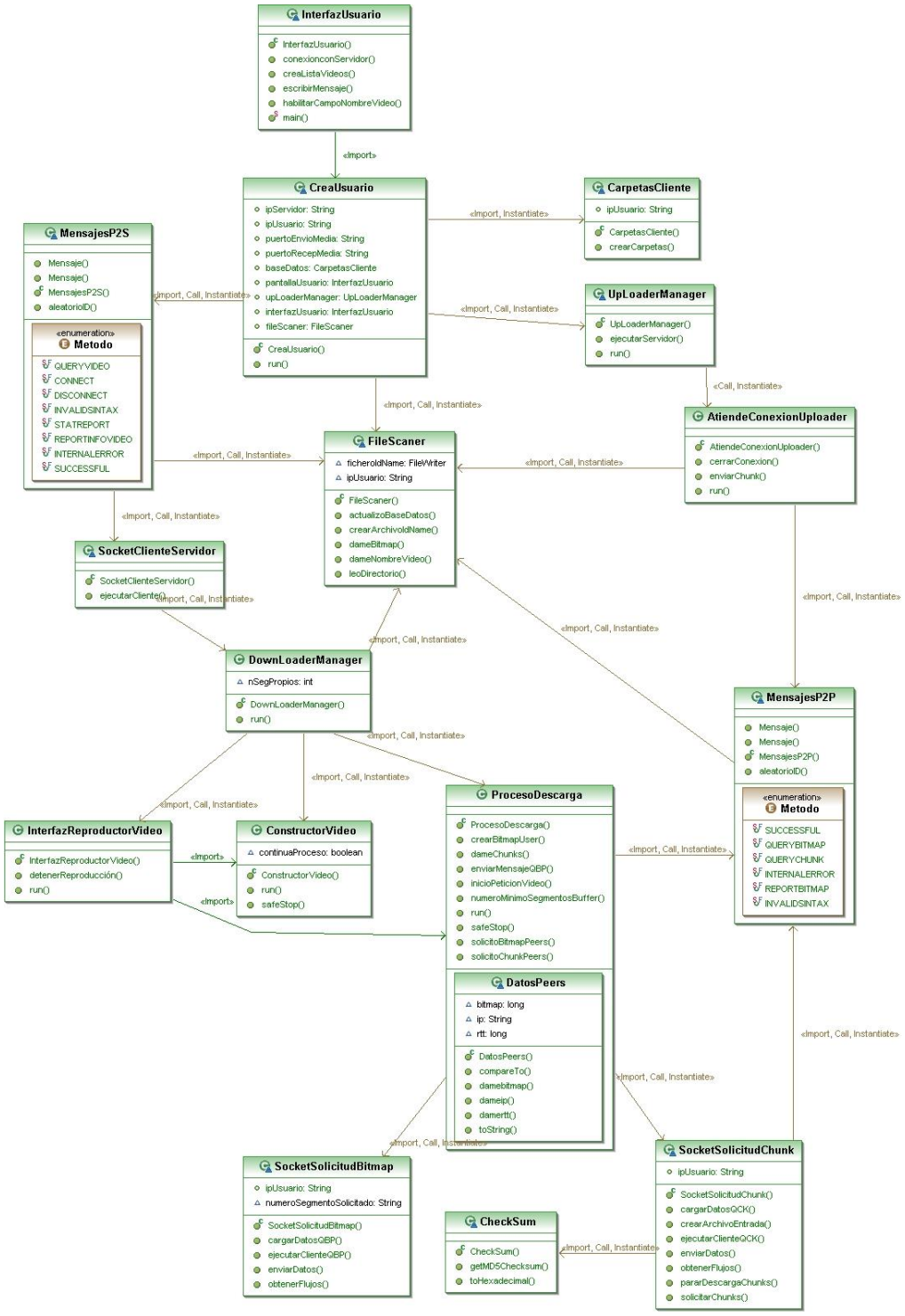
ServetSocketMedia y Atiende ConexionMedia: Crean el Servetsocket para la transferencia de datos de video con los peers.

Mensaje: Clase que implementa los métodos para crear los diferentes mensajes XML que necesitará enviar el servidor.

CheckSum: Esta clase es la encargada de calcular el md5 del array de bits que se la pasa.

VideoInfo: Esta clase permite obtener y modificar información de ficheros multimedia.

Diagrama de clases del Usuario



Descripción de las clases del Usuario.

InterfazUsuario: Esta clase crea la interfaz que usará el usuario para solicitar videos. Contiene el método main.

CreaUsuario: Una vez que el usuario se ha conectado al servidor, esta clase crea la aplicación completa del usuario.

MensajeP2S: Crea los mensajes entre el peer y el servidor.

SocketClienteServidor: Crea un socket con el servidor para los mensajes de control.

DowLoaderManager: Inicia el proceso de descarga de un video.

ProcesoDescarga: Con la información recibida del servidor, determina el mejor peer e inicia la solicitud de chunks a este peer.

SocketSolicitudBitmap: Crea el socket para la solicitud de bitmap a otros peers.

SocketSolicitudChunk: Crea el socket para la solicitud de chunk al peer seleccionado.

ContructorVideo: Clase que construye el vídeo con los segmentos recibidos por los peers fuente.

InterfazReproductorVideo: Clase que crea el interfaz a través del cual se reproducirá el video.

CarpetaCliente: Clase que crea la estructura de carpetas donde se almacenarán los videos descargados y la información de estos.

FileScaner: Clase que implementa los métodos necesarios para crear archivos txt y modificarlos.

UpLoaderManager y *AtiendeConexionUploaderManager*: Crean el ServetSocket, que estará escuchando cualquier solicitud de otro peer (bitmap o chunk).

MensajesP2P: Crea los mensajes que se intercambian entre peers.

CheckSum: Esta clase es la encargada de calcular el md5 del array de bits que se la pasa.

Anexo II. Secuencia de mensajes XML

A continuación mostramos como sería el intercambio de mensajes entre los diferentes elementos, basado en el esquema de la *Figura 6.* Ejemplo *petición de un vídeo* El Peer1 tiene descargado 304 segmentos del vídeo "RED", mientras que el Peer2 tiene 210 segmentos. Partiendo de este escenario, se muestra la secuencia de conexión de los 3 peers y la petición del vídeo "RED" por parte del Peer3.

- Mensajes enviados por el Usuario1
- Mensajes enviados por el Usuario2
- Mensajes enviados por el Usuario3
- Mensajes enviados por el Servidor

```
<?xml version="1.0" encoding="UTF-8"?>
<method transationID="-4224868218881575804">CONNECT
    <IPHost>127.0.0.11</IPHost>
</method>
<?xml version="1.0" encoding="UTF-8"?>
<method transationID="-4224868218881575804">SUCCESSFUL
</method>

<?xml version="1.0" encoding="UTF-8"?>
<method transationID="3062105143125950873">STATREPORT
    <IPHost>127.0.0.11</IPHost>
    <videoUser>
            <videoID>67a68a973f66452ad62c4c8cf41e616f</videoID>
    </videoUser>
</method>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<method transationID="3062105143125950873">VÍDEOLIST
    <cartelera>
        <vídeo>Vídeo_1.avi</vídeo>
        <vídeo>Vídeo_6.mp4</vídeo>
        <vídeo>Vídeo_4.mkv</vídeo>
        <vídeo>Moby Motivacion.wmv</vídeo>
        <vídeo>Vídeo_5.mp4</vídeo>
        <vídeo>Vídeo_9.mpg</vídeo>
        <vídeo>Shakira- Waka waka.wmv</vídeo>
        <vídeo>Vídeo_2.avi</vídeo>
        <vídeo>Vídeo_3.mkv</vídeo>
    </cartelera>
</method>

<?xml version="1.0" encoding="UTF-8"?>
<method transationID="-2832738935366420370">CONNECT
    <IPHost>127.0.0.22</IPHost>
</method>
<?xml version="1.0" encoding="UTF-8"?>
<method transationID="-2832738935366420370">SUCCESSFUL
</method>

<?xml version="1.0" encoding="UTF-8"?>
<method transationID="-5857282564260401017">STATREPORT
    <IPHost>127.0.0.22</IPHost>
    <videoUser>
            <videoID>67a68a973f66452ad62c4c8cf41e616f</videoID>
    </videoUser>
</method>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<method transationID="-5857282564260401017">VÍDEOLIST
    <cartelera>
        <vídeo>Vídeo_1.avi</vídeo>
        <vídeo>Vídeo_6.mp4</vídeo>
        <vídeo>Vídeo_4.mkv</vídeo>
        <vídeo>Moby Motivacion.wmv</vídeo>
        <vídeo>Vídeo_5.mp4</vídeo>
        <vídeo>Vídeo_9.mpg</vídeo>
        <vídeo>Shakira- Waka waka.wmv</vídeo>
        <vídeo>Vídeo_2.avi</vídeo>
        <vídeo>Vídeo_3.mkv</vídeo>
    </cartelera>
</method>

<?xml version="1.0" encoding="UTF-8"?>
<method transationID="-5289685811642340904">CONNECT
    <IPHost>127.0.0.33</IPHost>
</method>
<?xml version="1.0" encoding="UTF-8"?>
<method transationID="-5289685811642340904">SUCCESSFUL
</method>

<?xml version="1.0" encoding="UTF-8"?>
<method transationID="-5449672272108907800">STATREPORT
    <IPHost>127.0.0.33</IPHost>
    <videoUser />
</method>
<?xml version="1.0" encoding="UTF-8"?>
<method transationID="-5449672272108907800">VÍDEOLIST
    <cartelera>
        <vídeo>Vídeo_1.avi</vídeo>
        <vídeo>Vídeo_6.mp4</vídeo>
        <vídeo>Vídeo_4.mkv</vídeo>
        <vídeo>Moby Motivacion.wmv</vídeo>
        <vídeo>Vídeo_5.mp4</vídeo>
        <vídeo>Vídeo_9.mpg</vídeo><vídeo>Shakira- Waka waka.wmv</vídeo>
        <vídeo>Vídeo_2.avi</vídeo>
        <vídeo>Vídeo_3.mkv</vídeo>
    </cartelera>
</method>

<?xml version="1.0" encoding="UTF-8"?>
<method transationID="4977599472308850310">QUERYVÍDEO
```

```
        <IPHost>127.0.0.33</IPHost>
        <VideoName>Vídeo_9.mpg</VideoName>
    </method>

    <method transationID="4977599472308850310">REPORTINFOVÍDEO
        <videoID>67a68a973f66452ad62c4c8cf41e616f</videoID>
        <videoName>Vídeo_9.mpg</videoName>
        <bitRate>1113519</bitRate>
        <segmentos>1600
                <IDsegmento1>a37068b468a9fede37e0bf909e9e5cc0</IDsegmento1>
                <IDsegmento2>04fc88da3b367596915b10f3f2da6fbf</IDsegmento2>
                <IDsegmento3>9bf27ba0a2a0cf2f33611b9193567a35</IDsegmento3>
                “”””””””        “”””””””””””””””””        “”””””
                “”””””””        “”””””””””””””””””        “”””””
                “”””””””        “”””””””””””””””””        “”””””

                <IDsegmento1598>3ca1d794b90ca155b5eef181593fbfb8</IDsegmento1598>
                <IDsegmento1599>1b4041affa0044ccedf16a5c444af4ee</IDsegmento1599>
                <IDsegmento1600>b115ae253abf53de736ad845f29e50f0</IDsegmento1600>
        </segmentos>
        <peersdisponibles>
                <peerID>127.0.0.5</peerID>
                <peerID>127.0.0.11</peerID>
                <peerID>127.0.0.22</peerID>
        </peersdisponibles>
    </method>

<?xml version="1.0" encoding="UTF-8"?>
    <method transationID="5687638242632551495">QUERYBITMAP
        <videoID>67a68a973f66452ad62c4c8cf41e616f</videoID>
    </method>
    <?xml version="1.0" encoding="UTF-8"?>
    <method transationID="5687638242632551495">REPORTBITMAP
        <bitmap>210</bitmap>
    </method>

    <?xml version="1.0" encoding="UTF-8"?>
    <method transationID="5687638242632551495">QUERYBITMAP
        <videoID>67a68a973f66452ad62c4c8cf41e616f</videoID>
    </method>
    <?xml version="1.0" encoding="UTF-8"?>
    <method transationID="5687638242632551495">REPORTBITMAP
        <bitmap>304</bitmap>
    </method>

    <?xml version="1.0" encoding="UTF-8"?>
```

<method transationID="-4884512158252724388">**QUERYCHUNK**
<videoID>67a68a973f66452ad62c4c8cf41e616f</videoID>
<chunk>1</chunk>
</method>

“Peer2 envía CHUNK_1”

<?xml version="1.0" encoding="UTF-8"?>
<method transationID="79187438375958574 27">**QUERYCHUNK**
<videoID>67a68a973f66452ad62c4c8cf41e616f</videoID>
<chunk>2</chunk>
</method>

“Peer2 envía CHUNK_2”

“”””””””” “”””””””””””””””””” “”””””””
“”””””””” “”””””””””””””””””” “”””””””

<?xml version="1.0" encoding="UTF-8"?>
<method transationID="32907498439583364 31">**QUERYCHUNK**
<videoID>67a68a973f66452ad62c4c8cf41e616f</videoID>
<chunk>210</chunk>
</method>

“Peer2 envía CHUNK_210”

<?xml version="1.0" encoding="UTF-8"?>
<method transationID="8164946574914702564">**QUERYCHUNK**
<videoID>67a68a973f66452ad62c4c8cf41e616f</videoID>
<chunk>211</chunk>
</method>

“Peer1 envía CHUNK_211”

<?xml version="1.0" encoding="UTF-8"?>
<method transationID="-7275229220958745369">**QUERYCHUNK**
<videoID>67a68a973f66452ad62c4c8cf41e616f</videoID>
<chunk>212</chunk>
</method>

“Peer1 envía CHUNK_212”

<?xml version="1.0" encoding="UTF-8"?>
<method transationID="973473212938052977">**QUERYCHUNK**
<videoID>67a68a973f66452ad62c4c8cf41e616f</videoID>
<chunk>213</chunk>
</method>

“Peer2 envía CHUNK_213”

“”””””””” “”””””””””””””””””” “”””””””
“”””””””” “”””””””””””””””””” “”””””””

“”””””””” “”””””””””””””””””””” “”””””””

```
<?xml version="1.0" encoding="UTF-8"?>
<method transationID="-293869521843244763">QUERYCHUNK
    <videoID>67a68a973f66452ad62c4c8cf41e616f</videoID>
    <chunk>304</chunk>
</method>
```

“Peer2 envía CHUNK_304”

```
<?xml version="1.0" encoding="UTF-8"?>
<method transationID="-7816092855522460301">QUERYCHUNK
    <videoID>67a68a973f66452ad62c4c8cf41e616f</videoID>
    <chunk>305</chunk>
</method>
```

“Servidor envía CHUNK_305”

“”””””””” “”””””””””””””””””””” “”””””””

“”””””””” “”””””””””””””””””””” “”””””””

```
<?xml version="1.0" encoding="UTF-8"?>
<method transationID="2374001849638346098">QUERYCHUNK
    <videoID>67a68a973f66452ad62c4c8cf41e616f</videoID>
    <chunk>1600</chunk>
</method>
```

“Servidor envía CHUNK_1600”

```
<?xml version="1.0" encoding="UTF-8"?>
<method transationID="7185342700083239674">DISCONNECT
    <IPHost>127.0.0.33</IPHost>
</method>
<?xml version="1.0" encoding="UTF-8"?>
<method transationID="7185342700083239674">SUCCESSFUL
</method>

<?xml version="1.0" encoding="UTF-8"?>
<method transationID="-5473682585924423386">DISCONNECT
    <IPHost>127.0.0.22</IPHost>
</method>
<?xml version="1.0" encoding="UTF-8"?>
<method transationID="-5473682585924423386">SUCCESSFUL
</method>

<?xml version="1.0" encoding="UTF-8"?>
<method transationID="-3094404869769489225">DISCONNECT
    <IPHost>127.0.0.11</IPHost>
```

</method>
<?xml version="1.0" encoding="UTF-8"?>
<method transationID="-3094404869769489225">**SUCCESSFUL**
</method>