Escola Politècnica Superior de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO DE FIN DE CARRERA

TÍTULO DEL TFC: Desarrollo de un cliente DVB-IP con perfil Live Media Broadcast (LMB)

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad Telemática

AUTOR: Eugenio Viudez García

DIRECTOR: Ramon Borràs

SUPERVISOR: David Rincón Rivera

FECHA: 31 de Marzo de 2009

Título: Desarrollo de un cliente DVB-IP con perfil Live Media Broadcast (LMB)

Autor: Eugenio Viudez García

Director: Ramon Borràs

Fecha: 31 de Marzo de 2009

Resumen

En este proyecto se ha desarrollado un software cliente en lenguaje de programación C, capaz de visualizar canales DVB-IP (Digital Video Broadcast sobre redes IP).

DVB-IP es un estándar desarrollado por DVB para el envío de audio, video y otros datos a través de redes IP. La aplicación desarrollada cumple el perfil denominado LMB (Live Media Broadcast) definido en el mismo. Este perfil indica los pasos necesarios para la visualización de contenido "Live Media"

Se han desarrollado las funcionalidades definidas en el estándar para la selección de punto de entrada (Entry Point), selección del proveedor de servicios deseado, obtención del listado de canales disponibles en el proveedor de servicios y finalmente la visualización de dichos canales mediante la aplicación VLC.

Finalmente se ha implementado un escenario completo, compuesto en uno de sus extremos por un servidor DVB-IP, en el otro extremo el cliente desarrollado y entre ambos un router multicast intermedio, pudiendo realizar las pruebas y verificar el correcto funcionamiento de la solución desarrollada.

El software desarrollado podría ser integrado en un equipo con capacidad para interpretar el lenguaje C y realizar las funciones necesarias para la recepción de canales "Live Media", con cualquier proveedor de servicios que emita contenido DVB-IP.

Title: Development of DVB-IP client with Live Media Broadcast (LMB) profile.

Author: Eugenio Viudez García

Director: Ramon Borràs

Date: March, 31th 2009

Overview

This project has developed a software client in C programming language, capable of viewing DVB-IP (Digital Video Broadcast over IP networks).

DVB-IP is a standard developed by DVB for streaming of audio, video and data over IP networks. The application complies with the profile called LMB (Live Media Broadcast) defined in this standard, which includes the functionality necessary to display Live Medias.

We have developed the functionality defined in the standard for selection the Entry Point, selecting the desired service provider, obtain the list of available channels at the service provider and finally the visualization of these channels with VLC player.

Finally, we have implemented a complete scenario composed at one end by the server DVB-IP, an intermediate multicast router and at the other end the client developed, where we were able to perform testing and verification of proper operation of the solution developed.

The developed software can be integrated into equipment with the ability to interpret C language and perform the functions required for receiving channels "Media Live" with any service provider that delivers DVB-IP content.

DEDICATORIA

Quiero dedicar este TFC a mi madre Dora (mi bruji), por toda la paciencia que ha tenido siempre conmigo y el apoyo que me ha dado durante toda la carrera.

A mi sobrina Andrea, por darme todo el cariño del mundo recargándome de energía en los momentos más difíciles.

A mis hermanos Ismael y Mónica, junto con mis cuñados Antonia y Paco, que han sabido siempre darme el calor de una familia, cada uno a su manera pero siempre desde el corazón.

A mi gran compañera de universidad, Lidia Julià, que sin su ayuda no creo que hubiera terminado la carrera.

A todos los profesores que he tenido durante mi estancia en la EPSC.

Finalmente una dedicatoria especial a mi padre, Eugenio Viudez Agüera, a quien tanto le debo y tanto me ha enseñado en esta vida sin yo darme ni cuenta. Sé que habrías estado orgulloso, ojala hubieras podido estar aquí para verlo.

ÍNDICE

INTRODUCCIÓN 1

CAPÍTULO 1. BASES TEÓRICAS 3

1.1. Introducción a TV sobre IP (IPTV) 3

1.2. Encapsulado MPEG2-TS 4
1.2.1. Definición 4

1.3. Protocolos multicast para el Streaming sobre redes IP 5
1.3.1. IP Multicast 5
1.3.2. IGMP 6
1.3.3. Protocolos de enrutamiento multicast 7

1.4. XML (eXtensible Markup Language) 8

CAPÍTULO 2. INTRODUCCIÓN A DVB-IP 9

2.1. Que es DVB-IP? 9

2.2. Arquitectura 9
2.2.1. Perfiles DVB-IP 10
2.2.2. Pila de protocolos empleados en DVB-IP 11
2.2.3. Pasos basicos a seguir por el HNED 12

2.3. Service Discovery & Selection (SD&S) 12
2.3.1 Definición 12
2.3.2. Service Discovery 13
2.3.3. Service Selection 19

2.4. Mecanismos de transporte 20
2.4.1. DVB SD&S Transport Protocol (DVBSTP) 20
2.4.2. Uso de secciones en DVBSTP 22

CAPITULO 3. DISEÑO Y DESARROLLO 24

3.1. Introducción 24

3.2. Escenario 24
3.2.1. Servidor de metadatos XML con encapsulado DVBSTP 24
3.2.2. Servidor de streaming de video VLC 25
3.2.3. Router multicast XORP. 26
3.2.4. Visualización de videos con VLC. 26
3.2.5. Cliente DVB-IP. 26

3.3. Software empleado. 29
3.3.1. Software para el desarrollo y verificación. 29
3.3.2. Software integrado en el proyecto. 30

CAPITULO 4. PRUEBAS Y VERIFICACIÓN DE FUNCIONAMIENTO 32

4.1. Introducción 32

4.2. Envío documentos XML 32
4.2.1. Prueba envío correcto del SD&S 32
4.2.2. Verificación envío del SD&S 33
4.2.3. Verificación del envío todos los segmentos XML 34

4.3. Envío de los flujos de video mediante VLC 35

4.4. Control de flujos Multicast en el router XORP 37

4.5. Recepción documentos XML y videos en el cliente 37
4.5.1. Recepción documentos XML y videos en el cliente. 37
4.5.2. Pruebas de captura de los documentos XML 38
4.5.3. Extracción de la información de los documentos XML 40

4.6. Cliente completo DVB-IP 43

CONCLUSIONES Y LÍNEAS FUTURAS 44

Conclusiones. 44

Repercusión medioambiental 45

Aspectos éticos 46

Aspectos de seguridad 46

Líneas futuras 46

BIBLIOGRAFÍA 48

DVB-IP 48

IETF RFCs 48

Programación 49

Herramientas utilizadas. 49

Varios. 50

ANEXO I. DOCUMENTOS XML 52
ServiceDiscovery.xml 52
BroadcastRecord.xml 52
PackageRecord.xml 55

ANEXO II. TABLAS 56
Tabla de Audio y Video coding 56
Tabla de valores Payload ID en los segmentos 57
Tabla Service Provider(s) discovery record. 58
Tabla Service Provider(s) discovery record. 58
Tabla "TS Full SI" Discovery Information. 59
Tabla " TS Optional SI " Discovery Information. 60
Tabla Package Record. 61

ANEXO III. ARCHIVOS CONFIGURACIÓN 62
Configuración router XORP (config.boot) 62
Envío de flujos MPEG-TS de videos multicast con VLC. 65
Configuración DHCP. 67

ANEXO IV. CÓDIGO DISEÑADO.....68
Serverxml2.c 68
Capturador.c 72
ParserCompleto.c 76
ClienteCompleto.c 85
Capturador.c (integrado en ClienteCompleto.c) 86
ParserCompleto.c (Integrado en ClienteCompleto.c) 89

INTRODUCCIÓN

En muchos países se ha desarrollado en mayor o menor medida servicios de IPTV, como por ejemplo en España con Jazztelia TV [28], Imagenio [29], etc., o Reino Unido con BT Vision [38], para la entrega de varios servicios, como pueden ser televisión por Internet, servicios interactivos, videos bajo demanda, etc. Pero muchos de estos servicios no son estándar y se basan en tecnologías propietarias, como es el caso de BT Vision que usa una plataforma desarrollada por Microsoft llamada Microsoft Mediaroom [37]. Esto nos condiciona al uso de soluciones privadas diferentes para el uso de cada uno de estos servicios IPTV.

Una vez visto el estado actual en cuanto a la difusión de medios audiovisuales a través de Internet, es conde cobra sentido la aparición de DVB-IP. DVB-IP es un estándar desarrollado por DVB para el envío de Televisión digital a través de redes IP bidireccionales sobre redes de banda ancha. Una de estas tecnologías de banda ancha es ADSL, la cual ha supuesto un fácil y extenso uso de la red de redes (Internet) y se prevé que dicha tendencia sea la de usar esta conexión para la retransmisión de estos contenidos.

Una de las grandes ventajas de la estandarización que tiene el proyecto DVB-IP, es el abaratamiento de los costes de producción de los equipos pudiendo realizar grandes producciones para un amplio mercado internacional, además al tratarse de un estándar abierto se reducen los posibles costos del uso de tecnologías privadas.

El usuario final podrá beneficiarse de otra gran ventaja con el uso de DVB-IP, ya que podrá conectar un equipo que cumpliera el estándar, independientemente del ISP (Internet Service Provider) que tuviera, pudiendo seleccionar el proveedor de DVB-IP a visualizar

La propuesta de DVB-IP es crear un estándar abierto, el cual este basado en tecnologías ya conocidas, usando en gran proporción métodos ya conocidos en la distribución de servicio multimedia a través de medios de radiodifusión, Satélite (DVB-S), Cable (DVB-C) y Terrestre (DVB-T), los cuales ya están en gran uso en multitud de países.

En el Handbook DVB-IPTV [1] especifica los todos los protocolos y mecanismos que deberá cumplir el cliente DVB-IP y especifica varios tipos de servicios de IPTV, como por ejemplo "Live Media" o CoD (Content on Demand), pero para operadores o fabricantes que no requieran un completo uso de todas las funciones que se ofrecen en DVB-IP, se introduce el concepto de perfil. Estos perfiles se definen como niveles de funcionalidad, para que terminales con más o menos complejidad puedan interactuar con los servidores DVB-IP.

Según la definición del estándar, el perfil LMB ha de ser capaz de visualizar los canales “Live Media”, canales que se visualizan mediante streaming multicast

de videos encapsulados en MPEG-TS los cuales irán sobre paquetes UDP directamente o bien sobre UDP/RTP.

El presente Trabajo Fin de Carrera tiene como objetivo el implementar un software en lenguaje C el cual sea capaz de cumplir con el perfil LMB (Live Media Broadcast) determinado en el estándar DVB-IP.

El presente documento se ha desarrollado en diferentes capítulos los cuales se organizan de la siguiente manera:

Capítulo 1: En este capítulo se van a comentar algunas tecnologías que aparecerán a lo largo del proyecto para tener un conocimiento previo sobre ellas en el momento que se mencionen. Estas tecnologías son una introducción a IPTV, encapsulado MPEG2-TS, IP multicast, IGMP, protocolos de enrutamiento multicast y finalmente XML.

Capítulo 2: Se realiza una introducción a DVB-IP, mostrando su arquitectura, el funcionamiento del descubrimiento de servicios mediante el proceso Service Discovey & Selection. Así mismo se hace una mención a los métodos de transporte y encapsulación de los datos, finalmente obteniendo una tabla de los protocolos implicados en todo el estándar DVB-IP.

Capítulo 3: En este capítulo se muestra el escenario implementado en el proyecto y las herramientas empleadas tanto para el desarrollo como los elementos implementados dentro del propio proyecto.

Capítulo 4: Pruebas de correcto funcionamiento y verificación del estándar en el código implementado.

Capítulo 5: Finalmente se describen las conclusiones del proyecto, las líneas futuras a seguir, la repercusión medioambiente, los aspectos éticos y de seguridad del proyecto.

CAPÍTULO 1. BASES TEÓRICAS

En este capítulo se van a comentar algunas tecnologías que aparecerán a lo largo del proyecto para tener un conocimiento previo sobre ellas en el momento que se mencionen. Estas tecnologías son una introducción a IPTV, encapsulado MPEG2-TS, IP multicast, IGMP, protocolos de enrutamiento multicast y finalmente XML.

1.1. Introducción a TV sobre IP (IPTV)

En muchos países se ha desarrollado en mayor o menor medida servicios de IPTV. El aumento del ancho de banda de las redes IP y la aparición de nuevas tecnologías, como la ADSL, han permitido el despliegue de estos servicios, que hasta ahora estaban reservados para otros medios, como por ejemplo el cable o el satélite. La definición de IPTV (Internet Protocol Televisión) hace referencia únicamente al mecanismo de transmisión del servicio de televisión (servicios de alta calidad, comparable a la TV actual analógica o digital) a través de redes IP. IPTV responde a un entorno cerrado en el que el proveedor del servicio controla tanto la red de transmisión como los contenidos o el acceso a los mismos.

También existe lo que se denomina Internet TV (no confundir con IPTV), el cual esta basado en retransmisiones de video utilizando Internet para llegar a los usuarios. La televisión por Internet representa un entorno menos controlado, en el que tanto los contenidos como su acceso tienen un carácter más abierto. En contraposición tiene una baja calidad de visualización y no hay ningún tipo de garantías de calidad del servicio.

Los servicios que se pueden ofrecer a través de IPTV son variados, como pueden ser el "Live Media", PPV (Pay per View) o el CoD (Content on Demand). A continuación tenemos una descripción de los mismos.

Live Media: Por servicios "Live Media" se entiende la difusión de contenidos de forma común para todos los usuarios, es decir, los clásicos canales de televisión. En IPTV la transmisión de estos canales se realiza a través de canales multicast, ya que es la manera más eficiente de emitir el mismo contenido a un número grande de receptores (ver apartado 1.3)

CoD: Este servicio permite al usuario escoger entre un catálogo de contenidos y reproducir cualquiera de ellos en cualquier momento. La reproducción es enviada desde los servidores del proveedor al usuario de forma individual, de modo que este puede pausar, rebobinar o avanzar la reproducción a su voluntad. En IPTV la transmisión de estos canales se realiza a través de canales unicast.

1.2. Encapsulado MPEG2-TS

Para la entrega de video en el estándar DVB-IP [1] se define el encapsulado MPEG2-TS como “contenedor” de los datos multimedia, pudiendo ser la codificación de video y audio cualquiera de las especificadas en el estándar [5]. En el anexo II hemos introducido la tabla *Audio y Video coding*, con las codificaciones admitidas por DVB.

1.2.1. Definición

MPEG2 Transport Stream define la sintaxis y la semántica que han de adoptar los flujos MPEG de video y datos para su multiplexación. Este formato esta especificado en el Standard ISO/IEC 13818-1 [10]. Para su empleo en DVB, dicha estructura debe complementarse con la denominada: “Información del Servicio (SI)” (es decir, las tablas que permiten saber qué programa estamos viendo, qué flujos de video, audio y datos –por ejemplo, teletexto entre otras informaciones), que está especificada por DVB en la norma ETSI EN 300 468 [43].

Para comprender el funcionamiento de la multiplexación de los diferentes tipos de flujos, vamos a ver cómo está formado cada paquete Transport Stream en la figura 1.1

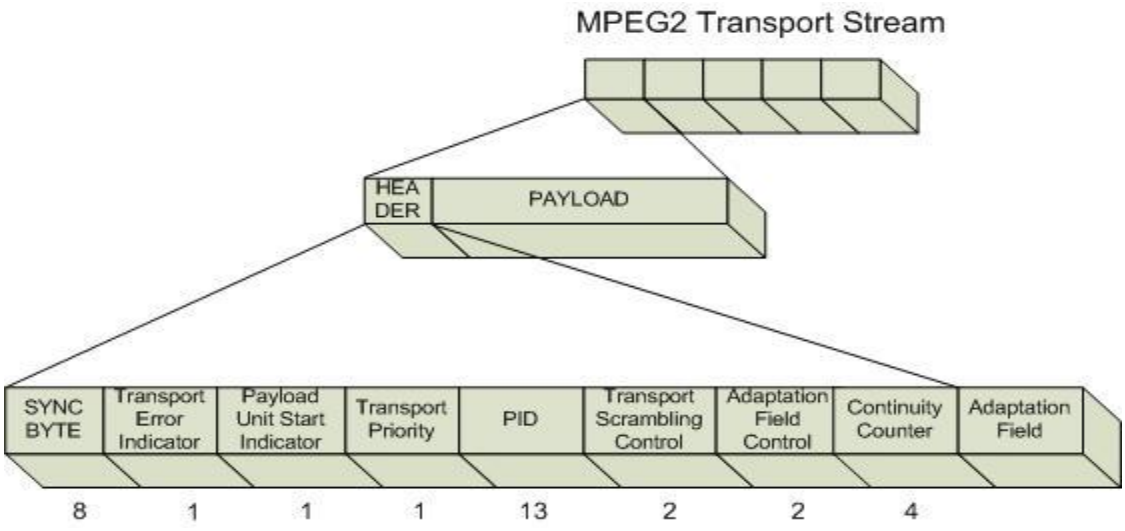

Fig. 1.1 Estructura paquete MPEG2-TS

Cada paquete tiene una medida fijada a 188 bytes, los cuales están formados por una cabecera (Header) y una carga (Payload).

En la cabecera podernos ver varios campos, de los más relevantes son:

- Sync Byte: Sirve para que el receptor pueda sincronizarse correctamente con los datos entrantes. Tiene el valor 0x47 y delimita el inicio de un paquete TS.

- PID: Es un identificador que nos dirá que tipo de información tenemos en la carga (video, audio, texto, etc.).

- Continuity Counter: El codificador lo incrementa en 1 cada vez que envía un paquete de la misma fuente. Esto permite que el decodificador sea capaz de deducir si ha habido una pérdida (o ganancia incluso) de un paquete de transporte y evitar errores

Dentro del Adaptation Fiel se deben destacar los dos siguientes:

- PCR (Program Clock Referente): Es una información de sincronización del reloj de 27 MHz del receptor necesaria para la descodificación del video, audio y datos. Se incluye periódicamente en los paquetes de transporte. El receptor necesita esta información – a una cadencia de unas 10 veces por segundo.

- Bytes “comodines”: Son bytes de relleno para conseguir una trama de 188 bytes de información en el supuesto de que no hubiera información suficiente para llenar el paquete.

1.3. Protocolos multicast para el Streaming sobre redes IP

A continuación se va a presentar una introducción a IP multicast, así como una descripción de los protocolos necesarios para el uso de IP multicast en DVB-IP.

1.3.1. IP Multicast

Para las transmisiones de los flujos de datos y sobre todo para la transmisión de los flujos de video, es donde cobra sentido la utilización de trafico Multicast, aunque este no nos será útil para todo tipo de situaciones, sí será el más conveniente para los Live Media.

La gran ventaja de usar multicast en comparación con unicast, es que tan solo se envía una única copia de la información y esta se hace llegar a los miembros del grupo multicast.

Transmisión Unicast: Es tráfico en el que se envía información de un origen concreto hacia un único destinatario. El uso de esta forma de transmisión no es un método eficiente por el consumo de ancho de banda, pero garantiza una conexión entre servidor y cliente. Serán los usados para contenidos VoD y se suelen usar bajo el protocolo Real Time Streaming Protocol (RTSP) [13].

Transmisión Broadcast: Se envía el tráfico hacia todos los hosts de una red. Este tipo de tráfico se emplea para redes locales puesto que en redes como Internet no está permitido este tipo de tráfico.

Transmisión Multicast: A diferencia del trafico Broadcast, se envían los datos a un grupo concreto de hosts, los cuales han solicitado dicho trafico con anterioridad hacia un servidor de datos Multicast. En este caso el ancho de banda es usado de una forma más eficiente para llegar hacia los hosts destinatarios, puesto que una sola copia de un flujo puede llegar a múltiples destinatarios. De acuerdo con el RFC 3171 las direcciones Multicast son desde la 224.0.0.0 a la 239.255.255.255. Formalmente a este rango se le llama “clase D”.

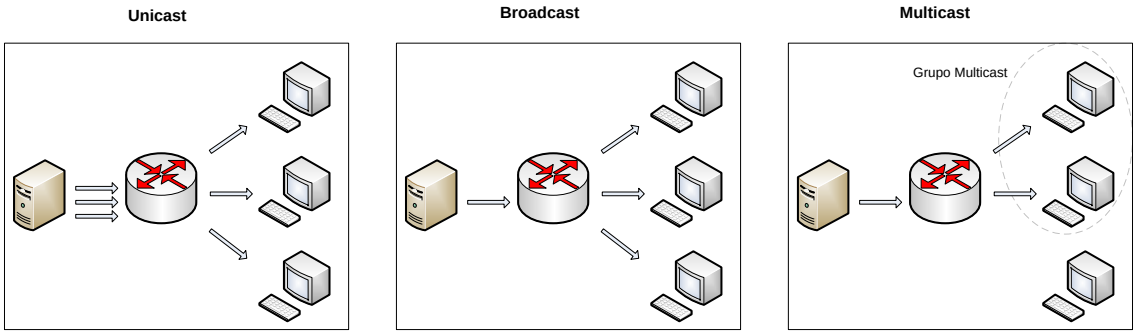

Fig. 1.2 Unicast, Broadcast y Multicast

1.3.2. IGMP

Internet Group Management Protocol (IGMP) es un protocolo usado por los hosts para solicitar añadirse a un grupo Multicast, de esta forma empezar a recibir el trafico del flujo de datos solicitado.

Un grupo multicast es el símil a lo que en Televisión convencional seria un canal de TV, un usuario se “sintoniza” a un canal de radiofrecuencia y empieza a recibir un canal de TV. Cuando un usuario está solicitando subscribirse a un grupo multicast, está solicitando recibir un cierto flujo multicast de entre los que están disponibles. En el caso particular de IPTV, cada grupo multicast está asignado a un programa de TV.

El ámbito en el cual tiene presencia el protocolo IGMP es entre el host cliente y el dispositivo que nos ofrece conectividad (el router de acceso, el primer router, al que está conectado directamente el terminal) con el Proveedor de Servicios, es decir, que este protocolo tan solo estará presente dentro de la red local (LAN).

Existen tres versiones de este protocolo, que se diferencian en la gestión de las altas y bajas del grupo multicast:

- IGMP v1: Es la primera versión que se definió. En esta versión tan solo se existen dos tipos de mensajes, “Membership Query” y “Membership Report”. El primero para consultar a los host si su pertenencia al grupo y el segundo para reportar desde el host su solicitud de añadir a un grupo o su solicitud para continuar perteneciendo a un grupo.
- IGMP v2: Con las mismas características que la v1, pero en esta versión se añade un nuevo mensaje “Leave Group”, el cual es usado para solicitar expresamente la exclusión por parte del host hacia un grupo en el cual estaba añadido.
- IGMP v3: Es la última versión, la cual da la capacidad de aunar solicitudes de pertenencia a varios grupos en un solo paquete. Los “Query” son enviados a la IP 224.0.0.1 (todos los host Multicast) y los “Response” son enviados hacia la 224.0.0.22 en la cual escuchan todos los routers con capacidad Multicast que implementen IGMP v3. Esta última versión es la que está definida en el estándar DVB-IP.

1.3.3. Protocolos de enrutamiento multicast

Para el enrutado de tráfico multicast en la red troncal (en el núcleo de la red, una vez pasado el router de acceso) en el estándar DVB-IP no se indica ningún protocolo específico, dejando este punto abierto a elección por el Proveedor de Servicios

Existen varios protocolos de enrutamiento multicast como pueden ser PIM-SM (Protocol Independent Multicast – Sparse Mode), MOSPF (Multicast Open Shortest Path First), etc., que podrían ser usados para en encaminamiento de los paquetes multicast a través de la red troncal del Proveedor de Servicios desde el origen de los datos multicast hasta el router local de los clientes. Para hacer un escenario realista del “testbed” de DVB-IP de este proyecto hemos elegido el protocolo PIM-SM.

Protocol Independent Multicast – Sparse Mode, es un protocolo de enrutamiento para redes multicast. Como su propio nombre indica es independiente del protocolo de enrutamiento unicast que se emplee. Está diseñado para una optimización de los recursos de red cuando tenemos flujos de datos multicast.

Por un lado, tendremos registrados todos los flujos de datos multicast, estos serán registrados en un router de la red multicast designado para esta función, el RP (Rendezvous Point)

Por otro lado, los clientes mediante el protocolo antes mencionado IGMP, realizan la petición de unión a un grupo multicast (dirección multicast), siendo entregado solo el trafico multicast deseado, evitando así la inundación de la red con todos los flujos de datos multicast.

1.4. XML (eXtensible Markup Language)

XML (eXtensible Markup Language) [41], es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C) [40]. Es una simplificación y adaptación del SGML [42] y permite definir la gramática de lenguajes específicos. Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable [27].

Existen tres términos comúnmente usados para describir las partes de un documento XML: etiquetas, elementos y atributos. Aquí está un documento de ejemplo que ilustra estos términos, figura 1.3:

```
<?xml version="1.0" encoding="UTF-8"?>
<direccion>
      <persona>
            <titulo>Sr.</titulo>
            <nombre>Pepito </nombre>
            <apellidos>Grillo </apellidos>
      </persona>
      <calle> C/ Rue de la Street </calle>
      <ciudad provincia="BCN"> Esplugas de Llobregat</ciudad>
      <codigo-postal> 08950 </codigo-postal>
</direccion>
```

Fig. 1.3 XML de ejemplo.

Etiqueta: Una etiqueta es un texto entre el símbolo menor que (<) y el símbolo mayor que (>). Existen etiquetas de inicio (como <nombre>) y etiquetas de fin (como </nombre>).

Elemento: Un elemento consta de la etiqueta de inicio, la etiqueta de fin y de todo aquello que este entre ambas. En el ejemplo anterior, el elemento <persona> contiene tres elementos hijos: <titulo>, <nombre>, y <apellidos>.

Atributo: Un atributo es un par nombre-valor dentro de la etiqueta de inicio de un elemento. En el ejemplo, provincia es un atributo del elemento <ciudad>,

En el estándar DVB-IP se indica que los documentos XML han de ser validos y well-formed. Para que un XML sea válido, se ha de ajustar a las reglas semánticas del XML-Schema definido en el estándar [1], y para que sea well-formed se debe ajustar a las normas de sintaxis XML [41], por ejemplo, si una etiqueta de inicio (<>) aparece sin la correspondiente etiqueta de final (</>), el XML no está bien formado.

CAPÍTULO 2. INTRODUCCIÓN A DVB-IP

2.1. Qué es DVB-IP?

DVB-IP es un estándar desarrollado por DVB para el envío de Televisión digital a través de redes IP bidireccionales sobre redes de banda ancha (por ejemplo, ADSL), en el cual se exponen las especificaciones técnicas de interoperabilidad.

Las especificaciones claves están publicadas en los documentos:

- Digital Video Broadcasting (DVB); Transport of MPEG-2 TS Based DVB Services over IP Based Networks (DVB-IPTV Phase 1.4 [1].

- Digital Video Broadcasting (DVB); Guidelines for the implementation of DVB-IP Phase 1 specifications, ETSI TS 102 542 V1.2.1. [2].

- Digital Video Broadcasting (DVB); DVB-IPTV Profiles for TS 102 034, ETSI TS 102 826 V1.1.1. [3].

- Digital Video Broadcasting (DVB); Specification for the use of Video and Audio Coding in Broadcasting Applications based on the MPEG-2 Transport Stream, ETSI TS 101 154 V1.8.1 (2007-07). [5].

2.2. Arquitectura

En la figura 2.1 podemos ver una arquitectura básica DVB-IP. En ella como puntos clave de referencia tenemos el Proveedor de Contenidos, el Proveedor de Servicios, el Home Network Gateway y el HNED (Home Network End Device).

Proveedor de Contenidos: Se refiere a la entidad que posee contenidos o tiene licencia para venderlos. Por ejemplo podría ser Telecinco o La Fox, que deciden ofrecer sus contenidos vía Internet.

Proveedor de Servicios: Es la entidad que proporciona el servicio al usuario final. Existen diferentes tipos de proveedor de servicios para DVB-IP, un ejemplo podría ser directamente el ISP (Internet Service Provider).

Home Network Gateway: Es el dispositivo encargado de interconectar la red del Service Provider con la red local del usuario, por ejemplo el router local.

Home Network End Device (HNED): Es el dispositivo conectado a la red local instalado en casa del usuario y donde generalmente termina el flujo de datos IP, aunque no necesariamente ha de ser el equipo final de los flujos de datos que no sean IP, ya que este equipo podría hacer una retransmisión de estos

datos recibidos hacia otra tecnología que no fuera IP (por ejemplo IEEE 1394 hacia un monitor de TV).

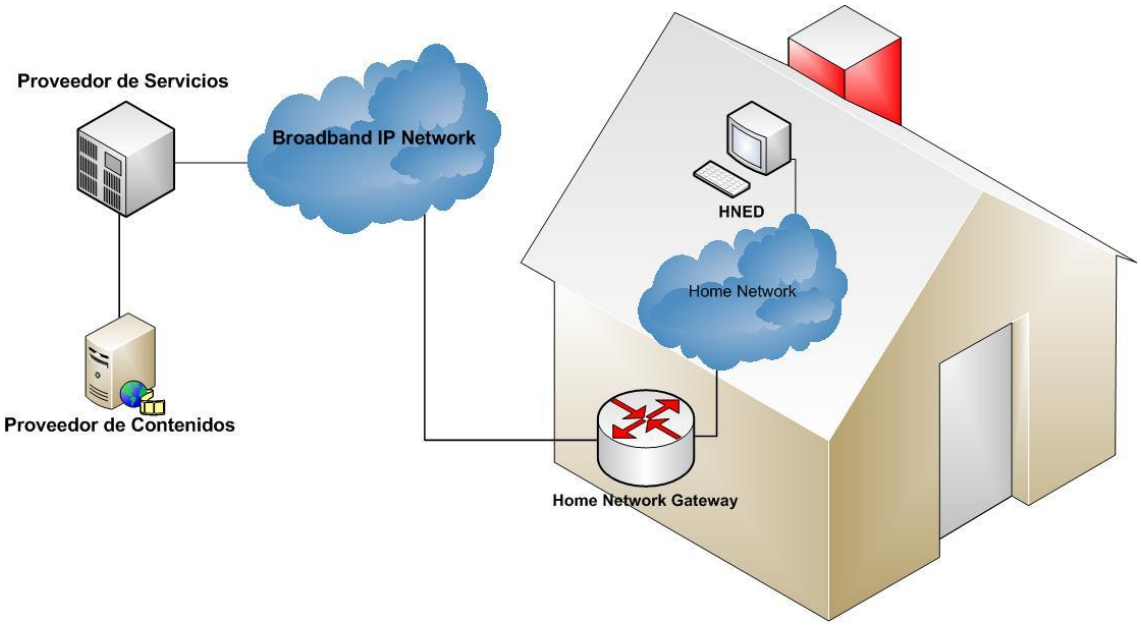

Fig. 2.1 Arquitectura básica DVB-IP.

2.2.1. Perfiles DVB-IP

Para operadores o fabricantes que no requieran un completo uso de todas las funciones que se ofrecen en DVB-IP, se introduce el concepto de perfil. Estos perfiles se definen como niveles de funcionalidad, para que terminales con más o menos complejidad puedan interactuar con los servidores DVB-IP. En la tabla 2.1. podemos ver estos tres niveles y sus funcionalidades.

Tabla 2.1. Perfiles DVB-IP.

Profiles	Modules				
	Transport	Connection	Format	Discovery	Metadata
Basic	UDP	IGMP	MPEG2	SD&S	SD&S XML data SI/PSI tables
LMB	UDP RTP/UDP	IGMP	Refer to TS 101 154	SD&S	SD&S XML data SI/PSI tables
CoD	UDP RTP/UDP	RTSP	Refer to TS 101 154	SD&S / BCG	SD&S XML data BCG-TVA

Basic: Este perfil solo trabaja con UDP a nivel de transporte, siendo su entorno de aplicación escenarios controlados (como por ejemplo una red Local) en los cuales no sea necesario el uso de RTP (Real Time Protocol) [14]. Los contenidos que se pueden visualizar son “Live Media” (es decir, canales

emitidos en multicast que son retransmitidos en vivo, similar a la emisiones de TV convencionales) ya que tan solo implementa IGMP a nivel de conectividad, además, estos contenidos tan solo podrán estar codificados en MPEG2.

LMB: Este perfil, además de las funcionalidades del perfil Basic, implementa la opción de trabajar con RTP/UDP, siendo posible su entorno de aplicación escenarios mas abiertos (como por ejemplo Internet a través de ADSL) en los cuales no se tenga tanto control ya que dispone de RTP. La codificación de los videos podrá ser cualquiera de las que se indican en el estándar TS 101 154 [5]. Del cual podemos ver un resumen en la tabla del anexo II.

CoD: Este perfil es el más completo de todos y el que cumple con todos los servicios posibles en DVB-P. Se pueden visualizar tanto canales "Live Media", como "Live Media" con Trick Mode (de similares características a los Live Media pero con las opciones de, Play, Stop, Forward, etc.) o servicios de Content on Demand, ya que implementa a nivel de conectividad RTSP [13]. A nivel de información de metadatos se podrá visualizar la EPG (Electronic Program Guide) ya que contempla el funcionamiento de BCG-TVA.

Para el desarrollo de este proyecto, como se indica en el titulo del mismo, centraremos los esfuerzos en realizar un cliente el cual cumpla con los requisitos establecidos para el Perfil LMB (Live Media Broadcast).

2.2.2. Pila de protocolos empleados en DVB-IP

A continuación, en la figura 2.2, se muestra la pila de protocolos de transporte de informes DVB-IP.

Esta pila de protocolos especifica los protocolos necesarios para transportar los elementos del servicio ofrecido vía red IP, independientemente de las capas físicas bajo la capa de red IP.

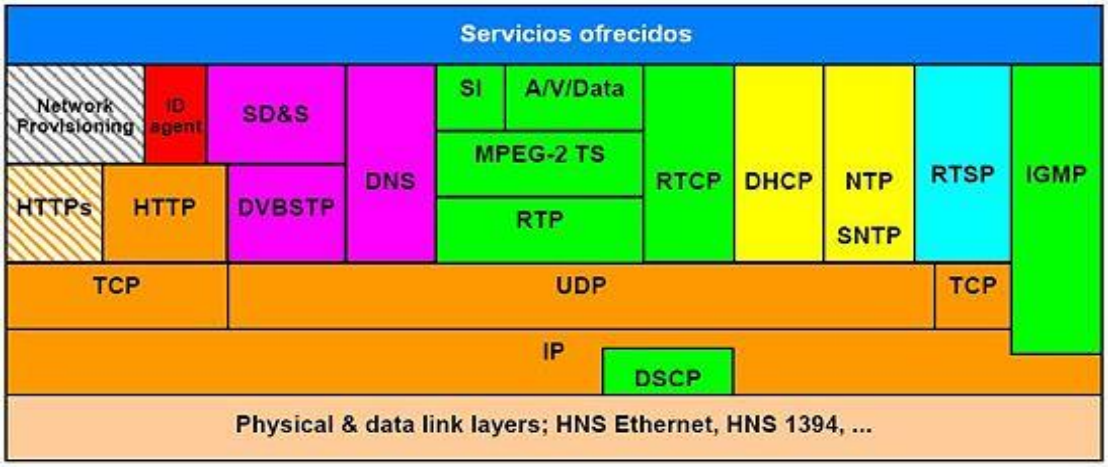

Fig. 2.2 Pila de protocolos para servicios DVB-IP, extraída de [1].

2.2.3. Pasos básicos a seguir por el HNED

Una vez tenemos definida la arquitectura DVB-IP, veamos un resumen de cuales son los pasos básicos que debe seguir un HNED para obtener servicios DVB-IP. En la figura 2.3 se muestra paso a paso los procesos que se han de seguir para el funcionamiento de un cliente (HNED) que cumpla el estándar DVB-IP.

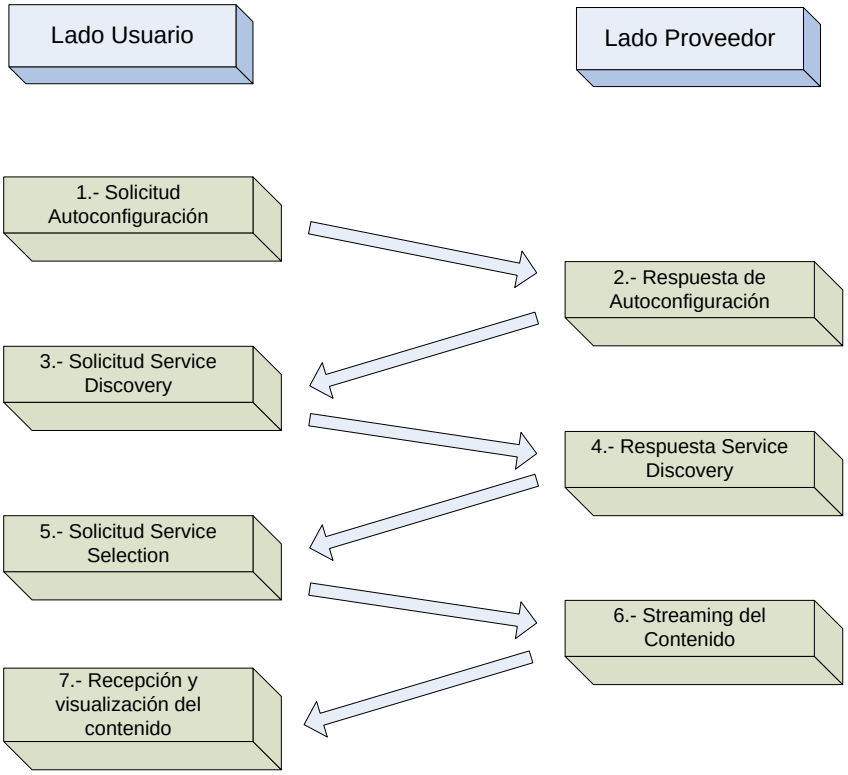

Fig. 2.3 Procedimiento a seguir en DVB-IP.

2.3. Service Discovery & Selection (SD&S)

2.3.1 Definición

En el estándar DVB-IP se definen mediante el SD&S los mecanismos utilizados a través de la red IP para el descubrimiento de servicios, su selección y su forma de entrega al usuario.

El método de transporte de la información puede ser en modo Pull o en modo Push. En el primero de los casos, Pull, el Terminal es el que inicia de manera activa la conexión con el servidor solicitando la información. Aunque ambos métodos se pueden realizar indistintamente en Unicast o en Multicast, la tendencia es que el modelo Pull se establece una conexión unicast entre el servidor y el cliente típicamente mediante TCP. En el modelo push, no se establece una conexión, sino que mediante UDP el servidor va emitiendo de forma continua la información en multicast, y el Terminal la recibe de manera pasiva (se suscribe al grupo multicast).

2.3.2. Service Discovery

En este apartado se definen los mecanismos utilizados para identificar los Proveedores de Servicios y los servicios en el contexto de descubrimiento de servicios. Es decir, cómo un usuario va a recibir la lista de proveedores y de canales que tiene disponibles.

2.3.2.1. Identificación de Proveedor de Servicios y de los Servicios

En este apartado se describen los mecanismos que se emplean para poder identificar al Proveedor de Servicios y a los servicios ofrecidos por este. El Proveedor de servicios se identifica únicamente con el nombre del dominio DNS que se haya registrado. Un nombre de dominio de un SP podría ser por ejemplo "canal-plus.com"

Para los servicios existen dos mecanismos básicos para su única identificación:

- Mediando DNS se le asigna un nombre único a cada servicio y se concatena este nombre de servicio con el nombre de dominio DNS del proveedor de servicios. Un ejemplo seria "canaldeportes.canal-plus.com"

- La tripleta DVB se trata de un trío de identificadores numéricos: original_network_id, transport_stream_id y service_id, con el cual permite distinguir entre el mismo servicio transportado por diferentes redes. Por ejemplo, la tripleta distinguiría Telecinco transportado por Hispasat o por Astra.

2.3.2.2. Tipos de datos SD&S.

En la figura 2.4 podemos ver un esquema de los distintos tipos de documentos XML que se definen para la entrega de la información sobre el SD&S.

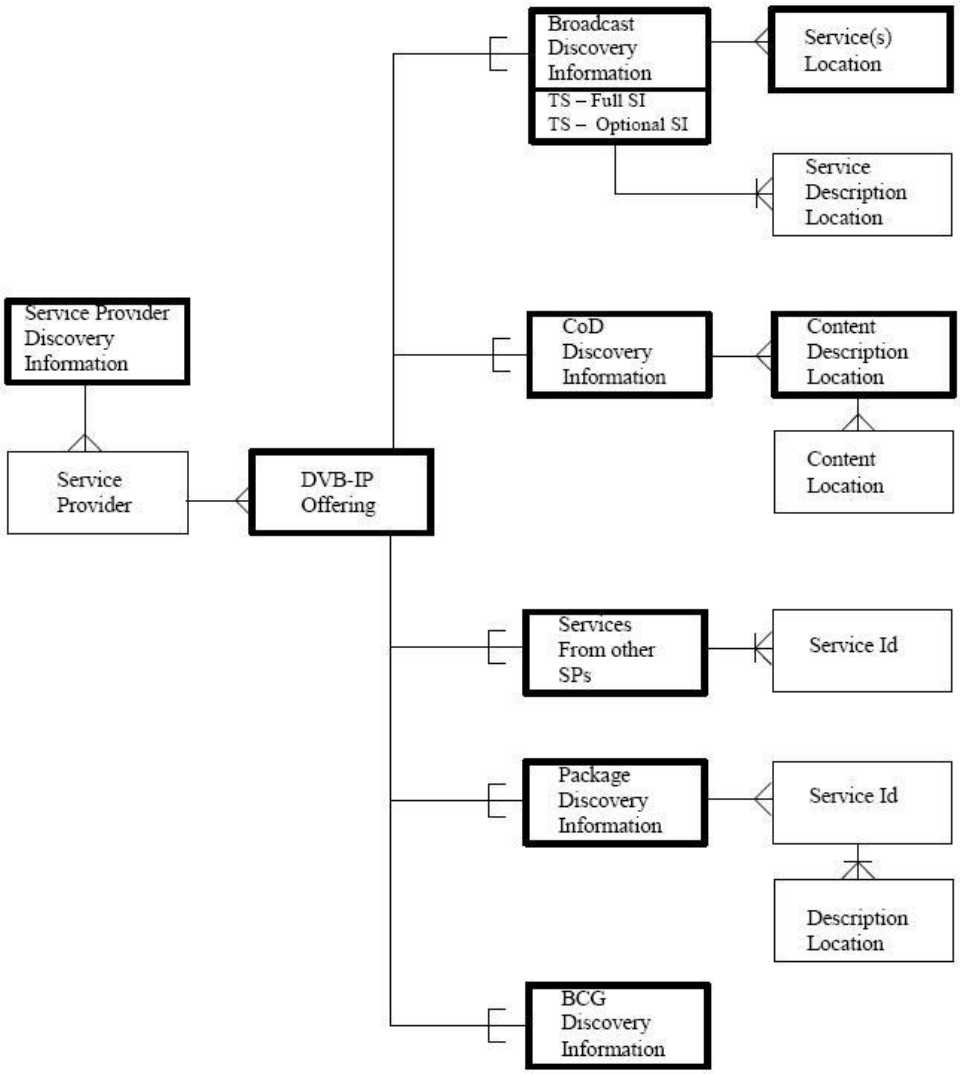

Fig. 2.4 Tipos de documentos SD&S, extraída de [1].

Service Provider Discovery Information, proporciona la información necesaria sobre la oferta de servicios DVB-IPTV del Proveedor de Servicios, bien sean Livemedia o Content on Demand

Bajo el nombre de DVB-IP Offering se aúnan 5 tipos de documentos los cuales proporcionan información sobre los servicios del Proveedor de Servicios.

- Broadcast Discovery Information, proporciona toda la información necesaria para encontrar los Live Media disponibles. Existen dos tipos distintos, “TS Full SI” o “TS Optional SI”. El primero, “TS Full SI”, trata de enviar la información mínima para que el cliente sea capaz de localizar cada uno de los servicios anunciados, este tipo se emplea en los casos en los que se envía la información del SI dentro del Transport Stream. El segundo, “TS Optional SI”, envía la misma información que “TS Full SI” y, además, contiene varios campos opcionales, este segundo tipo se

emplea cuando no se esta enviado información del SI en el flujo Transport Stream.

- CoD Discovery Information, este tipo de informe provee toda la información necesaria para descubrir los servidores CoD que nos ofrece el Proveedor de Servicios seleccionado, así como la localización de su catálogo de contenidos. El modelo de documento está definido para poder publicar servicios CoD de otros SP con los cuales tengan un acuerdo.

- Services from other SPs, como su propio nombre indica este informe hace referencia a los servicios ofrecidos por otro SP. Ya sea un sólo servicio como la oferta completa.

- Package Discovery Information, es utilizado por los proveedores de servicios que desea agrupar varios servicios y presentarlos como una sola entidad

- BCG Discovery Information, proporciona un medio para descubrir los lugares de la lista de las guías de contenido, ya sean Live media o CoD.

2.3.2.3. Fragmentación de los documentos SD&S

Los documentos XML del SD&S pueden ser de un tamaño considerable, por esto se ha visto necesario implementar la posibilidad de fragmentación de dichos informes en segmentos.

Cada uno de estos segmentos viene identificado por un Segment ID y un valor de 8 bits para la versión de dicho segmento. Cada vez que hay alguna modificación en el segmento, se cambia la versión del mismo, facilitando así la tarea de comprobación de actualizaciones de los segmentos al HNED.

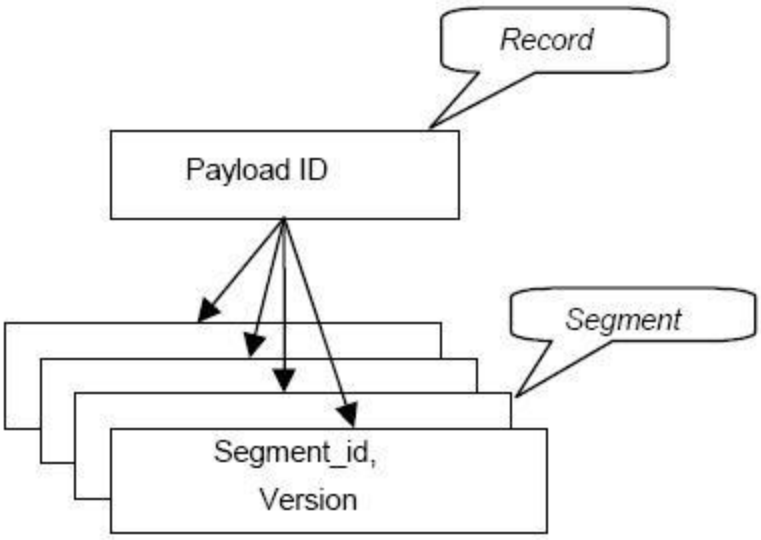

Fig. 2.5 Relación entre Record, Payload ID y segmento. Extraída de [1]

La información que contienen estos segmentos tan solo podrá pertenecer a uno de los tipos de documento SD&D descritos en el apartado anterior 2.1.2.1, vendrá identificada por un Payload ID especifico para cada tipo contenido en el segmento. En el anexo II podemos ver una tabla con los Payload ID que pueden tener estos segmentos.

Cada uno de los segmentos deberá contener un documento XML valido y *well-formed.*

2.3.2.4. Tiempo máximo de ciclo

El tiempo necesario para transmitir todos los segmentos que componen la serie completa del documento SD&S se llama el tiempo de ciclo. El tiempo máximo del ciclo será de 30 segundos.

2.3.2.5. Pasos a seguir en el Service Discovery

En el proceso de descubrimiento de servicios se definen varios pasos para conseguir obtener los servicios disponibles por cada Proveedor de Servicio, el objetivo es automatizar el descubrimiento de los servicios en los Proveedores de Servicio disponibles.

El proceso de descubrimiento de servicio se inicia determinando los puntos de entrada (Entry Points). Esto se especifica en el apartado siguiente 2.1.2.6.

Una vez obtenidos los Proveedores de Servicio disponibles, se obtiene la información sobre los servicios ofrecidos por cada uno de ellos.

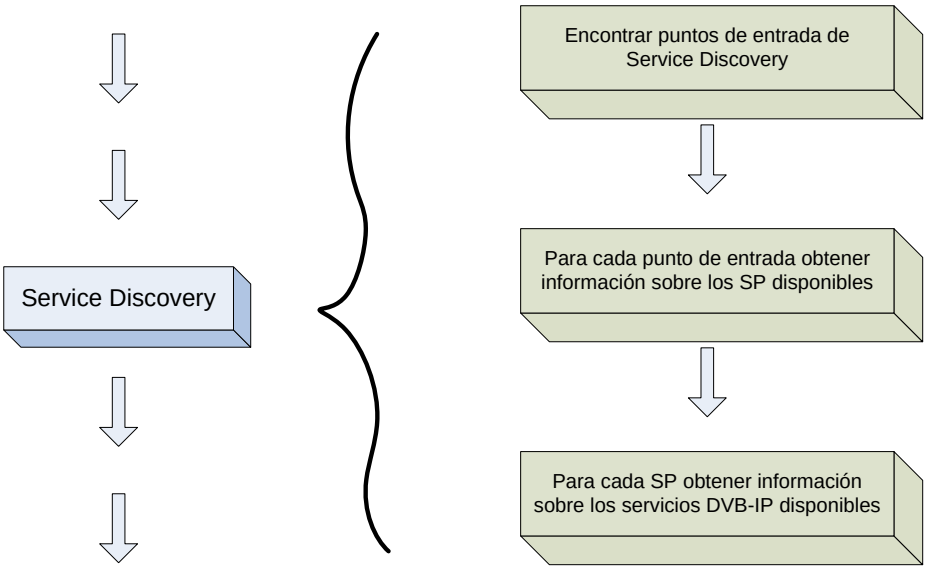

Fig. 2.6 Pasos a seguir en el descubrimiento de servicios.

2.3.2.6. Puntos de entrada (Entry Points)

El primer paso es encontrar los puntos de entrada SD&S, donde obtendremos la información sobre los distintos SP disponibles. El proceso de de localización de punto de entrada se realiza de forma priorizada, empezando en la primera de las opciones disponibles que existe y saltando a la siguiente en el caso de no obtener éxito en el punto de entrada actual.

A continuación se muestran los posibles puntos de entrada que pueden haber, estos puntos de entrada son:

- En el momento de adquirir su propia dirección IP el HNED mediante DHSCP. Puede ser proporcionado el nombre del dominio a través de la opción 15 de DHCP. Una lista de puntos de entrada SD&S se adquieren entonces vía DNS de acuerdo con el RFC 2782 [8]. El nombre del servicio es _dvbservdsc mientras el resto del nombre es el nombre del dominio proporcionado mediante DHCP. Por ejemplo, _dvbservdsc._tcp.midominio.com. En el caso de no haber obtenido ningún nombre de dominio a través de DHCP, se pasara al siguiente punto de entrada.

- Una dirección multicast registrada por IANA para tal propósito (DvbServDisc), con la dirección IP 224.0.23.14. En el caso de no recibir ningún paquete DVBSTP desde esta dirección IP multitas, se pasara al siguiente punto de entrada.

- Mediante DNS se puede adquirir una lista de puntos de entrada SD&S según el servicio de localización RFC 2782. El nombre del servicio es

_dvbservdsc, el protocolo puede ser TCP o UDP, mientras que el resto del nombre es el nombre de dominio mantenido por DVB para el servicio de descubrimiento, este nombre de dominio está establecido en services.dvb.org. Los nombres de servicio son _dvbservdsc._tcp.services.dvb.org o bien _dvbservdsc._udp.services.dvb.org. Esto requiere que el HNED soporte DNS SRV de acuerdo a la especificación en el RFC 2782 [8]. La organización DVB mantendrá el nombre de dominio para services.dvb.org y los nuevos proveedores de servicios deberán registrarse en DVB para añadirlos a la lista de DNS SRV. Estos servicios será publicados mediante HTTP para el protocolo TCP, mientras que para el servicio multicast se publican a través del protocolo UDP. En el caso de no obtener ninguna respuesta ante la consulta DNS, se pasara al siguiente punto de entrada.

- Si no se ha encontrado ningún punto de entrada a través de los pasos anteriores, el administrador del HNED podrá introducir directamente los puntos de entrada, la dirección IP, como parte de los datos de la configuración del terminal.

Si no se especifica un puerto, el puerto por defecto será 3937 (dvbservdscport), asignado por IANA.

2.3.2.7. Service Provider discovery Record

Una vez obtenido el punto de entrada por alguno de los métodos descrito en el apartado anterior 2.1.2.5, obtendremos un documento XML con la información sobre los SP disponibles. Esta entrada (Service Provider Discovery Record) mostrara un listado de los Proveedores de Servicio disponibles, así como la información necesaria para poder obtener los servicios DVB-IP disponible en cada uno de ellos. Esta entrada contendrá la información indicada en la tabla del anexo II.

2.3.2.8. DVB-IPTV Offering Record

Una vez decidido y seleccionado el Service Provider que deseamos, obtendremos del mismo el DVB-IPTV Offering Record, que deberá contener al menos los campos definidos en la tabla del anexo II, seguido de los campos relacionados con la oferta real del SP que vendrán indicados en el Broadcast discovery Record.

Un proveedor de servicios puede ofrecer dos tipos de servicios bien diferenciados, los Livemedia ("TS Full SI" o "TS SI Optional ") y/o Contenido bajo demanda (a través de la BCG Discovery Record).

2.3.2.9. Broadcast discovery Record

El “TS Full SI” Broadcast Discovery deriva de DVB-IP Offering Record, provee toda la información necesaria para encontrar los servicios Live Media disponibles que llevan integrados la información del SI (Service Information) en el Transport Stream. Este registro deberá incluir todos los atributos obligatorios que se indican en el Handbook [1] de los cuales en la tabla del anexo II hacemos una mención de los más relevantes para nuestro proyecto, ya que la tabla completa es de un tamaño considerable. Esta entrada también deberá contener los campos DVB-IP Offering a modo de cabecera.

"TS Optional SI" Broadcast Discovery Information implementa la misma información que el "TS full SI", además de los servicios de Service Description Location del SI. Básicamente contiene la misma información que el “TS FULL SI” pero con alguna información más sobre el contenido que se está enviando. Se emplea cuando no se está enviando la información del SI en el flujo Transport Stream.

2.3.2.10. Package discovery Record

El Package Discovery Record proporciona la posibilidad de agrupar los servicios a ser comercializados o aunar los servicios bajo una misma entidad. También nos proporciona el Service ID y el Description Location tal y como se aprecia en la tabla del anexo II

Otra de las funcionalidades de este servicio es la asignación de números de canales (Logical Channel Number) asociados a cada servicio DVB-IP que se ofrece.

2.3.3. Service Selection

Se definen dos métodos bien diferenciados para el acceso a los servicios de stream desde el HNED, estos son los siguientes:

- RTSP, para modo unicast
- IGMP, para modo multicast

Los servicios Live Media son transmitidos sobre IP multicast de forma continua, no es necesita ser iniciado por ningún HNED. Los dispositivos finales pueden solicitar el acceso o abandonar el acceso a un flujo de datos concreto mediante IGMP. El campo Service Location del paquete Broadcast Record proporciona la información necesaria para enviar los mensajes IGMP para cada servicio anunciado.

Opcionalmente los servicios de distribución de vídeo en tiempo real pueden elegir requerir al HNED que ejecute una serie de pasos, como activar una cuenta, acceso condicional (acceso solo a los canales que tenga el usuario

contratado), etc. En estos sistemas se usará RTSP que además servirá para adquirir parámetros requeridos por IGMP, por ejemplo, para el acceso a canales de pago que estén siendo retransmitidos en multicast, usaremos RTSP para obtener la información necesaria en el uso de IGMP y la visualización de los canales multicast.

Los servicios Live Media con el Trick Mode son similares a los Live Media pero proporcionan la posibilidad al usuario de control (avance, stop, etc.) y serán entregados mediante IP unicast usando RTSP.

2.4. Mecanismos de transporte

Se definen dos mecanismos para transmisión y envío de los documentos XML, uno para multicast y otro unicast.

- En el modo multicast se define un protocolo de transporte (DVBSTP) el cual se transporta a través de conexiones UDP. Este protocolo es el único en todo el estándar DVB-IP que ha sido creado, el resto de protocolos empleados en DVB-IP son estándares ya definidos. DVBSTP se explicara en el siguiente apartado. 1.5.1.

- Para el caso de unicast, se hará uso del protocolo HTTP, el cual es estándar y se usa para la navegación Web, para la transmisión de los documentos XML. A diferencia del protocolo DVBSTP que va sobre UDP, HTTP se trasporta sobre conexiones TCP.

Ambos métodos transportaran la misma información en los documentos XML, siendo la diferencia el método de transporte de los mismos.

2.4.1. DVB SD&S Transport Protocol (DVBSTP)

Cuando la información de descubrimiento de servicio es transmitida en modo multicast mediante paquetes UDP, se emplea el protocolo DVBSTP definido en este apartado. Los datos de la cabecera de este protocolo son los que se pueden ver a continuación en la tabla 2.2.

Tabla 2.2. Cabecera DVBSTP

<table>
<tr><td colspan="8">0</td><td colspan="8">1</td><td colspan="8">2</td><td colspan="8">3</td><td colspan="8">4</td><td colspan="8">5</td><td colspan="8">6</td><td colspan="8">7</td><td colspan="8">8</td><td colspan="8">9</td><td colspan="8">0</td><td colspan="8">1</td></tr>
<tr><td colspan="4">VER</td><td colspan="8">Resrv</td><td colspan="6">Enc</td><td colspan="2">C</td><td colspan="48">Total Segment Size</td></tr>
<tr><td colspan="16">Payload ID</td><td colspan="32">Segment ID</td><td colspan="16">Segment version</td></tr>
<tr><td colspan="24">Section Number</td><td colspan="32">Last Section Number</td><td colspan="12">Compr</td><td colspan="2">P</td><td colspan="14">HRD_LEN</td></tr>
<tr><td colspan="80">(Conditional) SP ID</td></tr>
<tr><td colspan="80">(Optional) Private Header Data</td></tr>
<tr><td colspan="80">Payload</td></tr>
<tr><td colspan="64">Payload</td><td colspan="16">(Optional) CRC</td></tr>
<tr><td colspan="64">(Optional) CRC (cont)</td></tr>
</table>

A continuación vemos una descripción sobre los campos encontrados en la cabecera de los paquetes DVBSTP.

- **Versión del Protocolo (Ver):** Versión del protocolo. Este campo de 2 bits valdrá “00”.
- **Reservado (Resrv):** Estos 3 bits están reservados y tendrán el valor “000”.
- **Encriptación (Enc):** Teste campo de 2 bits se usará para señalar la presencia de encriptación. Valdrá “00” para indicar que el payload no está encriptado.
- **Flag CRC (C):** Si el valor es “1”, esto indica la presencia de un CRC de 32bits al final del paquete. Este flag puede solo estar presente al final del paquete en un segmento, por ejemplo cuando section_number == last_section_number.
- **Tamaño Total del Segmento**: Campo de 24 bits que especifica el tamaño en bytes. Para datos no comprimidos (comprimidos es “000”) es el tamaño acumulado de la suma de todos los payloads de todas las secciones que componen el segmento (ignorando cabeceras y CRC). Para datos comprimidos usables en su forma comprimida, es el tamaño acumulado de todo el payload de todas las secciones comprimidas. Para datos comprimidos usables sólo en su forma descomprimida es el tamaño del segmento descomprimido por el algoritmo específico.
- **ID de Payload:** Un campo de 8 bits usado para identificar el tipo de datos cargados en el payload.
- **ID de Segmento:** Un campo de 16 bits identifica que segmento de datos es del payload.
- **Versión de Segmento:** Un valor de 8 bits usado para definir la versión actual del segmento transportado, se incrementará cada vez que se modifique y es módulo 256.
- **Número de Sección:** 12 bits que identifican el número de la sección. La primera sección en un segmento será 0.
- **Último Número de Sección:** 12 bits que especifican el último número de sección.
- **Compresión (Compr):** 3 bits que indican el esquema de compresión. Todos los segmentos de un payload deben llevar el mismo esquema de compresión. Los posibles valores están en la tabla 2.3.

Tabla 2.3. Valores de compresión

Compression	value Meaning	Total Segment Size Meaning
0	No Compression	Transmitted Size
1	BiM	Transmitted Size
10	GZIP	Transmitted Size
011 to 110	Reserved	
111	User Private	User Defined

- **ProviderID Flag (P):** 1bit. Flag que indica si el campo ServiceProviderID está presente. El valor “1” define la presencia del campo en la cabecera.
- **Private Header Length (HDR_LEN):** Campo de 4 bits que cuenta el número de palabras de 32 bits en la cabecera que siguen el campo de longitud de cabecera, o el campo Provider ID si está presente. Se usa para señalar la presencia de una cabecera privada, si no la hay el valor será “0000”.
- **Service Provider ID:** Un número de 32 bits que se usa para identificar al proveedor. Será una dirección Ipv4.
- **Datos de Cabecera Privada:** Son datos privados. Este campo será múltiplo de 4 bytes.
- **Payload:** El contenido del paquete, que es un número integral de bytes. El tamaño del contenido puede calcularse como la resta del tamaño del paquete menos el tamaño de la cabecera y el CRC (si está presente).
- **CRC:** Campo opcional de 32 bits.

2.4.2. Uso de secciones en DVBSTP

El tamaño de los segmentos, formados en la fragmentación de los informes, puede ser substancialmente mayor que el soportado por la red. Para permitir la entrega de estos datos, es necesario poder dividir los segmentos en unidades más pequeñas para su entrega. Para ello disponemos de este mecanismo que nos proporciona esta funcionalidad, tal como se puede ver en la figura 2.7.

Cada sección se enviara en un datagrama UDP. Estos datagramas UDP tan solo podrán contener información sobre una sección, no pudiendo introducir en un mismo datagrama UDP información sobre dos secciones diferentes.

La cantidad teórica de datos que puede ser encapsulada en cada paquete UDP está limitada por el tamaño máximo del datagrama IP (65535 bytes) menos las cabeceras UDP y RTP. Pero esto seguramente superaría la MTU (Maximum Transmisión Unit) de la red y obligaría a los routers de red a realizar fragmentación, aumentando la carga de trabajo a dichos routers. Para evitar esto, el tamaño real máximo que se emplea es el de la MTU del nivel de enlace que tengamos. Por ejemplo, si tenemos una trama Ethernet de 1500 bytes al quitarle las cabeceras (20 bytes de IP, 8 de UDP y 12 de RTP) tendremos un tamaño máximo de datos a introducir de 1460 bytes.

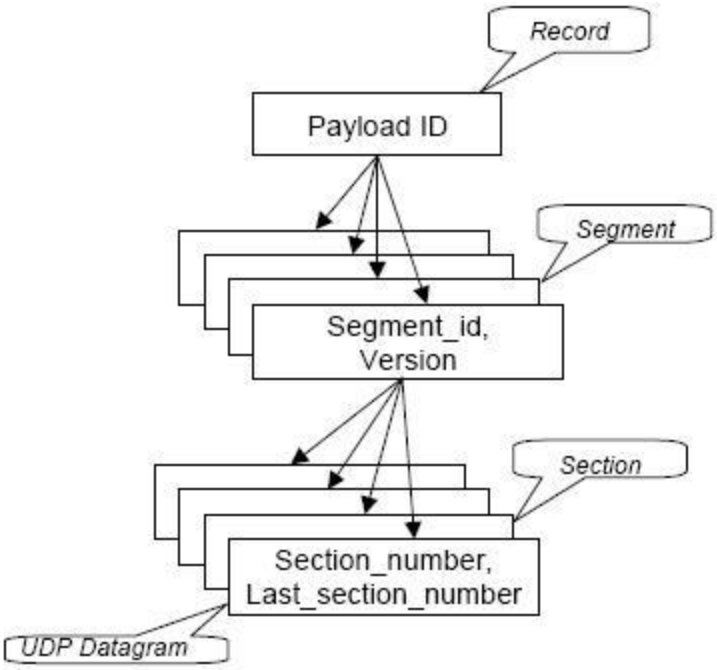

Fig. 2.7 Relación entre Record, segmento y sección. Extraída de [1].

Si el dispositivo final no soporta fragmentación, el Service Provider deberá marcar el flag “Do not Fragment” de IP, con el riesgo de que hayan routers que no los dejen pasar y devuelva un mensaje ICMP "fragmentation needed and DF set". En este caso el Service Provider deberá ajustar el tamaño del contenido al indicado en el RFC 791 [9], en el cual se indica que los hosts han de admitir como mínimo paquetes de 576 bytes.

CAPITULO 3. DISEÑO Y DESARROLLO

3.1. Introducción

El objetivo global a medio plazo es el de realizar un cliente HNED que cumpla con el perfil LMB de DVB-IP.

Los objetivos del proyecto eran el desarrollo de:

- Integración de un servidor de videos multicast, realizado con VideoLan Client (VLC) [26], aplicación la cual nos permite realizar el envío de streamings de video en MPEG-TS.
- El servidor de documentos XML encapsulando en DVBSTP, desarrollado en código C.
- Integración de un router con capacidad de enrutamiento multicast, implementado en un PC con dos tarjetas de red y el software XORP [27] en un sistema operativo Linux.
- Finalmente nuestro cliente DVB realizado en código C. Diseñándose los módulos Capturador y ParserCompleto para finalmente crear el ClienteCompleto.

3.2. Escenario

Tal y como se puede ver en la Figura 2.1, se ha integrado el servidor de video y se ha implementado el servidor de metadatos XML (el conjunto de ambos lo llamaremos servidor DVB-IP como se ve en la figura 2.1) en un extremos del router XORP y al otro extremo se ha puesto el cliente DVB-IP, al cual se le asignará dirección IP mediante DHCP.

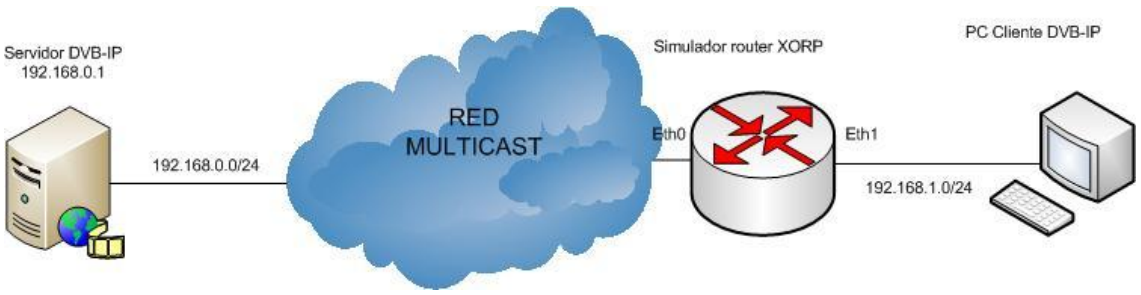

Fig. 2.1 Escenario implementado

3.2.1. Servidor de metadatos XML con encapsulado DVBSTP

Para el envío de los documentos XML hemos implementado el serverxml2, el cual lee del disco duro los tres documentos necesarios, que previamente

hemos generado manualmente, ServiceDiscovery.xml, BroadcastRecord.xml y PackageRecord.xml, los encapsula en DVBSTP y los retransmite por la dirección IP multicast correspondiente. En la figura 2.2 podemos ver un diagrama de bloques del funcionamiento del mismo.

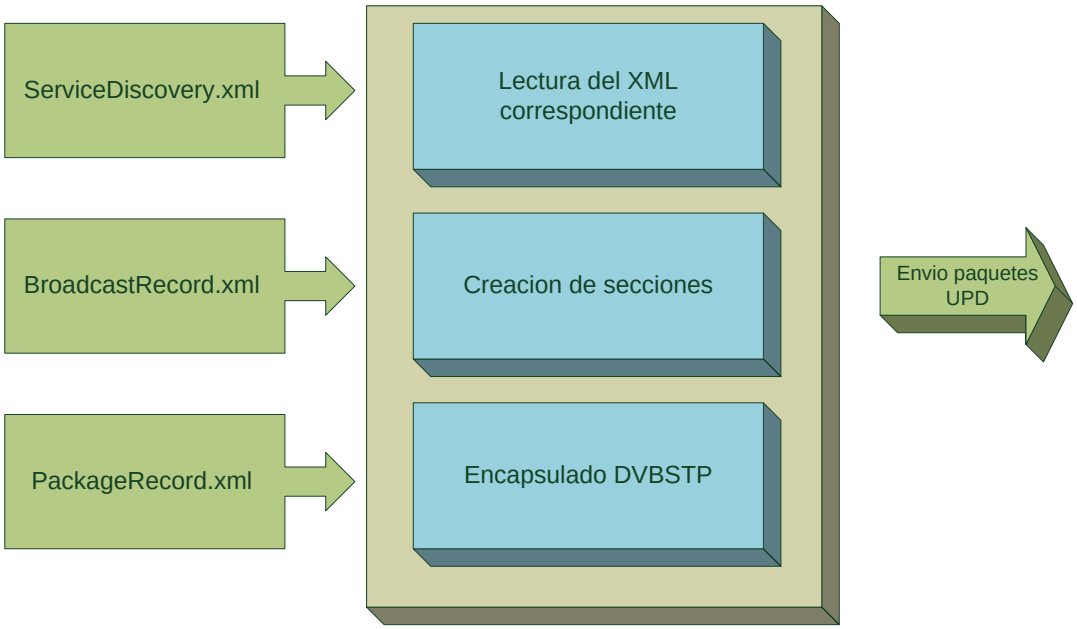

Fig. 2.2 Diagrama de bloques del Serverxml2.

El funcionamiento de este servidor, Consiste en leer el documento en cuestión, ver el tamaño de dicho documento, realizar el cálculo de las secciones necesarias a enviar, rellenar los campos DVBSTP de las cabeceras e ir retransmitiendo cada paquete, actualizando los campos necesarios (Section Number, PayloadID, etc.) en cada uno de los paquetes UDP que retransmite.

Se ha contemplado dos formas diferenciadas para el envío:

- El envío de los documentos de una forma controlada, pudiendo seleccionar cual de los XML deseamos enviar, la dirección IP multicast hacia la que enviar y el número de veces a enviar.

- El envío de forma continua de todos los XML.

3.2.2. Servidor de streaming de video VLC

El envío de los videos se realiza mediante el software VLC. Para poder realizar el envío de varios flujos de datos, se ha hecho uso del interface Telnet que dicho software tiene implementado, el cual nos ofrece la posibilidad de realizar el envío de varios flujos con una única aplicación abierta.

Los videos son enviados con encapsulación MPEG2-TS tal y como indica el Standard. Se ha optado por el envío directamente sobre UDP, puesto que el

envío en RTP/UDP no aportaba ninguna funcionalidad tal y como es nuestra implementación del proyecto, el cual esta bajo un entorno controlado, sin la probabilidad de que se pierdan o desordenen paquetes al tratarse de una red Ethernet. Usando como códec de videos el H.264 Main profile Level 3 para obtener una buena relación entre ancho de banda necesario y calidad de video, a una tasa de unos 2Mbps. Este códec está contemplado por el estándar DVB [5] tal y como podemos ver en el anexo II. Actualmente la mayoría de los proveedores realizan la transmisión de video sobre redes IP con esta codificación de video, como por ejemplo en el caso de Jazztelia [28] e Imagenio [29].

3.2.3. Router multicast XORP.

Para el enrutado multicast, se ha usado el software XORP [27] instalado sobre el sistema operativo Linux. Se ha recurrido a este software ya que nos permite realizar funciones de enrutado multicast, el empleo de IGMP para el control de los usuarios y PIM-SM para el encaminamiento de flujos de datos y videos multicast.

Para las funciones de servidor DHCP se ha instalado el paquete dhcpd3 en el sistema operativo Linux, así cumplir con uno de los puntos de autoconfiguración del estándar.

Los detalles de la configuración del router, así como del servidor DHCP la podemos ver en el Anexo III.

3.2.4. Visualización de videos con VLC.

Al igual que para el envío de videos, para la visualización de los mismos se ha hecho uso de la aplicación VLC. En la cual cargaremos una Lista de Reproducción donde podremos ver todos los canales disponibles y ordenados, pudiendo seleccionar el canal que se desea visualizar.

3.2.5. Cliente DVB-IP.

En el cliente de DVD-IP es donde más nos hemos centrado ya que era el centro del proyecto. Para la realización de todas las funciones necesarias se han diseñado dos módulos independientes, el Capturador y el ParserCompleto, los cuales finalmente se han modificado y unido en una única aplicación.

Capturador: Este modulo tiene la funcionalidad de capturar los paquetes UDP multicast que transportan información SD&S sobre DVBSTP, y validar los documentos XML generados. En la figura 2.3 podemos ver su funcionalidad en un diagrama de bloques.

En una primera instancia tendremos que seleccionar la dirección IP para el Entry Point, ofreciendo al usuario el uso de la IP registrada por la IANA para tal fin, de esta forma implementamos dos de las cuatro opciones contempladas en

el Standard para el Entry Point. Una vez seleccionada la IP multicast, se realiza una solicitud IGMP para añadirse al grupo multicast de dicha IP.

Seguidamente se genera el documento XML con los paquetes UDP capturados, una vez han sido desencapsulados de la cabecera DVBSTP. También tenemos la opción de validar y verificar que este bien formado el documento generado que se realizara mediante la aplicación xmlstarlet [24] que hemos integrado en nuestro cliente.

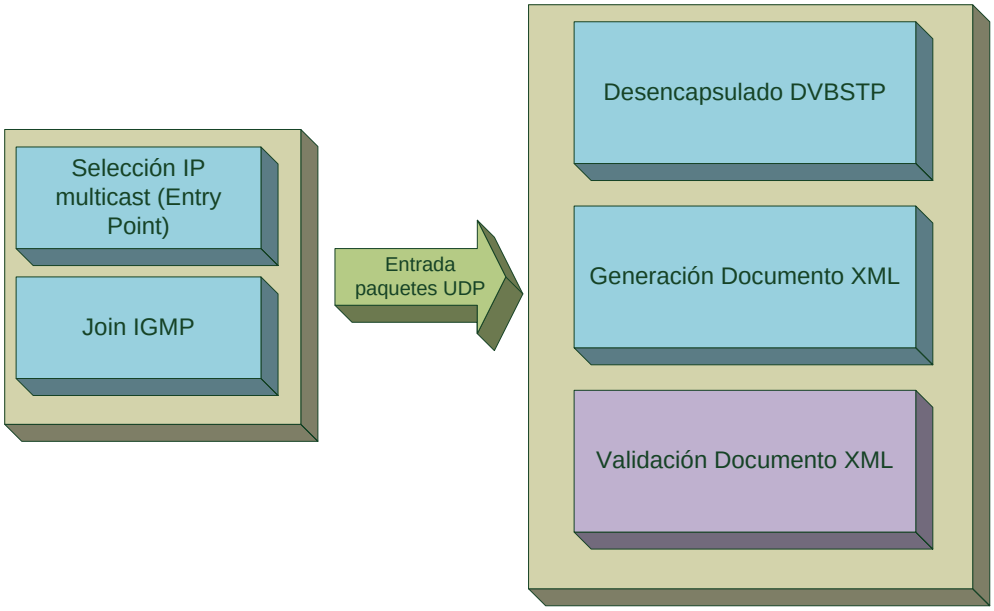

Fig. 2.3 Diagrama de bloques del Capturador.

ParserCompleto: Con este modulo podemos realizar el parseo[1] de dicho documento, realizado con la API Expat [23], la cual también se ha integrado en el cliente, para obtener la información relevante del documento XML. Se realiza una extracción de datos de forma diferente para cada uno de los tres XML que emplearemos en el proyecto, ya que la distribución de la información en cada uno de ellos es diferente. Además la información que deseamos de cada uno de ellos es completamente distinta. En la figura 2.4 podemos apreciar el diagrama de bloques funcional de este modulo.

La extracción de la información del documento XML la realizaremos en función de las etiquetas, elementos y atributos, la descripción de las cuales vimos en el apartado 1.4.

Mediante la API Expat [23] iremos analizando el documento XML, ésta se detiene cuando encuentra una etiqueta nueva y en función de si es un elemento o un atributo lo que deseamos extraer, haremos un tipo de llamada interna de esta misma API para extraer la información que deseamos.

[1] Decodificación y análisis de los documentos XML.

Cliente completo: Teniendo los dos módulos anteriores, tan solo quedaría crear el cliente final, que hiciera llamadas a cada uno de los módulos en el momento que fuera necesario. Se han modificado los módulos para poder usarlas como funciones dentro del propio cliente. Podemos ver su diagrama de bloques en la figura 2.5.

Finalmente, desde el cliente generamos una lista de reproducción con todos los canales disponibles, ejecutamos el VLC con esta lista y podremos visualizar los videos, además de poder seleccionar qué canal deseamos visualizar en cada momento.

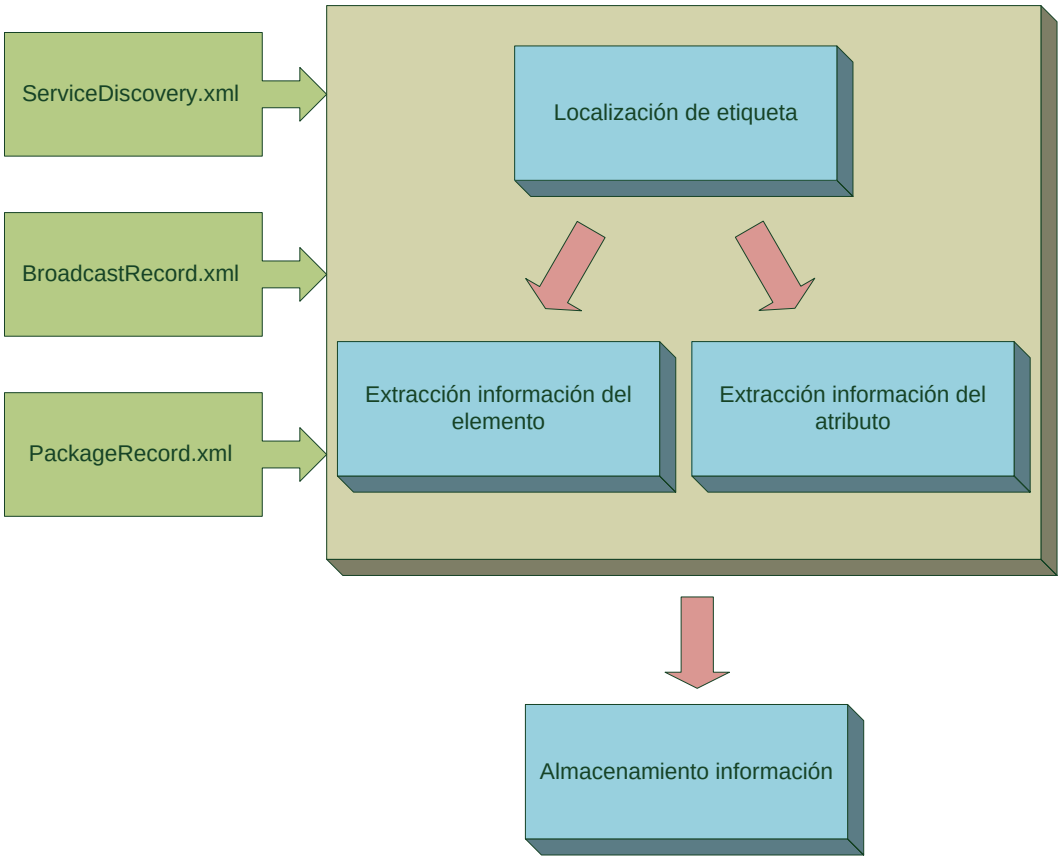

Fig. 2.4 Diagrama de bloques del ParserCompleto.

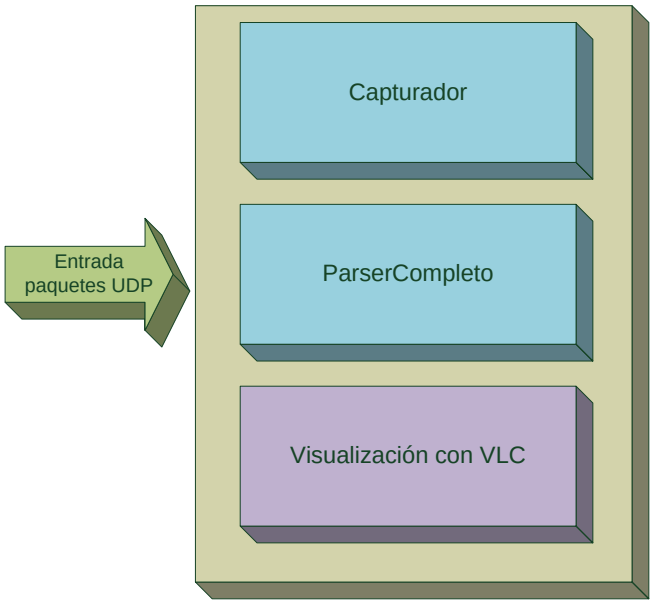

Fig. 2.5 Diagrama de bloques del ClienteCompleto.

3.3. Software empleado.

Para la implementación del proyecto ha sido necesario el uso de software ya desarrollado como herramientas para su diseño, así como el uso de API para integrar dentro de nuestro propio código.

Cabe mencionar que el sistema operativo con el cual hemos trabajado ha sido básicamente Windows XP home edition, pero aun así se ha diseñado el proyecto para que pueda ser implementado tanto en SO Windows como en Linux/Unix, habiendo sido testeado en Ubuntu 8.0.4.

3.3.1. Software para el desarrollo y verificación.

3.3.1.1. Wireshark [19]

Aplicación usada para la captura de paquetes y su posterior análisis. La aplicación permite verificar que se está realizando correctamente el envío de paquetes, verificar el protocolo implementado y un análisis a bajo nivel de cada byte enviado por la red. Este tipo de software también es conocido como software “sniffer”.

3.3.1.2. Notepad++ [20]

Es un editor de textos el cual tiene integrado un intérprete de comandos para varios lenguajes, entre ellos C. De esta forma nos era más cómoda la programación del código, así como una correcta exportación del mismo para implementar en los anexos.

3.3.1.3. Cygwin [21]

Cygwin es una colección de herramientas desarrollada por Cygnus Solutions para proporcionar un comportamiento similar a los sistemas Linux/Unix en Windows. De esta forma podemos desarrollar la aplicación en Windows y ser compatible también con la plataforma Linux/Unix.

3.3.1.4. Bloodshed Dev-C++ [22]

Bloodshed Dev-C++ es un entorno de desarrollo integrado para programar en lenguaje C/C++. Utilizado de similar forma que notepad++, para una programación más cómoda y visualización mejor del código C.

3.3.2. Software integrado en el proyecto.

Cabe hacer mención también del software que hemos conseguido integrar con el código del proyecto. Así como una mención hacia algún software sobre el cual nos hemos basado para la realización de nuestro código.

3.3.2.1. Expat [23]

Es una biblioteca de procesado XML. La cual hemos integrado completamente en nuestra aplicación, con la que podemos realizar el parseo de los documentos XML y obtener la información relevante en cada uno de los documentos XML.

3.3.2.2. XmlStarlet [24]

Es una utilidad que nos permite verificar si los documentos XML son validos y bien formados. También hemos integrado una llamada a esta aplicación dentro de nuestro código del proyecto, obteniendo el resultado por pantalla tanto si es un documento válido como si no lo es.

3.3.2.3. Infocast2Tools [25]

No es una aplicación como tal que hayamos empleado, pero sí que hemos usado parte de su código para las conexiones multicast y ha sugerido ideas para el desarrollo del proyecto. Por ese motivo creo que es digno de mención.

CAPITULO 4. PRUEBAS Y VERIFICACIÓN DE FUNCIONAMIENTO

4.1. Introducción

Este capítulo ilustra el correcto funcionamiento del proyecto implementado, tanto en el envío como en la recepción de los documentos XML y Videos. Se han ido realizando por partes los envíos y se acompañan de correspondientes capturas con Wireshark para verificar los resultados obtenidos.

4.2. Envío documentos XML

Los documentos XML han sido generados a mano, siguiendo como muestras, algunos de los ejemplos que se pueden encontrar en el documento Guidelines de DVB-IP [2].

Para el envío de los documentos XML se ha diseñado la aplicación serverxml2, la cual esta implementada en lenguaje C. Su ejecución es sencilla, basta con ejecutarla desde una ventana de MS-DOS o desde el propio Windows.

4.2.1. Prueba envío correcto del SD&S

Vamos a verificar el envío del ServiceDiscovery.xml usando nuestro servidor. En primer lugar ejecutamos la aplicación serverxml2, seleccionamos la primera opción 1, dejamos la dirección IP que nos da por defecto que es la registrada por la IANA para los servicios de Entry Point de DVB-IP (224.0.23.14) y por ultimo le indicamos que nos lo envíe una sola vez, tal y como podemos ver en la figura 4.1.

La información que nos muestra por pantalla el serverxml2 es la dirección IP multicast hacia la que envía el flujo de datos, el fichero XML que está enviando, el tamaño del XML, el tamaño de las secciones a enviar, el total de paquetes UDP para enviar todas las secciones, el tamaño del segmento en Hexadecimal para facilitar su visualizado mediante el sniffer, el numero de la última sección y el tamaño de la última sección enviada, ya que será menor que las anteriores.

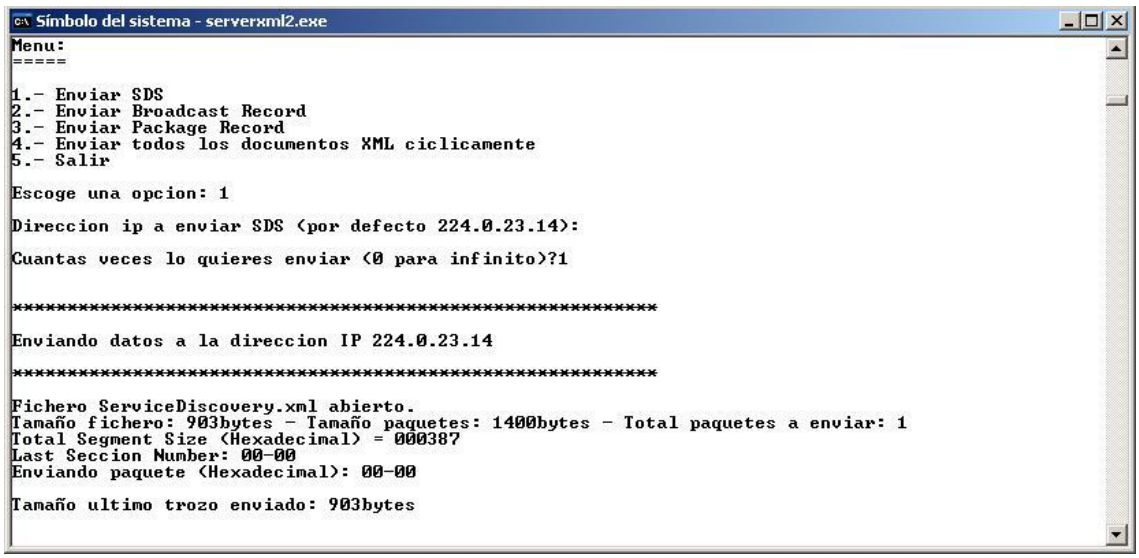

Fig. 4.1 Envío del SD&S mediante serverxml2.exe

4.2.2. Verificación envío del SD&S

Mientras realizamos el envío del ServiceDiscovery.xml, mediante Wireshark capturamos los paquetes para poder visualizar si el encapsulado DVBSTP es correcto como vemos en la figura 4.2.

De la captura podemos ver que se está enviado el documento XML hacia la dirección IP multicast correcta (224.0.23.14), el puerto de destino es correcto (3937) y podemos ver como dentro del paquete UDP tenemos el XML con la cabecera DVBSTP.

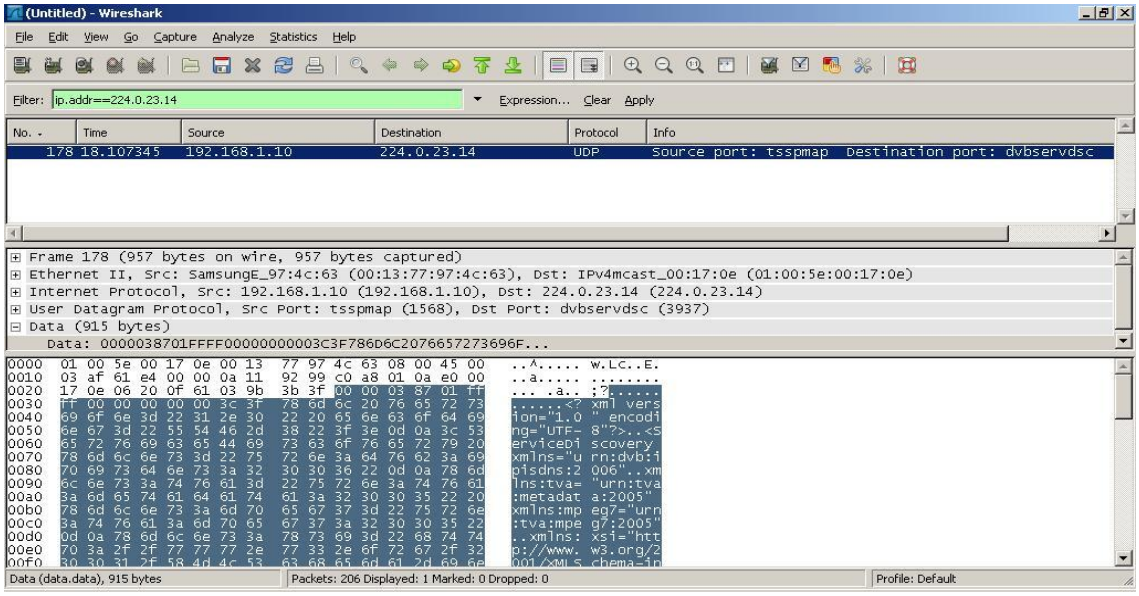

Fig. 4.2 Captura envío del SD&S mediante serverxml2.exe

La cabecera DVBSTP de 12 bytes que podemos observar en captura representada de forma hexadecimal podemos obtener los datos que se reflejan en la tabla 4.1.

Tabla 4.1. Cabecera DVBSTP capturada.

1	2	3	4	5	6	7	8	9	10	11	12
00	00	03	87	01	FF	FF	00	00	00	00	00

- Byte 1: Obtenemos la información referente a Versión (2 bits), Reservado (3 bits), Encriptación (2 bits) y Flag CRC (1 bit). Todos han de estar a 0 y así es.
- Bytes 2, 3 y 4: Indican el tamaño total del Segmento en hexadecimal, tal y como habíamos visto en el server el tamaño es 387, así que este campo también es correcto.
- Byte 5: Payload ID, según vimos en la tabla 1.1., para el SD&S el payload ID ha de ser 0x1.
- Bytes 6 y 7: Segment ID, hemos puesto el identificador FFFF para una localización visual más rápida.
- Byte 8: Versión del Segmento, hemos puesto que es la versión 0.
- Bytes 9 y 10: Numero de sección, el byte 9 y los 4 bits más significativos del byte 10 (12 bits en total) indican el numero de segmento de este paquete UDP. En este caso 0.
- Bytes 10 y 11: Ultimo numero de sección, indica cual es el ultimo numero de sección, en este caso 0 ya que tan solo habrá una sección.
- Byte 12: Compresión (3 bits), Flag de provider ID (1 bit), tamaño cabecera privada (4 bits). Todos han de ser 0, ya que no hay compresión ni vamos a enviar datos de cabecera privados.

4.2.3. Verificación del envío todos los segmentos XML

Una vez verificado que el serverxml2 realiza bien su función de envío de documentos XML y una correcta encapsulación DVBSTP, procedemos al envío de todos los documentos XML necesarios para el funcionamiento de DVB-IP.

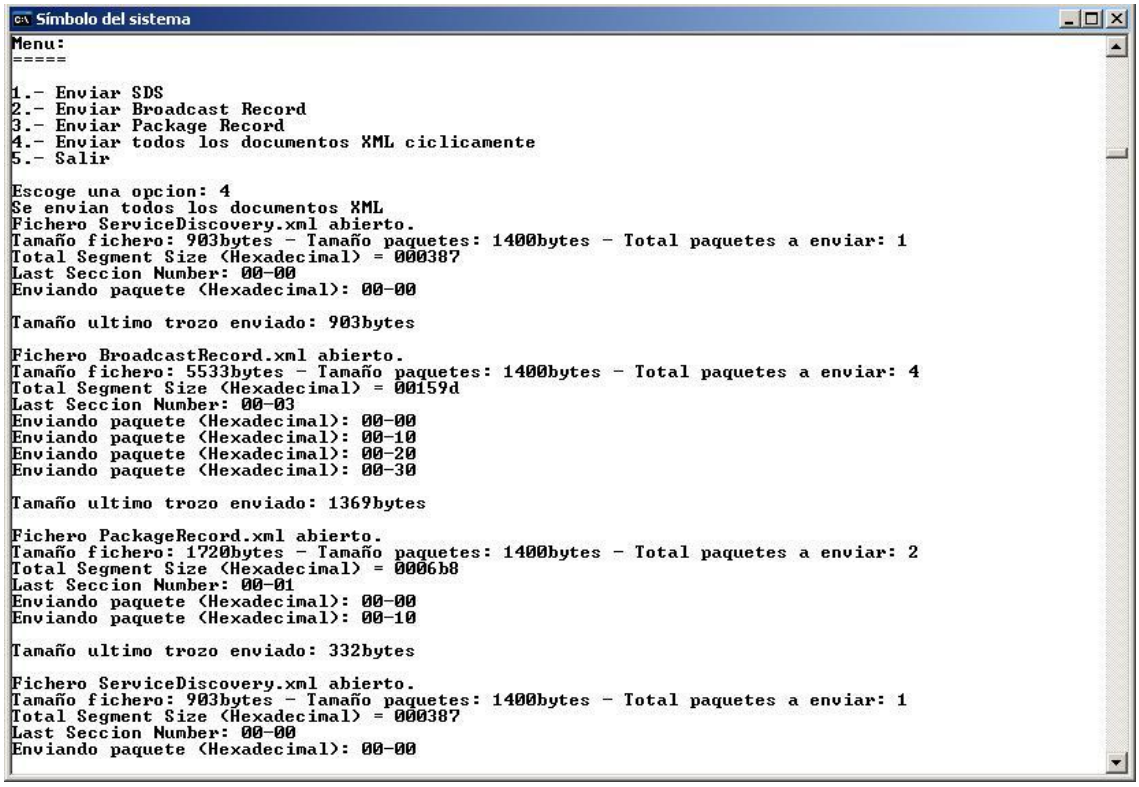

Fig. 4.3 Envío de todos los documentos XML cíclicamente.

Como podemos apreciar en la Figura 4.3, seleccionando la opción 4, vemos como se van enviando los tres documentos XML necesarios, ServiceDiscovery.xml, BroadcastRecord.xml y PackageRecord.xml, de forma cíclica.

Para poder cumplir con el tiempo de ciclo definido en el estándar y también para no saturar la red, se ha definido una variable “TIME_WAIT” expresada en milisegundos, dentro del código del serverxml2. Esta variable controla el tiempo de espera entre el envío de cada paquete UDP.

4.3. Envío de los flujos de video mediante VLC

Como se ha mencionado, el envío de los flujos de video se realizara mediante la aplicación VLC, ejecutando su interface telnet, la cual nos da la opción de enviar varios flujos con una sola sentencia de VLC.

Para poder ejecutar el interface telnet lo podemos realizar desde una ventana de MS-DOS ejecutando el comando:

C:\>vlc -I telnet

Una vez ejecutamos el interface telnet, debemos de conectarnos precisamente mediante un telnet a la dirección IP local (podemos usar también la IP de

loopback 127.0.0.1) por el puerto 4212, conectándonos a la “Shell” que tiene VLC para realizar los controles necesarios para los envíos.

C:\>telnet 127.0.0.1 4212

La contraseña por defecto es “admin”. Una vez estemos conectados mediante telnet, debemos de ejecutar los comandos necesarios para la creación de los flujos de video que deseamos, estos comandos los adjuntaremos al final del proyecto en el anexo III.

Cabe mencionar que el origen de estos videos, pueden ser videos que tengamos almacenados en local, o bien retransmisiones de alguna tarjeta de TV que tengamos en el PC (DVB-C, DVB-T, DVB-S, etc.).

Para las pruebas realizadas en el proyecto se han realizado envíos de videos almacenados en local, ya que no disponemos de un PC con tarjetas capturadoras de TV.

Para cerciorarnos que el envío de videos se está realizando correctamente, ejecutamos el envío de un video y capturamos los paquetes con Wireshark para ver si se están enviando correctamente. Fig. 4.4.

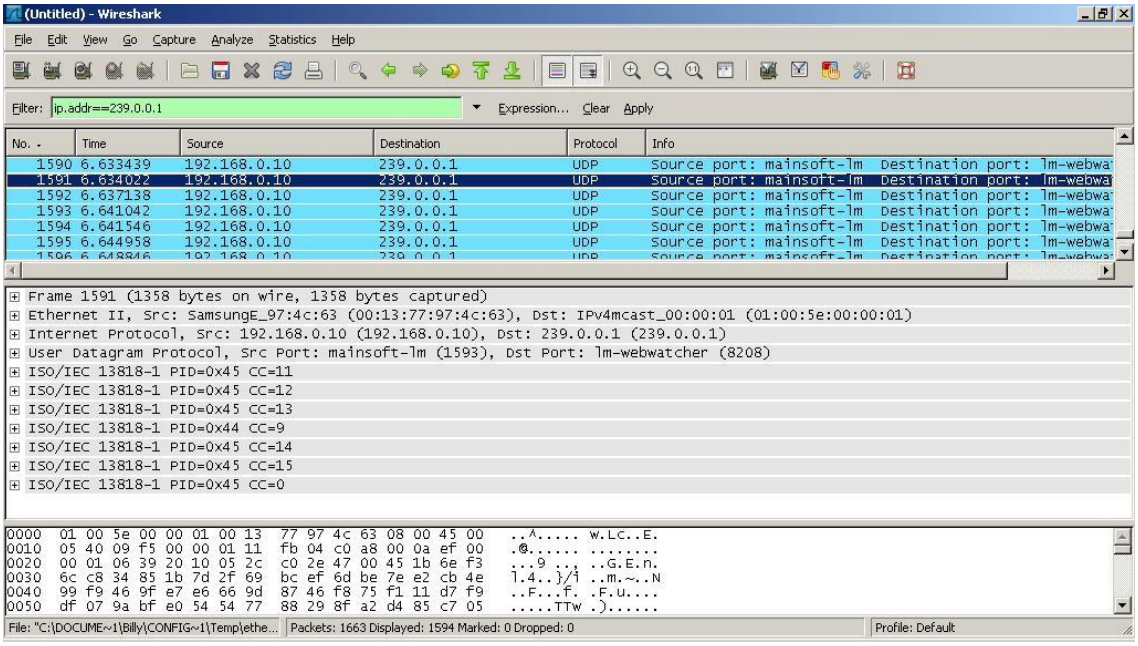

Fig. 4.4 Captura envío video MPEG-TS sobre UDP.

Como podemos apreciar, se están realizando correctamente el envío de video hacia la dirección IP Multicast 239.0.0.1 por el puerto 8208. Podemos observar conforme se está enviando encapsulado en MPEG-TS (paquetes de 188bytes) directamente sobre UDP, de ahí los 7 encapsulados que se pueden aprecia en la carga del paquete UDP.

Una vez verificado el correcto envío de un video, procedemos al envío de todos los videos y de los archivos de audio, ejecutando los comandos del anexo III.

4.4. Control de flujos Multicast en el router XORP

Para la instalación de esta aplicación, hemos partido de la base de un PC con la instalación del sistema operativo Linux, concretamente una Ubuntu 8.0.4, el cual dispone de dos tarjetas Ethernet (eth0 y eth1).

Una vez tenemos el sistema operativo, realizaremos la instalación del software XORP, esto lo realizaremos desde una consola de "Shell" con el siguiente comando:

#apt-get install xorp

Con esto ya tendremos disponible la aplicación en nuestro sistema operativo. El siguiente paso será ejecutar el modulo de dicha aplicación que nos permite configurar y ver el estado del router.

#xorpsh

Desde esta parte de la aplicación podemos configurar el comportamiento del router, así como la visualización de alguna información referente a la configuración y al tráfico que está gestionando.

El archivo de configuración se almacena en */etc/xorp/config.boot*, el cual hemos adjuntado en el Anexo III de este proyecto.

En nuestro escenario como el único router presente será el XORP, será el responsable de mantener la información sobre los flujos Multicast disponibles en la red. Esto lo podemos observar con el comando siguiente:

#showpim mfc

De este comando obtenemos un listado de todos los flujos multicast disponibles, así como la dirección IP unicast de la fuente de origen de dichos flujos.

4.5. Recepción documentos XML y videos en el cliente

4.5.1. Recepción documentos XML y videos en el cliente.

Para el diseño del cliente DVB-IP, se fueron realizando módulos para ir implementando las distintas funcionalidades necesarias. Se diseñó un modulo para la captura de los documentos XML (Capturador), otro módulo para la

extracción de la información de los documentos (ParserCompleto). De esta forma podíamos ir testeando la funcionalidad implementa. Finalmente se fusionaron en una aplicación (ClienteCompleto) la cual realiza todas las funciones de manera automática.

4.5.2. Pruebas de captura de los documentos XML

El primer paso para la captura de los documentos XML es definir la IP del Entry Point. En nuestro caso hemos dado dos opciones, la primera es la dirección IP (224.0.23.14) definida por el estándar DVB-IP y registrada por la IANA para este servicio. La segunda opción es que el usuario introduzca manualmente la dirección IP del Entry Point.

En el estándar se definen dos métodos mas para el Entry Point, como se explico en el apartado 2.1.2.6, que van ligados al uso de consultas DNS SRV. Se intentó integrar la aplicación adns [31] que proporciona la posibilidad de realizar esta funciones, pero sin éxito.

Una vez tenemos dicha dirección IP, el cliente debe de añadirse al grupo multicast de dicha dirección mediante IGMP. Tal y como vemos en la figura 4.5 el cliente ha solicitado añadirse al grupo 224.0.23.14. De esta forma el router dejara pasar dicho trafico multicast hacia el cliente.

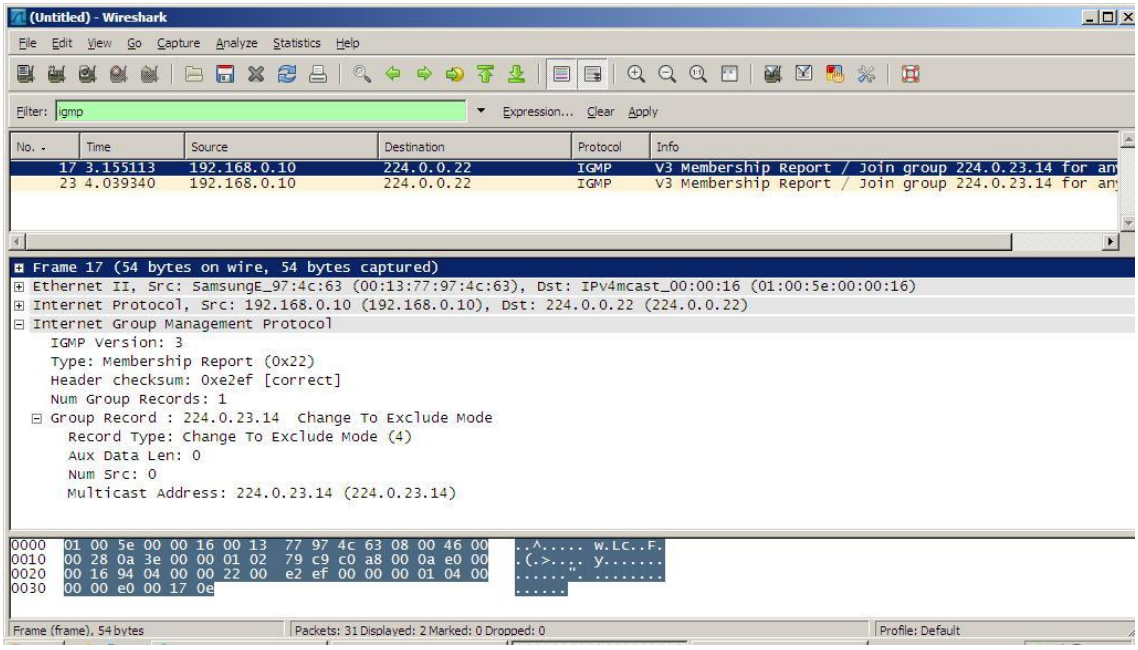

Fig. 4.5 Join IGMP hacia la IP multicast 224.0.23.14.

Cuando se empiezan a recibir los paquetes DVBSTP, el cliente deberá de ir capturándolos, desencapsulándolos y verificando que son paquetes deseados, así como verificar el PayloadID es el correcto. Para el caso del SD&S el Payload ID es el “0x01”.

El Capturador ira almacenando todas las secciones deseadas y controlando que tiene todas las secciones, solicitando dejar el grupo multicast en el momento en el que tiene todas las secciones. Finalmente debe de almacenar el documento ServiceDiscovery.xml para su posterior uso.

Para una mejor verificación visual, el Capturador mostrara por pantalla el resultado del documento XML en una pantalla de Internet Explorer como podemos apreciar en la figura 4.6.

Teniendo ya capturado y almacenado en un archivo el documento XML, pasamos a verificar su correcta captura, y también que sea un documento XML valido y “well-formed”. Para ello, utilizaremos la opción 2 del menú en nuestro Capturador, el cual hace una llamada a la aplicación xmlstarlet que nos informara si es válido el documento. Fig. 4.7.

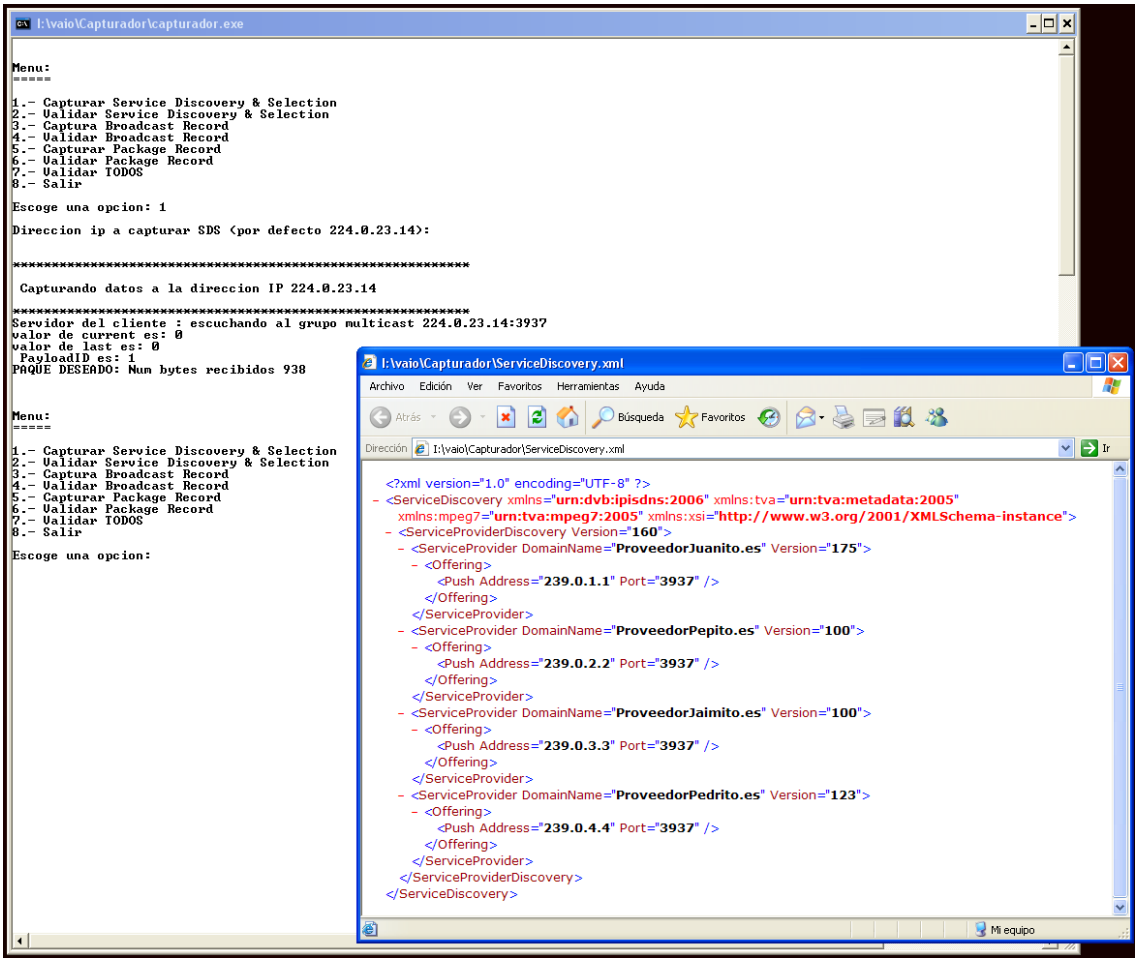

Fig. 4.6 Captura y visualización del ServiceDiscovery.xml.

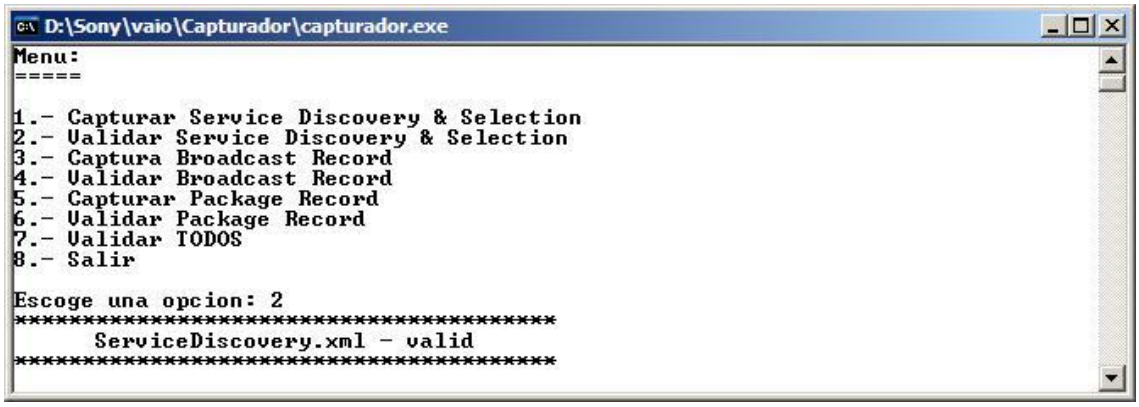

Fig. 4.7 Validación del ServiceDiscovery.xml.

Verificado el correcto funcionamiento del Capturador para el documento Service Discovery, podemos ir ejecutando el resto de las opciones del menú, para ir verificando las correctas capturas del Broadcast Record y del Package Record.

4.5.3. Extracción de la información de los documentos XML

De los documentos XML que hemos capturado en el apartado anterior, ahora debemos de extraer la información interesante y necesaria para el cliente. Esto lo realizaremos con el modulo ParserCompleto que hemos diseñado, la cual lleva integrada la API Expat. Esta API nos permite realizar esta operación mediante llamadas a funciones internas.

Del "parseo" del documento ServiceDiscovery.xml, opción 1 del menú, obtendremos un listado de Proveedores de Servicio y daremos la opción al usuario para que escoja cuál de ellos desea seleccionar. Una vez seleccionado el Proveedor de Servicios deseado, en en la figura 4.8 se muestra el ProveedorJuanito.es, mostraremos por pantalla los datos detallados del mismo y los almacenaremos. Estos datos serán necesarios para la captura de los siguientes XML en el cliente final.

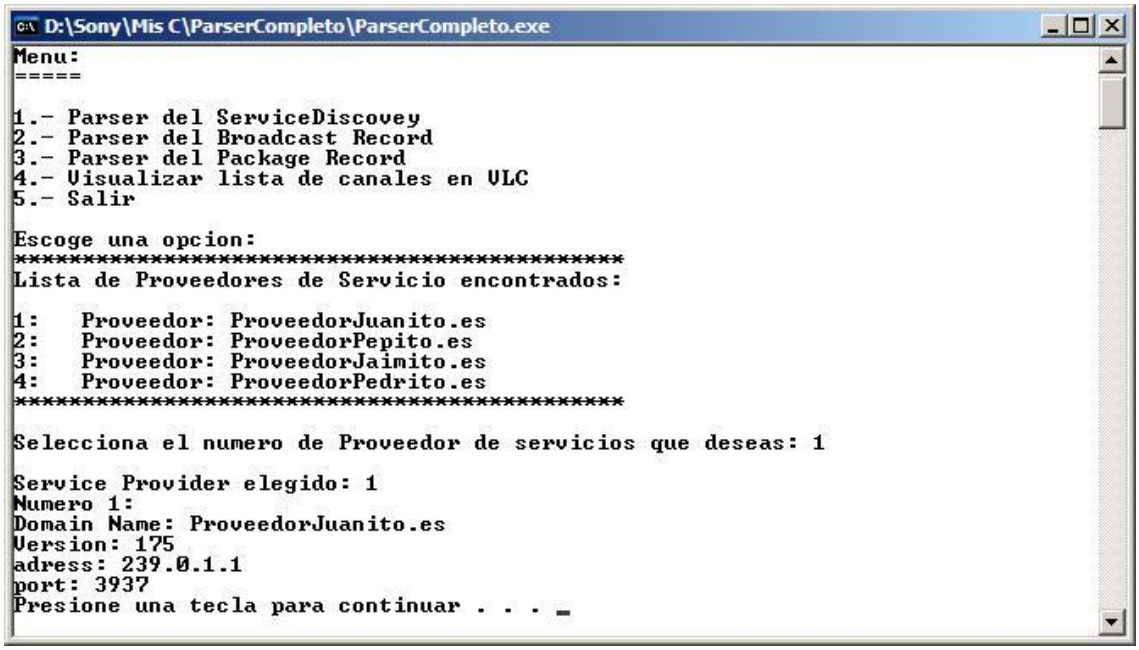

Fig. 4.8 Lista de Proveedores disponibles en el ServiceDiscovery.xml.

Con la opción 2 del menú, realizamos la extracción de la información interesante del Broadcast Record, mostrando por pantalla la más relevante sobre cada uno de los Live Media encontrados, así como un recuento final del número de servicios encontrados.

Como podemos observar en la figura 4.9 hemos obtenido la información necesaria sobre cada uno de los canales multicast de video disponible para el Proveedor de Servicios seleccionado con anterioridad.

El siguiente paso será extraer la obtención de datos interesantes del Package Record, donde tendremos la asociación entre el número lógico de canal y cada uno de los canales (Logical Channel Number). En la figura 4.10 vemos como se asocia cada servicio a un número lógico de canal.

Una vez tenemos todos los datos de todos los XML, estamos en disposición de poder visualizar los canales Live Media. Este paso lo realizaremos con el VLC.

Finalmente podemos visualizar los canales mediante una lista de reproducción, la cual generamos con los resultados extraídos en los pasos anteriores y visualizando dicha lista en el cliente VLC. Se dispone la posibilidad de ir cambiando de canal a visualizar desde el VLC tal como se ve en la figura. 4.11.

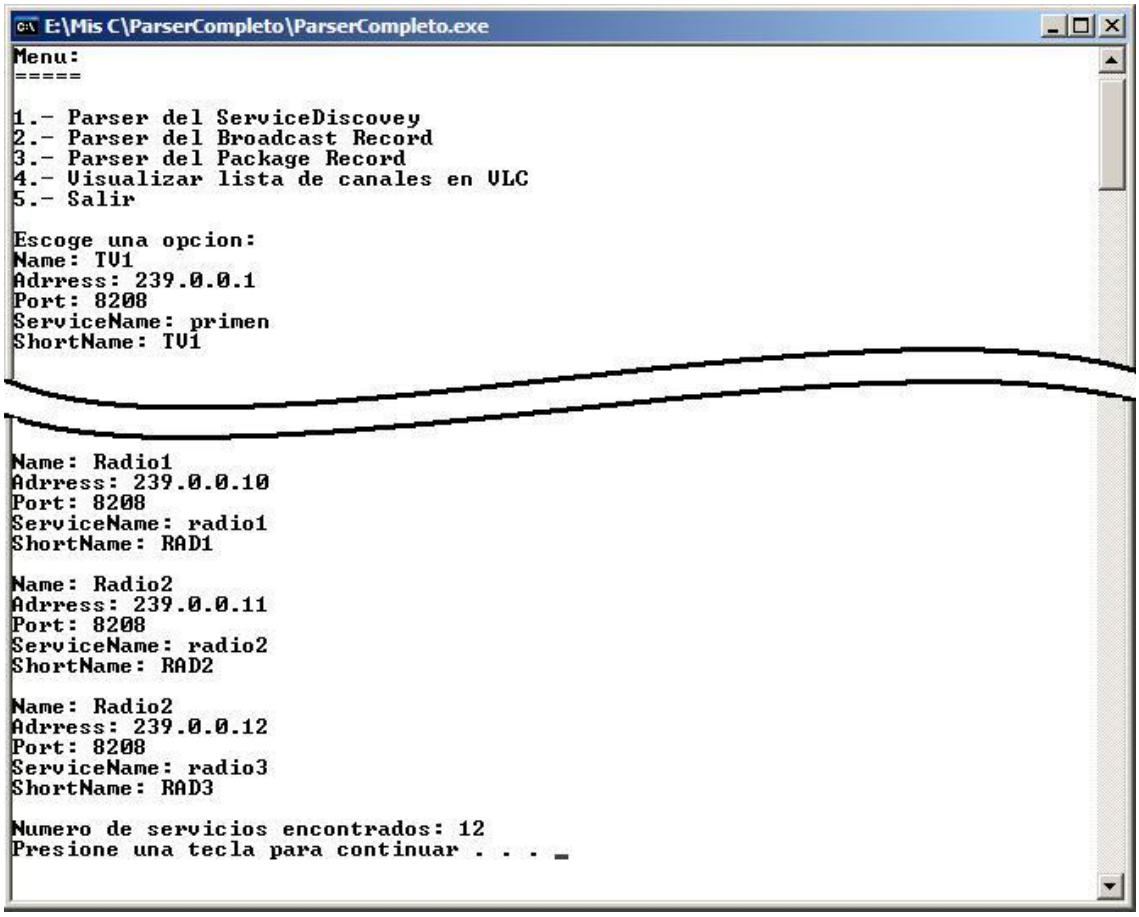

Fig. 4.9 Extracción de la información de los Live Media disponibles

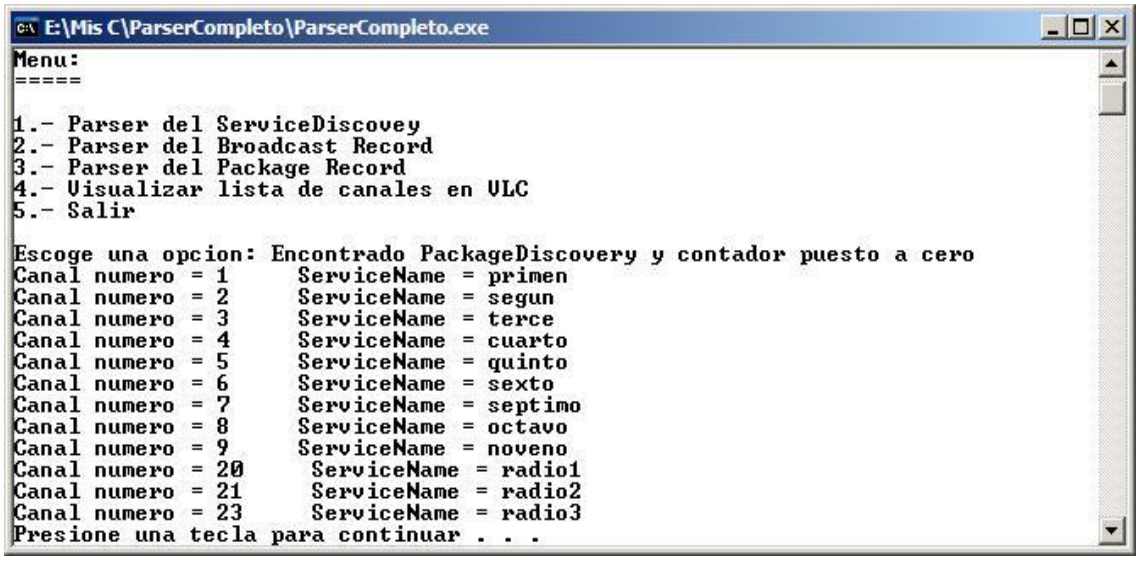

Fig. 4.10 Extracción de la información del Package Record

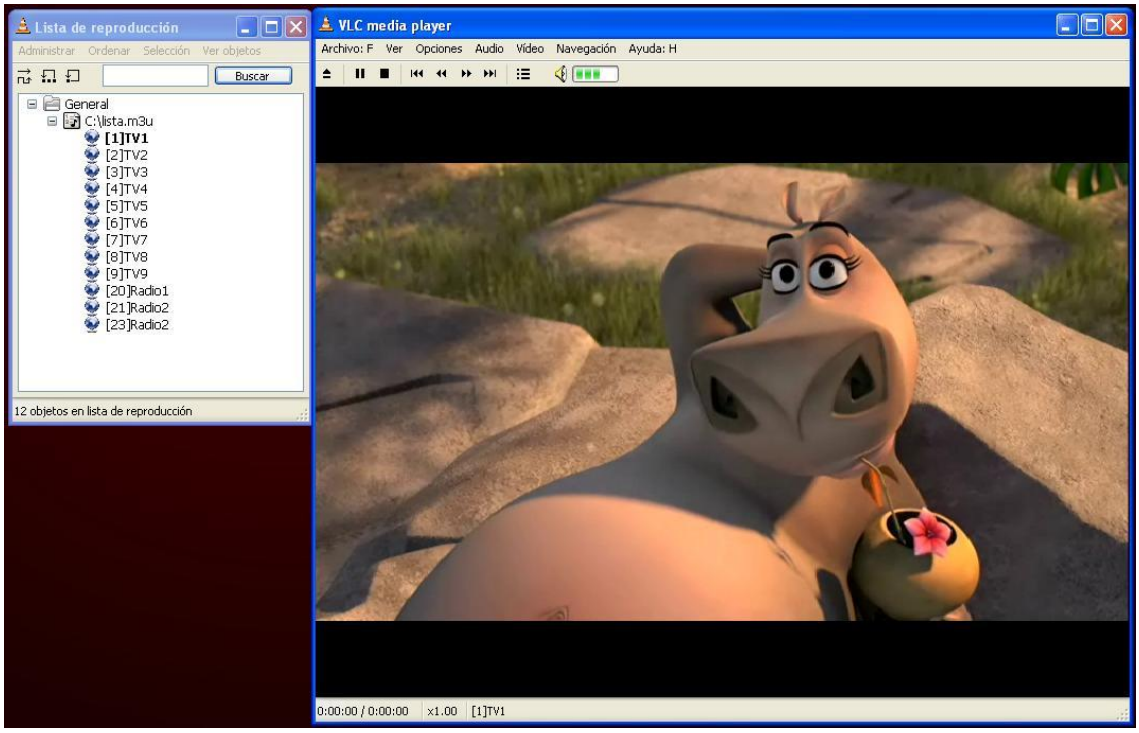

Fig. 4.11 Visualización lista de canales en VLC.

4.6. Cliente completo DVB-IP

El último paso a realizar, es la fusión de todos los módulos diseñados en un solo cliente completo, el cual realice todos los pasos descritos con anterioridad de forma automática. Se han modificado los bloques Capturador y ParserCompleto para integrarlos en única aplicación final ClienteCompleto.

La única opción que ha de seleccionar el usuario es la elección del Proveedor de Servicios que desea, a partir de ese momento se realizan las capturas de los documentos XML, se extrae la información necesaria y finalmente se muestra la lista de canales disponibles en el cliente VLC.

Podemos ver el código implementado en el Anexo IV.

CONCLUSIONES Y LÍNEAS FUTURAS

Conclusiones.

Se ha logrado diseñar un cliente DVB-IP el cual cumple con el estándar y en gran medida con el perfil LMB, también se ha implementado por necesidad del proyecto un servidor DVBSTP el cual no estaba en un principio pensado, pero aun así, se ha implementado y funciona correctamente.

Lo primero que se tuvo que realizar, fue una fase de documentación. Una revisión de los Estándares DVB-IP, Handbook [1], Guidelines [2], DVB-IPTV Profiles [3], etc., asimilados los conceptos básicos de DVB-IP. Estos están ligados a otros estándares ante los cuales también nos tuvimos que documentar, MPEG2-TS [13], XML, etc.

Con los conocimientos adquiridos, se inició el proceso de creación del cliente. En una primera fase, se necesitaba implementar un módulo el cual fuera capaz de capturar los paquetes UDP y realizar el desencapsulado DVBSTP. Aquí es donde se realizó la búsqueda de librerías del lenguaje de programación C [14] y [15], así como códigos de ejemplo en C para la captura de tráfico multicast. El resultado de este módulo fue el Capturador que hemos añadido en este proyecto visto en el apartado 2.4.4.

Cabe mencionar que para que nuestro cliente cumpliera el estándar al completo, debería de poder realizar consultas DNS sobre el servicio DVB explicado en el apartado 1.5.6 para encontrar los puntos de entrada (Entry Points). Lamentablemente no se ha conseguido esta funcionalidad ya que es un uso algo más complejo de los DNS. Se intentó integrar la aplicación adns [31] que proporciona la posibilidad de realizar consultas SRV sobre DNS, pero sin éxito. Tan solo se pudo conseguir realizar consultas DNS sencillas sobre registros A. Este punto queda pendiente para cumplir completamente con el estándar.

Teniendo ya diseñado un Capturador de tráfico, se necesitaba un servidor de datos que fuera capaz de enviar datos en multicast, y que además los encapsule en DVBSTP, para así poder realizar pruebas de envío y recepción de datos. De esta forma se creó el serverxml2 visto en el apartado 3.2.1.

Con un Capturador de trafico multicast y un servidor de trafico multicast, ya teníamos los elementos necesarios para hacer pruebas de envío, pero para hacer más reales las pruebas de envío y verificar que realmente se empleaba correctamente el uso de IGMP, era necesario integrar un router intermedio el cual uniera los dos segmentos multicast, es donde se hizo latente la necesidad del router XORP visto en el apartado 2.3.2.

Una vez que los datos multicast ya se enviaban de un extremo a otro y eran encapsulados de forma correcta, era el momento de realizar los XML con la información necesaria a enviar. Estos XML los hemos creado de forma manual, con la información necesaria y siguiendo algunos de los ejemplos que hay en el documento Guidelines del estándar DVB-IP [2].

Al disponer de los XML ya generados de forma manual, debíamos de buscar la forma de extraer la información relevante de ellos y almacenarla para su uso. En este momento fue cuando se hizo una búsqueda de "parsers" XML que nos pudieran ayudar a realizar de forma correcta y precisa esta función. Buscando información sobre "parsers" XML encontramos Expat, uno de los más utilizados en Internet (por ejemplo Mozilla lo usa [30]). Con esta API pudimos extraer la información de los XML y realizar pruebas de extracción de datos, donde apareció el módulo implementado en el proyecto, el ParserCompleto como hemos visto en el apartado 3.2.5.

Solventado el envío de metadatos, había que entrar en el envío y recepción de videos. En un principio este punto estaba bastante claro, al saber del buen funcionamiento y amplia funcionalidad del software libre VLC.

Para el envío de datos se busco la forma de poder enviar varios flujos de videos a través de IP multicast y que no consumieran grandes recursos en el servidor. Fue cuando encontramos la opción del interface telnet de VLC, que nos proporciona gran amplitud de funciones para el envío de videos a través de la red, con una sola instancia de VLC ejecutada. En el anexo III se pueden ver los comandos necesarios para el envío de flujos de video usando VLC.

La recepción de video se debía de realizar en función de la información obtenida de los XML, por ello, se creó la lista de reproducción a partir de la información que contenían los XML y visualizando esta lista en el VLC, teníamos una forma elegante y vistosa de mostrar la lista de canales disponibles, así como la posibilidad de visualizar el canal deseado.

Por último me gustaría añadir que se ha intentado realizar un entorno grafico GUI para una visualización más agradable y dejar el proyecto final más completo, se estuvo testeando el GTK+ [32], pero lamentablemente por falta de conocimientos y de tiempo este punto no ha sido posible de implementar.

En resumen, se dispone de un escenario completo DVB-IP formado por un servidor de metadatos que encapsula en DVBSTP y envía sobre UDP, un servidor de video de flujos MPEG-TS sobre multicast, un router con capacidad de encaminamiento multicast y un ClienteCompleto capaz de funcionar casi por completo con el estándar DVB-IP.

Repercusión medioambiental

Las repercusiones medioambientales que puedan estar ocasionadas por este proyecto no son más de las que puedan causar el uso cotidiano de cualquier ordenador, así como sus componentes y/o su fabricación. Se podría añadir que

si en un futuro se implementara esta forma de transmisión multimedia en frente de la tradicional por radiofrecuencia, incluso podríamos llegar a hablar de beneficios medioambientales, al liberar el aire de las transmisiones electromagnéticas actuales, quedando un amplio rango de frecuencias disponibles para cualquier otra función o simplemente un aire más “limpio”.

Aspectos éticos

No se aprecia ninguna implicación ética en este proyecto que deba destacar. Los únicos aspectos a tener en cuenta podrían ser los contenidos emitidos que no fueran apropiados para ciertas edades, implicaran alguna tendencia política, religión, etc., pero estos aspectos serían los mismos que para los actuales métodos de retransmisión de televisión.

Aspectos de seguridad

Para el desarrollo de este proyecto nos hemos centrado en el perfil LMB el cual no contempla ningún tipo de control de acceso a los usuarios, esto es debido a que el propio estándar DVB-IP en ningún momento lo contempla, por lo tanto en este aspecto no abría ningún problema de seguridad, ya que son de libre acceso para los cliente conectados al proveedor.

En referencia a la seguridad del servidor XML y del flujo de video, el único aspecto posible es que si es mal gestionado podemos inundar la red de tráfico multicast, esta función deberá ser controlada por el administrador.

Finalmente, cabría mencionar el problema de seguridad en cuanto a falsedad de identidad, ya que no hay ningún método para identificar si el tráfico que estamos recibiendo es lícito del servidor, pero esto no debería ser problema ya que para ello, el intruso debería realizar el envío de datos desde la propia red del proveedor de servicio, el cual ya ha de tener controlado los accesos a su red.

Líneas futuras

Como ya se ha mencionado en las conclusiones, el integrar un cliente DNS que sea capaz de realizar consultas SRV, sería una de las funcionalidades que completarían este proyecto, así como la creación de una interfaz gráfica para el uso y visualización del cliente DVB-IP.

Para completar aún más este proyecto otra línea futura sería la de implementar el modo *Pull,* el cual trabaja en Unicast y los documentos XML se envían a través de HTTP sobre TCP, estableciéndose una conexión entre el cliente y el servidor.

Otra posibilidad sería la de saltar al siguiente perfil de DVB-IP, el CoD, esto ya tendría una mayor complejidad, puesto que pasaríamos a trabajar con IGMP a hacerlo con RTSP. En cuanto a descubrimiento de servicio además del SD&S se debería de implementar la BCG (Broadband Content Guide), que proporciona información para los contenidos CoD y además de la EPG (Electronic Program Guide). En la ultima versión del estándar DVB-IP [1] se ha implementado como opcional la posibilidad del uso de peticiones SOAP para su funcionamiento. Así mismo se podría realizar en dos proyectos por separado el cliente y el servidor, ya que los dos serían de gran complejidad.

BIBLIOGRAFÍA

DVB-IP

[1] Digital Video Broadcasting (DVB); Transport of MPEG-2 TS Based DVB Services over IP Based Networks (DVB-IPTV Phase 1.4), DVB Bluebook A086r7.
http://www.dvb.org/technology/standards/a086r7.dTS102034.V1.4.1.DVB-IPTV1.4.zip

[2] Digital Video Broadcasting (DVB); Guidelines for the implementation of DVB-IP Phase 1 specifications, ETSI TS 102 542 V1.2.1.
http://pda.etsi.org/exchangefolder/ts_102542v010201p.pdf

[3] Digital Video Broadcasting (DVB); DVB-IPTV Profiles for TS 102 034, ETSI TS 102 826 V1.1.1.
http://pda.etsi.org/exchangefolder/ts_102826v010101p.pdf

[4] DVB-HN (Home Network) Reference Model Phase 1; DVB Document A109, February 2007.
http://www.dvb.org/technology/standards/a109.tm3690r2.DVB-HN_ref_model.pdf

[5] Digital Video Broadcasting (DVB); Specification for the use of Video and Audio Coding in Broadcasting Applications based on the MPEG-2 Transport Stream, ETSI TS 101 154 V1.8.1 (2007-07).
http://pda.etsi.org/exchangefolder/ts_101154v010801p.pdf

[6] Internet Protocol TV; Broadcast to Broadband - Open Standards for IPTV.
http://www.dvb.org/technology/fact_sheets/DVB-IPTV%20Fact%20Sheet.0908.pdf

IETF RFCs

[7] RFC 3171. IANA Guidelines for IPv4 Multicast Address Assignments.
http://www.ietf.org/rfc/rfc3171.txt

[8] RFC 2782. A DNS RR for specifying the location of services (DNS SRV).
http://www.ietf.org/rfc/rfc2782.txt

[9] RFC 791. Internet Protocol.
http://www.ietf.org/rfc/rfc0791.txt

[10] ISO/IEC 13818-1. Information technology — Generic coding of moving pictures and associated audio information: Systems.
http://www.iso.org

[11] RFC 2132. DHCP Options and BOOTP Vendor Extensions.
http://www.ietf.org/rfc/rfc2132.txt

[12] RFC 3376. Internet Group Management Protocol, Version 3.
http://www.ietf.org/internet-drafts/draft-ietf-idmr-igmp-v3-11.txt

[13] Real Time Streaming Protocol (RTSP)
http://www.ietf.org/rfc/rfc2326.txt

[14] RTP: A Transport Protocol for Real-Time Applications
http://tools.ietf.org/rfc/rfc3550.txt

Programación

[15] Curso de C.
http://www.elrincondelc.com/cursoc/cursoc.html

[16] Guía Beej de Programación en Redes.
http://www.arrakis.es/~dmrq/beej/index.html

[17] Schildt,H., *C. Manual De Referencia*. 4.ª Ed. Editorial McGraw-Hill, ISBN: 8448128958.

[18] Programación de sockets en C de Unix/Linux.
http://www.chuidiang.com/clinux/sockets/sockets_simp.php

Herramientas utilizadas.

[19] Wireshark.
http://www.wireshark.org

[20] Notepad++.
http://notepad-plus.sourceforge.net/es/site.htm

[21] Cygwin.
http://www.cygwin.com/

[22] Bloodshed Dev-C++.
http://www.bloodshed.net/devcpp.html

[23] Expat.
http://expat.sourceforge.net

[24] Xmlstarlet.
http://xmlstar.sourceforge.net

[25] Infocast2Tools .
http://sourceforge.net/projects/infocast2tools/

[26] VideoLan.
http://www.videolan.org/

[27] XORP (eXtensible Open Router Platform).
http://www.xorp.org/

Varios.

[28] Flujos de programa y de transporte MPEG-2 aplicación a DVB.
http://www.etc.upm.es/tsmpeg2d.pdf

[29] Extensible Markup Language.
http://es.wikipedia.org/wiki/XML

[30] DVB-IPI.
http://es.wikipedia.org/wiki/DVB-IPI

[31] Arquitectura de red Triple-Play; Jazztelia
http://www.jazztel.com/archivos/documentos/ficheros/26102006104525memoria%20anual_2005.pdf

[32] Imagenio: La nueva forma de ocio a la carta.
http://info.telefonica.es/sociedaddelainformacion/pdf/publicaciones/imagenio/imagenio.pdf

[33] XML In Mozilla.
http://www.mozilla.org/rdf/doc/xml.html

[34] GNU adns.
http://www.chiark.greenend.org.uk/~ian/adns

[35] The GTK+ Projetc.
http://www.gtk.org

[36] Digital Video Broadcasting Project (DVB).
http://www.dvb.org

[37] European Telecommunications Standards Institute (ETSI).
http://www.etsi.org

[38] Microsoft Mediaroom.
http://en.wikipedia.org/wiki/Microsoft_Mediaroom

[39] BT Vision.
http://www.btvision.bt.com

[40] World Wide Wed.
http://www.w3.org

[41] Extensible Markup Language.
http://www.w3.org/TR/xml/

[42] Standard Generalized Markup Language (SGML).
http://www.iso.org/iso/catalogue_detail.htm?csnumber=16387

[43] Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems.
http://pda.etsi.org/exchangefolder/en_300468v010901p.pdf

ANEXO I. Documentos XML

A continuación tenemos los documentos XML empleados para el desarrollo del proyecto. Los cuales son validos y bien formados (well-formed).

ServiceDiscovery.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ServiceDiscovery      xmlns="urn:dvb:ipisdns:2006"      xmlns:tva="urn:tva:metadata:2005"
xmlns:mpeg7="urn:tva:mpeg7:2005"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <ServiceProviderDiscovery Version="160">
       <ServiceProvider DomainName="ProveedorJuanito.es" Version="175">
              <Offering>
                     <Push Address="239.0.1.1" Port="3937"> </Push>
              </Offering>
       </ServiceProvider>
       <ServiceProvider DomainName="ProveedorPepito.es" Version="100">
              <Offering>
                     <Push Address="239.0.2.2" Port="3937"> </Push>
              </Offering>
       </ServiceProvider>
       <ServiceProvider DomainName="ProveedorJaimito.es" Version="100">
              <Offering>
                     <Push Address="239.0.3.3" Port="3937"> </Push>
              </Offering>
       </ServiceProvider>
       <ServiceProvider DomainName="ProveedorPedrito.es" Version="123">
              <Offering>
                     <Push Address="239.0.4.4" Port="3937"> </Push>
              </Offering>
       </ServiceProvider>
  </ServiceProviderDiscovery>
</ServiceDiscovery>
```

BroadcastRecord.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ServiceDiscovery xmlns="urn:dvb:ipisdns:2006" xmlns:urn="urn:tva:metadata:2005">
 <BroadcastDiscovery DomainName="eugenio.name" Version="10">
       <ServiceList>
              <SingleService>
                     <ServiceLocation>
                            <IPMulticastAddress Port="8208" Address="239.0.0.1" />
                     </ServiceLocation>
                     <TextualIdentifier ServiceName="primen"/>
                     <DVBTriplet OrigNetId="0" ServiceId="5002" TSId="201"/>
                     <SI ServiceType="">
                            <Name Language="SPA">TV1</Name>
                            <ShortName Language="SPA">TV1</ShortName>
                   <Description Language="SPA">Este es el primer canal </Description>
                     </SI>
              </SingleService>
              <SingleService>
                     <ServiceLocation>
                            <IPMulticastAddress Port="8208" Address="239.0.0.2" />
                     </ServiceLocation>
                     <TextualIdentifier ServiceName="segun"/>
                     <DVBTriplet OrigNetId="0" ServiceId="5002" TSId="202"/>
                     <SI ServiceType="">
                            <Name Language="SPA">TV2</Name>
                            <ShortName Language="SPA">TV2</ShortName>
                   <Description Language="SPA">Este es el segundo canal </Description>
                     </SI>
              </SingleService>
```

```
<SingleService>
        <ServiceLocation>
                <IPMulticastAddress Port="8208" Address="239.0.0.3" />
        </ServiceLocation>
        <TextualIdentifier ServiceName="terce"/>
        <DVBTriplet OrigNetId="0" ServiceId="5002" TSId="203"/>
        <SI ServiceType="">
                <Name Language="SPA">TV3</Name>
                <ShortName Language="SPA">TV3</ShortName>
      <Description Language="SPA">Este es el tercer canal </Description>
        </SI>
</SingleService>
<SingleService>
        <ServiceLocation>
                <IPMulticastAddress Port="8208" Address="239.0.0.4" />
        </ServiceLocation>
        <TextualIdentifier ServiceName="cuarto"/>
        <DVBTriplet OrigNetId="0" ServiceId="5002" TSId="204"/>
        <SI ServiceType="">
                <Name Language="SPA">TV4</Name>
                <ShortName Language="SPA">TV4</ShortName>
      <Description Language="SPA">Este es el cuarto canal </Description>
        </SI>
</SingleService>
<SingleService>
        <ServiceLocation>
                <IPMulticastAddress Port="8208" Address="239.0.0.5" />
        </ServiceLocation>
        <TextualIdentifier ServiceName="quinto"/>
        <DVBTriplet OrigNetId="0" ServiceId="5002" TSId="205"/>
        <SI ServiceType="">
                <Name Language="SPA">TV5</Name>
                <ShortName Language="SPA">TV5</ShortName>
      <Description Language="SPA">Este es el quinto canal </Description>
        </SI>
</SingleService>
<SingleService>
        <ServiceLocation>
                <IPMulticastAddress Port="8208" Address="239.0.0.6" />
        </ServiceLocation>
        <TextualIdentifier ServiceName="sexto"/>
        <DVBTriplet OrigNetId="0" ServiceId="5002" TSId="206"/>
        <SI ServiceType="">
                <Name Language="SPA">TV6</Name>
                <ShortName Language="SPA">TV6</ShortName>
      <Description Language="SPA">Este es el sexto canal </Description>
        </SI>
</SingleService>
<SingleService>
        <ServiceLocation>
                <IPMulticastAddress Port="8208" Address="239.0.0.7" />
        </ServiceLocation>
        <TextualIdentifier ServiceName="septimo"/>
        <DVBTriplet OrigNetId="0" ServiceId="5002" TSId="207"/>
        <SI ServiceType="">
                <Name Language="SPA">TV7</Name>
                <ShortName Language="SPA">TV7</ShortName>
      <Description Language="SPA">Este es el septimo canal
      </Description>
        </SI>
</SingleService>
<SingleService>
        <ServiceLocation>
                <IPMulticastAddress Port="8208" Address="239.0.0.8" />
        </ServiceLocation>
        <TextualIdentifier ServiceName="octavo"/>
        <DVBTriplet OrigNetId="0" ServiceId="5002" TSId="208"/>
        <SI ServiceType="">
                <Name Language="SPA">TV8</Name>
                <ShortName Language="SPA">TV8</ShortName>
      <Description Language="SPA">Este es el octavo canal </Description>
        </SI>
</SingleService>
<SingleService>
        <ServiceLocation>
                <IPMulticastAddress Port="8208" Address="239.0.0.9" />
        </ServiceLocation>
```

```
                        <TextualIdentifier ServiceName="noveno"/>
                        <DVBTriplet OrigNetId="0" ServiceId="5002" TSId="209"/>
                        <SI ServiceType="">
                                <Name Language="SPA">TV9</Name>
                                <ShortName Language="SPA">TV9</ShortName>
                      <Description Language="SPA">Este es el noveno canal </Description>
                        </SI>
                </SingleService>
                <SingleService>
                        <ServiceLocation>
                                <IPMulticastAddress Port="8208" Address="239.0.0.10" />
                        </ServiceLocation>
                        <TextualIdentifier ServiceName="radio1"/>
                        <DVBTriplet OrigNetId="1" ServiceId="5002" TSId="201"/>
                        <SI ServiceType="">
                                <Name Language="SPA">Radio1</Name>
                                <ShortName Language="SPA">RAD1</ShortName>
                      <Description Language="SPA">Esta es la primera radio </Description>
                        </SI>
                </SingleService>
                <SingleService>
                        <ServiceLocation>
                                <IPMulticastAddress Port="8208" Address="239.0.0.11" />
                        </ServiceLocation>
                        <TextualIdentifier ServiceName="radio2"/>
                        <DVBTriplet OrigNetId="1" ServiceId="5002" TSId="202"/>
                        <SI ServiceType="">
                                <Name Language="SPA">Radio2</Name>
                                <ShortName Language="SPA">RAD2</ShortName>
                      <Description Language="SPA">Esta es la segunda radio </Description>
                        </SI>
                </SingleService>
                <SingleService>
                        <ServiceLocation>
                                <IPMulticastAddress Port="8208" Address="239.0.0.12" />
                        </ServiceLocation>
                        <TextualIdentifier ServiceName="radio3"/>
                        <DVBTriplet OrigNetId="1" ServiceId="5002" TSId="203"/>
                        <SI ServiceType="">
                                <Name Language="SPA">Radio2</Name>
                                <ShortName Language="SPA">RAD3</ShortName>
                      <Description Language="SPA">Esta es la tercera radio </Description>
                        </SI>
                </SingleService>
        </ServiceList>
 </BroadcastDiscovery>
</ServiceDiscovery>
```

PackageRecord.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ServiceDiscovery xmlns="urn:dvb:ipisdns:2006">
	<PackageDiscovery DomainName="eugenio.name" Version="10">
		<Package Id="1">
			<PackageName Language="SPA">Paquete1</PackageName>
			<Service>
				<TextualID ServiceName="primen"/>
				<LogicalChannelNumber>1</LogicalChannelNumber>
			</Service>
			<Service>
				<TextualID ServiceName="segun"/>
				<LogicalChannelNumber>2</LogicalChannelNumber>
			</Service>
			<Service>
				<TextualID ServiceName="terce"/>
				<LogicalChannelNumber>3</LogicalChannelNumber>
			</Service>
			<Service>
				<TextualID ServiceName="cuarto"/>
				<LogicalChannelNumber>4</LogicalChannelNumber>
			</Service>
			<Service>
				<TextualID ServiceName="quinto"/>
				<LogicalChannelNumber>5</LogicalChannelNumber>
			</Service>
			<Service>
				<TextualID ServiceName="sexto"/>
				<LogicalChannelNumber>6</LogicalChannelNumber>
			</Service>
			<Service>
				<TextualID ServiceName="septimo"/>
				<LogicalChannelNumber>7</LogicalChannelNumber>
			</Service>
			<Service>
				<TextualID ServiceName="octavo"/>
				<LogicalChannelNumber>8</LogicalChannelNumber>
			</Service>
			<Service>
				<TextualID ServiceName="noveno"/>
				<LogicalChannelNumber>9</LogicalChannelNumber>
			</Service>
			<Service>
				<TextualID ServiceName="radio1"/>
				<LogicalChannelNumber>20</LogicalChannelNumber>
			</Service>
			<Service>
				<TextualID ServiceName="radio2"/>
				<LogicalChannelNumber>21</LogicalChannelNumber>
			</Service>
			<Service>
				<TextualID ServiceName="radio3"/>
				<LogicalChannelNumber>23</LogicalChannelNumber>
			</Service>
		</Package>
	</PackageDiscovery>
</ServiceDiscovery>
```

ANEXO II. Tablas

En este anexo se podrán encontrar las tablas descritas en el proyecto.

Tabla de Audio y Video coding

A continuación se muestra una tabla con las codificaciones de video admitidas por DVB-IP para la retransmisión de contenido multimedia, extraída de [5].

Video:
• MPEG-2 Main Profile at Main Level is used for MPEG-2 encoded SDTV.
• MPEG-2 Main Profile at High Level is used for MPEG-2 encoded HDTV.
• H.264/AVC Main Profile at Level 3 is used for H.264/AVC SDTV.
• H.264/AVC High Profile at Level 4 is used for H.264/AVC HDTV.
• VC-1 Advanced Profile at Level 1 is used for VC-1 SDTV.
• VC-1 Advanced Profile at Level 3 is used for VC-1 HDTV.
• The 25 Hz MPEG-2 SDTV IRD, 25 Hz H.264/AVC SDTV IRD and 25 Hz VC-1 SDTV IRD support 25 Hz frame rate.
• The 25 Hz MPEG-2 HDTV IRD, 25 Hz H.264/AVC HDTV IRD and 25 Hz VC-1 HDTV IRD support frame rates of 25 Hz or 50 Hz.
• The 30 Hz MPEG-2 SDTV IRD, 30 Hz H.264/AVC SDTV IRD and 30 Hz VC-1 SDTV IRD support frame rates of 24 000/1 001, 24, 30 000/1 001 and 30 Hz;
• The 30 Hz MPEG-2 HDTV IRD, 30 Hz H.264/AVC HDTV IRD and 30 Hz VC-1 HDTV IRD supports frame rates of 24 000/1 001, 24, 30 000/1 001, 30, 60 000/1 001 and 60 Hz.
• SDTV pictures may have either 4:3, 16:9 or 2.21:1 aspect ratio; IRDs support 4:3 and 16:9 and optionally 2.21:1 aspect ratio.
• MPEG-2 HDTV pictures have 16:9 or 2.21:1 aspect ratio; IRDs support 16:9 and optionally 2.21:1 aspect ratio.
• H.264/AVC HDTV pictures have 16:9 aspect ratio; IRDs support 16:9 aspect ratio.
• VC-1 HDTV pictures have 16:9 aspect ratio; IRDs support 16:9 aspect ratio.
Audio:
• Audio content complies with MPEG-1 Layer I, MPEG-1 Layer II, MPEG-2 Layer II backward compatible ,AC-3, Enhanced AC-3, DTS, MPEG-4 AAC, MPEG-4 HE-AAC or MPEG-4 HE-AACv2 audio.
• Sampling rates of 32 kHz, 44,1 kHz and 48 kHz are supported by IRDs.
• The encoded bit-stream does not use emphasis.
• IRDs may also optionally support full multi-channel decoding of MPEG-2 Layer II backwards compatible multi-channel audio.
• The use of Layer II encoding is recommended for MPEG-1 audio bit-streams.

Tabla de valores Payload ID en los segmentos

A continuación se muestra la tabla de valores que pueden tener los Payload ID de los segmentos, para indicar el contenido que contienen.

Payload ID value	SD&S record carried
0x00	Reserved
0x01	SD&S Service Provider Discovery Information
0x02	SD&S Broadcast Discovery Information
0x03	SD&S COD Discovery Information
0x04	SD&S Services from other SPs
0x05	SD&S Package Discovery Information
0x06	SD&S BCG Discovery Information
0x07	SD&S Regionalisation Discovery Information
0x08	SD&S RMS-FUS Firmware Announcement Message and RMS-FUS Firmware Update Announcements
0x09 to 0xA0	Reserved
0xA1 to 0xAF	BCG Payload ID values
0xB0	Reserved
0xB1	CDS XML download session description
0xB2 to OxEF	Reserved
0xF0 to 0xFF	User Private

Tabla Service Provider(s) discovery record.

En esta tabla se enumeran los registros que ha de contener el Service Provider Redord, se muestran tanto los registros obligatorios como los opcionales.

Element / Attribute Name	Element / Attribute Description	Mandated/ Optional/ Conditional
ServiceDiscovery type:	/ServiceDiscovery	
ServiceDiscovery@Version	Version of this record. A change in this value indicates a change in one of the ServiceProviderDiscovery Records.	O
ServiceProvider type (one entry per service provider):	/ServiceDiscovery/ServiceProviderDiscovery/ServiceProvider	
ServiceProvider@DomainName	An internet DNS domain name registered by the Service Provider that uniquely identifies the Service Provider	M
ServiceProvider@Version	Version of the Service Provider(s) Discovery record; the version number shall be incremented every time a change in any of the records that comprise the service discovery information for this Service Provider occurs.	M
ServiceProvider@LogoURI	Pointer to a Service Provider logo for potential display. The pointer shall be a URI [21].	O
Name	Name of the Service Provider for display in one or more languages; one Service Provider name is allowed per language code, and at least one language shall be provided (though not necessarily more than one).	M
Description	Description of the Service Provider for potential display in one or more languages; one description is allowed per language code.	O
OfferingListType type (one entry per offering):	/ServiceDiscovery/ServiceProviderDiscovery/ServiceProvider/Offering	
Push@Source Push@Address Push@Port	Port number and IP address of the multicast location of the DVB-IPTV Offering Records which describe the offerings that the Service Provider makes available. This element is optional.	O M (see note 1) M (see note 1)
Pull@Location	This URI [21] encodes the location of the DVB-IPTV Offering(s) Records which describe the offerings that the Service Provider makes available.	O
PayloadList type:	/ServiceDiscovery/ServiceProviderDiscovery/ServiceProvider/Offering/Pull/PayloadId /ServiceDiscovery/ServiceProviderDiscovery/ServiceProvider/Offering/Push/PayloadId	
PayloadId@Id	Indicates the type of service discovery information available at the DVB-IPTV offering location. For example, this can be of type broadcast discovery or CoD discovery. The different values of this field are set out in table 1 in clause 5.2.2.1.	O
Segment@ID	Indicates which segment carries service discovery information of type PayloadId@Id for this service provider.	C (see note 2)
Segment@Version	Version number of the segment identified by Segment@ID.	O

Tabla Service Provider(s) discovery record.

Element / Attribute Name	Element / Attribute Description	Mandated/ Optional/ Conditional
@DomainName	An internet DNS domain name registered by the Service Provider that uniquely identifies the Service Provider.	M
@Version	Version of the DVB-IPTV Offering record, the version number shall be incremented every time a change in the DVB-IPTV Offering record is made.	C (see note)
NOTE: The version number of the DVB-IPTV offering record is mandatory when the record is provided on request (i.e. "pull mode") and is optional when the record is multicasted (i.e. "push mode").		

Tabla "TS Full SI" Discovery Information.

Element / Attribute	Element / Attribute Description	Mandated/ Optional
BroadcastOffering type:	/BroadcastDiscovery	
IPServiceList type (one per service list):	/BroadcastDiscovery/ServiceList	
TextualIdentifier@ServiceName	A unique host name for the service within the service provider's domain.	M
DVBTriplet@OrigNetId	Identifies the network Id of the originating delivery system.	M
DVBTriplet@TSId	Identifies the Transport Stream.	M
DVBTriplet@ServiceId	Identifies a service from any other service within the TS. The service Id is the same as the program number in the corresponding program map table.	M
IPMulticastAddress@Address	Provides the multicast group address at which the service may be accessed.	M (see note)
IPMulticastAddress@Port	Provides the port at which the service may be accessed.	M (see note)

En los campos que indica “see note”, hace referencia a que son obligatorios estos campos cuando se hace el envío a través de multicast, tal y como es el caso de nuestro proyecto.

Tabla " TS Optional SI " Discovery Information.

Element / Attribute	Element / Attribute Description	Mandated/ Optional
BroadcastOffering type:	/BroadcastDiscovery	
IPServiceList type (one per service list):	/BroadcastDiscovery/ServiceList	
TextualIdentifier@ServiceName	A unique host name for the service within the service provider's domain.	M
DVBTriplet@OrigNetId	Identifies the network Id of the originating delivery system.	M
DVBTriplet@TSId	Identifies the Transport Stream.	M
DVBTriplet@ServiceId	Identifies a service from any other service within the TS. The service Id is the same as the program number in the corresponding program map table.	M
IPMulticastAddress@Address	Provides the multicast group address at which the service may be accessed.	M (see note)
IPMulticastAddress@Port	Provides the port at which the service may be accessed.	M (see note)
SI type:	/BroadcastDiscovery/ServiceList/SingleService/SI	
SI@ServiceType	Specifies the type of service; it shall be coded as per DVB SI standard 1. Examples are digital television service, digital radio sound service, mosaic service, data broadcast service, DVB MHP service, etc.	M (see note)
Name	Name of the service for display in one or more languages; one Service name is allowed per language code, and at least one language shall be provided (though not necessarily more than one).	M

En los campos que indica “see note”, hace referencia a que son obligatorios estos campos cuando se hace el envío a través de multicast como es nuestro caso, además de los campos SI para la “TS Optional SI”.

Tabla Package Record.

Element / Attribute Name	Element / Attribute Description	Mandated/ Optional
Packaged Services type:	/PackageDiscovery	
Package@Id	Identifies a package; this ID is allocated by the Service Provider	M
Package@Visible	A Boolean which indicates in combination with the PackageAvailability element, whether this package shall be presented to the user. The default value is true.	O
PackageName	Name of the package for display in one or more languages; one name per language code maximum.	M
PackageDescription	If present, this shall contain the identifier(s) of the BCG Record(s) for the BCG Discovery element that carries the information on this package.	O
PackageDescription@preferred	If present and set to true, specifies that this location contains the preferred BCG. The default value for this attribute is false. There shall be only one preferred BCG.	O
CountryAvailability	Gives a list of countries and/or groups of countries where the package is intended to be available, and/or a list of countries and/or groups where it is not. This field is deprecated and Package Availability should be used instead.	Deprecated
PackageReference	This shall be the Id(s) of package(s) that are included in the current package.	O
Service	List of services forming the package, comprising:	M
TextualID@ DomainName	An internet DNS domain name registered by the Service Provider that uniquely identifies the Service Provider. If this is omitted the Service Provider Domain Name from the inherited DVB-IPTV Offering is used.	O
TextualID@ServiceName	A unique host name for the service within the service provider's domain.	M
DVBTriplet	The DVB triplet by which the service may be known.	O
DescriptionLocation	The URI [21] of additional service description provided in the context of a package; this is not required to acquire a service.	O
DescriptionLocation @preferred	A Boolean flag which indicates if the description location indicated is the preferred location for this information. Only one location for any service may have this flag set to true.	O
LogicalChannelNumber	The logical channel number of the service.	O
PackageAvailability	This element provides support for regionalisation. It allows each package to have a list of 'cells' (regions) with which the package is associated. By default, the package is available everywhere. There shall be at most one PackageAvailability element for each CountryCode.	O
CountryCode	This element indicates the country for which the availability is being defined. This element shall be of the 2-letter format specified in ISO-3166 [58].	M
CountryCode@Availability	This flag indicates whether the package is available in the country specified by CountryCode. The default is TRUE. When TRUE, the package is available in the specified country with the exception of those regions identified by Cells. When FALSE, the package is not available in the specified country with the exception of those regions identified by Cells.	O
Cells	A list of string identifiers representing geographical regions in the country identified by CountryCode. The Cells listed represent the exception to the value supplied by the flag, i.e. the negation of the Availability flag applies to any listed cells.	O

ANEXO III. Archivos configuración

A continuacion se agrupan en una anexo los archivos de configuracion empleados durante el desarrollo del proyecto.

Configuración router XORP (config.boot)

En el PC que tiene instalado el software XORP, el archivo de configuración se encuentra almacenado en la dirección /etc/xorp/config.boot

Los parámetros básicos que hemos de configurar son las direcciones IP Unicast de los dos interfaces Ethernet, habilitar la opción multicast de IPv4 (mfea4, disable: false), habilitar el protocolo IGMP (igmp, disable: false), habilitar el protocolo de enrutamiento multicast PIM-SM (pimsm4, disable: false) y posteriormente ir ajustando las posibilidades que nos ofrecen cada uno de estos protocolos para ajustarlo a nuestras necesidades.

```
/*XORP Configuration File, v1.0*/
protocols {
    fib2mrib {
        disable: false
    }
    igmp {
        disable: false
        interface eth0 {
            vif eth0 {
                disable: false
                version: 3
                enable-ip-router-alert-option-check: false
                query-interval: 125
                query-last-member-interval: 1
                query-response-interval: 10
                robust-count: 2
            }
        }
        interface eth1 {
            vif eth1 {
                disable: false
                version: 3
                enable-ip-router-alert-option-check: false
                query-interval: 125
                query-last-member-interval: 1
                query-response-interval: 10
                robust-count: 2
            }
        }
        traceoptions {
            flag {
                all {
                    disable: true
```

```
            }
        }
    }
}
pimsm4 {
    disable: false
    interface eth0 {
        vif eth0 {
            disable: false
            dr-priority: 1
            hello-period: 30
            hello-triggered-delay: 5
        }
    }
    interface eth1 {
        vif eth1 {
            disable: false
            dr-priority: 1
            hello-period: 30
            hello-triggered-delay: 5
        }
    }
    interface "register_vif" {
        vif "register_vif" {
            disable: false
            dr-priority: 1
            hello-period: 30
            hello-triggered-delay: 5
        }
    }
    bootstrap {
        disable: false
        cand-bsr {
            scope-zone 224.0.0.0/4 {
                is-scope-zone: false
                cand-bsr-by-vif-name: "eth0"
                cand-bsr-by-vif-addr: 0.0.0.0
                bsr-priority: 1
                hash-mask-len: 30
            }
        }
        cand-rp {
            group-prefix 224.0.0.0/4 {
                is-scope-zone: false
                cand-rp-by-vif-name: "eth0"
                cand-rp-by-vif-addr: 0.0.0.0
                rp-priority: 192
                rp-holdtime: 150
            }
        }
    }
    switch-to-spt-threshold {
        disable: false
        interval: 100
        bytes: 102400
    }
    traceoptions {
        flag {
            all {
                disable: true
```

```
            }
        }
    }
}
fea {
    unicast-forwarding4 {
        disable: false
    }
    unicast-forwarding6 {
        disable: false
    }
}
interfaces {
    restore-original-config-on-shutdown: false
    interface eth0 {
        description: "Ethernet"
        disable: false
        discard: false
        unreachable: false
        management: false
        vif eth0 {
            disable: false
            address 192.168.0.1 {
                prefix-length: 24
                broadcast: 192.168.0.255
                disable: false
            }
        }
    }
    interface eth1 {
        description: "Ethernet"
        disable: false
        discard: false
        unreachable: false
        management: false
        vif eth1 {
            disable: false
            address 192.168.1.1 {
                prefix-length: 24
                broadcast: 192.168.1.255
                disable: false
            }
        }
    }
}
plumbing {
    mfea4 {
        disable: false
        interface eth0 {
            vif eth0 {
                disable: false
            }
        }
        interface eth1 {
            vif eth1 {
                disable: false
            }
        }
        interface "register_vif" {
```

```
            vif "register_vif" {
                disable: false
            }
        }
        traceoptions {
            flag {
                all {
                    disable: true
                }
            }
        }
    }
}
```

Envío de flujos MPEG-TS de videos multicast con VLC.

Para el envío de los flujos de video en MPEG-TS sobre UDP, se debe de entrar en el modo telnet del VLC.

C:\>vlc -I telnet

C:\>telnet 127.0.0.1 4212

La contraseña por defecto es “admin”.

Una vez estamos dentro debemos de configurar los flujos a enviar con los siguientes comandos:

```
new channel1 broadcast enabled
setup channel1 input .\videos\video1.ts
setup channel1 loop
setup channel1 output #standard{mux=ts,access=udp,dst=239.0.0.1:8208}

new channel2 broadcast enabled
setup channel2 input .\videos\video2.ts
setup channel2 loop
setup channel2 output #standard{mux=ts,access=udp,dst=239.0.0.2:8208}

new channel3 broadcast enabled
setup channel3 input .\videos\video3.ts
setup channel3 loop
setup channel3 output #standard{mux=ts,access=udp,dst=239.0.0.3:8208}

new channel4 broadcast enabled
setup channel4 input .\videos\video4.ts
setup channel4 loop
setup channel4 output #standard{mux=ts,access=udp,dst=239.0.0.4:8208}

new channel5 broadcast enabled
```

```
setup channel5 input .\videos\video5.ts
setup channel5 loop
setup channel5 output #standard{mux=ts,access=udp,dst=239.0.0.5:8208}

new channel6 broadcast enabled
setup channel6 input .\videos\video6.ts
setup channel6 loop
setup channel6 output #standard{mux=ts,access=udp,dst=239.0.0.6:8208}

new channel7 broadcast enabled
setup channel7 input .\videos\video7.ts
setup channel7 loop
setup channel7 output #standard{mux=ts,access=udp,dst=239.0.0.7:8208}

new channel8 broadcast enabled
setup channel8 input .\videos\video78.ts
setup channel8 loop
setup channel8 output #standard{mux=ts,access=udp,dst=239.0.0.8:8208}

new channel9 broadcast enabled
setup channel9 input .\videos\video9.ts
setup channel9 loop
setup channel9 output #standard{mux=ts,access=udp,dst=239.0.0.9:8208}

new channel10 broadcast enabled
setup channel10 input .\videos\radio1.mp3
setup channel10 loop
setup channel10 output #standard{mux=ts,access=udp,dst=239.0.0.10:8208}

new channel11 broadcast enabled
setup channel11 input .\videos\radio2.mp3
setup channel11 loop
setup channel11 output #standard{mux=ts,access=udp,dst=239.0.0.11:8208}

new channel12 broadcast enabled
setup channel12 input .\videos\radio3.mp3
setup channel12 loop
setup channel12 output #standard{mux=ts,access=udp,dst=239.0.0.12:8208}
```

Una vez tenemos configurados todos los flujos, debemos de iniciarlos:

```
control channel1 play
control channel2 play
control channel3 play
control channel4 play
control channel5 play
control channel6 play
control channel7 play
control channel8 play
control channel9 play
```

```
control channel10 play
control channel11 play
control channel12 play
```

También podemos ir parando algunos flujos en el caso necesario, cambiando el comando “play” por “stop”.

Configuración DHCP.

Para configurar el servidor DHCP, es necesario crear los siguientes archivos con la información que queramos que remita el servidor DHCP hacia los cliente, así como el interface/es donde se desee que este habilitado el servidor DHCP.

/etc/dhcp3/dhcpd.conf

```
#
# Archivo de configuración del servidor DHCP
#

option domain-name "example.org";
option domain-name-servers ns1.example.org, ns2.example.org;

default-lease-time 86400;
max-lease-time 604800;


subnet 192.168.1.0 netmask 255.255.255.0 {
        option routers                  192.168.1.1;
        option subnet-mask              255.255.255.0;

        option domain-name              "example.com";
        option domain-name-servers      192.168.1.1;

        option time-offset              -18000;

        range 192.168.1.10 192.168.1.100;
}
```

/etc/default/dhcp3-server

```
# Interfaces en los cuales estará escuchando el servidor DHCP

INTERFACES="eth1"
```

ANEXO IV. Código diseñado

A continuación se mostrara el código C diseñado para el proyecto.

Serverxml2.c

```
/*****************************************************************

                            SERVIDOR DE PAQUETES DVBSTP

Servidor diseñado para enviar por independiente los paquetes
- Service Selection & Discovery (SDS)
- Broadcast Record
- Package Record

Se puede indicar la dirección IP multicast queremos enviar cada uno de ellos
o bien con la ultima opción que envia constantemente una copia de cada uno de
estos documentos XML.

*****************************************************************/

#include <stdio.h>            // para fprintf()
#include <sys/socket.h>       // para socket(), connect(), send() y recv()
#include <arpa/inet.h>// para sockaddr_in and inet_addr()
#include <stdlib.h>           // para atoi() y exit()
#include <string.h>           // para memset()
#include <unistd.h>           // para sleep()
#include <dirent.h>           // para dir()
#include <sys/stat.h>
#define TAMA_DATA 1388        // Tamaño máximo útil del payload
#define SERVER_PORT 3937      // puerto DVBSTP por definición
#define TIME_WAIT 100         // Milisegundos de espera entre paquetes
#define IP_SDS "224.0.23.14"
#define IP_SP1 "239.0.1.1"

void DieWithError(char *errorMessage)
        {
        perror(errorMessage);
        exit(1);
        }

FILE * abrirFichero(char *NombreFichero)
        {
        FILE *elmio=NULL;
        char error[255];
        if ((elmio=fopen(NombreFichero, "r"))==NULL)        // Abrimos el fichero
                {
                sprintf(error,"Error abriendo fichero: %s",NombreFichero);
                DieWithError (error);
                }
        return elmio;
        }

struct dvbstp
        {
        unsigned char vrec[1];              // Version, Reservado, Encriptación y Flag
CRC
        unsigned char totalsize[3];         // Tamaño total del segmento
        unsigned char payloadID[1];         // ID del Payload
        unsigned char segmentID[2];         // ID del Segmento
        unsigned char segmentversion[1];// Version del Segmento
        unsigned char sn_lsm[3];            // Numero de sección y Ultimo numero de
sección
        unsigned char cph[1];              // Compresión, Flag ID de Proveedor y Tamaño
cabecera privada
        unsigned char buffer[TAMA_DATA];// Payload, hasta un máximo de 1400 bytes
        } paquete;
```

```
////////////////////////////////////////////////////
int enviar(char *ip, char tipo)
	{
	int sock;                                                // Socket
	struct sockaddr_in multicastAddr;     // Dirección Multicast
	char *multicastIP;                                       // Dirección IP Multicast
	unsigned short multicastPort;         // Server port
	char *Fichero;                                           // fichero a enviar
	unsigned char multicastTTL;                    // TTL of multicast packets
	FILE *pf1;
	unsigned int totalpartes=0, parteactual=0, partetemporal=0;
	long filetam=0,valor=0,filetamtemporal=0;
	struct stat fs;
	int i;
	int loquefalta;
	int TamCabecera=12;
	DIR           *dir_p;
	struct dirent *dir_entry_p;

	multicastIP = ip;                                                              //
Dirección IP multicast
	multicastPort = SERVER_PORT;                                  // Puerto multicast
	if (tipo==0x2) Fichero = "BroadcastRecord.xml";     //0x2  es  el  PayloadID  del
Broadcast Record
	if (tipo==0x5) Fichero = "PackageRecord.xml";       //0x5  es  el  PayloadID  del
Package Record
	if (tipo==0x1) Fichero = "ServiceDiscovery.xml";//0x1 es el PayloadID del SD&S
	paquete.vrec[0] = 0x0;                                                 //     Version,
Reservado, Encriptación y Flag CRC puestos a 0
	paquete.payloadID[0] = tipo;
	paquete.segmentID[0] = 0xff;
	paquete.segmentID[1] = 0xff;
	multicastTTL = 10;

	// Creación del socket para enviar/recibir datagramas
	if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
		DieWithError("socket() failed");

	// Configuración del TTL del paquete multicast
	if  (setsockopt(sock,  IPPROTO_IP,  IP_MULTICAST_TTL,  (void  *)  &multicastTTL,
sizeof(multicastTTL)) < 0)
		DieWithError("setsockopt() failed");

	// Construcción de la estructura multicast
	memset(&multicastAddr, 0, sizeof(multicastAddr));          //         Inicializar
estructura a cero
	multicastAddr.sin_family = AF_INET;                                            //
Internet address family
	multicastAddr.sin_addr.s_addr = inet_addr(multicastIP);    // IP Multicast
	multicastAddr.sin_port = htons(multicastPort);                     //       Puerto
Multicast

	pf1=abrirFichero(Fichero);
	if (pf1!=NULL) // Enviamos el fichero
		{
		if (stat(Fichero, &fs) == 0)
			filetam=fs.st_size;
		else
			DieWithError("No se puede calcular el tamaño del fichero\n");

		parteactual=0;
		totalpartes=filetam/TAMA_DATA;                                 //       Partes
completas
		if (filetam%TAMA DATA!=0) totalpartes++;     // Le sumamos la ultima
		printf   ("\nFichero   %s   abierto.\nTamaño   fichero:   %dbytes   -   Tamaño
paquetes:        %dbytes        -        Total        paquetes        a        enviar:
%d\n",Fichero,filetam,sizeof(paquete),totalpartes);

		// Anotamos el tamaño del Segmento
		filetamtemporal=filetam;
		paquete.totalsize[2]= (filetamtemporal%256);
		filetamtemporal = filetamtemporal/256;
		paquete.totalsize[1] = (filetamtemporal%256);
		filetamtemporal = filetamtemporal/256;
		paquete.totalsize[0] = (filetamtemporal%256);

	printf         ("Total         Segment         Size         (Hexadecimal)         =
```

```
%02x%02x%02x\n",paquete.totalsize[0],paquete.totalsize[1],paquete.totalsize[2]);

                // Anotamos el numero de Sección y el numero de la ultima Sección
                partetemporal=totalpartes-1;
                paquete.sn_lsm[2]= (partetemporal%256);

                partetemporal=partetemporal/256;
                paquete.sn_lsm[1]=(partetemporal%256);

                printf          ("Last          Seccion          Number:          %02x-
%02x\n",paquete.sn_lsm[1],paquete.sn_lsm[2]);

                while (totalpartes>1 && parteactual+1<totalpartes)  // Si no ocupa mas de
un paquete UDP vamos al ultimo bloque
                        {
                        fread(paquete.buffer,TAMA_DATA,1,pf1);    //    Leer    el    buffer
correspondiente al fichero

                        // Anotamos el numero de Sección y el numero de la ultima Sección
                        partetemporal=parteactual;
                        paquete.sn_lsm[1]=(partetemporal%256)<<4;
                        paquete.sn_lsm[0]=(partetemporal%256)>>4;

                        printf      ("Enviando      paquete      (Hexadecimal):      %02x-
%02x\n",paquete.sn_lsm[0],paquete.sn_lsm[1]);

                        // Construir el paquete para enviar
                        if (sendto(sock, &paquete, sizeof(paquete), 0, (struct sockaddr *)
&multicastAddr, sizeof(multicastAddr)) != sizeof(paquete))
                                DieWithError("sendto() se han enviado menos bytes de los
esperados.");

                        usleep(TIME_WAIT * 1000); // Paramos el tiempo que hemos definido
para evitar sobrecarga en la red
                        parteactual++;
                        }

                // El ultimo trozo hay que enviarlo de "lo que falta"
                loquefalta=filetam%TAMA_DATA;
                if (loquefalta==0) loquefalta=TAMA_DATA;
                valor=fread(paquete.buffer,loquefalta,1,pf1);    //    Leer    el    buffer
correspondiente al fichero

                partetemporal=parteactual;
                paquete.sn_lsm[1]=(partetemporal%256)<<4;
                paquete.sn_lsm[0]=(partetemporal%256)>>4;
                printf        ("Enviando        paquete        (Hexadecimal):        %02x-
%02x\n\n",paquete.sn_lsm[0],paquete.sn_lsm[1]);
                printf("Tamaño ultimo trozo enviado: %dbytes\n",loquefalta);

                // Construir el paquete para enviar
                valor=sendto(sock, &paquete, loquefalta+TamCabecera, 0, (struct sockaddr
*) &multicastAddr, sizeof(multicastAddr));
                if (valor!=loquefalta+TamCabecera)
                        {
                        printf("sendto() se han enviado menos bytes de los esperados en el
ultimo bloque: Quedaban:%d <> enviados:%d",filetam%TAMA_DATA,valor);
                        exit(1);
                        }
                usleep(TIME_WAIT  * 1000);// Paramos el tiempo que hemos definido para
evitar sobrecarga en la red
                fclose (pf1);
                }
        }

void mostrar_menu()
        {
        printf("\n\nMenu:\n=====\n\n");
        printf("1.- Enviar SDS\n");
        printf("2.- Enviar Broadcast Record\n");
        printf("3.- Enviar Package Record\n");
        printf("4.- Enviar todos los documentos XML ciclicamente\n");
        printf("5.- Salir\n\n");
        printf("Escoge una opcion: ");
        }

int main(int argc, char *argv[])
```

```
        {
        int opcion,cuantas,i;
        char ip[16]="";
        char tipo,c;
        do
                {
                mostrar_menu();
                scanf("%i",&opcion);
                switch (opcion)
                        {
                        case 1:printf  ("\nDireccion ip a enviar SDS (por defecto %s):
",IP_SDS);
                                        c=getchar();
                                        c=getchar();
                                        if (c=='\n')
                                                {
                                                strcpy(ip,IP_SDS);
                                                }
                                        else
                                                {
                                                ip[0]=c;
                                                scanf ("%s",&ip[1]);
                                                }
                                        printf  ("\nCuantas veces lo quieres enviar (0 para
infinito)?");
                                        scanf("%i",&cuantas);
                                        if (cuantas==0)
                                                {
                                                printf
("\n\n*******************************************************\n");
                                                printf  ("\nEnviando datos a la direccion IP
%s\n",ip);
                                                printf
("\n*******************************************************\n");
                                                for (;;)enviar(ip, 0x1);
                                                }
                                        else
                                                {
                                                printf
("\n\n*******************************************************\n");
                                                printf  ("\nEnviando datos a la direccion IP
%s\n",ip);
                                                printf
("\n*******************************************************\n");
                                                for (i=0;i<cuantas;i++) enviar(ip, 0x1);
                                                }
                                        break;
                        case 2:printf  ("\nDireccion ip a enviar Broadcast Record( por
defecto %s): ",IP_SP1);
                                        c=getchar();
                                        c=getchar();
                                        if (c=='\n') strcpy(ip,IP_SP1);
                                        else
                                                {
                                                ip[0]=c;
                                                scanf ("%s",&ip[1]);
                                                }
                                        printf  ("\nCuantas veces lo quieres enviar (0 para
infinito)?");
                                        scanf("%i",&cuantas);
                                        if (cuantas==0)
                                                {
                                                printf
("\n\n*******************************************************\n");
                                                printf  ("\nEnviando datos a la direccion IP
%s\n",ip);
                                                printf
("\n*******************************************************\n");
                                                for (;;)enviar(ip, 0x2);
                                                }
                                        else
                                                {
                                                printf
("\n\n*******************************************************\n");
                                                printf  ("\nEnviando datos a la direccion IP
%s\n",ip);
                                                printf
```

```
("\n*********************************************************\n");
                                        for (i=0;i<cuantas;i++)
                                                {
                                                enviar(ip, 0x2);
                                                }
                                        }
                                break;
                        case 3:printf  ("\nDireccion ip a enviar Package Record (por
defecto %s): ",IP_SP1);
                                c=getchar();
                                c=getchar();
                                if (c=='\n') strcpy(ip,IP_SP1);
                                else
                                        {
                                        ip[0]=c;
                                        scanf ("%s",&ip[1]);
                                        }
                                printf ("\nCuantas veces lo quieres enviar (0 para
infinito)?");
                                scanf("%i",&cuantas);
                                if (cuantas==0)
                                        {
                                        printf
("\n\n*********************************************************\n");
                                        printf ("\nEnviando datos a la direccion IP
%s\n",ip);
                                        printf
("\n*********************************************************\n");
                                        for (;;)enviar(ip, 0x5);
                                        }
                                else
                                        {
                                        printf
("\n\n*********************************************************\n");
                                        printf ("\nEnviando datos a la direccion IP
%s\n",ip);
                                        printf
("\n*********************************************************\n");
                                        for (i=0;i<cuantas;i++)
                                                {
                                                enviar(ip, 0x5);
                                                }
                                        }
                                break;
                        case 4: printf ("Se envian todos los documentos XML");
                                for (;;)
                                        {
                                        enviar(IP_SDS, 0x1);
                                        enviar(IP_SP1, 0x2);
                                        enviar(IP_SP1, 0x5);
                                        }
                        case 5:exit( 1 );
                        default:        printf( "Opcion no valida" );
                                        break;
                        }
                } while (opcion!=5);
        }
```

Capturador.c

```
/*****************************************************************

                        CAPTURADOR DE PAQUETES DVBSTP

Cliente diseñado para poder capturar por independiente los paquetes
- Service Selection & Discovery (SDS)
- Broadcast Record
- Package Record

Se ha de indicar por que direccion IP multicast queremos recibir cada uno de ellos

*****************************************************************/
```

```
#include <sys/types.h>
#include <sys/socket.h>              // para socket(), connect(), send() y recv()
#include <netinet/in.h>              // para sockaddr_in and inet_addr()
#include <arpa/inet.h>        // para sockaddr_in and inet_addr()
#include <netdb.h>
#include <stdio.h>                   // para fprintf()
#include <stdlib.h>                  // para atoi() y exit()
#include <unistd.h>           // para sleep()
#define SERVER_PORT 3937

int capturar (char *ip, char tipo)
        {
        int sd, rc, n, cliLen,fin=1;
        struct ip_mreq mreq;
        struct sockaddr_in cliAddr, servAddr;
        struct in_addr mcastAddr;
        struct hostent *h;
        char buf[1500];
        int contador=0;
        int currentSegID, lastSegID;
        char archivo[20];
        FILE *fd;
        char comando[30]="explorer ";

        /* Se obtiene la direccion IP multicast a escuchar */
        h=gethostbyname(ip);
        if(h==NULL)
                {
                printf ("\n**********************************************\n");
                printf("Direccion IP mal introducida '%s'\n",ip);
                printf ("**********************************************\n");
                return (1);
                }
        memcpy(&mcastAddr, h->h_addr_list[0],h->h_length);

        /* Verifica si es una direccion IP multicast */
        if(!IN MULTICAST(ntohl(mcastAddr.s_addr)))
                {
                printf("La direccion IP '%s' no es multicast\n", inet_ntoa(mcastAddr));
                return(1);
                }

        /* crear el socket */
        sd = socket(AF_INET,SOCK_DGRAM,0);
        if(sd<0)
                {
                printf("Imposible crear el Socket\n");
                return(1);
                }

        /* bind del puerto */
        servAddr.sin_family=AF_INET;
        servAddr.sin_addr.s_addr=htonl(INADDR_ANY);
        servAddr.sin_port=htons(SERVER_PORT);
        if(bind(sd,(struct sockaddr *) &servAddr, sizeof(servAddr))<0)
                {
                printf("No se puede realizar el bind al puerto %d \n",SERVER_PORT);
                return(1);
                }

        /* join al grupo multicast */
        mreq.imr_multiaddr.s_addr=mcastAddr.s_addr;
        mreq.imr_interface.s_addr=htonl(INADDR_ANY);
        rc = setsockopt(sd,IPPROTO_IP,IP_ADD_MEMBERSHIP,(void *) &mreq, sizeof(mreq));

        if(rc<0)
                {
                printf("No se ha podido realizar el join al grupo multicast
'%s'",inet_ntoa(mcastAddr));
                return (1);
                }
        else
                {
                printf("Servidor del cliente : escuchando al grupo multicast %s:%d\n",
inet_ntoa(mcastAddr), SERVER_PORT);
                cliLen = sizeof(cliAddr);
```

```
                if (tipo==0x2) strcpy (archivo,"BroadcastRecord.xml");
                if (tipo==0x5) strcpy (archivo,"PackageRecord.xml");
                if (tipo==0x1) strcpy (archivo,"ServiceDiscovery.xml");

                /* bucle para capturar paquetes */

                remove (archivo);
                fd = fopen(archivo, "a");
                fin = 1;
                while (fin)
                        {
                        n       =       recvfrom(sd,buf,sizeof(buf),0,(struct       sockaddr
*)&cliAddr,&cliLen); /* recibimos un paquete UDP*/
                        currentSegID = ((buf[8]&0x0000ff)<<4) + ((buf[9] & 0x0000ff)>> 4);
/* convertimos a int el Section Number*/
                        lastSegID   =   ((buf[9]&0x00000f)<<8)   +   (buf[10]&0x0000ff);   /*
convertimos a int el Last Section Number*/
                        printf ("valor de current es: %i\n", currentSegID);
                        printf ("valor de last es: %i\n", lastSegID);
                        printf (" PayloadID es: %x\n",buf[4]);

                        if (buf[4]==tipo)
                                {
                                if (currentSegID < lastSegID+1 && currentSegID == contador)
                                        {
                                        buf[n]='\0';
                                        if (buf[0]!=0)fwrite( &buf[12], n-16, 1, fd ); /*
si tiene CRC no capturamos los ultimos 32 bits*/
                                        else fwrite( &buf[12], n-12, 1, fd ); /* si no
tiene CRC capturamos todo el payload */
                                        printf   ("PAQUE   DESEADO:   Num   bytes   recibidos
%d\n\n",n);
                                        contador++;
                                        }
                                }
                        else   printf   ("***NO   ES   PAQUETE   DEL   PAYLOAD   QUE   DESEAMOS
****\n\n");

                        if   ((currentSegID   ==   lastSegID)   &&   (contador   ==   lastSegID+1))
fin=0; // Si tenemos todos los paquetes finalizamos bucle
                        }
                fclose(fd);
                close (sd);
                strcat (comando,archivo);
                system (comando);
                }
        return (0);
        }

int validar(int val)
        {
        if (val == 1)  // Validamos el BroadcastRecord.xml
                {
                if (!fork())
                        {
                        execlp ("xml","xml.exe", "val", "BroadcastRecord.xml", NULL);
                        exit (0) ;
                        }
                wait();
                }
        if (val == 2)  // Validamos el PackageRecord.xml
                {
                if (!fork())
                        {
                        execlp ("xml","xml.exe", "val", "PackageRecord.xml", NULL);
                        exit (0) ;
                        }
                wait();
                }
        if (val == 3)  // Validamos el ServiceDiscovery.xml
                {
                if (!fork())
                        {
                        execlp ("xml","xml.exe", "val", "ServiceDiscovery.xml", NULL);
                        exit (0) ;
                        }
                wait();
```

```
                }
        if (val == 4)  // Validamos todos los XML
                {
                if (!fork())
                        {
                        execlp ("xml","xml.exe", "val", "*.xml", NULL);
                        exit (0) ;
                        }
                wait();
                }
        return ;
        }

void mostrar_menu()
        {
        printf("\n\nMenu:\n=====\n\n");
        printf("1.- Capturar SDS\n");
        printf("2.- Validar SDS\n");
        printf("3.- Captura Broadcast Record\n");
        printf("4.- Validar Broadcast Record\n");
        printf("5.- Capturar Package Record\n");
        printf("6.- Validar Package Record\n");
        printf("7.- Validar TODOS\n");
        printf("8.- Salir\n\n");
        printf("Escoge una opcion: ");
        }

int main(int argc, char *argv[])
        {
        int opcion;
        char ip[16];
        char tipo;
        do
                {
                mostrar_menu();
                scanf("%i",&opcion);
                switch (opcion)
                        {
                        case 1:printf ("\nDireccion ip a capturar SDS: ");
                                scanf ("%s",ip);
                                capturar(ip, 0x1);
                                break;
                        case 2:printf("************************************* \n      "
);
                                validar(3);
                                printf("*************************************"
);
                                break;
                        case 3:printf ("\nDireccion ip a capturar Broadcast Record: ");
                                scanf ("%s",ip);
                                capturar(ip, 0x2);
                                break;
                        case 4: printf("************************************* \n      "
);
                                validar(1);
                                printf("*************************************"
);
                                break;
                        case 5: printf ("\nDireccion ip a capturar Package Record: ");
                                scanf ("%s",ip);
                                capturar(ip, 0x5);
                                break;
                        case 6: printf("************************************* \n      "
);
                                validar(2);
                                printf("*************************************"
);
                                break;
                        case 7: printf("************************************* \n      "
);
                                validar(4);
                                printf("*************************************"
);
                                break;
                        case 8: exit( 1 );
                        default:        printf( "Opcion no valida" );
                                        break;
```

```
                    }
                }
        while (opcion!=8);
        }
```

ParserCompleto.c

```
/*****************************************************************

                            PARSER XML DE DOCUMENTOS DVB-IP

Aplicacion diseñada para realizar el "parseo" de la informacion que contienen
los documentos XML. Obteniendo la informacion de interes de cada uno de ellos

*****************************************************************/

#include <stdio.h>
#include <expat.h>
#if defined(  amigaos  ) && defined(__USE_INLINE__)
#include <proto/expat.h>
#endif
#ifdef XML_LARGE_SIZE
#if defined(XML_USE_MSC_EXTENSIONS) && _MSC_VER < 1400
#define XML_FMT_INT_MOD "I64"
#else
#define XML_FMT_INT_MOD "ll"
#endif
#else
#define XML_FMT_INT_MOD "l"
#endif

#define BUFFSIZE        8192

char Buff[BUFFSIZE];
int Depth;
int HayLCN=0;
int HayShortName=0;
int Name=0;
FILE *fd;
FILE *fd2;
int j;
int contador=0;
int canalmax=0;

struct estructura_canales
        {
        char ServiceName[10];
        char canal[10];
        } canales;

struct tabla_canales
        {
        int contadorcanales;
        struct estructura_canales ncanal[1000];
        }tablacanales;

struct tabla_canales tablaaux;

struct estructura_Servicios
        {
        char Port[5];
        char Address[15];
        char ServiceName[10];
        int ServiceType;
        char Name[15];
        char Short[8];
        } servicios;

struct tabla_Servicios
        {
        int contadorServicios;
        struct estructura_Servicios serv[1000];
```

```
        }tablaservicios;

struct estructura_SDS
        {
        char DomainName[30];
        int Version;
        char Address[15];
        char Port[5];
        int numSP;
        } sds;

struct tabla_SDS
        {
        int count;
        struct estructura_SDS nsds[100];
        } tablaSP;

struct estructura_SDS SPelegido;

static void XMLCALL startSDS(void *data, const char *el, const char **attr)
        {
        int i;
        for (i = 0; i < Depth; i++)
        if (strcmp(el,"ServiceProviderDiscovery")==0)
                {
                printf ("Encontrado PackageDiscovery y contador puesto a cero\n\n");
                tablaSP.count=0;
                }
        if (strcmp(el, "Push")==0)
                {
                for (i = 0; attr[i]; i += 2)
                        {
                        if (strcmp(attr[i],"Address")==0) /* si es la direccion IP la
almacenamos en la estructura*/
                                {
                                strcpy (tablaSP.nsds[tablaSP.count].Address,attr[i+1]);
                                }
                        if (strcmp(attr[i],"Port")==0) /* si es el puerto lo almacenamos
en la estructura*/
                                {
                                strcpy (tablaSP.nsds[tablaSP.count].Port ,attr[i+1]);
                                }
                        }
                tablaSP.count++; /* Aumentamos contador de Services Providers*/
                tablaSP.nsds[tablaSP.count-1].numSP = tablaSP.count;
                }
        if (strcmp(el, "ServiceProvider")==0)
                {
                for(i=0;attr[i];i+=2)
                        {
                        if (strcmp(attr[i],"Version")==0)
                                {
                                tablaSP.nsds[tablaSP.count].Version = atoi (attr[i+1]); /*
guardamos el numero de version del SP*/
                                for(i=0;attr[i];i+=2)
                                        {
                                        if (strcmp(attr[i],"DomainName")==0) /* guardamos
el DomainName del SP*/
                                                {

        strcpy(tablaSP.nsds[tablaSP.count].DomainName,attr[i+1]);
                                                }
                                        }
                                }
                        }
                }
        Depth++;
        }

static void XMLCALL endSDS(void *data, const char *el)
        {
        Depth--;
        }

static void XMLCALL startLCN(void *data, const char *el, const char **attr)
        {
        int i;
```

```
        if (strcmp(el,"PackageDiscovery")==0)
                {
                printf ("Encontrado PackageDiscovery y contador puesto a cero\n");
                contador=0;
                }
        if (strcmp(el, "TextualID")==0)
                {
                if (strcmp (attr[0], "ServiceName")==0)
                        {
                        strcpy    (tablacanales.ncanal[contador].ServiceName,attr[1]);    /*
almacenamos el ServiceName */
                        }
                }
        if (strcmp(el, "LogicalChannelNumber")==0)
                {
                HayLCN=1; /* Ponemos marcador para obtener el valor del Logical Channel
Number*/
                }
        else (HayLCN=0);
        Depth++;
        }

static void XMLCALL endLCN(void *data, const char *el)
        {
        Depth--;
        }

static void XMLCALL charHandlerLCN(void *data, const XML_Char *s, int len)
        {
        if (HayLCN==1)
                {
                HayLCN=0;
                char* str = (char*)malloc( len + sizeof( char ) );
                memset( str, '\0', len + sizeof( char ) );
                strncpy( str, s, len );
                strcpy(tablacanales.ncanal[contador].canal,str); /* Se almacena el numero
de LCN*/
                contador++;
                tablacanales.contadorcanales=contador; /* Se aumenta contador de canales
encontrados */
                free( str );
                }
        }

static void XMLCALL startBroadcast(void *data, const char *el, const char **attr)
        {
        int i;

        for (i = 0; i < Depth; i++)
        if(strcmp(el, "ServiceList")==0)
                {
                tablaservicios.contadorServicios=0;  /* Contador de services*/
                }
        if(strcmp(el, "SingleService")==0)
                {
                tablaservicios.contadorServicios++;
                }
        if (strcmp(el, "IPMulticastAddress")==0)
                {
                for (i = 0; attr[i]; i += 2)
                        {
            if (strcmp (attr[i], "Port")==0)  // Guardamos el Puerto
                                {
                                strcpy
(tablaservicios.serv[tablaservicios.contadorServicios-1].Port,attr[i+1]);
                                }
                        if (strcmp(attr[i],"Address")==0)  // guardamos la IP Multicast
                                {

        strcpy(tablaservicios.serv[tablaservicios.contadorServicios-
1].Address,attr[i+1]);
                                }
                        }
                }
        if (strcmp(el, "ShortName")==0)
                {
```

```
                HayShortName=1;  // Marcador para ShortName
                }
        else (HayShortName=0);

        if (strcmp(el, "Name")==0)
                {
                Name=1;  // Marcador para Name
                }
        else (Name=0);

        if (strcmp(el, "TextualIdentifier")==0)
                {
                for (i = 0; attr[i]; i += 2)
                        {
                        if  (strcmp  (attr[i],  "ServiceName")==0)     //  Almacenamos  el
ServiceName
                                {
                                strcpy
(tablaservicios.serv[tablaservicios.contadorServicios-1].ServiceName,attr[i+1]);
                                }
                        }
                }
        Depth++;
        }

static void XMLCALL endBroadcast(void *data, const char *el)
        {
        Depth--;
        }

static void XMLCALL charHandlerBroadcast(void *data, const XML_Char *s, int len)
        {
        if (HayShortName==1)  // Si hay marcador de ShortName lo guardamos
                {
                HayShortName=0;
                char* str = (char*)malloc( len + sizeof( char ) );
                memset( str, '\0', len + sizeof( char ) );
                strncpy( str, s, len );
                strcpy(tablaservicios.serv[tablaservicios.contadorServicios-
1].Short,str);
                free( str );
                }

        if (Name==1) // Si hay marcador de Name lo guardamos
                {
                Name=0;
                char* str = (char*)malloc( len + sizeof( char ) );
                memset( str, '\0', len + sizeof( char ) );
                strncpy( str, s, len );
                strcpy(tablaservicios.serv[tablaservicios.contadorServicios-1].Name,str);
                free( str );
                }
        }

void guardarSDS ()
        {
        fd  =  fopen("parserSDS.log",  "w");  /*  almacenamos  en  un  log  el  resultado  del
parseo del SDS*/
        fwrite(&tablaSP,sizeof (tablaSP), 1, fd);
        fclose(fd);
        }

void leerSDS ()
        {
        int i;
        int option;

        fd = fopen( "parserSDS.log", "r" );
        if (fread( &tablaSP, sizeof(tablaSP), 1, fd ))
                {
                printf ("*******************************************\n");
                printf ("Lista de Proveedores de Servicio encontrados: \n\n");
                for (i=0;i<tablaSP.count;i++)
                        {
                        printf( "%i:   ", tablaSP.nsds[i].numSP);
                        printf( "Proveedor: %s    \n", tablaSP.nsds[i].DomainName);
                        }
```

```
                printf ("*******************************************\n\n");
                printf("Selecciona el numero de Proveedor de servicios que deseas: "); /*
Se solicita al usuario que elija un Service Provideer */
                scanf( "%i", &option );
                if ((option < 1) || (option > tablaSP.count))
                        {
                        printf ("\n***************\n");
                        printf ("opcion no valida\n");
                        printf ("***************\n");
                        option=0;
                        return;
                        }
                printf ("\nService Provider elegido: %i\n",option);
                SPelegido.numSP = tablaSP.nsds[option-1].numSP;
                printf( "Numero %i:   \n", SPelegido.numSP);
                strcpy(SPelegido.DomainName, tablaSP.nsds[option-1].DomainName);
                printf( "Domain Name: %s    \n", SPelegido.DomainName);
                SPelegido.Version = tablaSP.nsds[option-1].Version;
                printf( "Version: %i    \n", SPelegido.Version);
                strcpy (SPelegido.Address, tablaSP.nsds[option-1].Address);
                printf( "adress: %s    \n", SPelegido.Address);
                strcpy (SPelegido.Port,tablaSP.nsds[option-1].Port);
                printf("port: %s\n",SPelegido.Port);
                }
        else("No se puede abrir el archivo parserSDS.log");
  fclose(fd);
        }

void guardarBroadcast ()
        {
        fd = fopen("parser.log", "w"); /* almacenamos en un log el resultado del parseo
del Broadcast Record*/
        fwrite(&tablaservicios,sizeof (tablaservicios), 1, fd);
        fclose(fd);
        }

void leerBroadcast()
        {
        fd = fopen( "parser.log", "r" );
        if (fd==NULL)
                {
                printf( "No se puede abrir el fichero parser.log\n" );
                exit(1);
                }
        if (fread( &tablaservicios, sizeof(tablaservicios), 1, fd ))
                {
                for (j=0;j<tablaservicios.contadorServicios;j++)
                        {
                        printf( "Port: %s\n", tablaservicios.serv[j].Port);
                        printf( "Adrress: %s\n", tablaservicios.serv[j].Address);
                        printf( "ServiceName: %s\n", tablaservicios.serv[j].ServiceName);
                        printf( "Name: %s\n", tablaservicios.serv[j].Name);
                        printf( "ShortName: %s\n\n", tablaservicios.serv[j].Short);
                        }
                printf("contador servicios: %i\n",tablaservicios.contadorServicios);
                }
        else printf ("imposible leer parser.log");
        fclose(fd);
        }

void guardarLCN ()
        {
        fd = fopen("parserLCN.log", "w");
        if (fd==NULL)
                {
                printf( "No se puede abrir el fichero parserLCN.log\n" );
                exit(1);
                }
        fwrite(&tablacanales,sizeof (tablacanales), 1, fd);
        fclose(fd);
        }

void ordenarLCN()
        {
        canalmax=0;
        fd = fopen( "parserLCN.log", "r" );
        if (fd==NULL)
```

```
                {
                printf( "No se puede abrir el fichero parserLCN.log\n" );
                exit(1);
                }
        if (fread(&tablacanales, sizeof(tablacanales), 1, fd ))
                {
                tablaaux.contadorcanales=0;
                for (j=0;j<tablacanales.contadorcanales;j++)
                        {
                        strcpy  (tablaaux.ncanal[j].ServiceName,"");  //Icializamos  tabla
aux a vacio
                        strcpy  (tablaaux.ncanal[j].canal,"");  //Icializamos  tabla  aux  a
vacio
                        }
                for (j=0;j<tablacanales.contadorcanales;j++) // Copia solo canales que no
estan repetidos
                        {
                        if                                                              (strcmp
(tablaaux.ncanal[atoi(tablacanales.ncanal[j].canal)].canal,
tablacanales.ncanal[j].canal))
                                {
                                strcpy
(tablaaux.ncanal[atoi(tablacanales.ncanal[j].canal)].canal                              ,
tablacanales.ncanal[j].canal);

        strcpy(tablaaux.ncanal[atoi(tablacanales.ncanal[j].canal)].ServiceName,
tablacanales.ncanal[j].ServiceName);
                                tablaaux.contadorcanales++;
                                }
                        if (atoi(tablacanales.ncanal[j].canal) > canalmax)
                                {
                                canalmax= atoi(tablacanales.ncanal[j].canal);   // copiamos
el valor del canal mas alto
                                }
                        }
                for (j=0;j<canalmax+1;j++)
                        {
                        if (strcmp (tablaaux.ncanal[j].ServiceName,""))   // Mostrar solo
los que no son vacios
                                {
                                printf    ("Canal    numero    =    %i                    ",
atoi(tablaaux.ncanal[j].canal));
                                printf                    ("ServiceName                 =
%s\n",tablaaux.ncanal[j].ServiceName);
                                }
                        }
                }
        else printf ("imposible leer parserLCN.log");
        fclose(fd);
        }

void visualizar()
        {
        int i;
        fd  =  fopen("lista.m3u",  "w");  /*  Se  genera  una  lista  de  reproduccion  para
visualizar con VLC */
        if (tablaservicios.contadorServicios<1)
                {
                printf ("\n\nHa de realizar el parseo del Broadcast Record y del Package
Record antes para poder ver la lista de canales\n\n");
                system ("pause");
                return;
                }
        if (fd==NULL)
                {
                printf( "No se puede abrir el fichero lista.m3u\n" );
                exit(1);
                }
        for (i=1;i<canalmax+1;i++)
                {
                if (strcmp (tablaaux.ncanal[i].ServiceName,""))   // Mostrar solo los que
no son vacios
                        {
                        fputs ("#EXTINF:-1,[",fd);
                        fputs (tablaaux.ncanal[i].canal,fd);
                        fputs ("]",fd);
                        for (j=0;j<canalmax+1;j++)
```

```
                                {
                                if          (strcmp          (tablaaux.ncanal[i].ServiceName,
tablaservicios.serv[j].ServiceName)==0)
                                        {
                                        fputs (tablaservicios.serv[j].Name,fd);
                                        fputs ("\nudp://@",fd);
                                        fputs (tablaservicios.serv[j].Address,fd);
                                        fputs (":",fd);
                                        fputs (tablaservicios.serv[j].Port,fd);
                                        }
                                }
                        fputs ("\n",fd);
                        }
                }
        fclose(fd);
        printf ("lista de reproduccion realizada\n");
        system (".\\VLC\\vlc lista.m3u");
        }

void parserLCN()
        {
        //      **************  INICIO PARSER LCN      *****************
        XML_Parser p = XML_ParserCreate(NULL);
        if (! p)
                {
                fprintf(stderr,  "No  se  puede  reservar  memoria  suficiente  para  el
parser\n");
                exit(-1);
                }

        XML_SetElementHandler(p, startLCN, endLCN);
        XML_SetCharacterDataHandler(p, charHandlerLCN);

        fd2 = fopen( "PackageRecord.xml", "r" );
        if (fd2==NULL)
                {
                printf( "No se puede abrir el fichero PackageRecord.xml\n" );
                exit(-1);
                }

        for (;;)
                {
                int done;
                int len;

                len = (int)fread(Buff, 1, BUFFSIZE, fd2);
                if (ferror(fd2))
                        {
                        fprintf(stderr, "Error de lectura en PackageRecord.xml\n");
                        exit(-1);
                        }
                done = feof(fd2);
                if (XML_Parse(p, Buff, len, done) == XML_STATUS_ERROR)
                        {
                        fprintf(stderr,  "Parse  error  en  la  linea  %"  XML_FMT_INT_MOD
"u:\n%s\n", XML_GetCurrentLineNumber(p), XML_ErrorString(XML_GetErrorCode(p)));
                        exit(-1);
                        }
                if (done) break;
                }
        fclose(fd2);
        guardarLCN();
        ordenarLCN();
        XML_ParserFree(p);
        system ("pause");
//  **********************    FIN PARSER LCN  **********************
        }

void parserBroadcast()
        {
        // ********************** Inicio Parser Broadcast  ***********
        XML_Parser p2 = XML_ParserCreate(NULL);
        if (! p2)
                {
                fprintf(stderr,  "No  se  puede  reservar  memoria  suficiente  para  el
parser\n");
                exit(-1);
```

```
		}
	XML_SetElementHandler(p2, startBroadcast, endBroadcast);
	XML_SetCharacterDataHandler(p2, charHandlerBroadcast);

	fd2 = fopen( "BroadcastRecord.xml", "r" );
	for (;;)
		{
		int done;
		int len;
		len = (int)fread(Buff, 1, BUFFSIZE, fd2);

		if (ferror(fd2))
			{
			fprintf(stderr, "Error de lectura en BroadcastRecord.xml\n");
			exit(-1);
			}

		done = feof(fd2);

		if (XML_Parse(p2, Buff, len, done) == XML_STATUS_ERROR)
			{
			fprintf(stderr, "Parse error en la linea %" XML_FMT_INT_MOD
"u:\n%s\n", XML_GetCurrentLineNumber(p2), XML_ErrorString(XML_GetErrorCode(p2)));
			exit(-1);
			}

		if (done) break;
		}
	fclose(fd2);
	guardarBroadcast(tablaservicios);
	leerBroadcast (tablaservicios);
	XML_ParserFree(p2);
	system ("pause");
	// ************************ Fin Parser Broadcast ***************
	}

void parserSDS()
	{
	// ************************ INICIO PARSER SDS *****************
	XML_Parser p3 = XML_ParserCreate(NULL);
	if (! p3)
		{
		fprintf(stderr, "No se puede reservar memoria suficiente para el
parser\n");
		exit(-1);
		}

	XML_SetElementHandler(p3, startSDS, endSDS);
	fd2 = fopen( "ServiceDiscovery.xml", "r" );
	for (;;)
		{
		int done;
		int len;
		len = (int)fread(Buff, 1, BUFFSIZE, fd2);

		if (ferror(fd2))
			{
			fprintf(stderr, "Error de lectura en ServiceDiscovery.xml\n");
			exit(-1);
			}

		done = feof(fd2);

		if (XML_Parse(p3, Buff, len, done) == XML_STATUS_ERROR)
			{
			fprintf(stderr, "Parse error en la linea %" XML_FMT_INT_MOD
"u:\n%s\n", XML_GetCurrentLineNumber(p3), XML_ErrorString(XML_GetErrorCode(p3)));
			exit(-1);
			}

		if (done) break;
		}
	fclose(fd2);
	guardarSDS();
	leerSDS();
	XML_ParserFree(p3);
	system ("pause");
```

```
        // ******************** FIN PARSER SDS ************************
        }

 void mostrar_menu()
        {
        printf("\n\nMenu:\n=====\n\n");
        printf("1.- Parser del ServiceDiscovey\n");
        printf("2.- Parser del Broadcast Record\n");
        printf("3.- Parser del Package Record\n");
        printf("4.- Visualizar lista de canales en VLC\n");
        printf("5.- Salir\n\n");
        printf("Escoge una opcion: ");fflush(stdout);
        }

int main(int argc, char *argv[])
        {
        char opcion;
        do
                {
                system ("cls");
                mostrar menu();
                opcion = getch();
                switch ( opcion )
                        {
                        case '1':       parserSDS();
                                        break;
                        case '2':       parserBroadcast();
                                        break;
                        case '3':       parserLCN();
                                        break;
                        case '4':       visualizar();
                                        break;
                        case '5':       exit( 1 );
                        default:        printf( "Opcion no valida\n" );
                                         system ("pause");
                                        break;

                        }
                }
        while (opcion!='5');
        }
```

ClienteCompleto.c

```
/*****************************************************************

                        CLIENTE COMPLETO DVB-IP

Aplicacion donde se integran los diferentes modulos creados para la realización de todas
las funciones necesarias de una forma automatizada.

*****************************************************************/

#include <stdio.h>
#define IP_SDS "224.0.23.14"

struct estructura_SDS
        {
        char DomainName[30];
        int Version;
        char Address[15];
        char Port[5];
        int numSP;
        };

struct estructura_SDS SPelegido;

int main(int argc, char *argv[])
        {
        char ip[16]="";
        char c;

        printf ("\nDireccion ip del Entry Point (por defecto %s): ",IP_SDS);
        c=getchar();
        if (c=='\n')
                {
                strcpy(ip,IP_SDS);
                }
        else
                {
                ip[0]=c;
                scanf ("%s",&ip[1]);
                }
        capturar(IP_SDS, 0x1); // captura el SDS con la ip del DVB
        printf ("\ncapturado, ahora a validar\n");
        validar(3); // validamos el SDS
        printf ("\nvalidado, ahora a parserSDS\n");
        parserSDS();
        printf ("\nparseado SDS, ahora a capturar Broadcast\n");
        capturar(SPelegido.Address,0x2); // capturamos el Broadcast Record
        printf ("\ncapturado Broadcast, ahora a validar Broadcast\n");
        validar(1); // validamos el Broadcast Record
        printf ("\nvalidado Broadcast, ahora a parsear Broadcast\n");
        parserBroadcast();
        printf ("\nparseado Broadcast, ahora capturar Package Record\n");
        capturar(SPelegido.Address,0x5);// capturamos el Package Record
        printf ("\ncapturado Package Record, ahora a validar Package Record\n");
        validar(2); // validamos el Package Record
        printf ("\nvalidado Package Record, ahora a parsear Package Record\n");
        parserLCN();
        printf ("\nparseado Package Record, ahora a visualizar\n");
        visualizar();
        return 0;
}
```

Capturador.c (integrado en ClienteCompleto.c)

```
/*****************************************************************

        CAPTURADOR DE PAQUETES DVBSTP INTEGRADO EN CLIENTE COMPLETO

modulo diseñado para poder capturar los paquetes:
- Service Selection & Discovery (SDS)
- Broadcast Record
- Package Record

Se ha integrado este modulo en el cliente completo como funciones.

*****************************************************************/

#include <sys/types.h>
#include <sys/socket.h>              // para socket(), connect(), send() y recv()
#include <netinet/in.h>              // para sockaddr in and inet_addr()
#include <arpa/inet.h>         // para sockaddr_in and inet_addr()
#include <netdb.h>
#include <stdio.h>                   // para fprintf()
#include <stdlib.h>                  // para atoi() y exit()
#include <unistd.h>            // para sleep()
#define SERVER PORT 3937
#define IP_SDS "224.0.23.14"
#define IP_SP1 "239.0.1.1"

int capturar (char *ip, char tipo)
        {
        int sd, rc, n, cliLen,fin=1;
        struct ip_mreq mreq;
        struct sockaddr_in cliAddr, servAddr;
        struct in_addr mcastAddr;
        struct hostent *h;
        char buf[1500];
        int contador=0;
        int currentSegID, lastSegID;
        char archivo[20];
        FILE *fd;

        /* Se obtiene la direccion IP multicast a escuchar */
        h=gethostbyname(ip);
        if(h==NULL)
                {
                printf ("\n*********************************************\n");
                printf("Direccion IP mal introducida '%s'\n",ip);
                printf ("*********************************************\n");
                return (1);
                }
        memcpy(&mcastAddr, h->h_addr_list[0],h->h_length);

        /* Verifica si es una direccion IP multicast */
        if(!IN_MULTICAST(ntohl(mcastAddr.s_addr)))
                {
                printf("La direccion IP '%s' no es multicast\n", inet_ntoa(mcastAddr));
                return(1);
                }

        /* crear el socket */
        sd = socket(AF_INET,SOCK_DGRAM,0);
        if(sd<0)
                {
                printf("Imposible crear el Socket\n");
                return(1);
                }

        /* bind del puerto */
        servAddr.sin_family=AF_INET;
        servAddr.sin_addr.s_addr=htonl(INADDR_ANY);
        servAddr.sin port=htons(SERVER PORT);
        if(bind(sd,(struct sockaddr *) &servAddr, sizeof(servAddr))<0)
                {
                printf("No se puede realizar el bind al puerto %d \n",SERVER_PORT);
```

```
                return(1);
                }

        /* join al grupo multicast */
        mreq.imr_multiaddr.s_addr=mcastAddr.s_addr;
        mreq.imr_interface.s_addr=htonl(INADDR_ANY);
        rc = setsockopt(sd,IPPROTO_IP,IP_ADD_MEMBERSHIP,(void *) &mreq, sizeof(mreq));

        if(rc<0)
                {
                printf("No se ha podido realizar el join al grupo multicast
'%s'",inet_ntoa(mcastAddr));
                return (1);
                }
        else
                {
                printf("Cliente escuchando a la direccion IP multicast %s:%d\n",
inet ntoa(mcastAddr), SERVER PORT);
                cliLen = sizeof(cliAddr);

                if (tipo==0x2) strcpy (archivo,"BroadcastRecord.xml");
                if (tipo==0x5) strcpy (archivo,"PackageRecord.xml");
                if (tipo==0x1) strcpy (archivo,"ServiceDiscovery.xml");

                /* bucle para capturar paquetes */

                remove (archivo);
                fd = fopen(archivo, "a");
                fin = 1;
                while (fin)
                        {
                        n       =       recvfrom(sd,buf,sizeof(buf),0,(struct       sockaddr
*)&cliAddr,&cliLen); /* recibimos un paquete UDP*/
                        currentSegID = ((buf[8]&0x0000ff)<<4) + ((buf[9] & 0x0000ff)>> 4);
/* convertimos a int el Section Number*/
                        lastSegID   =   ((buf[9]&0x00000f)<<8)   +   (buf[10]&0x0000ff);   /*
convertimos a int el Last Section Number*/
                        printf ("valor de current es: %i\n", currentSegID);
                        printf ("valor de last es: %i\n", lastSegID);
                        printf (" PayloadID es: %x\n",buf[4]);

                        if (buf[4]==tipo)
                                {
                                if (currentSegID < lastSegID+1 && currentSegID == contador)
                                        {
                                        buf[n]='\0';
                                        if (buf[0]!=0)fwrite( &buf[12], n-16, 1, fd ); /*
si tiene CRC no capturamos los ultimos 32 bits*/
                                        else fwrite( &buf[12], n-12, 1, fd ); /* si no
tiene CRC capturamos todo el payload */
                                        printf   ("PAQUE   DESEADO:   Num   bytes   recibidos
%d\n\n",n);
                                        contador++;
                                        }
                                }
                        else   printf   ("***NO   ES   PAQUETE   DEL   PAYLOAD   QUE   DESEAMOS
****\n\n");

                        if   ((currentSegID   ==   lastSegID)   &&   (contador   ==   lastSegID+1))
fin=0; // Si tenemos todos los paquetes finalizamos bucle
                        }
                fclose(fd);
                close (sd);
                }
        return (0);
        }

int validar(int val)
        {
        if (val == 1)  // Validamos el BroadcastRecord.xml
                {
                if (!fork()) // creamos un proceso paralelo para la validacion
                        {
                        execlp ("xml","xml.exe", "val", "BroadcastRecord.xml", NULL);
                        exit (0) ;
                        }
                wait();
```

```
        }
if (val == 2)  // Validamos el PackageRecord.xml
        {
        if (!fork()) // creamos un proceso paralelo para la validacion
                {
                execlp ("xml","xml.exe", "val", "PackageRecord.xml", NULL);
                exit (0) ;
                }
        wait();
        }
if (val == 3)  // Validamos el ServiceDiscovery.xml
        {
        if (!fork()) // creamos un proceso paralelo para la validacion
                {
                execlp ("xml","xml.exe", "val", "ServiceDiscovery.xml", NULL);
                exit (0) ;
                }
        wait();
        }
if (val == 4)  // Validamos todos los XML
        {
        if (!fork()) // creamos un proceso paralelo para la validacion
                {
                execlp ("xml","xml.exe", "val", "*.xml", NULL);
                exit (0) ;
                }
        wait();
        }
return ;
}
```

ParserCompleto.c (Integrado en ClienteCompleto.c)

```
/*****************************************************************

        PARSER XML DE DOCUMENTOS DVB-IP INTEGRADO EN CLIENTE COMPLETO

Modulo diseñado para realizar el "parseo" de la informacion que contienen los documentos
XML. Obteniendo la informacion de interes de cada uno de ellos.

Se ha integrado este modulo en el cliente completo como funciones.

*****************************************************************/

#include <stdio.h>
#include <expat.h>
#if defined(  amigaos  ) && defined(__USE_INLINE__)
#include <proto/expat.h>
#endif
#ifdef XML_LARGE_SIZE
#if defined(XML_USE_MSC_EXTENSIONS) && _MSC_VER < 1400
#define XML_FMT_INT_MOD "I64"
#else
#define XML_FMT_INT_MOD "ll"
#endif
#else
#define XML_FMT_INT_MOD "l"
#endif

#define BUFFSIZE        8192

char Buff[BUFFSIZE];
int Depth;
int HayLCN=0;
int HayShortName=0;
int Name=0;
FILE *fd;
FILE *fd2;
int j;
int contador=0;
int canalmax=0;

struct estructura_canales
        {
        char ServiceName[10];
        char canal[10];
        } canales;

struct tabla_canales
        {
        int contadorcanales;
        struct estructura_canales ncanal[1000];
        }tablacanales;

struct tabla_canales tablaaux;

struct estructura_Servicios
        {
        char Port[5];
        char Address[15];
        char ServiceName[10];
        int ServiceType;
        char Name[15];
        char Short[8];
        } servicios;

struct tabla_Servicios
        {
        int contadorServicios;
        struct estructura_Servicios serv[1000];
        }tablaservicios;

struct estructura_SDS
        {
        char DomainName[30];
        int Version;
```

```
        char Address[15];
        char Port[5];
        int numSP;
        } sds;

struct tabla_SDS
        {
        int count;
        struct estructura_SDS nsds[100];
        } tablaSP;

struct estructura_SDS SPelegido;

static void XMLCALL startSDS(void *data, const char *el, const char **attr)
        {
        int i;
        for (i = 0; i < Depth; i++)
        if (strcmp(el,"ServiceProviderDiscovery")==0)
                {
                printf   ("Encontrado   ServiceProviderDiscovery   y   contador   puesto   a
cero\n\n");
                tablaSP.count=0;
                }
        if (strcmp(el, "Push")==0)
                {
                for (i = 0; attr[i]; i += 2)
                        {
                        if  (strcmp(attr[i],"Address")==0)  /*  si  es  la  direccion  IP  la
almacenamos en la estructura*/
                                {
                                strcpy (tablaSP.nsds[tablaSP.count].Address,attr[i+1]);
                                }
                        if  (strcmp(attr[i],"Port")==0)  /*  si es el puerto lo almacenamos
en la estructura*/
                                {
                                strcpy (tablaSP.nsds[tablaSP.count].Port ,attr[i+1]);
                                }
                        }
                tablaSP.count++; /* Aumentamos contador de Services Providers*/
                tablaSP.nsds[tablaSP.count-1].numSP = tablaSP.count;
                }
        if (strcmp(el, "ServiceProvider")==0)
                {
                for(i=0;attr[i];i+=2)
                        {
                        if (strcmp(attr[i],"Version")==0)
                                {
                                tablaSP.nsds[tablaSP.count].Version = atoi (attr[i+1]); /*
guardamos el numero de version del SP*/
                                for(i=0;attr[i];i+=2)
                                        {
                                        if  (strcmp(attr[i],"DomainName")==0)  /*  guardamos
el DomainName del SP*/
                                                {

        strcpy(tablaSP.nsds[tablaSP.count].DomainName,attr[i+1]);
                                                }
                                        }
                                }
                        }
                }
        Depth++;
        }

static void XMLCALL endSDS(void *data, const char *el)
        {
        Depth--;
        }

static void XMLCALL startLCN(void *data, const char *el, const char **attr)
        {
        int i;

        if (strcmp(el,"PackageDiscovery")==0)
                {
                printf ("Encontrado PackageDiscovery y contador puesto a cero\n");
                contador=0;
```

```
		}
	if (strcmp(el, "TextualID")==0)
		{
		if (strcmp (attr[0], "ServiceName")==0)
			{
			strcpy	(tablacanales.ncanal[contador].ServiceName,attr[1]);	/*
almacenamos el ServiceName */
			}
		}
	if (strcmp(el, "LogicalChannelNumber")==0)
		{
		HayLCN=1; /* Ponemos marcador para obtener el valor del Logical Channel
Number*/
		}
	else (HayLCN=0);
	Depth++;
	}

static void XMLCALL endLCN(void *data, const char *el)
	{
	Depth--;
	}

static void XMLCALL charHandlerLCN(void *data, const XML_Char *s, int len)
	{
	if (HayLCN==1)
		{
		HayLCN=0;
		char* str = (char*)malloc( len + sizeof( char ) );
		memset( str, '\0', len + sizeof( char ) );
		strncpy( str, s, len );
		strcpy(tablacanales.ncanal[contador].canal,str); /* Se almacena el numero
de LCN*/
		contador++;
		tablacanales.contadorcanales=contador; /* Se aumenta contador de canales
encontrados */
		free( str );
		}
	}

static void XMLCALL startBroadcast(void *data, const char *el, const char **attr)
	{
	int i;

	for (i = 0; i < Depth; i++)
	if(strcmp(el, "ServiceList")==0)
		{
		tablaservicios.contadorServicios=0;  /* Contador de services*/
		}
	if(strcmp(el, "SingleService")==0)
		{
		tablaservicios.contadorServicios++;
		}
	if (strcmp(el, "IPMulticastAddress")==0)
		{
		for (i = 0; attr[i]; i += 2)
			{
			if (strcmp (attr[i], "Port")==0)  // Guardamos el Puerto
				{
				strcpy
(tablaservicios.serv[tablaservicios.contadorServicios-1].Port,attr[i+1]);
				}
			if (strcmp(attr[i],"Address")==0)  // guardamos la IP Multicast
				{

	strcpy(tablaservicios.serv[tablaservicios.contadorServicios-
1].Address,attr[i+1]);
				}
			}
		}
	if (strcmp(el, "ShortName")==0)
		{
		HayShortName=1;  // Marcador para ShortName
		}
	else (HayShortName=0);

	if (strcmp(el, "Name")==0)
```

```
                {
                Name=1;  // Marcador para Name
                }
        else (Name=0);

        if (strcmp(el, "TextualIdentifier")==0)
                {
                for (i = 0; attr[i]; i += 2)
                        {
                        if  (strcmp  (attr[i],  "ServiceName")==0)     //  Almacenamos  el
ServiceName
                                {
                                strcpy
(tablaservicios.serv[tablaservicios.contadorServicios-1].ServiceName,attr[i+1]);
                                }
                        }
                }
        Depth++;
        }

static void XMLCALL endBroadcast(void *data, const char *el)
        {
        Depth--;
        }

static void XMLCALL charHandlerBroadcast(void *data, const XML_Char *s, int len)
        {
        if (HayShortName==1)  // Si hay marcador de ShortName lo guardamos
                {
                HayShortName=0;
                char* str = (char*)malloc( len + sizeof( char ) );
                memset( str, '\0', len + sizeof( char ) );
                strncpy( str, s, len );
                strcpy(tablaservicios.serv[tablaservicios.contadorServicios-
1].Short,str);
                free( str );
                }

        if (Name==1) // Si hay marcador de Name lo guardamos
                {
                Name=0;
                char* str = (char*)malloc( len + sizeof( char ) );
                memset( str, '\0', len + sizeof( char ) );
                strncpy( str, s, len );
                strcpy(tablaservicios.serv[tablaservicios.contadorServicios-1].Name,str);
                free( str );
                }
        }

void guardarSDS ()
        {
        fd  =  fopen("parserSDS.log",  "w"); /* almacenamos en un log el resultado del
parseo del SDS*/
        fwrite(&tablaSP,sizeof (tablaSP), 1, fd);
        fclose(fd);
        }

void leerSDS ()
        {
        int i;
        int option;

        fd = fopen( "parserSDS.log", "r" );
        if (fread( &tablaSP, sizeof(tablaSP), 1, fd ))
                {
                printf ("*********************************************\n");
                printf ("Lista de Proveedores de Servicio encontrados: \n\n");
                for (i=0;i<tablaSP.count;i++)
                        {
                        printf( "%i.-   ", tablaSP.nsds[i].numSP);
                        printf( "Proveedor: %s    \n", tablaSP.nsds[i].DomainName);
                        }
                printf ("*********************************************\n\n");
                printf("Selecciona el numero de Proveedor de servicios que deseas: "); /*
Se solicita al usuario que elija un Service Provideer */
                scanf( "%i", &option );
                if ((option < 1) || (option > tablaSP.count))
```

```
                        {
                        printf ("\n***************\n");
                        printf ("opcion no valida\n");
                        printf ("***************\n");
                        option=0;
                        return;
                        }
                printf ("\nService Provider elegido: %i\n",option);
                SPelegido.numSP = tablaSP.nsds[option-1].numSP;
                printf( "Numero %i:   \n", SPelegido.numSP);
                strcpy(SPelegido.DomainName, tablaSP.nsds[option-1].DomainName);
                printf( "Domain Name: %s    \n", SPelegido.DomainName);
                SPelegido.Version = tablaSP.nsds[option-1].Version;
                printf( "Version: %i    \n", SPelegido.Version);
                strcpy (SPelegido.Address, tablaSP.nsds[option-1].Address);
                printf( "adress: %s    \n", SPelegido.Address);
                strcpy (SPelegido.Port,tablaSP.nsds[option-1].Port);
                printf("port: %s\n",SPelegido.Port);
                }
        else("No se puede abrir el archivo parserSDS.log");
  fclose(fd);
        }

void guardarBroadcast ()
        {
        fd = fopen("parser.log", "w"); /* almacenamos en un log el resultado del parseo
del Broadcast Record*/
        fwrite(&tablaservicios,sizeof (tablaservicios), 1, fd);
        fclose(fd);
        }

void leerBroadcast()
        {
        fd = fopen( "parser.log", "r" );
        if (fd==NULL)
                {
                printf( "No se puede abrir el fichero parser.log\n" );
                exit(1);
                }
        if (fread( &tablaservicios, sizeof(tablaservicios), 1, fd ))
                {
                for (j=0;j<tablaservicios.contadorServicios;j++)
                        {
                        printf( "Port: %s\n", tablaservicios.serv[j].Port);
                        printf( "Adrress: %s\n", tablaservicios.serv[j].Address);
                        printf( "ServiceName: %s\n", tablaservicios.serv[j].ServiceName);
                        printf( "Name: %s\n", tablaservicios.serv[j].Name);
                        printf( "ShortName: %s\n\n", tablaservicios.serv[j].Short);
                        }
                printf("contador servicios: %i\n",tablaservicios.contadorServicios);
                }
        else printf ("imposible leer parser.log");
        fclose(fd);
        }

void guardarLCN ()
        {
        fd = fopen("parserLCN.log", "w");
        if (fd==NULL)
                {
                printf( "No se puede abrir el fichero parserLCN.log\n" );
                exit(1);
                }
        fwrite(&tablacanales,sizeof (tablacanales), 1, fd);
        fclose(fd);
        }

void ordenarLCN()
        {
        canalmax=0;
        fd = fopen( "parserLCN.log", "r" );
        if (fd==NULL)
                {
                printf( "No se puede abrir el fichero parserLCN.log\n" );
                exit(1);
                }
        if (fread(&tablacanales, sizeof(tablacanales), 1, fd ))
```

```
                {
                tablaaux.contadorcanales=0;
                for (j=0;j<tablacanales.contadorcanales;j++)
                        {
                        strcpy  (tablaaux.ncanal[j].ServiceName,"");  //Icializamos  tabla
aux a vacio
                        strcpy  (tablaaux.ncanal[j].canal,"");  //Icializamos  tabla  aux  a
vacio
                        }
                for (j=0;j<tablacanales.contadorcanales;j++) // Copia solo canales que no
estan repetidos
                        {
                        if                                                              (strcmp
(tablaaux.ncanal[atoi(tablacanales.ncanal[j].canal)].canal,
tablacanales.ncanal[j].canal))
                                {
                                strcpy
(tablaaux.ncanal[atoi(tablacanales.ncanal[j].canal)].canal                              ,
tablacanales.ncanal[j].canal);

        strcpy(tablaaux.ncanal[atoi(tablacanales.ncanal[j].canal)].ServiceName,
tablacanales.ncanal[j].ServiceName);
                                tablaaux.contadorcanales++;
                                }
                        if (atoi(tablacanales.ncanal[j].canal) > canalmax)
                                {
                                canalmax= atoi(tablacanales.ncanal[j].canal);   // copiamos
el valor del canal mas alto
                                }
                        }
                for (j=0;j<canalmax+1;j++)
                        {
                        if  (strcmp  (tablaaux.ncanal[j].ServiceName,""))   // Mostrar solo
los que no son vacios
                                {
                                printf    ("Canal    numero    =    %i                    ",
atoi(tablaaux.ncanal[j].canal));
                                printf                    ("ServiceName                 =
%s\n",tablaaux.ncanal[j].ServiceName);
                                }
                        }
                }
        else printf ("imposible leer parserLCN.log");
        fclose(fd);
        }

void visualizar()
        {
        int i;
        fd  =  fopen("lista.m3u",  "w");  /*  Se  genera  una  lista  de  reproduccion  para
visualizar con VLC */
        if (tablaservicios.contadorServicios<1)
                {
                printf ("\n\nHa de realizar el parseo del Broadcast Record y del Package
Record antes para poder ver la lista de canales\n\n");
                system ("pause");
                return;
                }
        if (fd==NULL)
                {
                printf( "No se puede abrir el fichero lista.m3u\n" );
                exit(1);
                }
        for (i=1;i<canalmax+1;i++)
                {
                if (strcmp (tablaaux.ncanal[i].ServiceName,""))   // Mostrar solo los que
no son vacios
                        {
                        fputs ("#EXTINF:-1,[",fd);
                        fputs (tablaaux.ncanal[i].canal,fd);
                        fputs ("]",fd);
                        for (j=0;j<canalmax+1;j++)
                                {
                                if          (strcmp          (tablaaux.ncanal[i].ServiceName,
tablaservicios.serv[j].ServiceName)==0)
                                        {
                                        fputs (tablaservicios.serv[j].Name,fd);
```

```
                                        fputs ("\nudp://@",fd);
                                        fputs (tablaservicios.serv[j].Address,fd);
                                        fputs (":",fd);
                                        fputs (tablaservicios.serv[j].Port,fd);
                                        }
                                }
                        fputs ("\n",fd);
                        }
                }
        fclose(fd);
        printf ("lista de reproduccion realizada\n");
        system (".//VLC//vlc lista.m3u");
        }

void parserLCN()
        {
        //      ******************  INICIO PARSER LCN      ********************
        XML_Parser p = XML_ParserCreate(NULL);
        if (! p)
                {
                fprintf(stderr,  "No  se  puede  reservar  memoria  suficiente  para  el
parser\n");
                exit(-1);
                }

        XML_SetElementHandler(p, startLCN, endLCN);
        XML_SetCharacterDataHandler(p, charHandlerLCN);

        fd2 = fopen( "PackageRecord.xml", "r" );
        if (fd2==NULL)
                {
                printf( "No se puede abrir el fichero PackageRecord.xml\n" );
                exit(-1);
                }

        for (;;)
                {
                int done;
                int len;

                len = (int)fread(Buff, 1, BUFFSIZE, fd2);
                if (ferror(fd2))
                        {
                        fprintf(stderr, "Error de lectura en PackageRecord.xml\n");
                        exit(-1);
                        }
                done = feof(fd2);
                if (XML_Parse(p, Buff, len, done) == XML_STATUS_ERROR)
                        {
                        fprintf(stderr,  "Parse  error  en  la  linea  %"  XML_FMT_INT_MOD
"u:\n%s\n", XML_GetCurrentLineNumber(p), XML_ErrorString(XML_GetErrorCode(p)));
                        exit(-1);
                        }
                if (done) break;
                }
        fclose(fd2);
        guardarLCN();
        ordenarLCN();
        XML_ParserFree(p);
//  ***********************    FIN PARSER LCN  **********************
        }

void parserBroadcast()
        {
        // ********************** Inicio Parser Broadcast  ***********
        XML_Parser p2 = XML_ParserCreate(NULL);
        if (! p2)
                {
                fprintf(stderr,  "No  se  puede  reservar  memoria  suficiente  para  el
parser\n");
                exit(-1);
                }
        XML_SetElementHandler(p2, startBroadcast, endBroadcast);
        XML_SetCharacterDataHandler(p2, charHandlerBroadcast);

        fd2 = fopen( "BroadcastRecord.xml", "r" );
        for (;;)
```

```
                {
                int done;
                int len;
                len = (int)fread(Buff, 1, BUFFSIZE, fd2);

                if (ferror(fd2))
                        {
                        fprintf(stderr, "Error de lectura en BroadcastRecord.xml\n");
                        exit(-1);
                        }

                done = feof(fd2);

                if (XML_Parse(p2, Buff, len, done) == XML_STATUS_ERROR)
                        {
                        fprintf(stderr,  "Parse  error  en  la  linea  %"  XML_FMT_INT_MOD
"u:\n%s\n", XML_GetCurrentLineNumber(p2), XML_ErrorString(XML_GetErrorCode(p2)));
                        exit(-1);
                        }

                if (done) break;
                }
        fclose(fd2);
        guardarBroadcast(tablaservicios);
        leerBroadcast (tablaservicios);
        XML_ParserFree(p2);
        // ********************** Fin Parser Broadcast ***************
        }

void parserSDS()
        {
        // ************************ INICIO PARSER SDS ***************
        XML_Parser p3 = XML_ParserCreate(NULL);
        if (! p3)
                {
                fprintf(stderr,  "No  se  puede  reservar  memoria  suficiente  para  el
parser\n");
                exit(-1);
                }

        XML_SetElementHandler(p3, startSDS, endSDS);
        fd2 = fopen( "ServiceDiscovery.xml", "r" );
        for (;;)
                {
                int done;
                int len;
                len = (int)fread(Buff, 1, BUFFSIZE, fd2);

                if (ferror(fd2))
                        {
                        fprintf(stderr, "Error de lectura en ServiceDiscovery.xml\n");
                        exit(-1);
                        }

                done = feof(fd2);

                if (XML_Parse(p3, Buff, len, done) == XML_STATUS_ERROR)
                        {
                        fprintf(stderr,  "Parse  error  en  la  linea  %"  XML_FMT_INT_MOD
"u:\n%s\n", XML_GetCurrentLineNumber(p3), XML_ErrorString(XML_GetErrorCode(p3)));
                        exit(-1);
                        }

                if (done) break;
                }
        fclose(fd2);
        guardarSDS();
        leerSDS();
        XML_ParserFree(p3);
        // ******************* FIN PARSER SDS ***********************
        }
```