TRABAJO DE FIN DE CARRERA

TÍTULO DEL TFC: Desarrollo de software de descubrimiento de servicio y protección en redes DVB-IP

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad Telemática

AUTOR: Toni Martínez Pereda

DIRECTOR: David Rincón Rivera

FECHA: 9 de Octubre de 2006

Título: Desarrollo de software de descubrimiento de servicio y protección en redes DVB-IP

Autor: Toni Martínez Pereda

Director: David Rincón Rivera

Fecha: 9 de Octubre de 2006

Resumen

Este proyecto consiste en la transmisión de flujos DVB (Digital Video Broadcast) a través de redes IP. Se hará gracias a un PC que dispone de dos tarjetas capturadoras de TV Digital, una terrestre (DVB-T) y otra satélite (DVB-S).

La primera parte del proyecto está destinada a la utilización básica de las tarjetas, donde se mostrará su funcionamiento y los resultados obtenidos de realizar envíos vía red IP.

La segunda parte del proyecto estará destinada al desarrollo de un software sobre Service Discovery basado en el estándar “Digital Video Broadcasting (DVB); Transport of MPEG-2 Based DVB Services over IP Based Networks”

Los objetivos del TFC serán la captura de trazas reales de emisiones, DVB-T y DVB-S, en abierto y su redifusión sobre IP así como el desarrollo de software en código C sobre Service Discovery y protección en redes DVB-IP.

Title: Service Discovery and DVB-IP networks protection software development

Author: Toni Martínez Pereda

Director: David Rincón Rivera

Date: October, 9th 2006

Overview

This project consists in DVB streaming (Digital Video Broadcasting) over IP based networks. It will be possible with a PC that allows two capturer cards of terrestrial and satellite Digital TV, one DVB-T and other one DVB-S.

The first part of the project is destined to the basic use of cards, where his operation and obtained results to make video streaming via network IP will be shown.

The second part of the project will be destined to a software development on Service Discovery based on the standard "Digital Video Broadcasting (DVB); Transport of MPEG-2 Based DVB Services to over IP Based Networks "

The objectives of the TFC will be the capture of real plans of emissions, DVB-T and DVB-S, in opened and to their redifusión on IP as well as the development of software on Service Discovery and protection in DVB-IP networks in C code.

ÍNDICE

INTRODUCCIÓN 1

1. ARQUITECTURA Y MODELOS 3

1.1. Transport Stream 3

1.2. Arquitectura 4

1.3. Modelos del servicio 5
1.3.1. Modelo por capas 5
1.3.2. Modelo de referencia Local 6

2. TRANSMISIÓN DVB EN REDES IP 8

2.1. Empaquetado de DVB sobre IP 8
2.1.1. Encapsulado sin corrección de errores 8
2.1.2. Encapsulado con corrección de errores 9
2.1.2.1. Implementación del entrelazador convolucional 10

3. SERVICE DISCOVERY & SELECTION 12

3.1. Introducción 12

3.2 Service Discovery & Selection 12
3.2.1 Fragmentación de informes SD&S 14
3.2.1.1. Segmentos 14
3.2.1.2. Secciones 15

3.3 Funcionamiento del Service Discovery 16
3.3.1. Paso 1. Encontrar puntos de entrada de SD 17
3.3.2. Paso 2. Para cada punto de entrada obtener información acerca de los SP disponibles 17
3.3.3. Paso 3. Para cada SP obtener información acerca de los servicios DVB-IP disponibles. 18
3.3.3.1. TS Full SI 19
3.3.3.2. Optional SI 20

3.4. Identificación del servicio 22

3.5. Selección de servicio 22

3.6. Transporte de SD&S 22
3.6.1. Sintaxis 22
3.6.2. Semántica 23
3.6.3. Pila de protocolos para el transporte de DVB sobre IP 24
3.6.4. Ejemplo de envío de SD&S 25

4. DISEÑO Y DESARROLLO DE LA APLICACIÓN 27

4.1. Escenario 27
4.1.1. Servidor 28
4.1.1.1. Bloque 1, core 28
4.1.1.2. Bloque 2, los analizadores 28
4.1.1.3. Bloque 3, el servidor de vídeo 29
4.1.2. Cliente 29
4.1.3. Conexión entre bloques 29

4.2. Software utilizado 31
4.2.1. Analizadores 31
4.2.2. Servidor de vídeo 31
4.2.3. Validador XML 32
4.2.4. Cliente de vídeo 32

4.3. Pruebas preliminares 32
4.3.1. Analizadores 32
4.3.1.1. Prueba decode_pat original 33
4.3.1.2. Prueba decode_pmt original 34
4.3.1.3. Prueba decode_pat modificado 35
4.3.1.4 Prueba decode_pmt modificado 36
4.3.2. Pruebas servidor de vídeo 38
4.3.2. Pruebas validador XML 40

5. PRUEBAS Y RESULTADOS 41

5.1. Software SD 41
5.1.1. Servidor 41
5.1.1.1. Captura 42
5.1.1.2. Análisis 42
5.1.1.3. Servicio 43
5.1.2. Cliente 44
5.1.2.1. Recepción 45
5.1.2.2. Petición 46

5.2. Modos FEC 47

6. CONCLUSIONES Y LÍNEAS FUTURAS 50

6.1. Conclusiones 50

6.2. Repercusión medioambiental 52

6.3. Aspectos éticos 52

6.4. Aspectos de seguridad 52

6.5. Líneas futuras 52

BIBLIOGRAFÍA 54

GLOSARIO 57

ANEXO I. LIBDVBPSI 58

ANEXO II. XML-SCHEMA 60

ANEXO III. TABLAS 62

ANEXO IV. CÓDIGO 67

INTRODUCCIÓN

El avance tecnológico visto en los últimos años también ha influido en la televisión. La televisión digital ha ido desbancando a la analógica progresivamente, de tal manera que en un futuro próximo la hará desaparecer.

La televisión digital, a diferencia de la televisión tradicional (que envía sus ondas de manera analógica), codifica sus señales de forma binaria, habilitando así la posibilidad de crear vías de retorno entre consumidor y proveedor de contenidos, teniendo así la posibilidad de crear aplicaciones interactivas.

El proyecto DVB, *Digital Video Broadcasting*, nació en 1990 con la idea de definir un nuevo sistema capaz de cubrir las necesidades de transmisión de televisión en su formato digital y crear así un estándar a nivel mundial.

El estándar DVB, en sus diferentes versiones, define aspectos que abarcan desde la capa más física, donde se describen las modulaciones a utilizar para los diferentes medios de transmisión, hasta la señalización empleada para la sintonización y reproducción del flujo de video, pasando por temas de sincronización y encriptación.

Actualmente existen varios tipos de estándares de transmisión basados en DVB, cada uno desarrollado según el medio de transmisión de la señal y/o en los receptores. Éstos son DVB-C (*DVB-Cable*, transmite sobre cable), DVB-S (DVB-Satélite, transmitido mediante satélite), DVB-T (*DVB-Terrestre*), DVB-H (*DVB-Handheld*, basado en DVB-T pero enfocado a terminales móviles) y DVB-IP (DVB sobre redes IP).

Este proyecto se basa en el estándar "*Digital Video Broadcasting (DVB); Transport of MPEG-2 Based DVB Services over IP Based Networks*", que trata de la última tecnología nombrada anteriormente, DVB-IP. El estándar se divide en 12 apartados y 4 anexos y nos proporciona un sistema de especificaciones técnicas que cubre distintas áreas.

Los tres primeros apartados incluyen introducción, referencias y definiciones, abreviaciones y anotaciones. A continuación, el apartado 4 dicta las especificaciones para que DVB-IP pueda ser soportado en el cliente. El siguiente apartado define SD (Descubrimiento de servicio o *Service Discovery*), su formato de datos y los protocolos usados para el envío de esta información. La cláusula 6 explica como usar RTSP para controlar servicios CoD y opcionalmente darse de alta en grupos multicast. Pasa a tratar de los modos de entrega de los servicios LMB (*Live Media Broadcast*) y CoD (*Content on Demand)* en el punto 7. El siguiente punto está basado en la configuración del cliente final. El noveno apartado trata de la identificación del cliente final en la red, mientras que la cláusula 10 define una administración de la red opcional con el fin de proveer automáticamente al usuario final con funcionalidades

adicionales. Las dos últimas cláusulas, 11 y 12, tratan de dos tipos distintos de clientes finales, uno basado en Ethernet y otro en IEEE 1394.

A partir del estándar nombrado se marcaron los objetivos a seguir en el proyecto. El primer objetivo a cumplir será el de realizar una introducción a dicho estándar, haciendo especial énfasis en los aspectos que posteriormente servirán al desarrollo de un software basado en él. Esta introducción también deberá hacer referencia al transporte de DVB.

El siguiente objetivo a cumplir será el desarrollo de software de descubrimiento de servicio (basado en la cláusula 5 del estándar). Para ello, el primer paso será el de diseñar un escenario óptimo para el desarrollo del software. Una vez diseñado el escenario ya se tendrá constancia de los bloques que componen éste y se podrá pasar a buscar software libre que nos pueda servir. Una vez desarrollado el escenario y obtenido software de utilidad se pasarán a hacer pruebas preliminares para ver su funcionamiento y las modificaciones necesarias. Finalmente se deberán unir los bloques creados en una única aplicación.

El último objetivo del proyecto será el desarrollo de software para protección en redes DVB-IP. Para ello deberemos retomar el código y las pruebas realizadas por Gustavo Cerdà [29]. Una vez familiarizados con dicho código se pasará a modificar para implementar la aplicación deseada.

A continuación se definen los capítulos que componen esta memoria:

- Capítulo 1- Se realiza una pequeña introducción a DVB y TS y se comienza el estudio del estándar, concretamente la arquitectura y modelos a seguir.

- Capítulo 2- En este capítulo se describe el sistema de encapsulado necesario para la multidifusión de vídeo digital sobre redes IP.

- Capítulo 3- Se presenta todo el diseño del mecanismo *Service Discovery & Selection.*

- Capítulo 4- Se describe el escenario diseñado para el desarrollo del software basado en SD&S. Además se describen cada uno de los bloques que componen dicho software.

- Capítulo 5- Se muestran las pruebas realizadas en los escenarios de transmisión montados y los resultados obtenidos en los mismos.

- Capítulo 6- Como último punto se describen las conclusiones del proyecto, las líneas futuras a seguir, la repercusión medioambiente y los aspectos éticos y de seguridad de la aplicación.

1. ARQUITECTURA Y MODELOS

1.1. Transport Stream

Transport Stream, "*MPEG-2 Transport Stream"* (TS), es un formato especificado en el estándar ISO/IEC 13818-1 sobre MPEG-2. El objetivo de TS es permitir la multiplexación de varios flujos de vídeo y audio y su sincronización a la salida. En el conjunto de estándares de televisión digital, DVB, cada estándar define los esquemas de codificación de canal y de modulación para el medio de transmisión de que se trate (satélite, cable, terrestre, etc.), pero en todos los casos la codificación de fuente es una adaptación del estándar MPEG-2, más concretamente de TS.

El paquete es la unidad básica de datos en el TS. Está formado por una cabecera y por el *payload*. En la cabecera destacan un byte de sincronía, cuyo valor es 0x47 y un PID de 13-bits. Los paquetes TS tienen un tamaño de 188 bytes, excepto los que utilizan Reed Solomon[1] como método de corrección de errores. Estos paquetes tienen un tamaño de 204 bytes al utilizar RS(204,188), tal y como se podrá ver en el apartado 2.1.2 de la presente memoria.

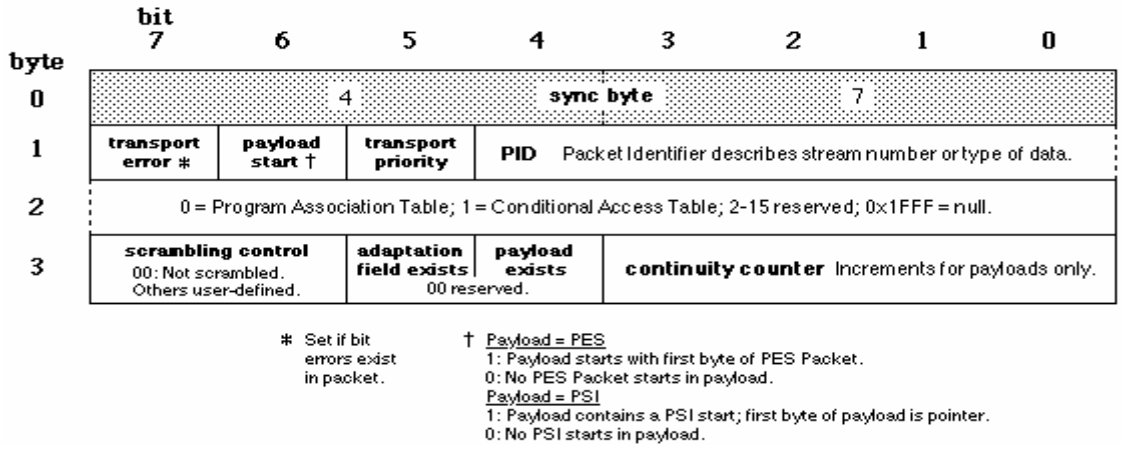


Fig. 1.1 TS Packet, extraída de [34]

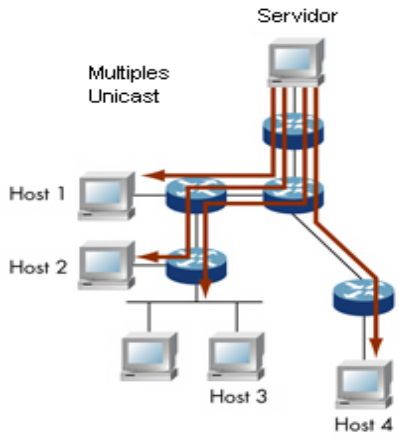
El campo PID de la cabecera identificará el tipo de información contenida en el payload, ya sea una tabla con información sobre el flujo contenido (Anexo III) o un programa transportado. Programa es precisamente el término utilizado por TS para denominar los distintos PID de vídeo, audio, etc. asociados en uno solo que los identifique. En cuanto al transporte de tablas, hay que destacar el transporte de tablas PAT, *Program Association Table*, y PMT, *Program Map Table*, ya que son las que transportan la información acerca de todos los flujos contenidos en el TS.

[1] Reed Solomon es un código bloque utilizado para la corrección de errores

Uno de los estándares DVB surgidos de una adaptación de TS es DVB-IP. DVB-IP se trata de un estándar basado en DVB, desarrollado para la transmisión sobre redes IP. Al tratarse de transmisión de flujos sobre redes IP tendrá un protocolo de transporte, que veremos en el apartado 2 del trabajo. También deberá disponer de una arquitectura de difusión que se muestra en el apartado siguiente. DVB-IP utiliza Reed Solomon como método de corrección de errores

1.2. Arquitectura

Para la difusión de DVB sobre redes IP el servidor deberá difundir el flujo de vídeo a todos los clientes. Si utilizamos una arquitectura unicast (Figura 1.2), en caso de disponer de muchos clientes, se puede convertir en una tarea inviable dado el ancho de banda utilizado; para reducirlo podemos utilizar la técnica multicast (Figura 1.3) ya que un mismo flujo se repartirá por todos los clientes, aunque para ello se necesitará una red que la soporte.



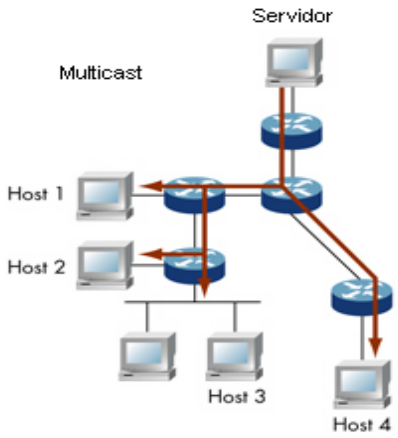


Fig. 1.2 Técnica unicast, extraída de [4] **Fig. 1.3** Técnica mcast, extraída de [4]

Si utilizamos tecnología multicast ese gran ancho de banda se verá reducido a un solo flujo que se irá abriendo camino entre los routers. En este caso se crearán grupos multicast para que los clientes se puedan unir a dicho grupo y recibir el flujo que esté enviando el servidor.

Para crear estos grupos se utiliza el protocolo IGMP, *Internet Group Management Protocol*, que se encarga tanto de crearlos como de la gestión de los mismos. Un cliente que quiera unirse a un grupo multicast simplemente deberá enviar un paquete Join al router (paquete que especifica que quiere unirse al grupo concreto) y, en caso de ser aceptado, pasará a formar parte de dicho grupo. Para más información acerca del protocolo IGMP consultar el RFC 3376 [28].

Al transmitir el flujo de vídeo necesitaremos un protocolo que lo controle, tanto la calidad de servicio deseada como el correcto envío del flujo y su correcta reconstrucción en el cliente con el fin de poder reproducirlo correctamente. Para todo esto se utiliza el protocolo RTP [23], *Real Time Protocol*.

1.3. Modelos del servicio

Para aclarar el sistema de gestión de los servicios DVB sobre las redes IP existen distintos modelos esquematizados, de los cuales vamos a describir el modelo por capas y el modelo de referencia local.

1.3.1. Modelo por capas

Este modelo por capas es una visión general sobre el número de interfaces entre el proveedor de contenidos, *Content Provider*, y el cliente final (*Home* en la figura 1.4). Define los componentes necesarios para entregar los servicios DVB sobre redes IP.

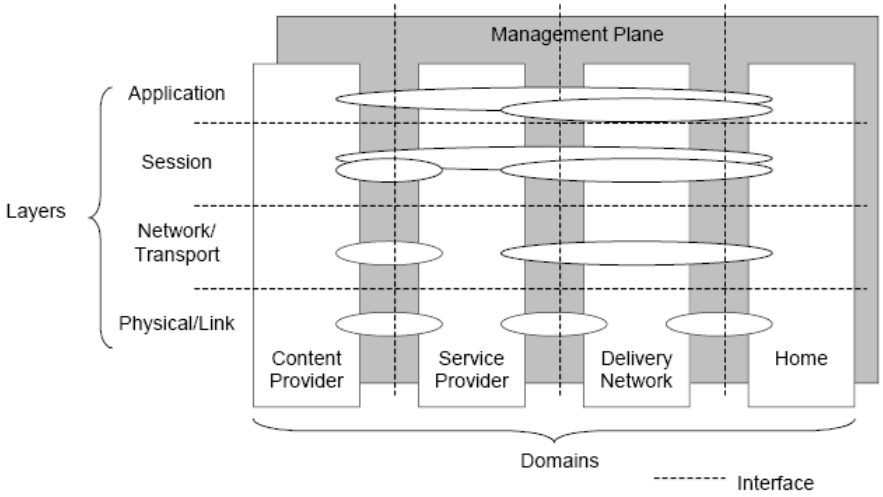


Fig. 1.4 Modelo por capas, extraída de [7]

Componentes del modelo por capas:

- *Content Provider*, CP (Proveedor de Contenidos): Se refiere a la entidad que posee contenidos o tiene licencia para venderlos. Como por ejemplo una productora de televisión que decide ofrecer sus canales vía internet.

- *Service Provider*, SP (Proveedor de Servicios): Se trata de la entidad que provee el servicio al usuario final, por tanto, será quien trate con el CP. Se tratará del intermediario entre la productora y el cliente final.

- *Delivery Network*, DN (Red de entrega): Es la entidad que conecta los clientes, Home, con los SP. Normalmente está compuesta por redes de acceso y redes troncales.

- *Home* (Usuario final): Es el lugar de consumo de los servicios antes mencionados, como puede ser un ordenador personal.

1.3.2. Modelo de referencia Local

Este esquema muestra los detalles de las interficies entre la red de acceso, la red local y el usuario final.

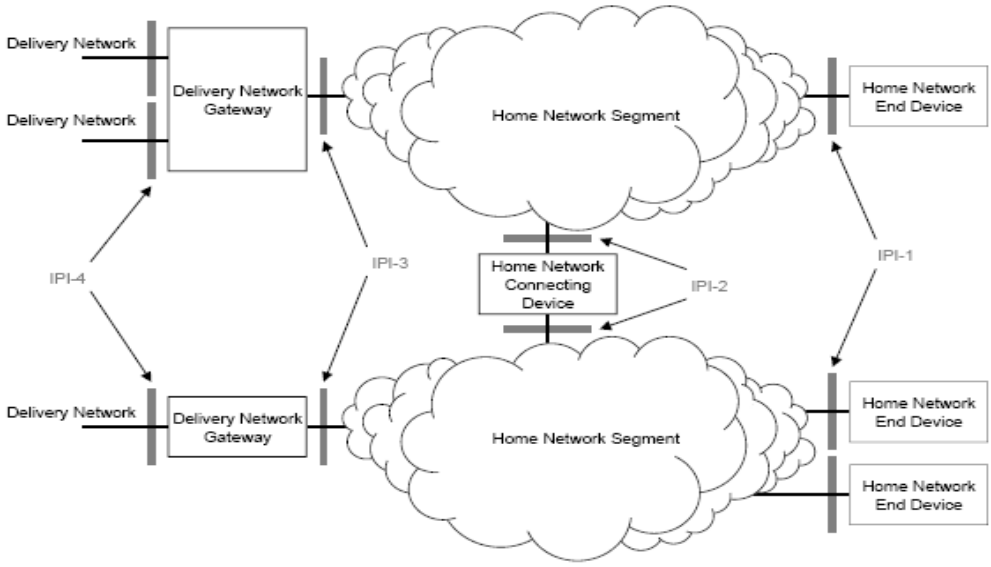


Fig. 1.5 Modelo de referencia local, extraída de [7]

Este modelo hace referencia al dominio del usuario final, *Home* en el modelo por capas visto en el apartado 1.3.1. Además muestra la interconexión con el dominio DN. Por tanto refleja los elementos que pueden haber en una casa y la relación que puede haber entre ellos, que en conjunto forman una red casera (*Home Network*).

El modelo descrito contiene los siguientes elementos:

- *Delivery Network Gateway*, DNG: Se trata del dispositivo encargado de interconectar la red externa con la interna, la red de entrega, DN, con segmentos de la red casera (Home Network Segment). Como por ejemplo el router del SP.

- *Home Network Segment*, HNS: este elemento consiste en un enlace simple entre capas y provee conexión entre el HNED y/o componentes conectados.

- *Home Network Connecting Device*, HNCD: este dispositivo, que contiene uno o más componentes conectores, conecta 2 o más HNS entre ellos. Como por ejemplo nuestro router o gateway.

- *Home Network End Device*, HNED: es el dispositivo al que está conectado el HN y que es donde típicamente acaba el flujo IP. Esto no implica que sea el punto final del flujo de información no-IP, por ejemplo, un stream DVB sobre IP puede ser convertido a flujo DVB según IEEE 1394.

En el caso de que no exista el dispositivo DNG el HNED estará conectado directamente al Delivery Network

2. Transmisión DVB en redes IP

2.1. Empaquetado de DVB sobre IP

Tomando como punto de partida el flujo de transporte de MPEG2, el empaquetado del flujo DVB sobre redes IP se define en el RFC 2250 [27]. Si se decide realizar la transmisión vía internet, red Best-Effort[2], habrá que tener en cuenta las pérdidas resultantes por caídas de routers, desbordamientos de colas, ráfagas de pérdidas, etc. Es por ello que el SP tendrá la opción de encapsular el flujo de dos modos distintos, sin corrección de errores o bien con corrección de errores.

2.1.1. Encapsulado sin corrección de errores

En cada paquete IP se introducirá un número entero de paquetes MPEG-2 TS, de 188 bytes cada uno, por lo que tendremos un paquete como el mostrado en la Figura 2.1. Podemos ver como lo componen los paquetes MPEG-2 TS y las cabeceras IP, UDP y RTP.

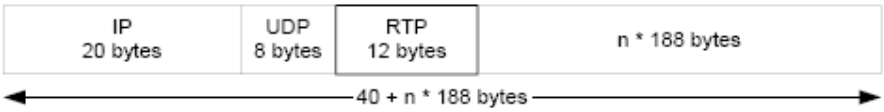


Fig. 2.1 Encapsulación MPEG-2, extraída de [11]

El número de paquetes, n en la Figura 2.1, vendrá determinado según el tamaño máximo de paquete IP. El protocolo IP limita el tamaño máximo de paquete a 65535 bytes, aunque no es recomendable superar la máxima unidad de transmisión (MTU, *Maximum Transmission Unit*) del enlace, para evitar las fragmentaciones de los routers y así elevar la demora, las posibles pérdidas, etc. Por lo tanto en caso de utilizar una red Ethernet este valor será de 1500 bytes, que nos permitirá transmitir 7 paquetes del TS en cada paquete IP, con lo que obtendremos un tamaño final de 1356 bytes.

[2] Best-Effort, BE, se refiere a un tipo de red que no provee ningún mecanismo para recuperar datos perdidos o corruptos.

2.1.2. Encapsulado con corrección de errores

En caso de que el SP encapsule con corrección de errores conseguirá reducir las pérdidas. Esta corrección de errores se aplica antes del encapsulado sobre los datos TS. El método de corrección de errores utiliza primero Reed Solomon (204, 188) y segundo un entrelazador convolucional posterior.

El codificador RS(204,188) es un código bloque. Utilizado en los estándar DVB-C, DVB-S y DVB-T, añade a cada paquete de 188 bytes, 16 bytes extra formando un paquete de 204 bytes. Este mecanismo puede corregir hasta 8 bytes incorrectos y detectar hasta 16 bytes erróneos en un paquete de 204 bytes, aunque sólo se utiliza como corrector.

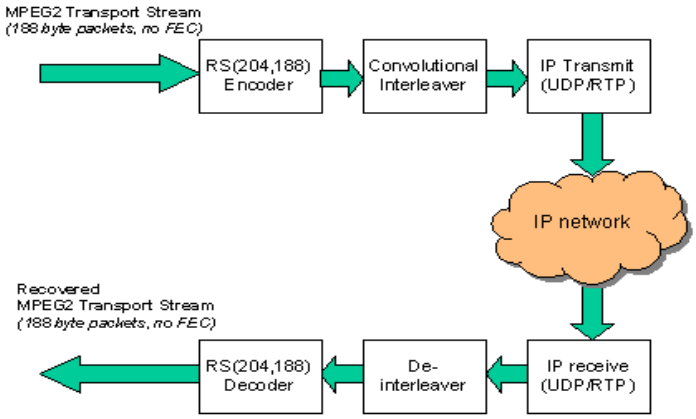


Fig. 2.2 Proceso de encapsulado con corrección de errores, extraída de [29]

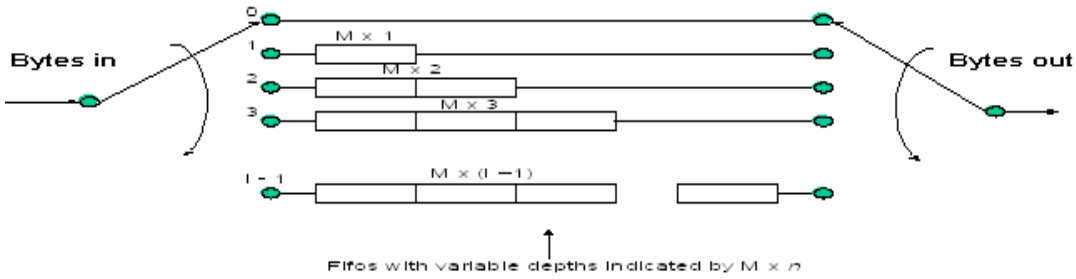


Fig. 2.3 Entrelazador convolucional, extraída de [29]

El entrelazador convolucional está destinado a reducir los efectos provocados por las ráfagas de errores, provocadas, por ejemplo, por la pérdida momentánea de conexión por parte del emisor. Esto lo consigue entrelazando de tal manera los bytes de datos que los bytes que en un principio estaban juntos ahora estarán separados, de tal manera que se evitarán las ráfagas de errores ya que se dispersarán los errores. El estándar define 5 modos de FEC, con diferentes características de retardo, uso de memoria y tolerancia respecto a fallos, como se puede ver en la tabla 2.1.

Tabla 2.1. Modos FEC, extraída de [29]

Modo	N	M	I	Máxima longitud de ráfaga (bytes)	Bytes necesarios para buffer (bytes)
0	204	4	51	408	10200
1	204	2	102	816	20604
2	204	1	204	1632	41412
3	408	2	204	3264	82824
4	612	3	204	4896	124236

En la Tabla 2.1 el valor N se refiere al número de bytes del registro de desplazamiento, el valor M a la unidad de desplazamientos para cada camino e I se refiere a la profundidad de entrelazado; por lo que tendremos un retardo de jxM para cada camino, donde j será el número de camino actual (Figura 2.3)

Existen 5 modos de los cuales los tres primeros, 0, 1 y 2, poseen una corrección de errores menor, por lo que se recomiendan para entornos Ethernet dada su baja tasa de pérdidas. Por otro lado existen los modos 3 y 4, con una corrección de errores mayor y por lo tanto más pesado de entrelazar y con más redundancia, causando un mayor retardo. Estos dos modos se recomiendan para aplicaciones menos tolerantes a fallos.

Una vez concluído el proceso de corrección de errores queda empaquetar el contenido, 204 bytes resultantes, en los paquetes IP. Para ello deberemos especificar que hemos realizado la corrección de errores introduciendo una cabecera FEC de 4 bytes seguida de la cabecera RTP.

Esta cabecera cuenta con dos campos significativos, el contador de paquetes de TS y el indicador de modo. El primero es un simple contador creciente útil para reordenar los paquetes y detectar pérdidas. El campo modo indica el tipo de entrelazado utilizado, en un rango [0,4] como muestra la Tabla 2.1, estando reservados los campos del 0 al 255.

En el siguiente apartado se hace referencia a la implementación de los distintos tipos de entrelazado mostrados.

2.1.2.1. Implementación del entrelazador convolucional

En el TFC Transmisión de flujos DVB/MPEG sobre redes IP elaborado por Gustavo Cerdá en 2004, del que se ha tomado referencia, ya se habían contemplado los tres primeros modos vistos en la tabla 2.1, modos recomendados para Ethernet ya que poseen una corrección menor debido a la baja pérdida en LAN. Con los dos restantes, diseñados para aplicaciones menos tolerantes a fallos. Se hará una pequeña revisión de su código, implementado en dvbstream, para que sea compatible con los 5 modos de entrelazado.

El primer paso será crear una matriz que nos sirva a modo de interleaving para poder realizar el entrelazado mostrado en la figura 2.3, los valores de esta matriz se han elegido para poder dar cabida a cualquiera de los 5 modos.

```
unsigned char entrelazador [612][204];
```

Tabla 2.2. Relación bytes necesarios-bytes utilizados

Modo de entrelazado	Bytes necesarios	Bytes utilizados	Diferencia
0	10200	124848	114648
1	20604	124848	104244
2	41412	124848	83436
3	82824	124848	42024
4	124236	124848	612

Como se puede ver en la Tabla 2.2 la máxima diferencia es de 112KBytes, que no suponen un problema dadas las dimensiones de las memorias actuales.

A continuación el código del entrelazador convolucional:

```
indi = 0;
  while (indi<1428) {
      for (j = 0; j<INTERI ;j++) {
      entrelazador[INTERM*j][j] = codeword2 [indi];
      salida[indi] = entrelazador [0][j];
      for (k = 1 ; k < INTERI ; k++) entrelazador [k-1][j] =
entrelazador [k][j];
      indi++;
      }
  }
```

Donde INTERI y INTERM corresponden a los valores mostrados en la Tabla 2.1 según que modo de entrelazado queramos, y que deberemos modificar en las siguientes líneas del fichero fuente.

```
#define INTERI = 204
#define INTERM = 1
#define writes(f,x) write((f),(x),strlen(x))
```

3. SERVICE DISCOVERY & SELECTION

3.1. Introducción

Service Discovery & Selection (SD&S) es un mecanismo que consiste en la presentación de una guía electrónica de programas disponibles (DVB). Le permitirá al usuario disponer de todos los datos concernientes a los servicios disponibles, todos ellos en formato XML (a lo que denominaremos informe SD&S). En este informe el cliente podrá ver la ubicación del servidor, como contactar con él (dirección IP y puerto), el nombre de cada uno de los programas ofrecidos, su descripción, lenguaje de emisión, si dispone de teletexto, etc. Una vez el cliente se decida a recibir un programa SD&S le permitirá hacer dicha petición y recibir el flujo en su máquina.

Para la entrega de la información de SD están soportados dos tipos de transporte, el modo unicast y el modo multicast. En caso de utilizar unicast estaremos utilizando un modelo *pull* ya que se tratará de un esquema de petición-respuesta entre cliente y servidor; en cambio en caso de utilizar multicast estaremos utilizando un modelo *push* donde el cliente simplemente irá recibiendo la información del SP que le permita seleccionar uno u otro servicio.

La información se puede cargar en cualquiera de los dos modos, siendo XML el formato empleado para difundir estos informes, que además deberán ser válidos y bien formado (*well-formed*). En el anexo II del presente proyecto se pueden ver los tipos de informes empleados.

3.2 Service Discovery & Selection

Para la entrega de información de SD&S el estándar describe distintos tipos de informes:

- Un tipo que contiene información SD&S relativa al SP como tal, a este informe se le refiere como *Service Provider Discovery Information.*

- Cuatro tipos de SD&S sobre los servicios ofrecidos por el SP. A estos informes se les denomina *DVB-I Offering* y existen 4 tipos como podemos ver en la Figura 3.1.

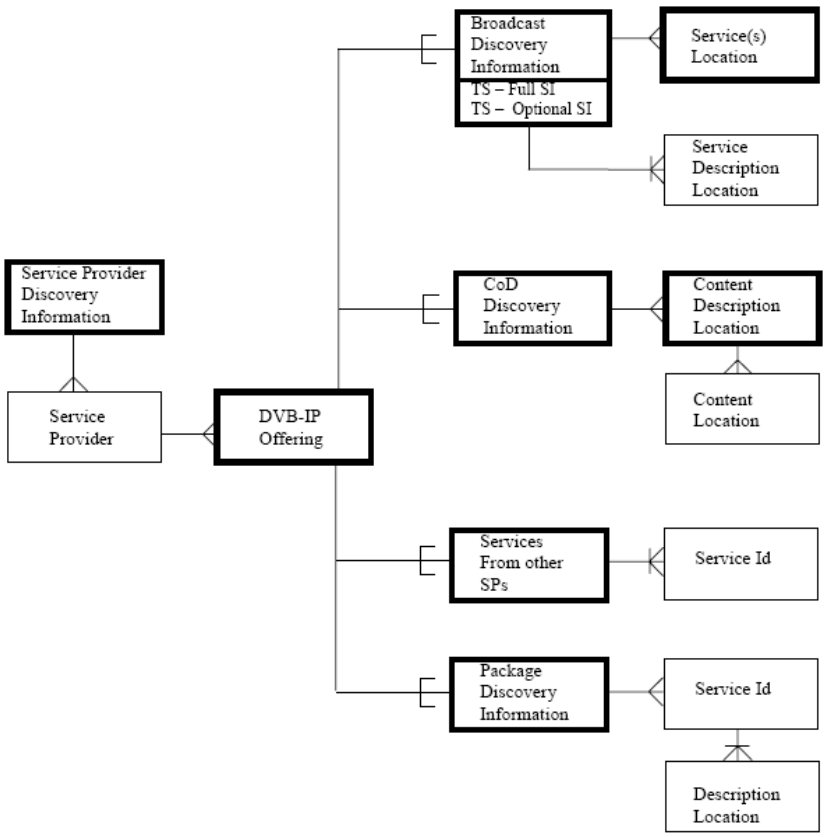


Fig. 3.1 Tipos de informes, extraída de [7]

Los cuatro tipos de informes que ofrecen información sobre los servicios ofrecidos por el SP son los siguientes:

- *Broadcast Discovery Information,* diseñado para entornos como el escenario utilizado en el software desarrollado. *Broadcast Discovery Information*, además, puede ser de dos tipos distintos, “TS Full SI” o bien “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. El segundo, “TS Optional SI”, envía la misma información que “TS Full SI” y, además, contiene varios campos opcionales que pueden dar más información acerca del servicio al cliente.

- *CoD Discovery Information*, este tipo de informe provee toda la información necesaria para descubrir los servidores CoD disponibles en la red, así como la localización de su catálogo de contenidos. No provee información alguna sobre los contenidos en sí.

- *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*, este informe proporciona la posibilidad de agrupar un conjunto de servicios como una única entidad.

Cada uno de estos tipos de informes de descubrimiento de servicio se identifican mediante un byte llamado PayloadID, que irá en la cabecera del paquete formado por Service Discovery para enviar los informes. Los distintos valores se pueden ver en la Tabla 3.1.

Tabla 3.1. Valores del Payload ID

Payload ID	**Informe SD&S transportado**
0x00	Reservado
0x01	SP Discovery Information
0x02	Broadcast Discovery Information
0x03	CoD Discovery Information
0x04	Services from other SP
0x05	Package Discovery Information
0x06-0xEF	Reservado
0xF0-0xFF	Usuario privado

Una vez enviemos cualquiera de estos informes si deseamos enviar una modificación o actualización del mismo deberemos especificarlo incrementando en una unidad el campo Version presente en cada uno de los tipos de informes SD&S.

3.2.1 Fragmentación de informes SD&S

3.2.1.1. Segmentos

Los informes SD&S se pueden segmentar ya que tienen un tamaño considerable. A cada uno de estos segmentos se le asignará un identificador para saber su número de segmento en el informe SD&S. El tamaño del identificador será de 16 bits. También existe un campo Version que consta de 8 bits.

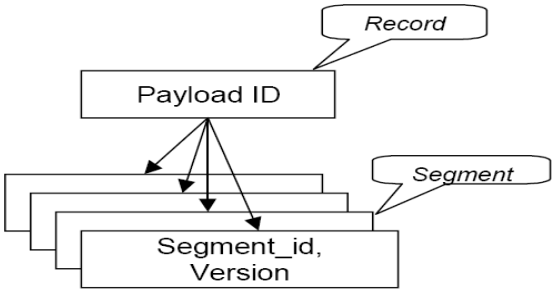


Fig. 3.3 Esquema de división en segmentos, extraída de [7]

Los informes SPDiscovery no se pueden segmentar cuando se usa *pull mode* o modo unicast. En todos los demás casos, el informe XML se puede segmentar para facilitar el proceso del HNED, no siendo indispensable esta segmentación.

Cada paquete sólo podrá contener segmentos de uno de los tipos de informes (BroadcastDiscovery, o CoDDiscovery, o ServicesFromOtherSP, o PackageDiscovery, o bien ServiceProviderDiscovery). Además, cada uno de estos segmentos debe ser tener un formato XML válido y bien-formado (*well-formed*).

Si el escenario consta de varios SP que comparten una misma dirección multicast, cada SP al formar un paquete con informes marcará con un Provider ID la cabecera DVBSTP (*Digital Video Broadcasting SD&S Transport Protocol)*. DVBSTP es un protocolo de transporte diseñado para el transporte de informes SD&S y se comenta en el punto 2.6.

En caso de segmentar un informe SD&S de un SP, al tiempo necesario para transmitir todos los segmentos se le llama Cycle Time. El tiempo máximo de ciclo (Maximum Cycle Time) es de 30s. Un segmento se considerará borrado si no llega un paquete para ese segmento en menos de 2 Max.Cycle Time.

3.2.1.2. Secciones

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 eficiente de 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.

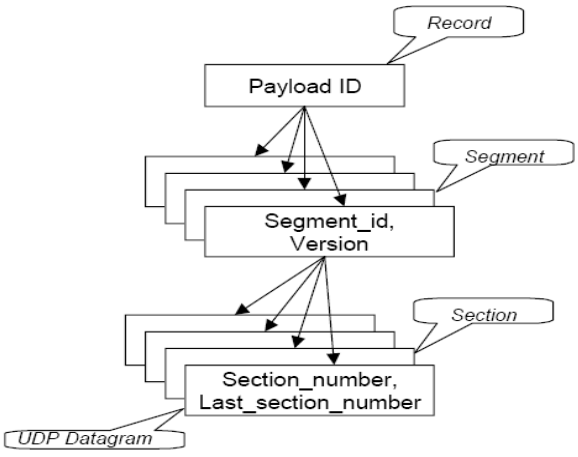


Fig. 3.4 Esquema de división en secciones, extraída de [7]

La cantidad de datos que puede ser encapsulada en cada paquete UDP está limitada por el tamaño máximo del datagrama IP menos las cabeceras UDP y RTP. 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.

Se recomienda fijar el tamaño máximo de paquete UDP sin exceder el MTU. Si pasamos del tamaño MTU obtendremos como respuesta la fragmentación por parte de los routers, lo que nos llevará a un aumento del retardo y más riesgo de pérdidas en la red.

Si el dispositivo final no soporta fragmentación el SP 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 SP deberá ajustar el tamaño del contenido.

3.3 Funcionamiento del Service Discovery

El estándar de Service Discovery define una serie de pasos para comunicar el cliente y el servidor y así conseguir que este último obtenga el servicio de vídeo deseado.

El proceso da comienzo con el descubrimiento de Proveedores de Servicio (SP) que ofrezcan servicios DVB-IP y continúa con el descubrimiento de los servicios disponibles de cada SP, de modo que primero se encontrarán los servidores disponibles y, una vez conocidos, se pasará a consultar los servicios disponibles de cada uno de ellas.

La totalidad de los pasos para el descubrimiento del servicio se pueden ver en la Figura 3.5.

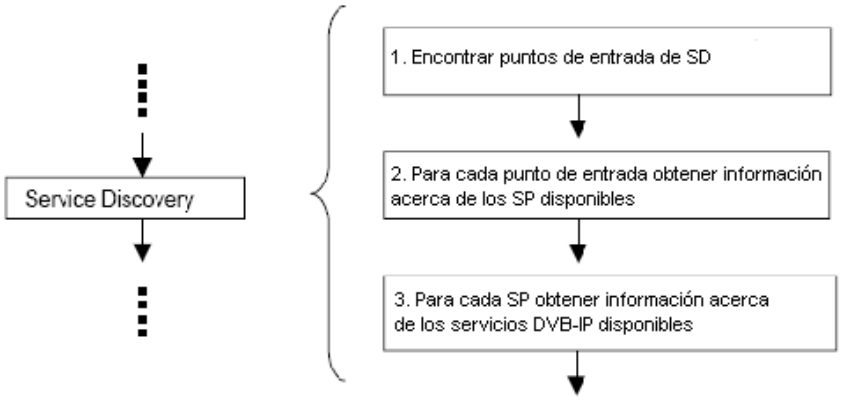


Fig. 3.5 Funcionamiento de SD, extraída de [7]

3.3.1. Paso 1. Encontrar puntos de entrada de SD

El primer paso en el proceso de descubrimiento de servicio trata de encontrar los puntos de entrada de la información del protocolo. Los puntos de entrada de SD&S pueden ser los siguientes:

- Una dirección muticast registrada con IANA que es 224.0.23.14 (DvbServDisc), *well-known*[3].

- Una lista de puntos de entrada de SD&S se puede adquirir vía DNS de acuerdo con el RFC 2782 [25]. El nombre del servicio es _dvbservdsc, el protocolo puede ser tcp o udp, mientras el resto del nombre es el nombre del dominio mantenido por DVB para el SD. Este nombre se fija por services.dvb.org. Las operaciones de búsqueda serán _dvbservdsc._tcp.services.dvb.org o _dvbservdsc._udp.services.dvb.org. Esta opción requiere que el HNED soporte SRV.

- Cuando el HNED se conecta a la red para saber su propia IP se le puede proveer de nombres de dominio vía DHCP [24] opción 15, que especifica el nombre de dominio que el cliente deberá usar cuando utilice el servicio DNS. Una lista de puntos de entrada SD&S se adquieren entonces vía DNS de acuerdo con el RFC 2782 [25]. El nombre del servicio es _dvbservdsc mientras el resto del nombre es el Domain Name provisto vía DHCP opción 15. Por ejemplo, _dvbservdsc._tcp.example.com. Esta opción requiere que el HNED soporte SRV.

- Otro modo es que el administrador del HNED proporcione directamente los puntos de entrada, la dirección IP, como parte de los datos de la configuración del terminal.

En el software desarrollado se optará por una dirección multicast conocida, por lo tanto se usará la última opción, una dirección multicast proporcionada directamente por el administrador del equipo.

Si no se especifica un puerto, el puerto por defecto será 3937 (dvbservdscport), asignado por IANA [26].

3.3.2. Paso 2. Para cada punto de entrada obtener información acerca de los SP disponibles

Una vez tengamos un punto de entrada de información pasaremos a recibir informes en el cliente detallando la información necesaria para que este pueda elegir un servicio. La primera elección deberá ser la de un SP en concreto para saber de que servicios dispone.

---

[3] *well-known* se le dice a los valores predeterminados reservados para servicios conocidos. En este caso, por ejemplo, el valor de una dirección multicast.

En nuestro caso sólo tendremos un SP por lo que este último paso de elegir un SP en concreto no estará presente y será siempre el mismo el que nos envíe información.

La “carta de presentación” de los SP se traduce en un informe en formato XML (SPDiscoveryInformation) con los campos mostrados en la Tabla 3.2.

Tabla 3.2. SP Discovery Record

SP Discovery Record	**Descripción del atributo**	**Obligatorio(Ob)/ Opcional(Op)**
Nombre de dominio del SP	Nombre de dominio registrado por el SP que lo identifique	Ob
Número de versión	Versión del informe, se incrementa para cada cambio	Ob
Nombre multilenguaje del SP	Nombre del SP para mostrar en uno o varios idiomas	Ob
Descripción multilenguaje del SP	Descripción del SP en uno o varios idiomas	Op
Logo del SP	Puntero a un logo de SP	Op
Localización	Localización de los informes de las ofertas del SP	Op
ID de la carga útil	Ref. Tabla 3.1	Op
Lista de IDs de los segmentos y su versión	Indica qué segmento transporta la información SD	Op (en modelo *push*)

La SPDiscoveryInformation puede entregarse tanto en modo multicast, modelo *push*, como en modo unicast, modelo *pull*, o petición-respuesta. Los dos modelos están soportados.

3.3.3. Paso 3. Para cada SP obtener información acerca de los servicios DVB-IP disponibles.

Una vez decidido el SP del que queremos obtener la información pasaremos a recibir los informes acerca de los servicios de que dispone, para ello recibiremos informes DVB-IP Offering Record.

Tal como se ha comentado con anterioridad existen dos tipos de entrega de estos informes, “TS Full SI” o bien “TS Optional SI”. Para cada uno de estos tipos existe un tipo de informe distinto, aunque disponen de muchos campos en común y una “cabecera” común (Tabla 3.3).

Tabla 3.3. DVB-IP Offering Record

DVB-IP Offering Record	**Descripción de los atributos**	**Obligatorio(Ob)/ Opcional(Op)**
Nombre de dominio del SP	Nombre de dominio registrado por el SP que lo identifique	Ob
Número de versión	Versión del informe, se incrementará para cada cambio sufrido	Op (en modelo push)

3.3.3.1. TS Full SI

El TS Full SI Bcast Discovery (Tabla 3.4) deriva de DVB-IP Offering Record. Provee toda la información necesaria para encontrar los servicios disponibles que lleva el SI.

Tabla 3.4. Broadcast Discovery Record – TS Full SI

Broadcast Discovery Record - TS Full SI	**Descripción de los atributos**	**Obligatorio(Ob)/ Opcional(Op)**
Localización del servicio	Transporta la dirección física de las descripciones agregadas del servicio	Op
Servicio	Para cada servicio...	
Identificador de servicio	Identificador único del servicio	Ob
Identificador textual del servicio	Identificador único del servicio. Es un nombre de host único	Op
Nombre de dominio del SP	Nombre de dominio registrado por el SP que lo identifique	Ob
Hostname del servicio	Un nombre de host único para el servicio	Ob
Tripleta DVB	Tripleta identificadora del servicio DVB (original_network_id, transport_stream_id and service_id)	Ob
Identificador de la red	Identificador de la red del sistema de entrega	Ob
Identificador del TS	Identifica el TS	Ob

Broadcast Discovery Record - TS Full SI	**Descripción de los atributos**	**Obligatorio(Ob)/ Opcional(Op)**
Identificador del servicio	Identifica el servicio respecto a cualquier otro dentro del TS. Corresponde al número de programa dentro de la tabla PMT	Ob
Tasa máxima	Especifica la tasa máxima del servicio	Op
Localizador del servicio	Localizador que especifica donde puede encontrarse el servicio. Tanto IGMP como RTSP están soportados. Uno de los dos debe estar especificado en este campo	Ob
Protocolo IGMP	Indica el uso de IGMP en el servicio	Op
Dirección	Dirección IP multicast	Ob**
Puerto	Número de puerto	Ob**
Dirección origen	Dirección IP unicast del origen del TS	Op
Protocolo RTSP	Indica el uso de RTSP	Op
Identificador textual del servicio	Identificador único del servicio. Es un nombre de host único	Op

** En este caso significa que si se usa IGMP estos campos serán obligatorios

3.3.3.2. Optional SI

El "TS - Optional SI" Broadcast Discovery Information Record (Tabla 3.5) deriva del DVB IP Offering Record. Éste provee toda la información necesaria para crear una lista de servicios disponibles con la suficiente información para que el usuario pueda elegir y acceder al servicio, además de otros campos opcionales.

Tabla 3.5. Broadcast Discovery Record – TS Optional SI

Broadcast Discovery Record – TS Opt. SI	**Descripción de los atributos**	**Obligatorio(Ob)/ Opcional(Op)**
Localización del servicio	Transporta la dirección física de las descripciones agregadas del servicio	Op
Servicio	Para cada servicio	
Identificador de servicio	Identificador único del servicio	Ob
Identificador textual del servicio	Identificador único del servicio. Es un nombre de host único	Op

Broadcast Discovery Record – TS Opt. SI	**Descripción de los atributos**	**Obligatorio(Ob)/ Opcional(Op)**
Nombre de dominio del SP	Nombre de dominio registrado por el SP que lo identifique	Ob
Hostname del servicio	Un nombre de host único para el servicio	Ob
Tripleta DVB	Tripleta identificadora del servicio DVB	Ob
Identificador de la red	Identificador de la red del sistema de entrega	Ob
Identificador del TS	Identifica el TS	Ob
Identificador del servicio	Identifica el servicio respecto a cualquier otro dentro del TS. Corresponde al número de programa dentro de la tabla PMT	Ob
Tasa máxima	Especifica la tasa máxima del servicio	Op
Disponibilidad de países	Lista de países y/o grupos de países donde el servicio está disponible	Op
Descriptor de soporte de anuncio	Identifica el tipo de anuncios que soporta el servicio	Op
Descriptor de servicio de reemplazo	Identifica un servicio de reemplazo de servicios que se puede seleccionar automáticamente por el IRD cuando la decodificación del servicio falla	Op
Origen SI primario	Indica qué fuente de la información para dar prioridad en caso de que las tablas DVB-SI estén presentes	Op
Localizador del servicio	Localizador que especifica donde puede encontrarse el servicio. Tanto IGMP como RTSP están soportados, uno de los dos debe estar especificado en este campo	Ob
Protocolo IGMP	Indica el uso de IGMP en el servicio	Op
Dirección	Dirección IP multicast	Ob**
Puerto	Número de puerto	Ob**
Dirección origen	Dirección IP unicast del origen del TS	Op
Protocolo RTSP	Indica el uso de RTSP	Op
Identificador textual del servicio	Identificador único del servicio, es un nombre de host único	Op

** En este caso significa que si se usa IGMP estos campos serán obligatorios

3.4. Identificación del servicio

Para que el protocolo pueda funcionar deberemos poder identificar al SP y a los servicios ofrecidos por este. El SP se identifica únicamente con el nombre del dominio DNS que se haya registrado.

Para los servicios existen dos mecanismos básicos para su única identificación, de los que se usará la tripleta DVB. Se trata de un trío de identificadores numéricos: original_network_id, transport_stream_id y service_id, con lo cual permite distinguir entre el mismo servicio transportado por diferentes redes. Por ejemplo, el trío distinguiría TVE1 transportado por Hispasat o por Astra. Esta tripleta está presente en los informes anteriormente mostrados en las Tablas 3.4 y 3.5.

3.5. Selección de servicio

El usuario final puede acceder al servicio por las siguientes vías:

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

En los servicios que utilizan tecnología multicast para su transmisión, el flujo enviado es continuo y no necesita ser iniciado por ningún HNED. Los dispositivos finales pueden unirse y dejar el servicio multicast simplemente utilizando el mensaje IGMP pertinente. El campo Localizador de Servicio, presente en los informes anteriormente mostrados en las Tablas 3.4 y 3.5, da toda la información necesaria para editar el mensaje IGMP.

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, etc. En estos sistemas se usará RTSP que además servirá para adquirir parámetros requeridos por IGMP.

Los servicios que utilizan tecnología unicast se entienden para un único cliente y este deberá iniciar una sesión contra el servidor. En este caso se utilizará RTSP.

3.6. Transporte de SD&S

El estándar define dos mecanismos de transporte de informes SD&S, uno para multicast y otro para unicast. Para multicast se define DVBSTP para entregar los informes XML. DVBSTP añade una cabecera para controlar la sucesión de segmentos, de secciones, etc.

3.6.1. Sintaxis

La cabecera del protocolo DVBSTP es la que se muestra a continuación:

Tabla 3.6. Cabecera DVBSTP

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*0*	*1*
Ver		*Resrv*			*Enc*		*C*	*Total Segment Size*																							
Payload ID								*Segment ID*																*Segment Version*							
Section Number															*Last Section Number*									*Compr*			P	*HDR_LEN*			
(Conditional) SP ID																															
(Optional) Private Header Data																															
Payload																															
Payload																								*(Optional) CRC*							
(Optional) CRC (Cont)																															

3.6.2. Semántica

A continuación se comentan cada uno de los campos:

- **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, p.ej. Cuando section_number == last_section_number.
- **Tamaño Total del Segmento:** Aun 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, p.ej. Puedes tener múltiples informes Broadcast Discovery Information, y cada una tendrá asignada una única ID.
- **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 3.7.

Tabla 3.7. Valores de compresión

Valor de compresión	Significado	Significado del tamaño total del segmento
000	Sin compresión	Tamaño transmitido
001 a 110	Reservado	
111	Privado	Definido por el usuario

- **ProviderID Flag (P):** 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. n optional 32-bit CRC.

3.6.3. Pila de protocolos para el transporte de DVB sobre IP

A continuación, en la figura 3.6, se muestra la pila de protocolos de transporte de informes SD&S.

La capa superior, Service offering, hace referencia al servicio ofrecido por los SP, por lo tanto se refiere a los programas, la información sobre los programas,... en resumen, los informes ofrecidos por el SP.

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

Los informes SD&S se transportan de acuerdo con el protocolo especificado en el estándar según su dirección IP destino, según si es multicast o unicast. Para servicios multicast, *push model,* se transportan en paquetes IP de acuerdo al protocolo de transporte DVBSTP, que a su vez se transporta típicamente sobre UDP. Para servicios unicast, *pull model*, la información SD&S se transporta vía HTTP.

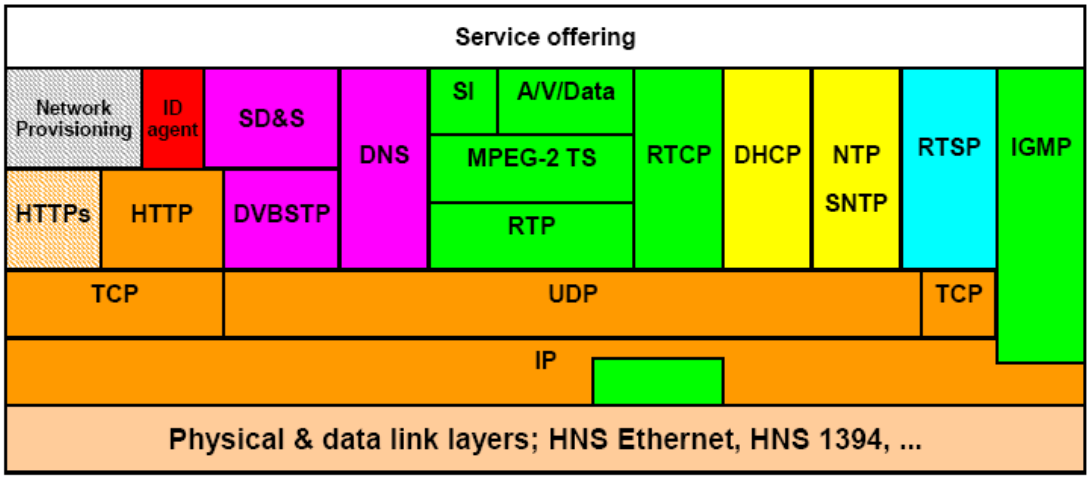


Fig. 3.6 Pila de protocolos, extraída de [11]

3.6.4. Ejemplo de envío de SD&S

Partiendo de la sintaxis del protocolo DVBSTP mostrada en el punto 2.6.1. paso a rellenar todos los campos de la cabecera (Tabla 3.8) con los valores típicos y contando con un tamaño de ejemplo de informe XML de 501 bytes.

Estos valores típicos son los siguientes, donde los no comentados se introducen por defecto a la espera de futuras utilidades.

- **Versión**: 2 bits = “00”
- **Reservado**: 3 bits = “000”
- **Encriptación**: 2 bits = ”00”. Indicando que el payload no está encriptado.
- **CRC**: 1 bit = “0”. Como no habrá CRC debe valer 0
- **Tamaño total del segmento**: 24 bits = “111110101”. Equivalentes a 501 bytes de datos útiles
- I**D de payload**: 8 bits = “0x01”. En este caso, y de acuerdo con la Tabla 3.1, valdrá 0x01 para el caso de ser un SP Discovery Information
- **ID del segmento**: 16 bits = “0x0001”. Como no se realizarán fragmentaciones ni de segmentos ni de secciones este y los 3 siguientes campos valdrán 1 todos ellos.

- **Versión del Segmento**: 8 bits = “0x01”
- **Número de Sección**: 12 bits = “00000000001”
- **Último Número de Sección**: 12 bits = “000000000001”
- **Compresión**: 3 bits = “000”. Como no se realizará compresión su valor debe ser el indicado según la Tabla 3.7.
- **ProviderID Flag**: 1 bit = “0”. No estará presente el campo SPID en la cabecera.
- **HDR_LEN**: 4 bits = “0000”. Valor que toma cuando no hay presencia de cabecera privada.
- **Service Provider ID**: Este campo no estará presente (ProviderID = 0)
- **Cabecera Privada**: Tampoco estará presente (HDR_LEN = 0000)
- **Payload**: El informe en sí.
- **CRC**: No estará presente (Flag CRC = 0)

Tabla 3.8. Ejemplo paquete DVBSTP

0	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***0***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***0***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***0***	***1***
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
(Conditional) SP ID (No)																															
(Optional) Private Header Data (No)																															
Payload																															
Payload																								(Optional) CRC							
(Optional) CRC (Cont)																															

Una vez formada la cabecera anterior e introducidas las secciones del informe deberíamos empaquetarlo todo según la pila de protocolos mostrada en el punto 2.6.3. como se muestra en la Figura 3.7.

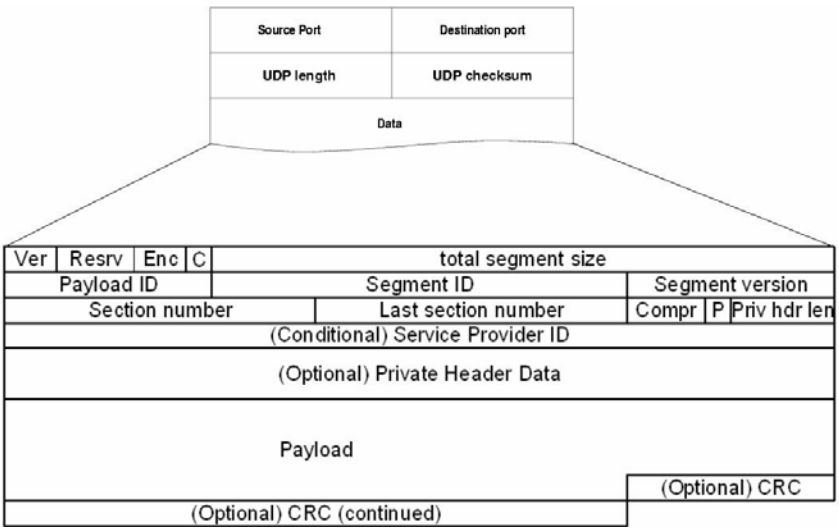


Fig. 3.7 Encapsulación del paquete DVBSTP formado

4. DISEÑO Y DESARROLLO DE LA APLICACIÓN

En este punto se explica el diseño y desarrollo de uno de los objetivos de este proyecto, un software basado en el apartado de Service Discovery del estándar *ETSI TS 102 033 v1.1.1: DVB; Transport of MPEG-2 Based DVB Services over IP Based Networks*.

Primero se describirá el escenario montado para realizar las pruebas del software desarrollado, luego se comentarán el software libre utilizado y finalmente se detallarán las pruebas realizadas.

4.1. Escenario

El escenario desarrollado constará de dos bloques, servidor y cliente, donde el bloque servidor lo componen otros tres bloques. Lo formarán 2 máquinas conectadas a la red de la escuela:

- El primer equipo funciona sobre un sistema operativo Linux Debian.
- El segundo equipo funciona con Linux Ubuntu 5.03.
- Estos dos equipos conectados a la red

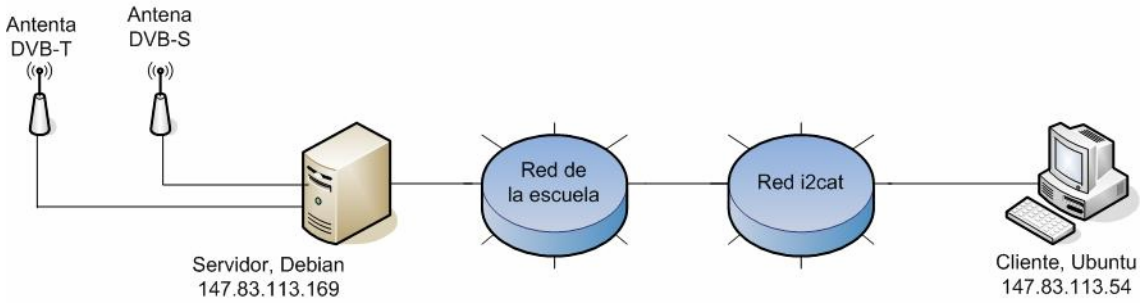


Fig. 4.1. Escenario montado poner

El primer equipo realizará las funciones de servidor de SD y de servidor de vídeo. Es donde se colocarán las tarjetas DVB (DVB-S y DVB-T) y el código desarrollado. Este sistema operativo debe tener las funciones de red multicast habilitadas para poder efectuar sus funciones de streaming.

El segundo equipo realizará las funciones de cliente y deberá tener instalado, a parte del software desarrollado, el cliente de vídeo VLC.

4.1.1. Servidor

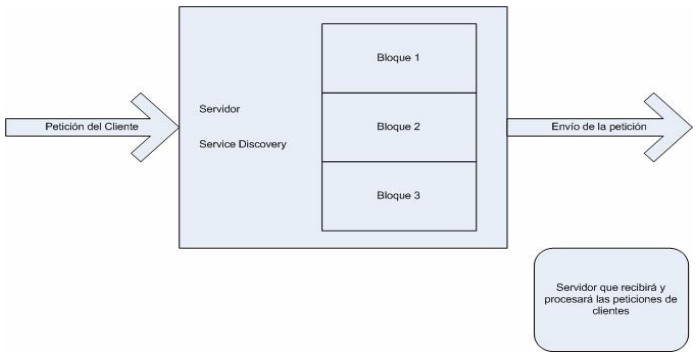


Fig. 4.2 Bloque general del servidor

Este primer bloque se refiere al servidor en su totalidad, la aplicación completa vista desde fuera. Como ya he dicho anteriormente, este bloque se compone de tres más.

4.1.1.1. Bloque 1, core

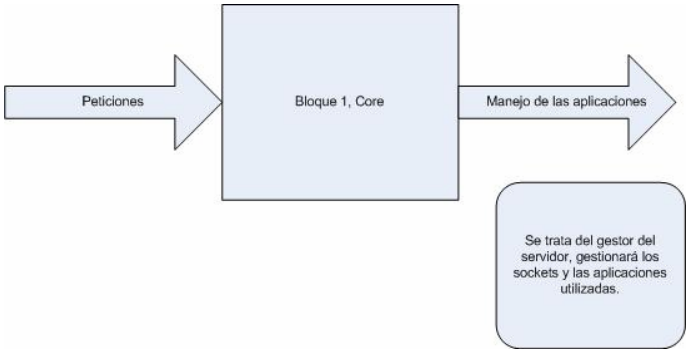


Fig. 4.3 Bloque core

El bloque 1 es el core del servidor. Se encargará de procesar las peticiones de los clientes, manejar los sockets y llamar a la aplicación pertinente en cada caso.

4.1.1.2. Bloque 2, los analizadores

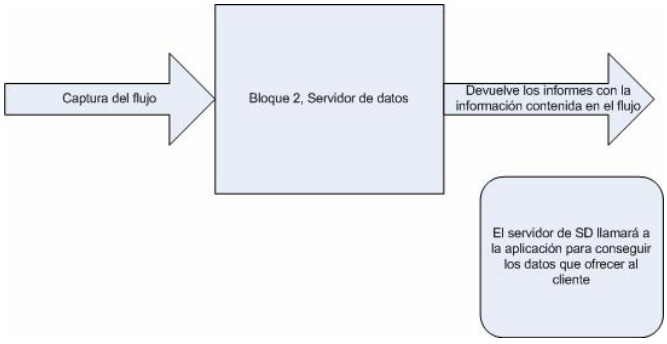


Fig. 4.4 Bloque analizador

El bloque 2, los analizadores, será llamado por el bloque 1 del servidor en caso de que necesite obtener datos del flujo de vídeo, estos datos serán los necesarios para poder ofrecer el servicio al cliente. Con los datos obtenidos el servidor podrá formar los informes SD&S para enviarlos a la dirección multicast y también podrá enviar flujos de vídeo correctamente.

4.1.1.3. Bloque 3, el servidor de vídeo

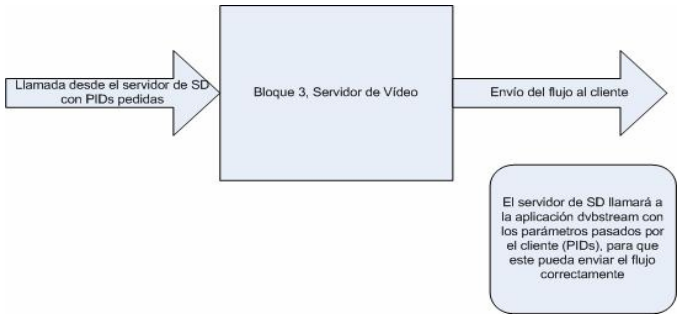


Fig. 4.5 Bloque servidor de vídeo

El bloque 3, servidor de vídeo es el encargado de "tratar" con las tarjetas receptoras de DVB-T y DVB-S, por lo que será el encargado de realizar una captura inicial a partir de la cual se obtendrán los informes. También será el encargado de difundir flujo de vídeo sobre la LAN en caso necesario.

4.1.2. Cliente

Por otro lado tenemos el cliente que al escuchar a la dirección multicast asociada a la aplicación recibirá los informes. Una vez recibidos hará peticiones para recibir el programa deseado.

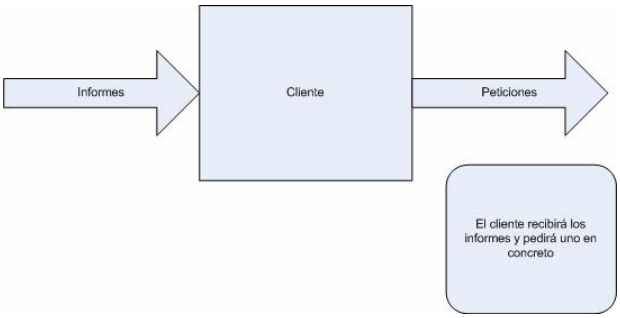


Fig.4.6 Bloque Cliente

4.1.3. Conexión entre bloques

En este capítulo se muestra la conexión entre los bloques anteriormente mencionados así como su funcionalidad exacta.

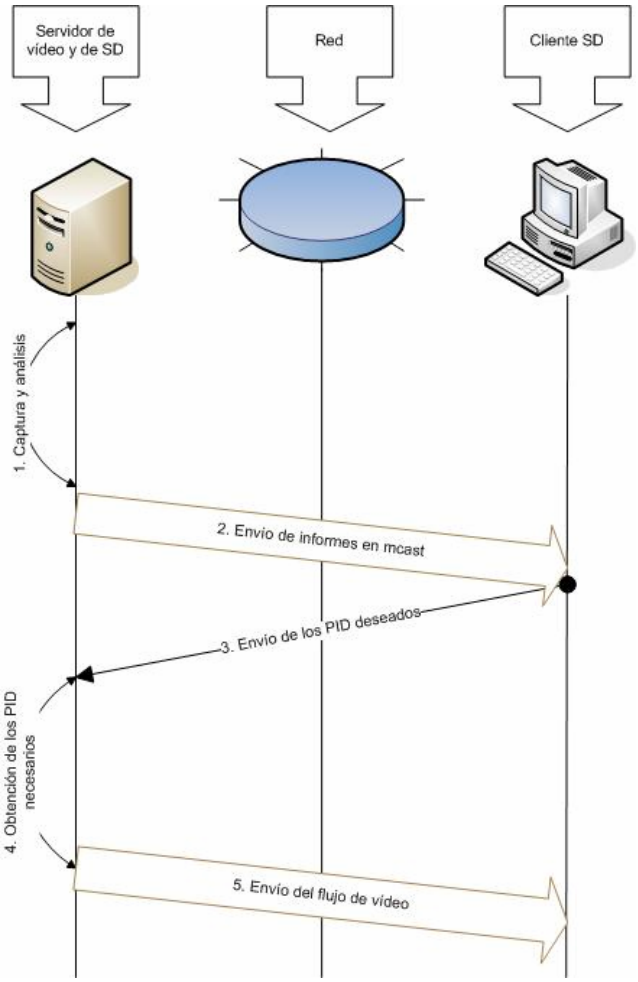


Fig. 4.7 Conexión entre bloques

1. El servidor mediante los bloques 2 y 3 realizará una captura del flujo entrante para poder analizarlo y así formar los informes SD&S.

2. Una vez formados los informes los enviará a una dirección multicast (bloque 1 del servidor).

3. El cliente (bloque cliente) activará su aplicación y recibirá y visualizará el contenido del informe. En caso de que desee recibir un programa deberá enviar un mensaje al servidor indicando los PIDs asociados a dicho programa.

4. El servidor, a la escucha (bloque 1), recibirá los PIDs del cliente y deberá volver a analizar el flujo (bloque 2) para obtener todos los datos necesarios para poder enviar el flujo correctamente.

5. El servidor enviará el programa deseado al cliente (bloque 3).

4.2. Software utilizado

El software principal, tanto cliente como servidor, lo he desarrollado a partir de clientes y servidores UDP ya creados y modificándolos según mis necesidades.

Para los analizadores y para el servidor de vídeo he utilizado, y modificado según mi conveniencia, dos aplicaciones de libre distribución. En cuanto al cliente de vídeo se ha utilizado VLC.

4.2.1. Analizadores

Para analizar el flujo de vídeo he utilizado dos aplicaciones de la librería libdvbpsi [21], decode_pat, decode_pmt y decode_sdt. Libdvbpsi es una librería multiplataforma desarrollada por los creadores de VideoLan que sirve para decodificar y generar tablas de TS y DVB.

En un principio estas aplicaciones devolvían una serie de datos de los que no todos eran necesarios para entregarlos al cliente según el estándar, del mismo modo que tampoco era correcto el modo de presentarlos, de tal manera que he modificado estas aplicaciones a fin de que devuelvan directamente el informe XML especificado por el estándar.

De este modo el servidor al llamar a los analizadores formará directamente el informe XML que debe enviar a los clientes.

4.2.2. Servidor de vídeo

Dvbstream [19] es un software de libre distribución que permite tanto actuar como servidor como de cliente de vídeo. Nos permite, por un lado recibir el flujo entrante desde una tarjeta receptora DVB, (ya sea satélite DVB-S, cable DVB-C o terrestre DVB-T) y lo distribuye sobre una Red de Área Local, LAN, o bien grabarlo en un archivo.

Hay varios flags que se pueden utilizar que nos ofrecen diversas opciones:

Tabla 4.1. Flags dvbstream

Descripción de la función	Flag a utilizar	Características
Selección de la tarjeta de entrada	-c 0/1	0 --> Satélite 1 --> Terrestre
Frecuencia	-f frecuencia	
Polaridad	-p V/H	V --> Vertical H --> Horizontal

Descripción de la función	Flag a utilizar	Características
SR	-s	
PID vídeo	-v PID	
PID audio	-a PID	
Análisis de flujos	-analyse	
Redireccionar el flujo	-i @IP	
Transformar en PS Stream	-ps	

4.2.3. Validador XML

Tal como se comenta en el apartado 3.1 los informes enviados en formato XML deben ser válidos y bien formados. Para esto utilizaré una aplicación para linux, *xmlstarlet* [32].

4.2.4. Cliente de vídeo

En el cliente he utilizado VLC como reproductor de vídeo, al ser una máquina corriendo con S.O. Debian he utilizado la aplicación aptitude para instalar el software. A parte de instalar el software en sí deberemos instalar (\$*apt-get install XXX*) todos los plugin disponibles para su correcto funcionamiento. Los plugins disponibles se pueden ver en la figura 4.8.

```
vlc-plugin-alsa - ALSA audio output plugin for VLC
vlc-plugin-arts - aRts audio output plugin for VLC
vlc-plugin-esd - Esound audio output plugin for VLC
vlc-plugin-ggi - GGI video output plugin for VLC
vlc-plugin-glide - Glide video output plugin for VLC
vlc-plugin-sdl - SDL video and audio output plugin for VLC
vlc-plugin-svgalib - SVGAlib video output plugin for VLC
```

Fig. 4.8 Plugins de VLC

4.3. Pruebas preliminares

4.3.1. Analizadores

Para hacer las pruebas preliminares con los analizadores de flujo de vídeo se ha realizado una captura del canal 61 (frecuencia = 794000 Khz) de Barcelona, correspondiente al TS de Televisió de Catalunya, flujo que incluye los canales TV3, K3/33, 3/24 y Canal300. Esta captura, canal61.ts, la realizaremos mediante el servidor de vídeo con la siguiente sentencia:

$dvbstream -c 1 -f 794000 -s 27500 8192 -o > canal61.ts

Sentencia, que nos permite capturar todos los PIDs del flujo contenido en la frecuencia indicada al indicar el PID 8192. Como paso previo analizaremos la captura con TSReader con el fin de tener una referencia para comprobar el correcto funcionamiento de la aplicación (figura 4.9).

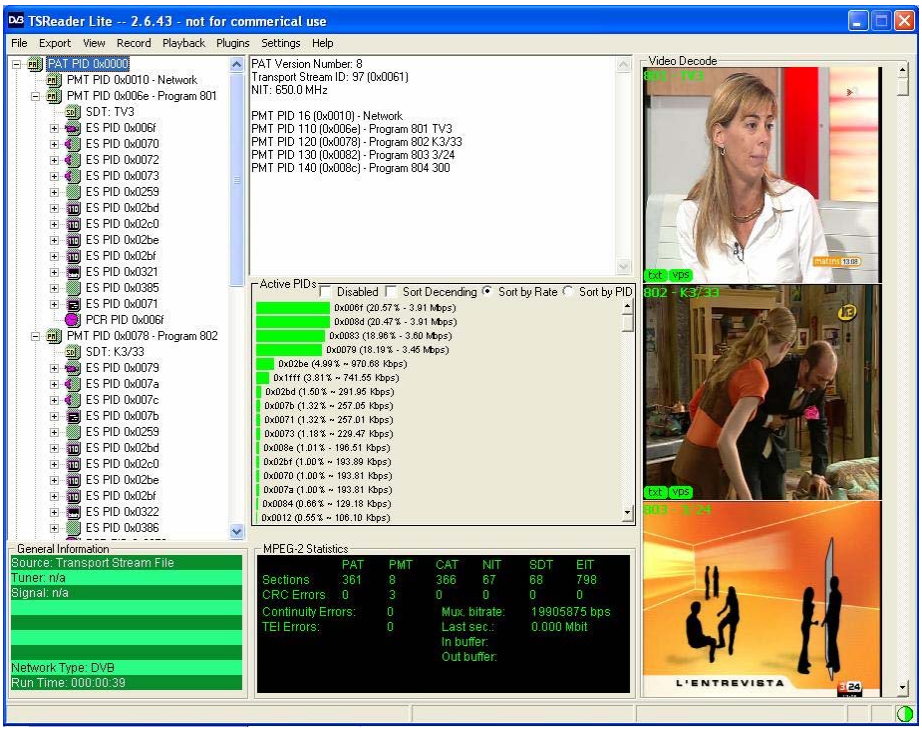


Fig. 4.9 Captura de TSReader analizando la captura canal61.ts

A partir de esta captura haré una serie de pruebas, antes de modificar los analizadores, para ver sus resultados y su modo de empleo. Cabe destacar que cuando llamamos a estas aplicaciones salen errores por pantalla, errores correspondientes a la reordenación por los números de serie de los paquetes TS, posiblemente debidas a la pérdida momentanea del canal de radio.

4.3.1.1. Prueba decode_pat original

Modo de empleo: *$decode_pat fichero.ts*

En este caso, igual que en las siguientes pruebas preliminares realizadas con los analizadores, redirigiré la salida a un fichero para tenerlo como referencia en cuanto a los datos que nos ofrecen los distintos analizadores.

Esta aplicación extrae la tabla PAT del flujo (Anexo III), tabla que contiene los PID de todos y cada uno de los programas que contiene el flujo analizado.

```
decode_pat canal61.ts > canal61.pat
libdvbpsi error (PSI decoder): TS discontinuity (received 0, expected
12)
[...]
```

con lo que obtengo canal61.pat,

```
New PAT

transport_stream_id : 97

version_number      : 6
    | program_number @ [NIT|PMT]_PID
    |                 0 @ 0x10 (16)
    |           801 @ 0x63 (110)
    |           802 @ 0xc8 (120)
    |           803 @ 0x12c (130)
    |           804 @ 0x1f3 (140)

active              : 1
```

En el resultado obtenido podemos ver el ID del TS, la versión del informe y los programas disponibles así como su correspondiente PID de tabla PMT. Ej: TS id : 97, Version : 6, Nº Prog. : 801, PMT PID 110.

4.3.1.2. Prueba decode_pmt original

Modo de empleo: *$decode_pmt fichero.ts program_number pmt_pid*

Donde los valores de *program_number* y *pmt_pid* los obtendré de los valores obtenidos con decode_pat.

```
$decode_pmt canal61.ts 801 110 > canal61.pmt
libdvbpsi error (PSI decoder): TS discontinuity (received 10, expected
0)
libdvbpsi error (PSI decoder): TS discontinuity (received 13, expected
12)
libdvbpsi error (PSI decoder): TS discontinuity (received 7, expected
6)
libdvbpsi error (PSI decoder): TS discontinuity (received 6, expected
5)
libdvbpsi error (PSI decoder): TS discontinuity (received 13, expected
13)
libdvbpsi error (PSI decoder): TS discontinuity (received 6, expected
5)
```

obtengo canal61.pmt,

```
New active PMT
  program_number : 801
  version_number : 2
  PCR_PID        : 0xc9 (110)
    | type @ elementary_PID
    | 0x02 @ 0xc9 (111)
    |  ] 0x11 : "ÿ"
    | 0x03 @ 0xcb (112)
    | 0x03 @ 0xcc (113)
    | 0x06 @ 0xca (114)
    |  ] 0x56 : "esl    d"
[...]
```

Una vez comprobado el modo de empleo de las aplicaciones originales paso a comentar las aplicaciones modificadas, modificación necesaria para que estas aplicaciones devolvieran exactamente los valores necesarios en formato de informe XML.

En el resultado obtenido podemos ver los PIDs de vídeo, audio, etc. del flujo seleccionado, que en este caso lo hemos especificado indicando el programa número 801 y el PID de la tabla PMT (110).

4.3.1.3. Prueba decode_pat modificado

Modo de empleo: *$decode_pat fichero.ts > ficheroPAT.txt*

```
$decode_pat canal61.ts > canal61.pat
[...]
```

en la salida, canal61.pat, obtengo directamente el informe XML, SD&S, que mandará el servidor a los clientes.

```
<ServiceDiscovery> <ServiceProvider DomainName="bean.upc.es" Version="8"
anyURI="proyecto_Toni.com">
<Name Language="SPA">SD</Name>
<Description Language="SPA">Servidor SD bean</Description>
<Offering><Push Source="84.88.32.52" Address="224.0.1.2" Port="1600">
<PayloadId Id="01"><Segment ID="0000" Version="1"/></PayloadId>
</Push>
</Offering>
</ServiceProvider>
<BroadcastOffering DomainName="toniProyecto.com" Version="1">
<ServiceList Version="1">
<DescriptionLocation></DescriptionLocation><SingleService><ServiceLocation><IP
MulticastAddress Port="1600">224.0.1.2</IPMulticastAddress></ServiceLocation>
<TextualIdentifier ServiceName="_"></TextualIdentifier>
<DVBTriplet OrigNetId="0x54582000" TSId="97" ServiceId="16"></DVBTriplet>
<SI ServiceType=""
PrimarySISource=""><Description>prog_ID:0</Description></SI></SingleService>
<SingleService><ServiceLocation><IPMulticastAddress
Port="1600">224.0.1.2</IPMulticastAddress></ServiceLocation>
<TextualIdentifier ServiceName="_"></TextualIdentifier>
<DVBTriplet OrigNetId="0x54582000" TSId="97" ServiceId="110"></DVBTriplet>
<SI ServiceType=""
PrimarySISource=""><Description>prog_ID:801</Description></SI></SingleService>
```

```
<SingleService><ServiceLocation><IPMulticastAddress
Port="1600">224.0.1.2</IPMulticastAddress></ServiceLocation>
<TextualIdentifier ServiceName="_"></TextualIdentifier>
<DVBTriplet OrigNetId="0x54582000" TSId="97" ServiceId="120"></DVBTriplet>
<SI ServiceType=""
PrimarySISource=""><Description>prog_ID:802</Description></SI></SingleService>
<SingleService><ServiceLocation><IPMulticastAddress
Port="1600">224.0.1.2</IPMulticastAddress></ServiceLocation>
<TextualIdentifier ServiceName="_"></TextualIdentifier>
<DVBTriplet OrigNetId="0x54582000" TSId="97" ServiceId="130"></DVBTriplet>
<SI ServiceType=""
PrimarySISource=""><Description>prog_ID:803</Description></SI></SingleService>
<SingleService><ServiceLocation><IPMulticastAddress
Port="1600">224.0.1.2</IPMulticastAddress></ServiceLocation>
<TextualIdentifier ServiceName="_"></TextualIdentifier>
<DVBTriplet OrigNetId="0x54582000" TSId="97" ServiceId="140"></DVBTriplet>
<SI ServiceType=""
PrimarySISource=""><Description>prog_ID:804</Description></SI></SingleService>
</ServiceList>
</BroadcastOffering>
</ServiceDiscovery>
```

4.3.1.4 Prueba decode_pmt modificado

Modo de empleo: *$decode_pmt fichero.ts program_number pmt_pid*

```
$decode_pmt canal61.ts 801 110 > canal61.pmt
libdvbpsi error (PSI decoder): TS discontinuity (received 10, expected 0)
[...]
```

en la salida, canal61.pmt, obtengo directamente los PIDs asociados al programa seleccionado:

```
111
112
113
114
115
601
701
901
801
702
703
704
```

Para comprobar su correcto funcionamiento corroboraremos los datos obtenidos cargando el fichero de vídeo con TSReader[4] [30] (Figura 4.10). Obtenemos los mismos resultados:

```
Program Number: 801
PCR on PID 111 (0x006f)
PMT Version: 16
```

[4] TSReader es un analizador, manipulador, decodificador y grabador de flujos MPEG-2 [30].

```
Stream Type: 0x02 MPEG-2 Video
 Elementary Stream PID 111 (0x006f)


Stream Type: 0x03 MPEG-1 Audio
 Elementary Stream PID 112 (0x0070)


Stream Type: 0x03 MPEG-1 Audio
 Elementary Stream PID 114 (0x0072)


Stream Type: 0x06 Dolby AC3 Audio
 Elementary Stream PID 115 (0x0073)


Stream Type: 0x0c ISO/IEC 13818-6 type C
 Elementary Stream PID 601 (0x0259)


Stream Type: 0x0b ISO/IEC 13818-6 type B
 Elementary Stream PID 701 (0x02bd)


Stream Type: 0x0b ISO/IEC 13818-6 type B
 Elementary Stream PID 704 (0x02c0)


Stream Type: 0x0b ISO/IEC 13818-6 type B
 Elementary Stream PID 702 (0x02be)


Stream Type: 0x0b ISO/IEC 13818-6 type B
 Elementary Stream PID 703 (0x02bf)


Stream Type: 0x06 ISO/IEC 13818-1 PES packets containing private data
 Elementary Stream PID 801 (0x0321)


Stream Type: 0x05 ISO/IEC 13818-1 private_sections
 Elementary Stream PID 901 (0x0385)


Stream Type: 0x06 Teletext/VBI
 Elementary Stream PID 113 (0x0071)
```

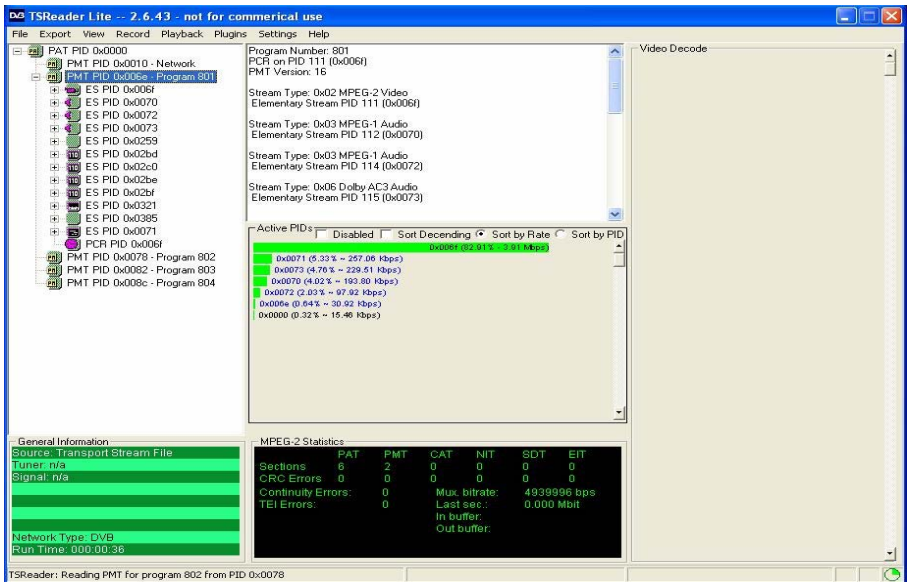


Fig 4.10 Captura de TSReader analizando la traza obtenida

Comparando los resultados de decode_pmt original y modificado, podemos ver como en la aplicación modificada se nos devuelven más valores. Mientras que en el original tenemos PIDs [111-115], en el modificado tenemos además PIDs 601, [701-704], 801 y 901. En la captura de TSReader podemos ver a que corresponde cada uno de estos PIDs.

Los PIDs [111- 115] corresponden a los canales de vídeo, audio y teletexto, mientras que el resto corresponden a los estándares ISO/IEC 13818-6 y ISO/IEC 13818-1:

- ISO/IEC 13818-6 Describe extensiones para DSM-CC (Comando Digital de herramientas de almacenamiento y control)
- ISO/IEC 13818-1 Sistema - describe sincronización y multiplexado de video y audio.

Al ser estándares de pago [33], no los he conseguido y por lo tanto no he profundizado más en el tema.

4.3.2. Pruebas servidor de vídeo

Para hacer las pruebas se ha utilizado software de distribución libre, en concreto *dvbstream* [19] y *VideoLan Client* [20].

Como ya he comentado dvbstream es un servidor de vídeo que nos permitirá redirigir el flujo entrante de las tarjetas DVB-T y DVB-S disponibles hacia nuestra LAN o bien obtener capturas para analizarlas. En este caso realizaré las pruebas con el flujo entrante DVB-T únicamente.

VideoLan también es un software de libre distribución que nos permitirá recibir el flujo unicast o multicast y reproducirlo en nuestra máquina cliente.

Dvbstream lo ejecutaremos desde la máquina servidor (debian) y mientras que en la otra máquina, cliente, con Ubuntu tendremos instalado VLC para que actúe como receptor del vídeo.

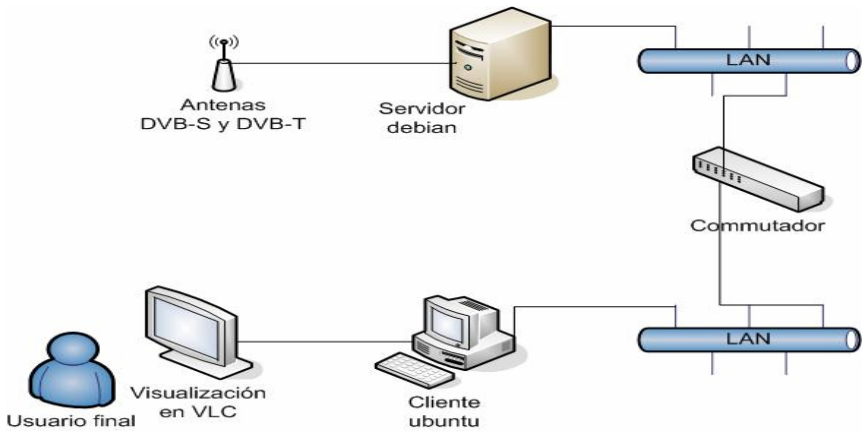


Fig. 4.11 Escenario montado

En nuestro caso para poder redireccionar el flujo entrante por la tarjeta de video terrestre utilizaremos la siguiente sentencia, usando los PIDs obtenidos en las pruebas de los analizadores.

Para que VLC pueda reproducir correctamente el flujo de vídeo que le enviemos con dvbstream necesitamos, a parte de los PIDs de vídeo o audio asociados al programa, enviar el PID de la tabla PMT asociada al flujo y el de la tabla PAT.

Para obtener todos estos PIDs necesarios primero analizaremos el flujo con decode_pat con lo que obtendremos el PMT PID asociado a cada programa. Con este valor podremos llamar a decode_pmt que nos devolverá todos los PIDs asociados.

$dvbstream -c 1 -f 794000 -s 27500 0 110 111 -i @IP

Con la opción "-c 1" elegimos el flujo que entra por la tarjeta terrestre, he hecho las pruebas con el flujo de TV3 que se corresponde con la frecuencia 794000 Hz y los PID 0, 110 y 111, PIDs obtenidos en las pruebas anteriores con los analizadores. Para que el vídeo se pueda reproducir correctamente en el cliente VideoLan deberemos enviar el PID 0 correspondiente a las tablas PAT y el PID PMT que corresponde a las tablas PMT (en este caso 110). Como última opción elegiremos la @IP a la que queremos redireccionar el flujo.

Con esta sentencia podemos ver, con ethereal[5], como el cliente recibe los paquetes RTP y lo visualizaremos de la siguiente forma:

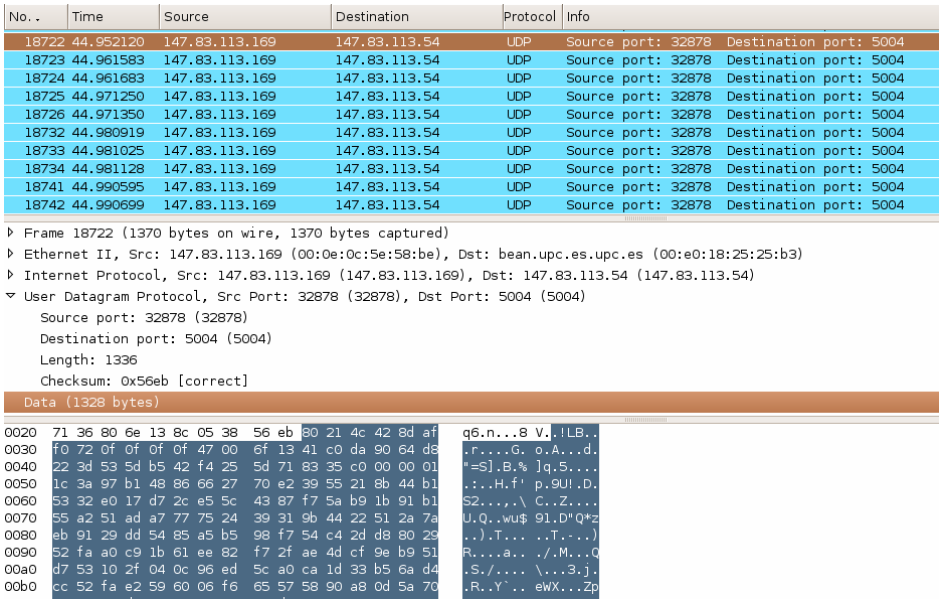


Fig. 4.12 Captura dvbstream enviando flujo de vídeo

[5] Ethereal es un analizador de protocolos, utilizado para realizar análisis y solucionar problemas en redes de comunicaciones, para desarrollo de software y protocolos [28].

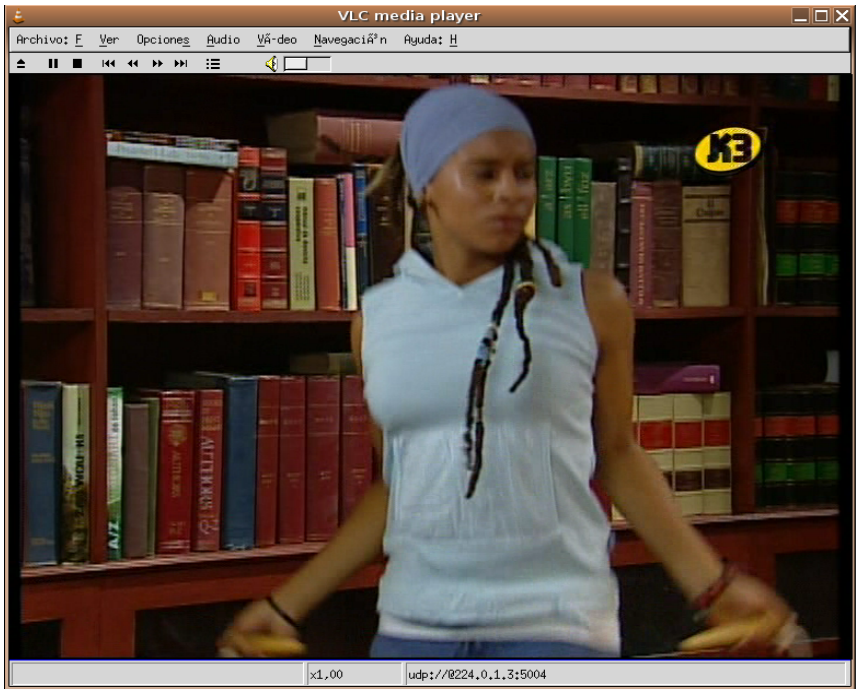


Fig. 4.13 Reproducción del flujo

4.3.2. Pruebas validador XML

Para validar los informes XML hay que llamar a la aplicación tal como se ve en la figura 4.14.

```
rooty@bean:~$ xmlstarlet
XMLStarlet Toolkit: Command line utilities for XML
Usage: xml [<options>] <command> [<cmd-options>]
where <command> is one of:
  ed    (or edit)       - Edit/Update XML document(s)
  sel   (or select)     - Select data or query XML document(s) (XPATH, etc)
  tr    (or transform)  - Transform XML document(s) using XSLT
  val   (or validate)   - Validate XML document(s) (well-formed/DTD/XSD/RelaxNG)
  fo    (or format)     - Format XML document(s)
  el    (or elements)   - Display element structure of XML document
  c14n  (or canonic)    - XML canonicalization
  ls    (or list)       - List directory as XML
  esc   (or escape)     - Escape special XML characters
  unesc (or unescape)   - Unescape special XML characters
  pyx   (or xmln)       - Convert XML into PYX format (based on ESIS - ISO 8879)
  p2x   (or depyx)      - Convert PYX into XML
<options> are:
  --version             - show version
  --help                - show help
Wherever file name mentioned in command help it is assumed
that URL can be used instead as well.

Type: xml <command> --help <ENTER> for command help

XMLStarlet is a command line toolkit to query/edit/check/transform
XML documents (for more information see http://xmlstar.sourceforge.net/)
rooty@bean:~$ xmlstarlet val informe.xml
informe.xml - valid
rooty@bean:~$
```

Fig 4.14 Llamada a xmlstarlet

Para el ejemplo de muestra se ha utilizado un informe creado por el servidor SD (informe.xml). A la salida nos dirá si el informe es válido o no.

5. PRUEBAS Y RESULTADOS

5.1. Software SD

Una vez desarrollado el software completo paso a realizar las pruebas pertinentes para comprobar su correcto funcionamiento. Por una parte comprobaremos el buen funcionamiento del servidor y por otro el del cliente. Los pasos seguidos por el software son los descritos en el punto 4.1.3.

Como ya he comentado, la versión final del software está basada en la técnica multicast; pero previamente, debido a la imposibilidad de utilizar multicast ya que los routers no lo soportaban, tuve que desarrollar una versión unicast que, aunque no cumple el estándar en cuando a comunicación unicast (vía HTTP), me sirvió para ir progresando mientras esperaba poder utilizar multicast. Es por eso que también incluiré en el CD del proyecto esta versión unicast.

5.1.1. Servidor

Arrancamos el servidor SD, al llamarlo debemos pasarle por parámetro el nombre que queremos darle a la captura que la aplicación realizará como primer paso. También le pasaremos la dirección IP del grupo multicast que usaremos para comunicarnos con los clientes. Esta dirección será el punto de entrada de información del cliente por lo que este deberá saberla de antemano.

```
debian:/home/dvb/multicast/server# ./super_mcast
uso ./super_mcast <video> <mcastGroup>
debian:/home/dvb/multicast/server# ./super_mcast video.ts 224.0.1.2


dvbstream v0.6 - (C) Dave Chapman 2001-2004
Released under the GPL.
Latest version available from http://www.linuxstb.org/
Tuning to 794000 Hz
Using DVB card "Hauppauge Nova-T DVB-T", freq=794000
tuning DVB-T (in United Kingdom) to 794000000 Hz, Bandwidth: 8
Getting frontend status
Event:  Frequency: 794000000
Bit error rate: 0
Signal strength: 10023
SNR: 65535
UNC: 0
FE_STATUS: FE_HAS_SIGNAL FE_HAS_LOCK FE_HAS_CARRIER FE_HAS_VITERBI FE_HAS_SYNC
dvbstream will stop after 10 seconds (0 minutes)
Output to stdout
Streaming 2 streams


libdvbpsi error (PSI decoder): TS discontinuity (received 2, expected 0)
Caught signal 1 - closing cleanly.


mcastClient : sending data on multicast group '224.0.1.2' (224.0.1.2)


mcastServer : listening to mgroup 224.0.23.14:1500
```

Fig. 5.1 Llamada desde consola a la aplicación del servidor

El servidor realizará una captura del flujo, formará los informes y los enviará cada 30 segundos. Finalmente se pondrá a la espera de recibir alguna petición. Por pantalla podremos ver todos estos pasos. Para hacer estas pruebas se ha definido por defecto en el servidor la frecuencia del canal 61 que contiene los canales TV3, K3/33, 3/24 y Canal300 (frecuencia = 794000 Khz).

5.1.1.1. Captura

Su primer paso es hacer una captura del TS completo, la captura se realizará durante 10 segundos, tiempo suficiente para poder analizar la captura.

```
dvbstream v0.6 - (C) Dave Chapman 2001-2004
Released under the GPL.
Latest version available from http://www.linuxstb.org/
Tuning to 794000 Hz
Using DVB card "Hauppauge Nova-T DVB-T", freq=794000
tuning DVB-T (in United Kingdom) to 794000000 Hz, Bandwidth: 8
Getting frontend status
Event:  Frequency: 794000000
Bit error rate: 0
Signal strength: 10023
SNR: 65535
UNC: 0
FE_STATUS: FE_HAS_SIGNAL FE_HAS_LOCK FE_HAS_CARRIER FE_HAS_VITERBI FE_HAS_SYNC
dvbstream will stop after 10 seconds (0 minutes)
Output to stdout
Streaming 2 streams
```

Fig 5.2 Captura del flujo durante 10 segundos

5.1.1.2. Análisis

El segundo paso será llamar a decode_pat para que haga el informe SD&S basándose en la captura anterior, para luego mandarlo, cada treinta segundos (apartado 3.2.1.1), a la dirección multicast indicada como parámetro, como se puede ver en la figura 5.4. En la figura 5.3 se puede ver que la llamada al analizador devuelve un error, error que, como antes se ha comentado, debe ser debido la reordenación de los números de serie de los paquetes TS, posiblemente debidas a la pérdida momentanea del canal de radio. En la figura 5.5 se puede ver la captura con ethereal [28] correspondiente al envío de los informes mediante DVBSTP, donde se pueden apreciar los valores de la cabecera tal como se explicaba en el apartado 3.6.4.

```
libdvbpsi error (PSI decoder): TS discontinuity (received 8, expected 0)
```

Fig 5.3 Llamada a decode_pat

```
mcastClient : sending data on multicast group '224.0.1.2' (224.0.1.2)
```

Fig 5.4 Envío de los informes

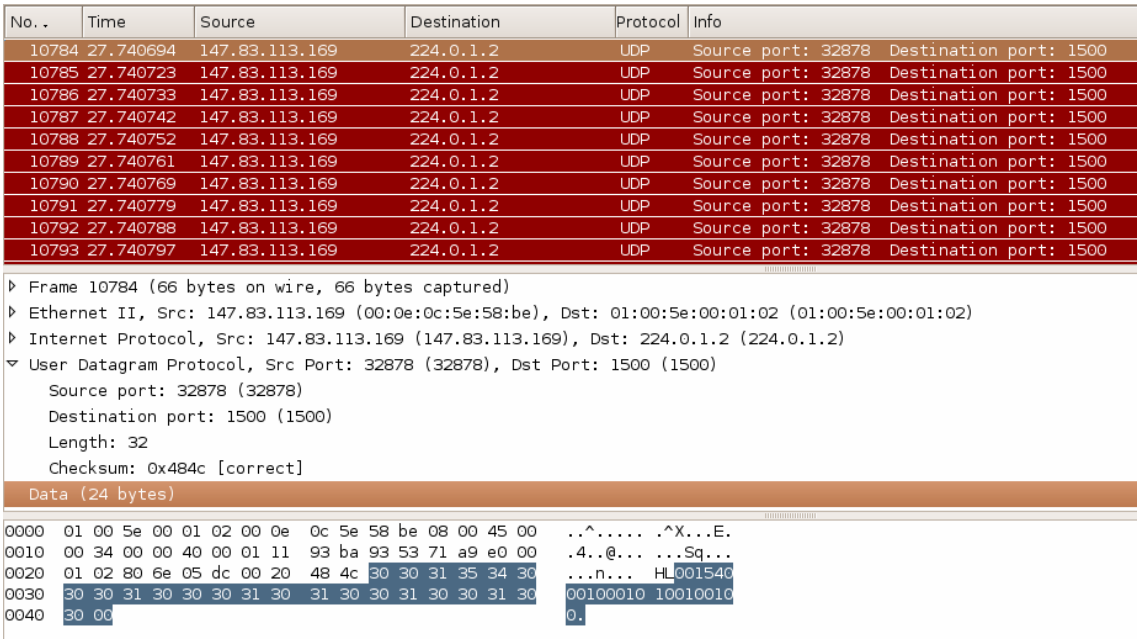


Fig. 5.5. Captura con ethereal del envío de paquetes DVBSTP

En la captura del primer paquete con ethereal se puede apreciar la cabecera del paquete DVBSTP (00154000100010100100100) que corresponde a los siguientes valores, basándonos en la tabla 3.6.

Tabla 5.1. Cabecera DVBSTP

0	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***0***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***0***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***0***	***1***
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Al no haber fragmentación todos los valores están introducidos por defecto excepto el del tamaño del informe.

5.1.1.3. Servicio

Finalmente, el servidor se mantendrá a la escucha en la dirección multicast 224.0.23.14 (ver apartado 3.3.1) y al recibir la petición del cliente pasará a procesarla. Primero deberá volver a analizar el flujo con decode_pmt a fin de obtener los PIDs asociados al programa elegido por el cliente.

Seguidamente enviará el flujo con todos los PIDs asociados a la dirección multicast 224.0.1.4. Como se puede ver en las figuras 5.6 y 5.7

```
mcastServer : listening to mgroup 224.0.23.14:1500
libdvbpsi error (PSI decoder): TS discontinuity (received 8, expected 0)
121
122
123
124
601
701
802
902
702
703
704
dvbstream v0.6 - (C) Dave Chapman 2001-2004
Released under the GPL.
Latest version available from http://www.linuxstb.org/
Tuning to 794000 Hz
Using DVB card "Hauppauge Nova-T DVB-T", freq=794000
tuning DVB-T (in United Kingdom) to 794000000 Hz, Bandwidth: 8
Getting frontend status
Event:  Frequency: 794000000
Bit error rate: 0
Signal strength: 10023
SNR: 65535
UNC: 0
FE_STATUS: FE_HAS_SIGNAL FE_HAS_LOCK FE_HAS_CARRIER FE_HAS_VITERBI FE_HAS_SYNC
dvbstream will stop after -1 seconds (71582788 minutes)
Using 224.0.1.4:5004:2
version=2
Streaming 13 streams
```

Fig 5.6 Procesado de petición

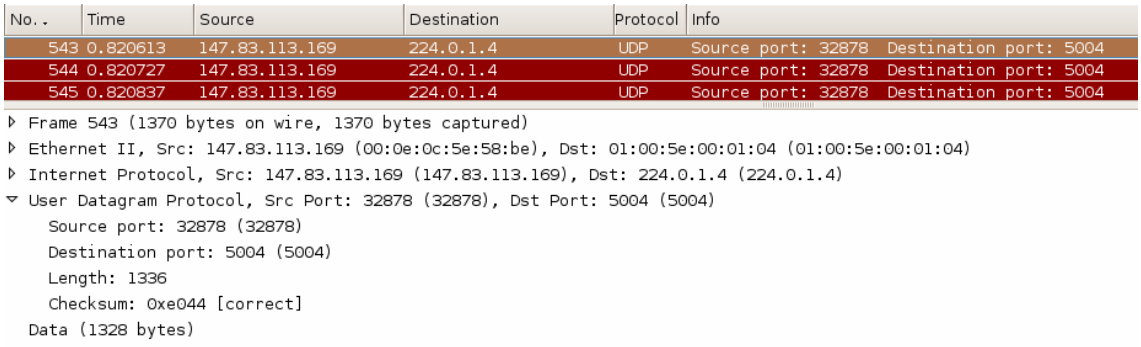


Fig 5.7. Envío del flujo

5.1.2. Cliente

El funcionamiento del cliente consta de dos fases, una primera donde deberá recibir los informes enviados pro el servidor y otra donde realizará una petición.

5.1.2.1. Recepción

Como ya he dicho la primera fase del cliente consiste en la recepción de los informes provenientes del servidor. Antes de llamar a la aplicación deberemos borrar el fichero *informe.xml* en caso de tenerlo, una vez hecho esto, cuando llamemos a la aplicación deberemos pasarle por parámetro la dirección multicast. Una vez recibidos los mostrará por pantalla mediante Firefox.

La aplicación utilizada para la recepción de los informes deberemos cortarla una vez obtenido éste ya que el servidor SD enviará dicho informe cada 30 segundos y con recibirlo una vez tendremos suficiente.

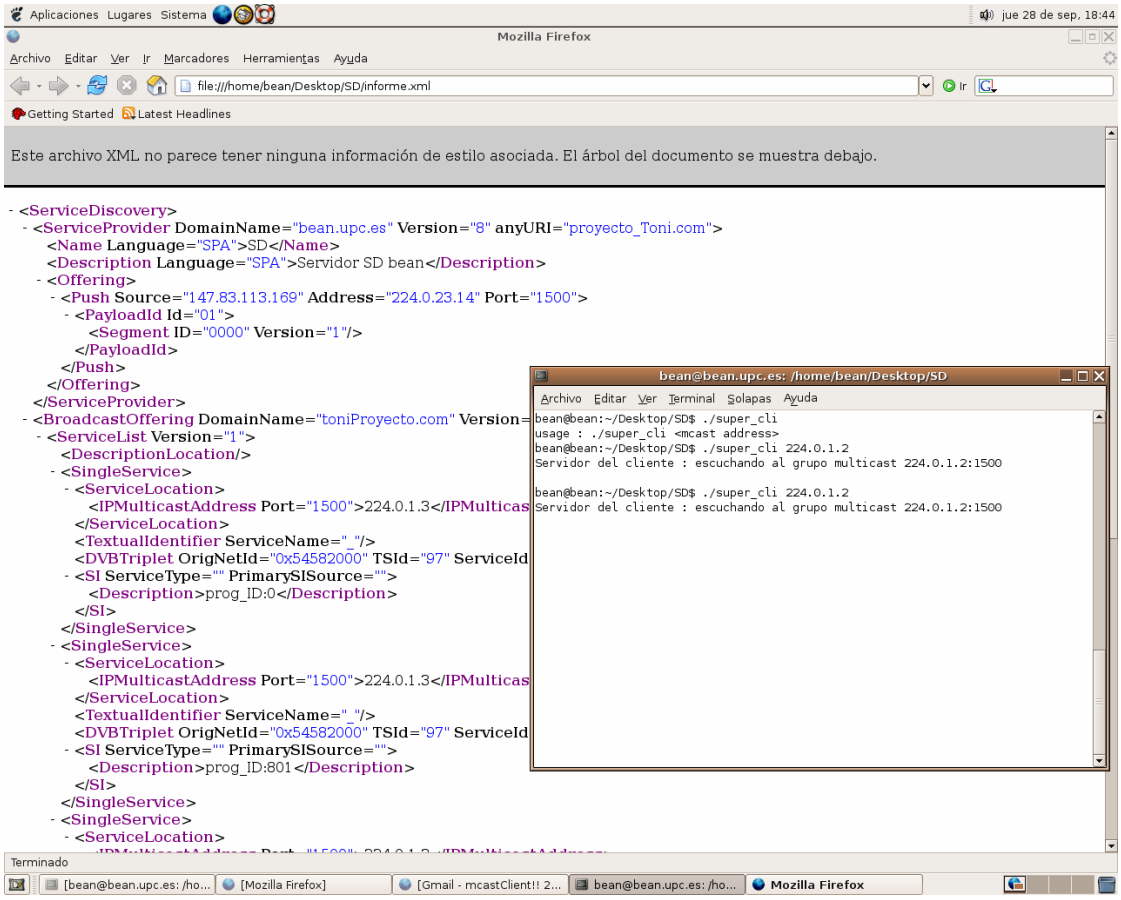


Fig 5.8. Recepción del informe

El informe recibido por el cliente deberá ser válido y *well-formed*, para esto se ha integrado la aplicación *xmlstarlet,* vista en el punto 4.3.2., en el código desarrollado.

5.1.2.2. Petición

Una vez obtenido el informe el cliente podrá realizar una petición concreta, como en la figura 5.9 donde se pueden comprobar los valores enviados (801 y 110). Una vez enviada se abrirá directamente el reproductor de vídeo VideoLanClient que reproducirá el flujo (figura 5.10).

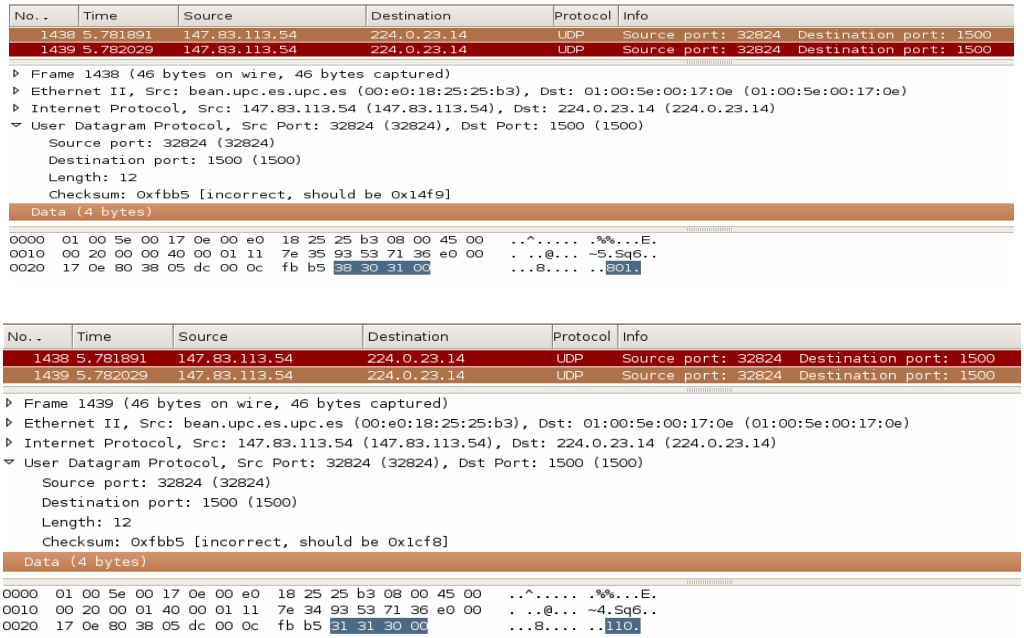


Fig. 5.9 Envió de la petición

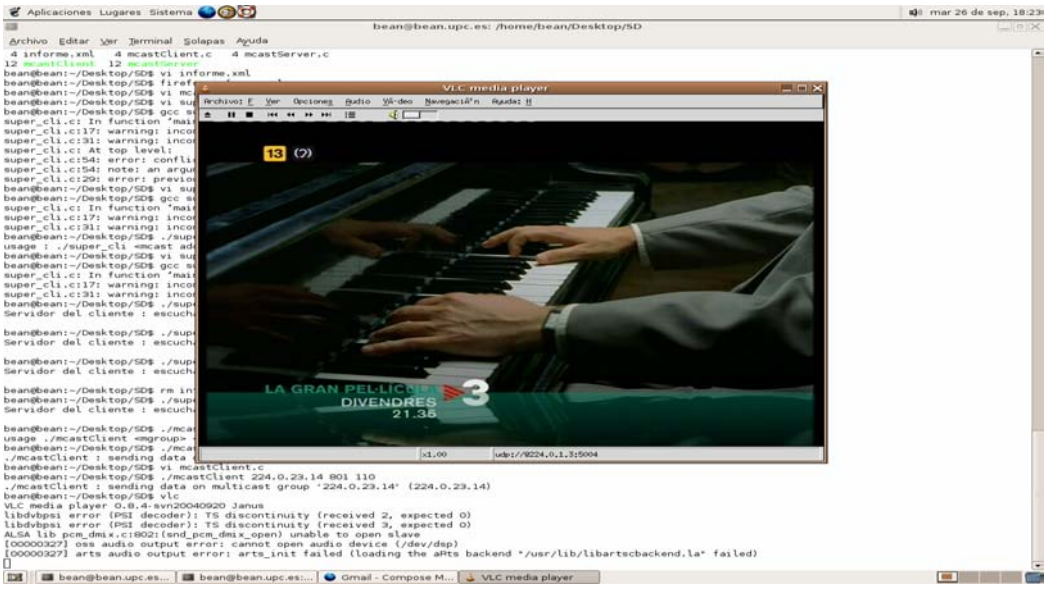


Fig 5.10 Reproducción del vídeo

5.2. Modos FEC

Una vez desarrollado el código del entrelazador se pasa a hacer pruebas para comprobar su correcto funcionamiento.

La primera prueba se realiza en modo 0, que, aunque ya se realizó en el TFC de otro compañero servirá de comparativa con respecto a los modos 3 y 4 desarrollados. En la figura 5.11 se puede ver una captura con ethereal del modo 0.

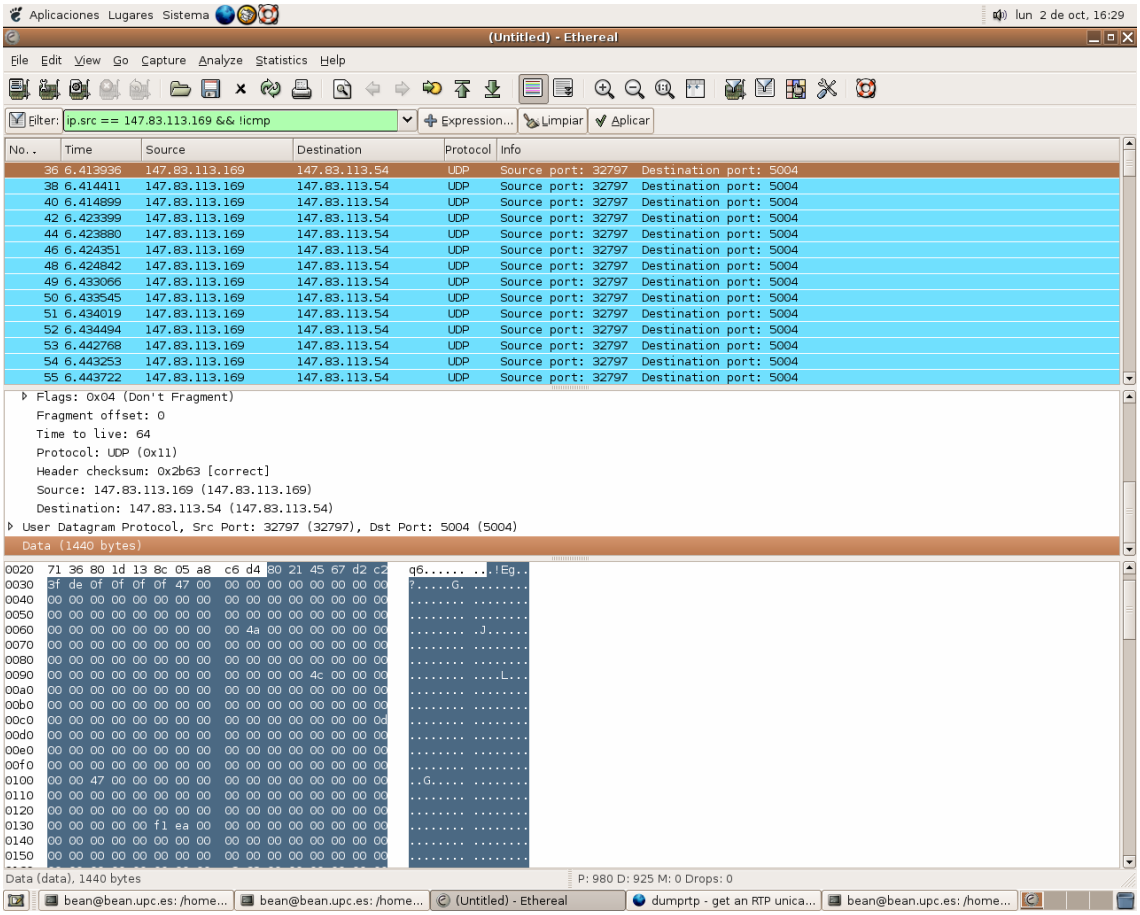


Fig. 5.11 Captura de paquete FEC modo 0 resaltando el campo de datos del primer paquete enviado con ethereal

En la figura 5.11 se puede ver como, al iniciar la transmisión, la matriz de entrelazado se encuentra llena de ceros y se irá llenando al entrar cada uno de los bytes de información a enviar. De este modo cada 51 bytes iremos viendo aparecer los primeros bytes (hay que recordar que en el modo 0 el valor de l es de 51).

La siguiente prueba será probar los modos de entrelazado 3 y 4 que se muestran en las figuras 5.12 y 5.13.

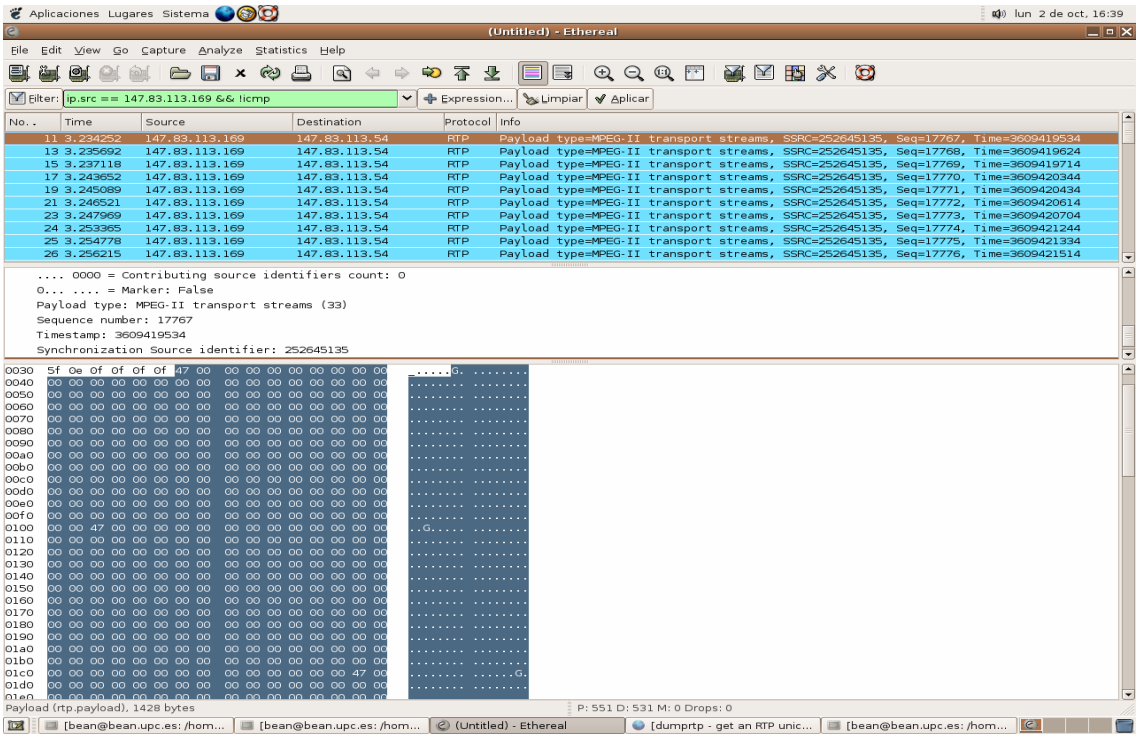


Fig. 5.12 Captura de paquete FEC modo 3 resaltando el campo de datos del primer paquete enviado con ethereal

Como se puede observar en la captura de ethereal de la figura 5.12 en este caso al tener “l” un valor de 204 en el entrelazador convolucional veremos como los valores se irán introduciendo cada 204 bytes. De la misma forma, al tener “M” un valor de 2, tardará 612 bytes en aparecer el primer par de bytes seguidos.

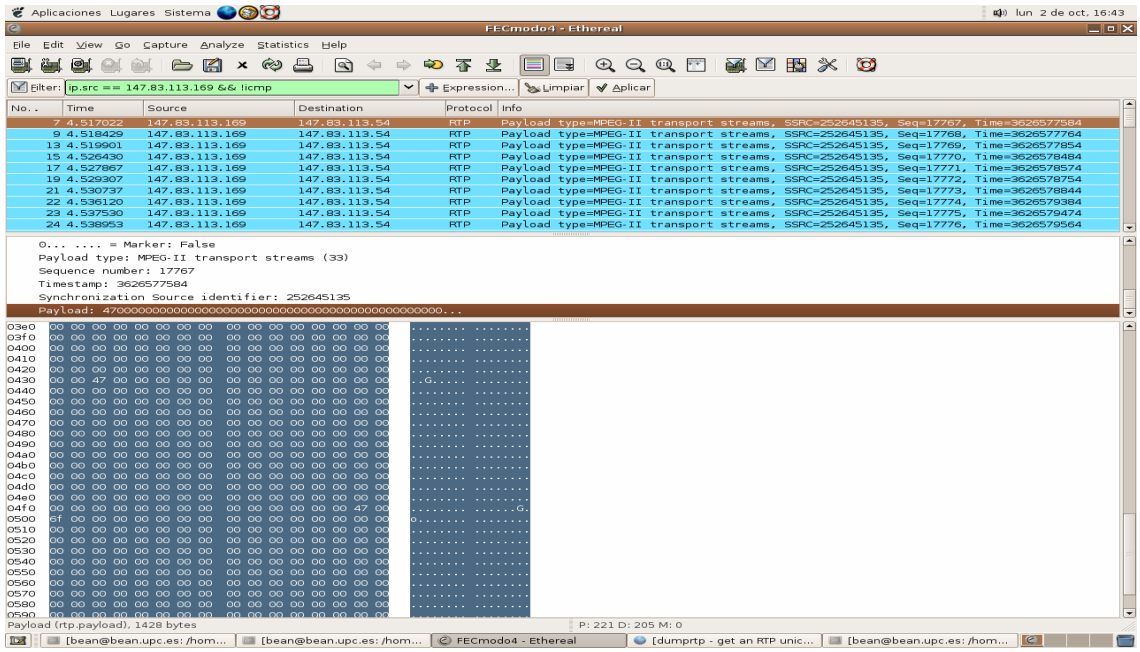


Fig. 5.13 Captura de paquete FEC modo 3 resaltando el campo de datos del primer paquete enviado con ethereal

En este caso, de la misma forma que se ha visto en el modo 3, veremos como irán apareciendo los primeros valores cada 204 bytes, mientras que el primer par de bytes seguidos se dará pasadas las cuatro vueltas, 816 bytes.

En la siguiente figura se puede observar el análisis del flujo RTP con ethereal:

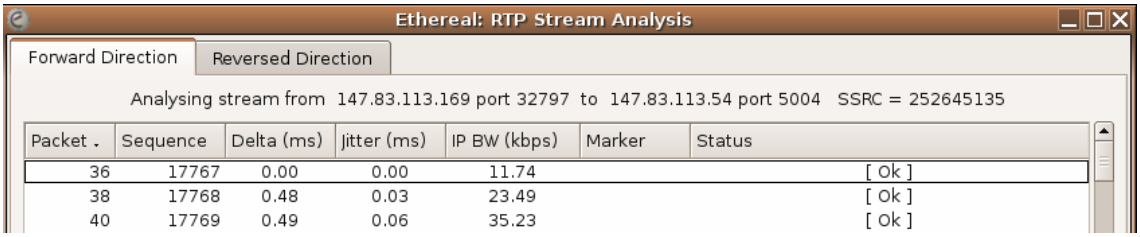


Fig. 5.14 Anàlisis del flujo RTP con FEC con ethereal

El jitter se mantiene bajo debido a que ambas máquinas pertenecen a la misma red. También se observa bastante retardo en la transmisión de los paquetes. Esto se explica porque el procesado que se introduce con las operaciones del entrelazador aumenta la carga del procesador y por tanto el retardo de envío de paquetes.

6. CONCLUSIONES Y LÍNEAS FUTURAS

6.1. Conclusiones

El objetivo principal de este proyecto ha sido el desarrollo de un software Service Discovery, en C y sobre plataforma linux, que cumpliera el estándar de la *ETSI*, *DVB; Transport of MPEG-2 Based DVB Services over IP Based Networks [7].* Hay que destacar que el software desarrollado no cumple todo el estándar, el código desarrollado está basado en el modelo *push* de descubrimiento de servicio. Hay varios aspectos que no se han tocado, como el modelo *pull* o la fragmentación de informes.

Para el desarrollo del mismo primero hubo que hacer un primer estudio sobre el estándar, ver todas las normas dictadas, todas las posibilidades existentes y el mecanismo utilizado. Una vez estudiado se supieron qué tipo de aplicaciones se necesitarían para desarrollar la aplicación, en concreto un servidor de vídeo y un analizador de flujos de vídeo.

De este modo se empezaron a hacer pruebas con dvbstream [19], un servidor de vídeo recomendado por el tutor y por compañeros que anteriormente lo habían utilizado con la misma máquina con fines parecidos, tal como se ha visto en el punto 4.3.2.

Una vez obtenidas varias capturas se pasó a probar la redifusión del flujo sobre una LAN, por lo que hubo que pasar a buscar un reproductor de vídeo libre que pudiera reproducir el flujo entrante de la red y se encontró VideoLan [20] que se adecuaba exactamente al perfil que se buscaba. Con VideoLan se pudo pasar a hacer las pruebas de envío y recepción de flujo de vídeo entre un servidor con dvbstream y un cliente con VLC.

Una vez conseguido el objetivo de difundir flujo de vídeo sobre una LAN, véase Anexo IV, se pasó a hacer un estudio más profundo del estándar. Éste declara que la información ofrecida por el servidor debe enviarse en formato XML y qué datos hay que enviar al cliente en cada uno de los informes. Para obtener dichos datos se tuvo que buscar analizadores que se adecuaran a las necesidades y se encontró la librería libdvbpsi, que precisamente tiene aplicaciones que extraen todos los datos necesarios del flujo de vídeo, librería, por cierto, utilizada en VLC.

Estas aplicaciones devolvían directamente las tablas PAT y PMT, innecesario para cumplir el estándar, por lo que hubo que modificarlas para obtener únicamente lo que pedía el estándar y así obtener directamente los informes SD&S formados. Hay que destacar que los informes SD&S que forman la aplicación no rellenan todos los campos, únicamente los obligatorios, dejando presentes y vacíos los opcionales.

Con esto sólo quedaba encontrar el modo de comunicación entre cliente y servidor. El estándar define que existen dos tipos de entregar los informes, unicast y multicast. Se decidió que el proyecto se enfocase hacia el modo multicast por lo que se tuvo que buscar y modificar un cliente y un servidor multicast UDP, tal como especifica el estándar.

Con esto ya teníamos cada uno de los bloques (Punto 4.1) por separado. Faltaba unirlos, por lo que hubo que crear una aplicación principal que llamara a cada uno de ellos en el momento adecuado (*super_mcast)*. Con esto se obtuvo la aplicación completa. Tras esto se han realizado las pruebas de funcionamiento de la aplicación para asegurar su correcto funcionamiento.

Por un lado se comprobaron todos los pasos del servidor. Primero, debía realizar una captura del flujo a servir; segundo, debía analizar dicha captura y, mediante los analizadores, obtener un informe SD&S en formato XML con los datos obtenidas; tercero, el servidor debía enviar el informe formado vía multicast cada 30 segundos; cuarto, una vez iniciados los envíos periódicos de los informes el servidor se mantendría a la escucha de peticiones de los clientes en modo multicast; finalmente, el servidor debía procesar las peticiones recibidas y reenviar el flujo solicitado vía multicast para que el cliente pudiera reproducirlo. Se comprobó el buen funcionamiento de todos los pasos nombrados, con lo que se pudo pasar a comprobar el correcto funcionamiento de la aplicación cliente.

El cliente debía, primero, activar un servidor multicast para recibir los informes del servidor y, una vez recibidos, mostrarlos al cliente y, segundo, ser capaz de enviar peticiones vía multicast al servidor y reproducir el flujo que a la postre recibiría. Las pruebas realizadas fueron satisfactorias, mostrando los informes con Firefox y el flujo con VLC, con lo que se pudo pasar a realizar las pruebas entre servidor y cliente.

Tal y como se ha podido ver en el apartado 5.1 del presente trabajo, las pruebas servidor-cliente fueron satisfactorias. Además del buen funcionamiento de la comunicación cliente-servidor también se comprobó como la aplicación implementada cumple las normas del estándar; formando informes XML válidos y bien formados y enviándolos sobre DVBSTP, habiendo comprobado la correcta formación de la cabecera.

Capítulo a parte hay que destacar los problemas surgidos con los equipos disponibles en la universidad para el desarrollo del software y más de un traslado de los mismos. En un principio para realizar capturas se debía transportar el equipo hasta un laboratorio con las tomas adecuadas de DVB-T y DVB-S, tomas que en un principio tampoco estaban disponibles. Este traslado debía hacerse teniendo en cuenta los horarios de clase y los de otro compañero que también necesitaba el equipo para su proyecto. Una vez conseguidas las tomas finales en las inmediaciones de i2cat se pasó a su traslado final, no sin antes tener que esperar otro tiempo a que se instalaran las tomas DVB finales. Cuando ya se hizo la conexión final hubo problemas con multicast ya que el equipo no se conectó a la red de la escuela que finalmente se solventaron cambiándolo de red. También destacar que los equipos

disponibles para el desarrollo del proyecto, tanto el anteriormente mencionado como otro disponible para utilizarlo de cliente han sufrido varios percances teniendo que reinstalar el S.O., arreglar el entorno gráfico, etc.

6.2. Repercusión medioambiental

Al tratarse de una aplicación para PC no veo una repercusión medioambiental directa. Quizá el hecho de que la utilización de este software consiga la reproducción de televisión en un PC implique un ahorro de recursos materiales por la posibilidad de utilización de monitores en lugar de televisores convencionales y por la posibilidad de la información digital en lugar de la información escrita.

6.3. Aspectos éticos

No se contempla ninguna implicación ética en el uso del software desarrollado. Únicamente el servicio de flujos de vídeo de contenido pirata puede considerarse éticamente inapropiado. En cuanto al código desarrollado lo aporto como código de libre distribución.

6.4. Aspectos de seguridad

No existen problemas de seguridad respecto a la aplicación desarrollada, siempre y cuando el administrador disponga de un sistema seguridad óptimo en el servidor. Al margen de la seguridad respecto a intrusos, existe la posibilidad de que debido a un mal uso de la aplicación por parte del administrador ésta pueda provocar flooding en la red debido a la gran cantidad de flujo de datos enviado.

6.5. Líneas futuras

El primer objetivo que se debería marcar para proseguir la formación completa de un software de descubrimiento de servicio debería ser implementar la fragmentación de los informes XML (segmentos y secciones). Como ya se ha comentado en el punto 2.2.1 los informes XML, de tener un tamaño demasiado grande, se pueden fragmentar en segmentos y estos, a su vez, en secciones.

Los informes se deberán fragmentar siempre respetando el obtener informes XML *well-formed* y válidos, y especificando en cada segmento o sección su versión e ID, e ir incrementando la versión en caso necesario. Esta

fragmentación se hará en caso de que sea necesario para su transporte por la red (para evitar la fragmentación de los paquetes por parte de los routers), por lo que la aplicación debería consultar al router la MTU de la red o su política de fragmentación para así dividir el informe o no.

Otra posible línea futura sería la de implementar todos los tipos de informes disponibles vistos en el apartado 3.2.: *CoD Discovery Information, Services From other SPs y Package Discovery Inormation*. Tipos de informes que corresponden a distintos escenarios, por lo que se debería modificar el escenario de este trabajo incluyendo más de un servidor, un sistema de CoD, etc.

También sería interesante implementar el modo unicast del estándar [7]. En el modo unicast los informes viajan sobre HTTP, realizando un esquema de petición-respuesta, cambiando totalmente el protocolo de comunicación utilizado en el modo multicast desarrollado en este trabajo.

Otra línea futura a seguir es la de modificar el servidor desarrollado para poder procesar las peticiones de varios clientes y poder enviar flujo a todos ellos, ya que en este proyecto el escenario utilizado contaba con un solo cliente.

Finalmente, también se podría implementar un cliente RTSP basándose en la cláusula 6 del estándar [7]. El desarrollo de un cliente RTSP, aunque no se ha hecho especial mención en este proyecto, es interesante debido a la posibilidad de controlar remotamente un servidor de flujo de vídeo. Este control se realiza mediante los siguientes comandos:

- Describe: con este comando obtendremos la URL RTSP y el tipo de datos obtenidos.
- Setup: con setup indicaremos como debe ser transportado un programa.
- Play: con play visualizaremos el flujo de vídeo.
- Pause: con este comando pararemos la visualización del flujo de vídeo.
- Record: sirve para enviar un flujo al vídeo para grabarlo
- Teardown: sirve para finalizar la sesión

Bibliografía

Programación

[1] *Tutoriales de programación en C*
http://www.ulpgc.es/otros/tutoriales/mtutor/indice.html

[2] *Sockets,*
http://www.elrincondelprogramador.com/default.asp?pag=articulos/leer.asp&id=2

[3] *Sockets mcast,* http://pont.net/socket/

[4] *Desarrollo multicast*
http://www.ixiacom.com/library/white_papers/display?skey=multicast

[5] *Código libre,* http://www.koders.com

[6] *Ejemplos de código,* http://docencia.ac.upc.es/EPSC/SO

DVB / MPEG

[7] *Digital Video Broadcasting (DVB); Transport of MPEG-2 Based DVB Services over IP Based Networks, ETSI TS 102 034 v1.1.1 (2005-03),* http://webapp.etsi.org/action/PU/20050329/ts_102034v010101p.pdf

[8] *Digital Video Broadcasting (DVB); Guidelines on implementation and usage of Service Information (SI), ETSI TS 101 211 v1.5.1 (2003-01),* http://webapp.etsi.org/action/PU/20060221/tr_101211v010701p.pdf

[9] *Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems, ETSI EN 300 468 v1.5.1 (2003-05),* http://webapp.etsi.org/action/OP/OP20060428/en_300468v010701o.pdf

[10] *Flujos de programa y de transporte MPEG-2: Aplicación a DVB, UPM Rev.02 Jun-01,* http://www.etc.upm.es/tsmpeg2d.pdf

[11] *«DVB-IP» Rémi Houdaille, Thomson R&D France Brussels, 13 December 2005,* http://www.magellan-itea.org/docs/workshop/DVB-IP_Thomson.pdf

[12] *ETSI,* http://www.etsi.org

[13] *Canales TDT,* http://www.espacioutil.com/tu_espacioutil1-3.asp

[14] *Canales TDT,* http://nomeriasdeti.no-

ip.com/index.php?top_tab=1&section_type=2&num=&task=show_comments&id_info=63

[15] *Canales TDT,* www.tdt.es

[16] *Canales Satélite,* http://www.satmania.com

[17] *Transmisión de MPEG-2,* http://www.erg.abdn.ac.uk/research/future-net/digital-video/mpeg2-trans.html

[34] *MPEG Systems FAQ,* http://www.chiariglione.org/mpeg/faq/mp2-sys/mp2-sys.htm#mp2-1

Herramientas utilizadas

[18] *Página de las librerías DVBtools,* http://sourceforge.net/projects/dvbtools

[19] *Página de los desarrolladores de DvbStream,* http://www.dchapman.com/dvbstream/

[20] *Página oficial de VideoLan,* http://www.videolan.org

[21] *Página de los desarrolladores de libdvbpsi,* http://developers.videolan.org/libdvbpsi/

[22] *Página de ethereal,* http://www.ethereal.com/

[32] *Página de los desarrolladores de xmlstarlet,* http://xmlstar.sourceforge.net/

RFCs y otros

[23] *RFC 1889 - RTP: A Transport Protocol for Real-Time Applications*

[24] *RFC 2132 - DHCP Options and BOOTP Vendor Extensions*

[25] *RFC 2782 - A DNS RR for specifying the location of services (DNS SRV)*

[26] *Asignación de puertos de IANA,* http://www.iana.org/assignments/port-numbers

[27] *RFC 2250 - RTP Payload Format for MPEG1/MPEG2 Video*

[28] *RFC 3376 - Internet Group Management Protocol, Version 3*

[29] Cerdá Brufau G., *Transmisión de flujos DVB/MPEG sobre redes IP*, 2004

[30] *Página de TSReader,* www.coolstf.com/tsreader/

[31] Hervé Benoit, Televisión Digital, MPEG-1 MPEG-2 y Sistema europeo DVB, Paraninfo, España (1998)

[33] International Organization of Standarization, http://www.iso.org

Glosario

CoD	Content on Demand
CP	Content Provider
CRC	Cyclic Redundancy Check
DHCP	Dinamic Host Configure Protocol
DN	Delivery Network
DNG	Delivery Network Gateway
DVB	Digital Video Broadcasting
DVB-C	Digital Video Broadcasting- Cable
DVB-H	Digital Video Broadcasting- Handheld
DVB-IPI	Digital Video Broadcasting- IP Infraestructure
DVB-S	Digital Video Broadcasting- Satellite
DVB-T	Digital Video Broadcasting- Terrestrial
DVBSTP	Digital Video Broadcasting SD&S Transport Protocol
ETSI	European Telecommunications Estándars Institute
FEC	Forward Error Correction
HN	Home Network
HNCD	Home Network Connecting Device
HNED	Home Network End Device
HNS	Home Network Segment
HTTP	Hyper Text Transfer Protocol
IANA	Internet Assigned Numbers Authority
IEEE	Institute of Electrical and Electronics Engineers
IGMP	Internet Group Management Protocol
IP	Internet Protocol
IPI	Internet Protocol Infraestructure
IPv4	Internet Protocol Version 4
ISO	Internacional Oranization of Standardization
MPEG	Moving Pictures Expert Group
MTU	Maximum Transmission Unit
PAT	Program Association Table
PMT	Program Map Table
RFC	Request For Comments
RTP	Real Time Protocol
SD	Service Discovery
SD&S	Service Discovery and Selection
SP	Service Provider
SRV	Service Location
TS	Transport Stream
UDP	User Datagram Protocol
VLC	VideoLan Client
WWW	World Wide Web
XML	eXtensible Markup Language

ANEXO I. LIBDVBPSI

Libdvbpsi es una librería multiplataforma desarrollada por los creadores de VideoLan que sirve para decodificar y generar tablas de TS y DVB.

A continuación se muestran los pasos para una correcta instalación.

Instalación librería libdvbpsi con apt

El primer paso será buscar con la aplicación aptitude la versión disponible de la librería y su posterior instalación.

Busco e instalo la versión más reciente de la librería

```
$apt-cache search libdvbpsi

$apt-get install libdvbpsi4

debian:~/Desktop/dvbstream# apt-get install libdvbpsi4
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
libdvbpsi4 ya está en su versión más reciente.
0 actualizados, 0 se instalarán, 0 para eliminar y 411 no actualizados.
```

Libdvbpsi4 parece instalada pero no aparecen decode_pat ni decode_pmt en sus directorios, por contra se usará libdvbpsi3-0.1.4, por lo que le hare un PATH para más comodidad.

Una vez instalada la librería pasamos a seguir los pasos indicados en INSTALL.txt

Configuración libdvbpsi

```
./configure --enable-release

libdvbpsi configuration
-----------------------
libdvbpsi version     : 0.1.4
debug                 : false
release               : true
```

Building "libdvbpsi"

```
$make

make[2]: Leaving directory `/home/dvb/libdvbpsi3-0.1.4/misc'
make[2]: Entering directory `/home/dvb/libdvbpsi3-0.1.4'
make[2]: Leaving directory `/home/dvb/libdvbpsi3-0.1.4'
make[1]: Leaving directory `/home/dvb/libdvbpsi3-0.1.4'
```

Instalando y usando "libdvbpsi"

```
$make install

make[1]: Leaving directory `/home/dvb/libdvbpsi3-0.1.4/misc'
make[1]: Entering directory `/home/dvb/libdvbpsi3-0.1.4'
make[2]: Entering directory `/home/dvb/libdvbpsi3-0.1.4'
make[2]: No se hace nada para `install-exec-am'.
make[2]: No se hace nada para `install-data-am'.
make[2]: Leaving directory `/home/dvb/libdvbpsi3-0.1.4'
make[1]: Leaving directory `/home/dvb/libdvbpsi3-0.1.4'
```

Para modificar decode_pat

Practicamente este es el punto más importante. Si decidimos modificar los ejemplos ofrecidos por libdvbpsi deberemos ejecutar la siguiente sentencia para poder disfrutar de dichos cambios. Copiaremos nuestro decode_pat.c en /home/dvb/libdvbpsi3-0.1.4/examples/ y ejecutar

```
$make install
```

Para decode_pmt o decode_sdt realizaremos la misma operación.

ANEXO II. XML-Schema

En este anexo se mostrarán XML-Schema pertenecientes a los dos tipos de informe utilizados en la aplicación, además de los pertenecientes a los tipos más importantes de cada uno de ellos.

Tipos de Elementos

SP Discovery Record

```
<xsd:complexType name="ServiceProvider">
    <xsd:sequence>
        <xsd:element name="ServiceProvider" maxOccurs="unbounded">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="Name" type="dvb:MultilingualType" maxOccurs="unbounded"/>
                    <xsd:element name="Description" type="dvb:MultilingualType" minOccurs="0"
maxOccurs="unbounded"/>
                    <xsd:element name="Offering" type="dvb:OfferingListType" minOccurs="0"/>
                </xsd:sequence>
                <xsd:attribute name="DomainName" type="dvb:DomainType" use="required"/>
                <xsd:attribute name="Version" type="dvb:Version" use="required"/>
                <xsd:attribute name="LogoURI" type="xsd:anyURI" use="optional"/>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
```

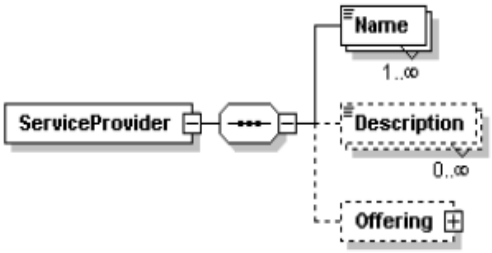


Fig 2.1 XML-Schema del SP Discovery Record, extraída de [7]

DVB-IP Offering Record

```
<xsd:complexType name="BroadcastOffering">
    <xsd:complexContent>
        <xsd:extension base="dvb:OfferingBase">
            <xsd:sequence>
                <xsd:element name="ServiceList" type="dvb:IPServiceList" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

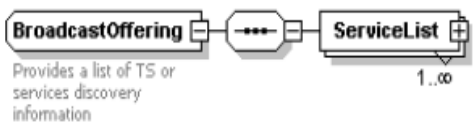


Fig. 2.2 XML-Schema del DVB-IP Offering Record, extraída de [7]

Tipos complejos y grupos de atributos

DVBTriplet

```
<xsd:complexType name="DVBTriplet">
    <xsd:attribute name="OrigNetId" type="dvb:OrigNetId" use="required"/>
    <xsd:attribute name="TSId" type="dvb:TSId" use="required"/>
    <xsd:attribute name="ServiceId" type="dvb:ServiceId" use="required"/>
</xsd:complexType>
```

Fig. 2.3 XML-Schema del tipo complejo DVBTriplet, extraída de [7]

IPServiceList

```
<xsd:complexType name="IPServiceList">
    <xsd:sequence>
        <xsd:element name="ServicesDescriptionLocation" type="dvb:DescriptionLocation" minOccurs="0"
maxOccurs="unbounded"/>
        <xsd:sequence>
            <xsd:element name="SingleService" type="dvb:IPService" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:sequence>
</xsd:complexType>
```

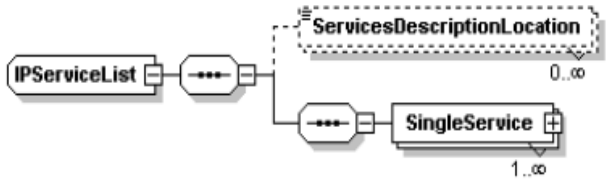


Fig. 2.4 XML-Schema del tipo complejo IPServiceList, extraída de [7]

IPService

```
<xsd:complexType name="IPService">
    <xsd:sequence>
        <xsd:element name="ServiceLocation" type="dvb:ServiceLocation"/>
        <xsd:element name="TextualIdentifier" type="dvb:TextualIdentifier"/>
        <xsd:element name="DVBTriplet" type="dvb:DVBTriplet"/>
        <xsd:element name="MaxBitrate" type="xsd:positiveInteger" minOccurs="0"/>
        <xsd:element name="SI" type="dvb:SI" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
```

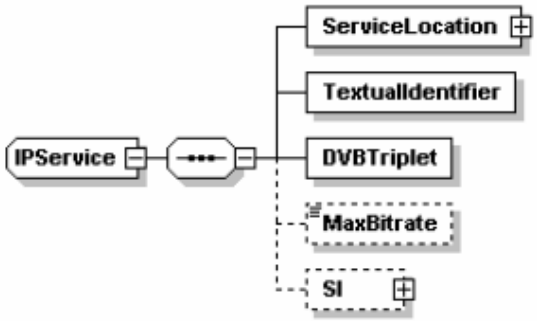


Fig. 2.5 XML-Schema del tipo complejo IPService, extraída de [7]

ANEXO III. Tablas

Tablas MPEG-2

Las tablas PAT y PMT serán necesarias para poder desarrollar el código de Service Discovery ya que poseen la información necesaria para poder informar al usuario acerca de los servicios disponibles.

Tabla PAT

- *Program Association Table (PAT)*:

Esta tabla, de inclusión obligatoria, es transportada por los paquetes con PID=0x0000 y contiene una lista completa de todos los programas disponibles en el *Transport Stream*. Cada programa aparece junto con el valor del PID de los paquetes que a su vez contienen la tabla con los datos que identifican a dicho programa (*Program Map Table, PMT*).

La tabla *PAT* debe transmitirse sin cifrar aunque todos los demás programas lo estén.

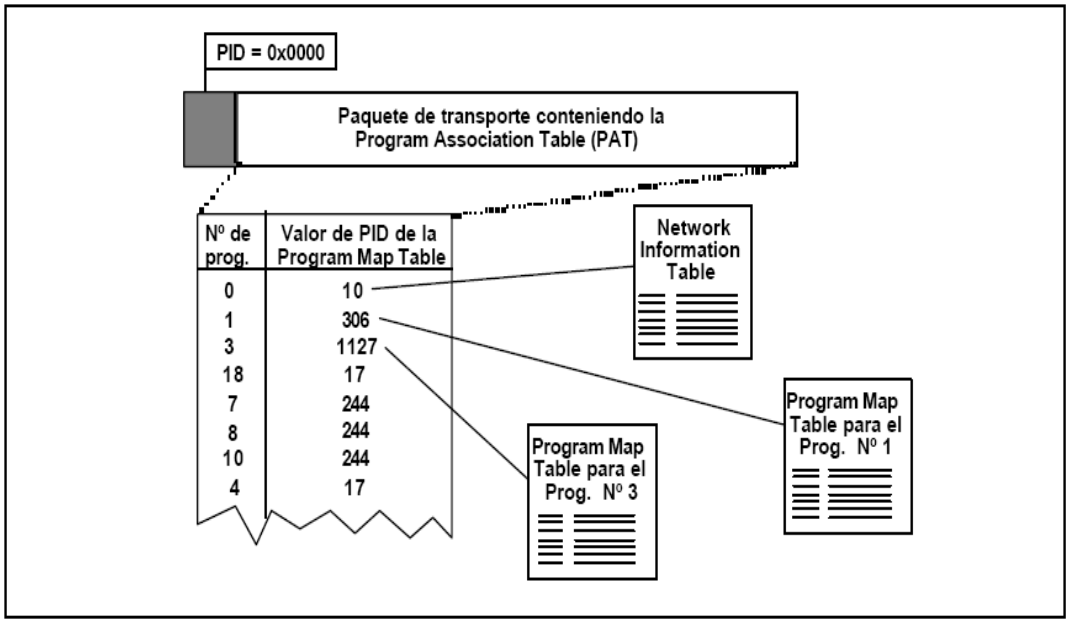


Fig. 3.1 Tabla PAT, extraída de [10]

Como se muestra a modo de ejemplo en la figura 3.1, una sola Program Map Table (la nº 244) puede contener los detalles de varios programas (los numerados como 7, 8 y 10).

Tabla PMT

- *Program Map Table (PMT)*:

Cada programa audiovisual incluido en un *Transport Stream* tiene una tabla *PMT* asociada con él. Dicha tabla proporciona detalles acerca del programa y de los flujos elementales que comprende. Según MPEG-2, las tablas PMT pueden ser transportadas por paquetes con valores de PID arbitrarios, exceptuando los valores 0x0000, reservado para *PAT*, y 0x0001 reservado para *CAT*. Sin embargo, las especificaciones DVB-SI también restringen el uso de los valores de PID que van desde 0x0002 hasta 0x001F.

En la figura 3.2 se muestra un ejemplo de tabla *PMT*. Mediante ella, un decodificador puede determinar que el flujo elemental codificado de vídeo correspondiente al programa nº 3 se encuentra en los paquetes de transporte identificados mediante el PID=726 y que el flujo elemental cuyos paquetes están etiquetados con PID=57 lleva el audio correspondiente en inglés.

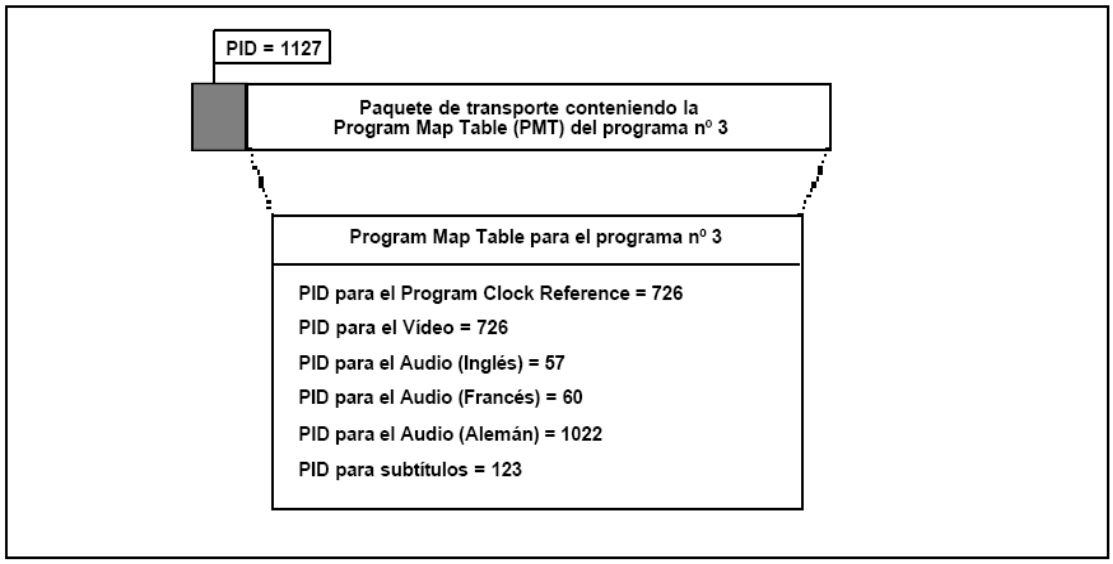


Fig. 3.2 Tabla PMT, extraída de [10]

En las tablas *PMT* los datos de los PID no pueden estar cifrados. Sin embargo, dichas tablas *PMT* también pueden contener información *privada* relativa al programa, que eventualmente sí puede estar cifrada (por ejemplo datos para el control de acceso).

Tabla CAT

- *Conditional Access Table (CAT)*:

Esta tabla debe estar presente si al menos un programa del múltiplex es de acceso
condicional. Se transporta por los paquetes con PID=0x0001, y proporciona detalles de los
sistemas de cifrado empleados, así como los valores de los PID de los paquetes de transporte
que contienen la información del control de acceso condicional.

Los datos para el acceso condicional se envían en forma de *"Entitlement Management Messages (EMM)"*. En estos *"EMM"* se especifican los niveles de autorización o los servicios a que pueden acceder determinados decodificadores, y pueden ir dirigidos a decodificadores individuales o a grupos de ellos. El formato de esta información no está especificado en MPEG-2, puesto que depende del tipo de sistema de cifrado empleado.

Tablas principales de DVB-SI

- *Network Information Table (NIT)*

Esta tabla, si está presente, constituye por definición el Programa nº 0 del múltiplex y está considerada como de datos privados, es decir, definidos por el radiodifusor (proveedor de servicios) y no por MPEG. Se transporta por los paquetes identificados con PID=0x0010. Esta tabla proporciona información acerca de la red física usada para transmitir el “Transport Sream”, como por ejemplo: frecuencias del canal, detalles del transpondedor del satélite, características de modulación, detalles de redes alternativas disponibles, etc.

- *Service Description Table (SDT)*

Contiene datos que describen los servicios en el sistema, como por ejemplo: nombres de los servicios, nombre del proveedor y otros parámetros asociados a cada servicio de un mismo múltiplex. Se transporta por paquetes identificados con PID=0x0011.

- *Event Information Table (EIT)*

Se utiliza para transmitir información relativa a los acontecimientos en curso o futuros en el múltiplex MPEG recibido en la actualidad, y eventualmente sobre otros múltiplex MPEG, tal como: denominación, hora de comienzo, duración, etc.. Se transporta por los paquetes identificados con PID=0x0012

- *Time & Date Table (TDT)*

Esta tabla proporciona información relativa a la hora y fecha del momento, y se utiliza para poner en hora el reloj interno del receptor. La citada información se incluye en una tabla específica debido a las frecuentes actualizaciones de la misma. Se transporta por paquetes identificados mediante PID=0x0014

Tablas opcionales de DVB-SI

- *Bouquet Association Table (BAT)*

El término “bouquet” se usa para referirse a una “Colección de servicios comercializados como entidad única”. Las tablas BAT proporcionan información relativa a los “bouquets”: Además de informar del nombre del bouquet aportan la lista de los servicios disponibles en cada bouquet.
Se transportan por paquetes identificados con PID=0x011.

- *Running Status Table (RST)*

Las tablas RST actualizan de forma rápida la información relativa a la situación de un acontecimiento (que está o no sucediendo). Se transmiten una sola vez y no de forma repetitiva por los paquetes identificados mediante PID=0x0013.

- *Time Offset Table (TOT)*:

Proporciona información relativa a la fecha y hora real así como a la diferencia horaria local (“local time offset”). Se actualiza frecuentemente, siendo transmitida por paquetes identificados con PID=0x0014 (como las TDT).

- *Stuffing Tables (ST)*

Estas tablas de “relleno” se emplean para invalidar tablas que ya no sirven. Por ello usan paquetes que comparten valores de PID con otros tipos de tablas: 0x0010; 0x0011; 0x0012 0x0013 y 0x0014.

ANEXO IV. Código

Para la realización de este proyecto se ha tenido que desarrollar software en lenguaje C, y se ha tenido que modificar aplicaciones existentes que cumplieran las especificaciones de nuestros objetivos.

El desarrollo final se ha traducido en un cliente y un servidor SD&S y en las modificaciones de la librería libdvbpsi y de dvbstream para tener los modos FEC 3 y 4. En cuanto al cliente y servidor SD&S a continuación se pueden ver las versiones multicast mientras que la unicast que se hizo en un principio se incluirá en el CD entregado.

Servidor Service Discovery

super_mcast

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h> /* close */

#define SERVER_PORT 3937
#define MAX_MSG 100

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

  int a, j, k, forkID, final, forkID2, final2;
  FILE *fd;
  char xml[255];
  char header[24];

  if(argc<3) {
    printf("uso %s <video> <mcastGroup>\n",argv[0]);
    exit(1);
  }

  for (j=0; j<4; j++)
  {
      forkID = fork();
      if (forkID ==0){
            switch(j){

                  case 0:
                        if (dvbstreamCase0(argv[1])!=1)
                              printf("Error al llamar a
dvbstreamCase0");
                        exit(0);
                        break;

                  case 1:
                        if (deco(argv[1])!=1)
                              printf("Error al llamar a deco()\n");
```

```
                        exit(0);
                        break;

                  case 2:
                        if (envio(argv[2])!=1)
                              printf("Error al llamar a envio\n");
                        //exit(0);
                        break;

                  case 3:
                        if (dvbstreamCase3(argv[2], argv[1])!=1)
                              printf("Error al llamar a
dvbstreamCase3\n");
                        //exit(0);
                        break;

                  default:
                        printf ("ERROR");
                        //exit(0);
                        break;
            }
      }sleep(15);
  }

  for(j=0; j<5; j++)
      forkID = wait (&final);
}

int dvbstreamCase0 (char video[]){

      FILE *fd;
      int a;

      close(1);
      fd= fopen(video,"w+");
      a = execlp("dvbstream", "dvbstream", "-c", "1", "-f", "794000",
"-s", "27500", "8192","-n", "10", "-o",">", video, NULL);
      fclose(fd);
      return(1);
}

int dvbstreamCase3 (char addr[], char video[]){
      int a;
      a = execlp("./mcastServer", "mcastServer", addr, video, NULL);
      return(1);
}

int deco (char video[]){

      FILE *fd;
      int a, j;

      //sleep(5);
      close(1);
      fd= fopen("video.pat", "w+");
      a = execlp("/home/dvb/libdvbpsi3-0.1.4/examples/decode_pat",
"decode_pat", video, NULL);
      fclose(fd);
      return(1);
}
```

```
int envio (char ip[]){

      int b;

      b= execlp ("./mcastClient", "mcastClient", ip, NULL);
      return (1);
}

int sdt (char video[]){

      int c;
      FILE *fd;

      close(1);
      fd= fopen("/home/dvb/libdvbpsi3-0.1.4/examples/video.sdt",
"w+");
      c = execlp("/home/dvb/libdvbpsi3-0.1.4/examples/decode_sdt",
"decode_sdt", video, NULL);
      fclose(fd);
      return(1);
}
```

mcastClient

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h> /* close */

#define SERVER_PORT 1500
#define MAX_MSG 100

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

  int sd, rc, i;
  unsigned char ttl = 1;
  struct sockaddr_in cliAddr, servAddr;
  struct hostent *h;
  char xml[1024];
  FILE *fd, *fd2;

  if(argc<2) {
    printf("usage %s <mgroup> \n",argv[0]);
    exit(1);
  }

  h = gethostbyname(argv[1]);
  if(h==NULL) {
    printf("%s : unknown host '%s'\n",argv[0],argv[2]);
    exit(1);
  }

  servAddr.sin_family = h->h_addrtype;
  memcpy((char *) &servAddr.sin_addr.s_addr, h->h_addr_list[0],h-
>h_length);
  servAddr.sin_port = htons(SERVER_PORT);
```

```
  /* check if dest address is multicast */
  if(!IN_MULTICAST(ntohl(servAddr.sin_addr.s_addr))) {
    printf("%s : given address '%s' is not multicast \n",argv[0],
         inet_ntoa(servAddr.sin_addr));
    exit(1);
  }

  /* create socket */
  sd = socket(AF_INET,SOCK_DGRAM,0);
  if (sd<0) {
    printf("%s : cannot open socket\n",argv[0]);
    exit(1);
  }

  /* bind any port number */
  cliAddr.sin_family = AF_INET;
  cliAddr.sin_addr.s_addr = htonl(INADDR_ANY);
  cliAddr.sin_port = htons(0);
  if(bind(sd,(struct sockaddr *) &cliAddr,sizeof(cliAddr))<0) {
    perror("bind");
    exit(1);
  }

  if(setsockopt(sd,IPPROTO_IP,IP_MULTICAST_TTL, &ttl,sizeof(ttl))<0) {
    printf("%s : cannot set ttl = %d \n",argv[0],ttl);
    exit(1);
  }

  printf("%s : sending data on multicast group '%s' (%s)\n",argv[0],
       h->h_name,inet_ntoa(*(struct in_addr *) h->h_addr_list[0]));

  /* send data */
  header();
  sleep(2);
  fd= fopen("video.pat", "r+");
  fd2= fopen("header", "r+");


    while(1){
          rewind(fd);
          rewind(fd2);

          while ((fscanf(fd2, "%s", xml))!=EOF)
            sendto(sd,xml,strlen(xml)+1,0,(struct sockaddr *)
&servAddr, sizeof(servAddr));

          while((fscanf(fd,"%s", xml))!=EOF){
            sendto(sd,xml,strlen(xml)+1,0,(struct sockaddr *)
&servAddr, sizeof(servAddr));
            sleep(1);
          }
          sleep(30);//Maximum Time Cycle
    }
    return (1);
}

int header(){

      FILE *fd, *fd2;
      unsigned long int count =0;
      char xml[1024];
```

```
      fd2= fopen("video.pat", "r+");
      while ((fscanf(fd2, "%s", xml))!=EOF)
            count+=sizeof(xml);
      fclose(fd2);

      fd = fopen("header", "w+");
      fprintf( fd, "00");
      fprintf (fd, "%X", count);
      fprintf (fd, "0100010100100100");

      fclose(fd);
      return (1);
}
```

mcastServer

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h> /* close */

#define SERVER_PORT 1500
#define MAX_MSG 100

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

  int sd, rc, n, cliLen;

  struct ip_mreq mreq;
  struct sockaddr_in cliAddr, servAddr;
  struct in_addr mcastAddr;
  struct hostent *h;
  char msg[MAX_MSG];
  char hola[50];

  char PID[50], PMTPID[50], progPID[50], PIDs[50];
  FILE *fdPMT, *paso;
  int j, forkID, final, a, k;
  int i=0;
  char params[20][50];

  if(argc!=3) {
    printf("usage : %s <mcast address> <video>\n",argv[0]);
    exit(0);
  }

  /* get mcast address to listen to */
  h=gethostbyname(argv[1]);
  if(h==NULL) {
    printf("%s : unknown group '%s'\n",argv[0],argv[1]);
    exit(1);
  }

  memcpy(&mcastAddr, h->h_addr_list[0],h->h_length);

  /* check given address is multicast */
```

```
  if(!IN_MULTICAST(ntohl(mcastAddr.s_addr))) {
    printf("%s : given address '%s' is not multicast\n",argv[0],
         inet_ntoa(mcastAddr));
    exit(1);
  }

  /* create socket */
  sd = socket(AF_INET,SOCK_DGRAM,0);
  if(sd<0) {
    printf("%s : cannot create socket\n",argv[0]);
    exit(1);
  }

  /* bind port */
  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("%s : cannot bind port %d \n",argv[0],SERVER_PORT);
    exit(1);
  }

  /* join multicast group */
  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("%s : cannot join multicast group '%s'",argv[0],
         inet_ntoa(mcastAddr));
    exit(1);
  }
  else {
    printf("%s : listening to mgroup %s:%d\n",
         argv[0],inet_ntoa(mcastAddr), SERVER_PORT);

    /* infinite server loop */
    while(1) {

      cliLen=sizeof(cliAddr);
      n = recvfrom(sd,PIDs,MAX_MSG,0,(struct sockaddr *)
&cliAddr,&cliLen);
      if(n<n) {
      printf("%s : cannot receive data\n",argv[0]);
      continue;
      }

      if (i==0)
            strcpy(progPID, PIDs);
      else{
            i=0;
            strcpy (PMTPID, PIDs);
            for (j=0; j<2; j++){

                  forkID=fork();
                  if (forkID==0){
                        switch(j){

                              case 0:
                                    decoPMT(argv[2], progPID, PMTPID);
```

```
                                    sleep(5);
                                    break;

                              case 1:
                                    fdPMT = fopen ("video.pmt", "r");

                                    k=0;

                                    while
((fscanf(fdPMT,"%s",PID))!=EOF){
                                          strcpy(params[k],PID);
                                          printf("%s\n", params[k]);
                                          k++;
                                    }
                                    a = execlp("dvbstream",
"dvbstream", "-c", "1", "-f", "794000", "-s", "27500", params[0],
params[1], params[2], params[3], params[4], params[5], params[6],
params[7], params[8], params[9], params[10], params[11], params[12],
params[13], params[14], params[15], params[16], params[17],
params[18], params[19], PMTPID, "0", "-i",
inet_ntoa(cliAddr.sin_addr), NULL);

                                         break;

                              default:
                                   printf("ERROR");
                                   break;
                        }

                  }sleep(1);
            }
            for (j=0; j<2; j++)
                  forkID = wait (&final);
      }i++;
    }
    exit(0);
  }
}

int decoPMT (char video[], int valor, char pmt[]){

              FILE *fd;
            int a;
              close(1);
            fd= fopen("video.pmt", "w+");
            a = execlp("/home/dvb/libdvbpsi3-
0.1.4/examples/decode_pmt", "decode_pmt", video, valor, pmt, NULL);
            fclose(fd);
            return(1);
}
```

Cliente Service Discovery

super_cli

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h> /* close */

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


 int i, forkID, final;


 if(argc!=2) {
 printf("usage : %s <mcast address>\n",argv[0]);
 exit(0);
 }


 for (i=0; i<2; i++)
 {
 forkID = fork();
 if (forkID==0)
 switch(i){

                        case 0:

                              if (CliServ(argv[1])!=1)
                                    printf    ("Error    al    llamar    a
CliServ()");
                              exit(0);
                              break;

                        case 1:

                              if (presentacion()!=1)
                                    printf    ("Error    al    llamar    a
presentacion()");
                              exit (0);
                              break;

                        default:

                              printf ("ERROR");
                              exit(0);
                              break;
                  }sleep(30);
      }

      for (i=0; i<2; i++)
            forkID = wait (&final);

}
```

```
int CliServ (char ip[]){

      int a;

      a = execlp ("./mcastServer", "mcastServer", ip, NULL);
      return(1);
}

int presentacion (){

      int a;

      a = execlp ("firefox", "firefox", "informe.xml", NULL);
      return (1);
}
```

mcastClient

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h> /* close */

#define SERVER_PORT 1500
#define MAX_MSG 100

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

  int sd, rc, i;
  unsigned char ttl = 1;
  struct sockaddr_in cliAddr, servAddr;
  struct hostent *h;

  int a;

  if(argc!=4) {
    printf("usage %s <mgroup> <progPID> <PMTPID>\n",argv[0]);
    exit(1);
  }

  h = gethostbyname(argv[1]);
  if(h==NULL) {
    printf("%s : unknown host '%s'\n",argv[0],argv[2]);
    exit(1);
  }

  servAddr.sin_family = h->h_addrtype;
  memcpy((char   *)   &servAddr.sin_addr.s_addr,   h->h_addr_list[0],h-
>h_length);
  servAddr.sin_port = htons(SERVER_PORT);

  /* check if dest address is multicast */
  if(!IN_MULTICAST(ntohl(servAddr.sin_addr.s_addr))) {
    printf("%s : given address '%s' is not multicast \n",argv[0],
         inet_ntoa(servAddr.sin_addr));
    exit(1);
  }
```

```
  /* create socket */
  sd = socket(AF_INET,SOCK_DGRAM,0);
  if (sd<0) {
    printf("%s : cannot open socket\n",argv[0]);
    exit(1);
  }

  /* bind any port number */
  cliAddr.sin_family = AF_INET;
  cliAddr.sin_addr.s_addr = htonl(INADDR_ANY);
  cliAddr.sin_port = htons(0);
  if(bind(sd,(struct sockaddr *) &cliAddr,sizeof(cliAddr))<0) {
    perror("bind");
    exit(1);
  }

  if(setsockopt(sd,IPPROTO_IP,IP_MULTICAST_TTL, &ttl,sizeof(ttl))<0) {
    printf("%s : cannot set ttl = %d \n",argv[0],ttl);
    exit(1);
  }

  printf("%s : sending data on multicast group '%s' (%s)\n",argv[0],
       h->h_name,inet_ntoa(*(struct in_addr *) h->h_addr_list[0]));


  /* send data */
  for(i=2;i<argc;i++) {
    rc = sendto(sd,argv[i],strlen(argv[i])+1,0,
            (struct sockaddr *) &servAddr, sizeof(servAddr));

    if (rc<0) {
      printf("%s : cannot send data %d\n",argv[0],i-1);
      close(sd);
      //exit(1);
    }

  }/* end for */

  a = execlp("vlc", "vlc", "udp://@224.0.1.4:5004", NULL);
  /* close socket and exit */
  close(sd);
}
```

mcastServer

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h> /* close */

#define SERVER_PORT 1500
#define MAX_MSG 100

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

  int sd, rc, n, cliLen;
  struct ip_mreq mreq;
```

```
  struct sockaddr_in cliAddr, servAddr;
  struct in_addr mcastAddr;
  struct hostent *h;

  char buf[1024 + 1];
  int a, i=0;

  if(argc!=2) {
    printf("usage : %s <mcast address>\n",argv[0]);
    exit(0);
  }

  /* get mcast address to listen to */
  h=gethostbyname(argv[1]);
  if(h==NULL) {
    printf("%s : unknown group '%s'\n",argv[0],argv[1]);
    exit(1);
  }

  memcpy(&mcastAddr, h->h_addr_list[0],h->h_length);

  /* check given address is multicast */
  if(!IN_MULTICAST(ntohl(mcastAddr.s_addr))) {
    printf("%s : given address '%s' is not multicast\n",argv[0],
         inet_ntoa(mcastAddr));
    exit(1);
  }

  /* create socket */
  sd = socket(AF_INET,SOCK_DGRAM,0);
  if(sd<0) {
    printf("%s : cannot create socket\n",argv[0]);
    exit(1);
  }

  /* bind port */
  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("%s : cannot bind port %d \n",argv[0],SERVER_PORT);
    exit(1);
  }

  /* join multicast group */
  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("%s : cannot join multicast group '%s'",argv[0],
         inet_ntoa(mcastAddr));
    exit(1);
  }
  else {
    printf("Servidor del cliente : escuchando al grupo multicast
%s:%d\n", inet_ntoa(mcastAddr), SERVER_PORT);

    cliLen = sizeof(cliAddr);
```

```
    /* infinite server loop */
    while(1) {

      n = recvfrom(sd,buf,1024,0,(struct sockaddr *)
&cliAddr,&cliLen);
      if (i > 0)
            informe(buf);
      i++;
    }/* end of infinite server loop */

//    a= execlp("firefox", "firefox", "informe.xml", NULL);
  }
  exit(0);
}

int informe (char buf[]){

      FILE *fd;

      fd = fopen("informe.xml", "a");
      fprintf(fd, "%s ", buf);
      fclose(fd);
}
```

Libdvbpsi

decode_pat

```
#include "config.h"

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

#if defined(HAVE_INTTYPES_H)
#include <inttypes.h>
#elif defined(HAVE_STDINT_H)
#include <stdint.h>
#endif

/* The libdvbpsi distribution defines DVBPSI_DIST */
#ifdef DVBPSI_DIST
#include "../src/dvbpsi.h"
#include "../src/psi.h"
#include "../src/tables/pat.h"
#else
#include <dvbpsi/dvbpsi.h>
#include <dvbpsi/psi.h>
#include <dvbpsi/pat.h>
#endif


/*****************************************************************
********
 * ReadPacket
```

```
*****************************************************************
*******/
int ReadPacket(int i_fd, uint8_t* p_dst)
{
  int i = 187;
  int i_rc = 1;

  p_dst[0] = 0;

  while((p_dst[0] != 0x47) && (i_rc > 0))
  {
    i_rc = read(i_fd, p_dst, 1);
  }

  while((i != 0) && (i_rc > 0))
  {
    i_rc = read(i_fd, p_dst + 188 - i, i);
    if(i_rc >= 0)
      i -= i_rc;
  }

  return (i == 0) ? 1 : 0;
}


/*****************************************************************
********
 * DumpPAT

*****************************************************************
*******/
void DumpPAT(void* p_zero, dvbpsi_pat_t* p_pat)
{

//    int fd;
//    uint8_t Txt[188];
//    int bo;

      //bo = ReadPacket(fd, Txt);


  dvbpsi_pat_program_t* p_program = p_pat->p_first_program;
  //printf(  "New PAT\n");

  //printf(  "transport_stream_id \n%d\n", p_pat->i_ts_id);
  printSP(p_pat->i_version);

 // fd = open("video.sdt", O_RDONLY);

  printBroadcastOffering();
  while(p_program)
  {

       /* ReadPacket(fd,Txt);
        printf ("TXT:%s\n", Txt);*/
      printSingleService("_", p_program->i_pid, p_pat->i_ts_id,
p_program->i_number);
      //printDVBTriplet(int TS_ID, int PMT_ID)
      //printf("program_number\n%d\nPID\n%d\n", p_program->i_number,
p_program->i_pid);
```

```
    p_program = p_program->p_next;
  }
  //fclose(fd);
  printCerrar();
  //printf(  "  active              : %d\n", p_pat->b_current_next);
  dvbpsi_DeletePAT(p_pat);
  //sleep(1);
}


/*****************************************************************************
********
 * main

*****************************************************************************
*******/
int main(int i_argc, char* pa_argv[])
{
  int i_fd;
  uint8_t data[188];
  dvbpsi_handle h_dvbpsi;
  int b_ok;


  if(i_argc != 2){
      printf("Falta el fichero de video!\n\n");
        return 1;
        }

  i_fd = open(pa_argv[1], 0); //aquí le pasas el argumento a dumpPAT

  h_dvbpsi = dvbpsi_AttachPAT(DumpPAT, NULL);

  b_ok = ReadPacket(i_fd, data);

  while(b_ok)
  {
    uint16_t i_pid = ((uint16_t)(data[1] & 0x1f) << 8) + data[2];
    /*  printf("%i\t",i_pid); */

    if(i_pid == 0x0){
      dvbpsi_PushPacket(h_dvbpsi, data);
    }

    b_ok = ReadPacket(i_fd, data);
  }

  dvbpsi_DetachPAT(h_dvbpsi);

  return 0;
}


int printSP(int version){

      printf("<ServiceDiscovery> <ServiceProvider
DomainName=\"bean.upc.es\" Version=\"%d\"
anyURI=\"proyecto_Toni.com\">\n", version);

      printName();
      printDescription();
```

```
      printOffering();

      printf("</ServiceProvider>\n");
      //printf("</ServiceDiscovery>\n\n");
      return(1);
}

int printName(){

      printf("<Name Language=\"SPA\">SD</Name>\n");
      return(1);
}

int printDescription(){

      printf("<Description Language=\"SPA\">Servidor SD
bean</Description>\n");
      return(1);
}

int printOffering(){

      printf("<Offering><Push Source=\"84.88.32.52\"
Address=\"224.0.1.2\" Port=\"1600\">\n");

      printPayloadId();

      printf("</Push>\n");
      printf("</Offering>\n");
      return(1);
}

int printPayloadId(){

      printf("<PayloadId Id=\"01\"><Segment ID=\"0000\"
Version=\"1\"/></PayloadId>\n");
      return(1);
}

int printBroadcastOffering(){

      printf("<BroadcastOffering DomainName=\"toniProyecto.com\"
Version=\"1\">\n");

      printServiceList();
      return (1);
}

int printServiceList(){

      printf("<ServiceList Version=\"1\">\n");
      printf("<DescriptionLocation></DescriptionLocation>");
      //printSingleService();
      return (1);
}

int printSingleService(char Txt[], int PMT_ID, int TS_ID, int
prog_ID){

      printf("<SingleService>");
```

```
      printServiceLocation();
      printTextualIdentifier(Txt);
      printDVBTriplet(TS_ID, PMT_ID);
      //opcional:
      //printf("<MaxBitrate></MaxBitrate>\n");
      printSI(prog_ID);

      printf("</SingleService>\n");
      return (1);
}

int printServiceLocation(){

      printf("<ServiceLocation>");

      printIPMulticastAddress();

      printf("</ServiceLocation>\n");
      return (1);
}

int printIPMulticastAddress(){

      //El port no esta definido en el xml-schema del estándar Anexo
C.3.14 pero en la tabla del mismo estándar, Table 4. "TS FUll SI"
Discovery Information marca que es un campo obligatorio por lo que lo
incluiré como atributo del elemento IPMulticastAddress
      printf("<IPMulticastAddress
Port=\"1600\">224.0.1.2</IPMulticastAddress>");

      //o bien
      //printf("<RTSPURL></RTSPURL\n");

      return(1);
}

int printTextualIdentifier(char name[]){

      printf("<TextualIdentifier
ServiceName=\"%s\"></TextualIdentifier>\n",  name);

      return(1);
}

int printDVBTriplet(int TS_ID, int PMT_ID){

      printf("<DVBTriplet OrigNetId=\"0x54582000\" TSId=\"%d\"
ServiceId=\"%d\"></DVBTriplet>\n", TS_ID, PMT_ID);
      //OrigNetID= "0x1234" para 0.0.18.52, para 84.88.32.0 será
0x54582000
      return(1);
}

int printCerrar(){



      printf("</ServiceList>\n");
      printf("</BroadcastOffering>\n");
      printf("</ServiceDiscovery>\n\n");
      return(1);
```

```
}

int printSI(int prog_ID){

      printf("<SI ServiceType=\"\" PrimarySISource=\"\">");
      printSIDescription(prog_ID);
      //printServiceDescriptionLocation();
      //printContentGenre();
      //printCountryAvailability();
      //printReplacementService();
      //printMosaicDescription();
      //printAnnouncementSupport();

      printf("</SI>");
      return(1);
}

int printServiceDescriptionLocation(){

      printf
("<ServiceDescriptionLocation></ServiceDescriptionLocation>");

 return(1);
}

int printSIDescription(int prog_ID){

      printf("<Description>prog_ID:%d</Description>", prog_ID);
      return (1);
}

int printContentGenre(){

        printf ("<ContentGenre></ContentGenre>");
 return(1);
}

int printCountryAvailability(){

              printf ("<CountryAvailability></CountryAvailability>");

 return(1);
}

int printReplacementService(){

              printf ("<ReplacementService></ReplacementService>");
 return(1);
}

int printMosaicDescription(){

              printf ("<MosaicDescription></MosaicDescription>");
 return(1);
}

int printAnnouncementSupport(){

              printf ("<AnnouncementSupport></AnnouncementSupport>");
            return(1);
}
```

decode_pmt

```
#include "config.h"

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

#if defined(HAVE_INTTYPES_H)
#include <inttypes.h>
#elif defined(HAVE_STDINT_H)
#include <stdint.h>
#endif

/* The libdvbpsi distribution defines DVBPSI_DIST */
#ifdef DVBPSI_DIST
#include "../src/dvbpsi.h"
#include "../src/psi.h"
#include "../src/descriptor.h"
#include "../src/tables/pmt.h"
#else
#include <dvbpsi/dvbpsi.h>
#include <dvbpsi/psi.h>
#include <dvbpsi/descriptor.h>
#include <dvbpsi/pmt.h>
#endif


/*****************************************************************************
********
 * ReadPacket

*****************************************************************************
*******/
int ReadPacket(int i_fd, uint8_t* p_dst)
{
  int i = 187;
  int i_rc = 1;

  p_dst[0] = 0;

  while((p_dst[0] != 0x47) && (i_rc > 0))
  {
    i_rc = read(i_fd, p_dst, 1);
  }

  while((i != 0) && (i_rc > 0))
  {
    i_rc = read(i_fd, p_dst + 188 - i, i);
    if(i_rc >= 0)
      i -= i_rc;
  }

  return (i == 0) ? 1 : 0;
}


/*****************************************************************************
********
 * DumpDescriptors
```

```
*****************************************************************
*******/
void DumpDescriptors(const char* str, dvbpsi_descriptor_t*
p_descriptor)
{
  while(p_descriptor)
  {
    int i;
    printf("%s 0x%02x : \"", str, p_descriptor->i_tag);
    for(i = 0; i < p_descriptor->i_length; i++)
      printf("%c", p_descriptor->p_data[i]);
    printf("\"\n");
    p_descriptor = p_descriptor->p_next;
  }
};


/*****************************************************************
********
 * DumpPMT

*****************************************************************
*******/
void DumpPMT(void* p_zero, dvbpsi_pmt_t* p_pmt)
{
  dvbpsi_pmt_es_t* p_es = p_pmt->p_first_es;

 // printf ("%d", p_pmt->i_pcr_pid);


 /* printf(  "\n");
  printf(  "New active PMT\n");
  printf(  "  program_number : %d\n",
         p_pmt->i_program_number);
  printf(  "  version_number : %d\n",
         p_pmt->i_version);
  printf(  "  PCR_PID        : 0x%x (%d)\n",
         p_pmt->i_pcr_pid, p_pmt->i_pcr_pid);
  DumpDescriptors("    ]", p_pmt->p_first_descriptor);
  printf(  "    | type @ elementary_PID\n");*/
  while(p_es)
  {
        printf("%d\n", p_es->i_pid);
        /*
    printf("    | 0x%02x @ 0x%x (%d)\n",
           p_es->i_type, p_es->i_pid, p_es->i_pid);
    DumpDescriptors("    |  ]", p_es->p_first_descriptor);*/
    p_es = p_es->p_next;
  }
  dvbpsi_DeletePMT(p_pmt);
}


/*****************************************************************
********
 * main

*****************************************************************
*******/
int main(int i_argc, char* pa_argv[])
```

```
{
  int i_fd;
  uint8_t data[188];
  dvbpsi_handle h_dvbpsi;
  int b_ok;
  uint16_t i_program_number, i_pmt_pid;

  if(i_argc != 4)
    return 1;

  i_fd = open(pa_argv[1], 0);
  i_program_number = atoi(pa_argv[2]);
  i_pmt_pid = atoi(pa_argv[3]);

  h_dvbpsi = dvbpsi_AttachPMT(i_program_number, DumpPMT, NULL);

  b_ok = ReadPacket(i_fd, data);

  while(b_ok)
  {
    uint16_t i_pid = ((uint16_t)(data[1] & 0x1f) << 8) + data[2];
    if(i_pid == i_pmt_pid)
      dvbpsi_PushPacket(h_dvbpsi, data);
    b_ok = ReadPacket(i_fd, data);
  }

  dvbpsi_DetachPMT(h_dvbpsi);
  return 0;
}
```

dvbstream con FEC modo 3

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <sys/poll.h>
#include <sys/stat.h>
#include <resolv.h>
#include <fcntl.h>
#include <unistd.h>
#include <signal.h>
#include <values.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

// DVB includes:
#ifdef NEWSTRUCT
#include <linux/dvb/dmx.h>
#include <linux/dvb/frontend.h>
#else
#include <ost/dmx.h>
#include <ost/sec.h>
#include <ost/frontend.h>
#endif

#include "ecc.h"
```

```
#include "rs.c"
#include "galois.c"
#include "berlekamp.c"

#include "rtp.h"
#include "mpegtools/transform.h"
#include "mpegtools/remux.h"

#include "tune.h"

// The default telnet port.
#define DEFAULT_PORT 12345

#define USAGE "\nUSAGE: dvbstream tpid1 tpid2 tpid3 .. tpid8\n\n"
#define PACKET_SIZE 188
#define PARIDAD 16

/* Thanks to Giancarlo Baracchino for this fix */
#define MTU 1500
#define IP_HEADER_SIZE 20
#define UDP_HEADER_SIZE 8
#define RTP_HEADER_SIZE 12
#define IPTOS_PREC_CRITIC_ECP 0x48
#define MAX_RTP_SIZE (MTU-IP_HEADER_SIZE-UDP_HEADER_SIZE-
RTP_HEADER_SIZE)
/*definiciones del entrelazador*/
#define INTERI = 51
#define INTERM = 4
#define writes(f,x) write((f),(x),strlen(x))

/* Signal handling code shamelessly copied from VDR by Klaus
Schmidinger
   - see http://www.cadsoft.de/people/kls/vdr/index.htm */

#ifdef NEWSTRUCT
char*
frontenddev[4]={"/dev/dvb/adapter0/frontend0","/dev/dvb/adapter1/front
end0","/dev/dvb/adapter2/frontend0","/dev/dvb/adapter3/frontend0"};
char*
dvrdev[4]={"/dev/dvb/adapter0/dvr0","/dev/dvb/adapter1/dvr0","/dev/dvb
/adapter2/dvr0","/dev/dvb/adapter3/dvr0"};
char*
demuxdev[4]={"/dev/dvb/adapter0/demux0","/dev/dvb/adapter1/demux0","/d
ev/dvb/adapter2/demux0","/dev/dvb/adapter3/demux0"};
#else
char*
frontenddev[4]={"/dev/ost/frontend0","/dev/ost/frontend1","/dev/ost/fr
ontend2","/dev/ost/frontend3"};
char*
dvrdev[4]={"/dev/ost/dvr0","/dev/ost/dvr1","/dev/ost/dvr2","/dev/ost/d
vr3"};
char*
secdev[4]={"/dev/ost/sec0","/dev/ost/sec1","/dev/ost/sec2","/dev/ost/s
ec3"};
char*
demuxdev[4]={"/dev/ost/demux0","/dev/ost/demux1","/dev/ost/demux2","/d
ev/ost/demux3"};
#endif

int card=0;
int Interrupted=0;
```

```
fe_spectral_inversion_t specInv=INVERSION_AUTO;
int tone=-1;
fe_modulation_t modulation=CONSTELLATION_DEFAULT;
fe_transmit_mode_t TransmissionMode=TRANSMISSION_MODE_DEFAULT;
fe_bandwidth_t bandWidth=BANDWIDTH_DEFAULT;
fe_guard_interval_t guardInterval=GUARD_INTERVAL_DEFAULT;
fe_code_rate_t HP_CodeRate=HP_CODERATE_DEFAULT;
unsigned int diseqc=0;
char pol=0;

int open_fe(int* fd_frontend,int* fd_sec) {

    if((*fd_frontend = open(frontenddev[card],O_RDWR)) < 0){
        perror("FRONTEND DEVICE: ");
        return -1;
    }
#ifdef NEWSTRUCT
    fd_sec=0;
#else
    if (fd_sec!=0) {
      if((*fd_sec = open(secdev[card],O_RDWR)) < 0){
          perror("SEC DEVICE: ");
          return -1;
      }
    }
#endif
    return 1;
}

static void SignalHandler(int signum) {
  if (signum != SIGPIPE) {
    Interrupted=signum;
  }
  signal(signum,SignalHandler);
}

long getmsec() {
  struct timeval tv;
  gettimeofday(&tv,(struct timezone*) NULL);
  return(tv.tv_sec%1000000)*1000 + tv.tv_usec/1000;
}

// There seems to be a limit of 8 simultaneous filters in the driver
#ifdef NEWSTRUCT
  #define MAX_CHANNELS 16
#else
  #define MAX_CHANNELS 8
#endif


void set_ts_filt(int fd,uint16_t pid, dmx_pes_type_t pestype)
{
  struct dmx_pes_filter_params pesFilterParams;

  fprintf(stderr,"Setting filter for PID %d\n",pid);
  pesFilterParams.pid     = pid;
  pesFilterParams.input   = DMX_IN_FRONTEND;
  pesFilterParams.output  = DMX_OUT_TS_TAP;
#ifdef NEWSTRUCT
  pesFilterParams.pes_type = pestype;
#else
```

```
  pesFilterParams.pesType = pestype;
#endif
  pesFilterParams.flags   = DMX_IMMEDIATE_START;

  if (ioctl(fd, DMX_SET_PES_FILTER, &pesFilterParams) < 0)  {
    fprintf(stderr,"FILTER %i: ",pid);
    perror("DMX SET PES FILTER");
  }
}

void make_nonblock(int f) {
  int oldflags;

  if ((oldflags=fcntl(f,F_GETFL,0)) < 0) {
    perror("F_GETFL");
  }
  oldflags|=O_NONBLOCK;
  if (fcntl(f,F_SETFL,oldflags) < 0) {
    perror("F_SETFL");
  }
}

typedef enum {STREAM_ON,STREAM_OFF} state_t;


  int socketIn, ns;
  int pids[MAX_CHANNELS];
  int pestypes[MAX_CHANNELS];
  unsigned char hi_mappids[8192];
  unsigned char lo_mappids[8192];
  int fd_sec;
  int fd_frontend;
  int pid,pid2;
  int connectionOpen;
  int fromlen;
  char hostname[64];
  char in_ch;
  struct hostent *hp;
  struct sockaddr_in name, fsin;
  int ReUseAddr=1;
  int oldflags;
  int npids = 0;
  int fd[MAX_CHANNELS];
  int to_stdout = 0; /* to stdout instead of rtp stream */

  /* rtp */
  struct rtpheader hdr;
  struct sockaddr_in sOut;
  int socketOut;

  ipack pa, pv;

#define IPACKS 2048
#define TS_SIZE 188
#define IN_SIZE TS_SIZE

int process_telnet() {
  char cmd[1024];
  int cmd_i=0;
  int i;
  char* ch;
```

```
  dmx_pes_type_t pestype;
  unsigned long freq=0;
  unsigned long srate=0;

    /* Open a new telnet session if a client is trying to connect */
    if (ns==-1) {
      if ((ns = accept(socketIn, (struct sockaddr *)&fsin, &fromlen))
> 0) {
        make_nonblock(ns);
        cmd_i=0;
        cmd[0]=0;
        printf("Opened connection\n");
        writes(ns,"220-DVBSTREAM - ");
        writes(ns,hostname);
        writes(ns,"\r\nDONE\r\n");
        connectionOpen=1;
      }
    }

    /* If a telnet session is open, receive and process any input */
    if (connectionOpen) {
      /* Read in at most a line of text - any ctrl character ends the
line */
      while (read(ns,&in_ch,1)>0) {
          if (in_ch < 32) break;
          /* Prevent buffer overflows */
          if (cmd_i < 1024-1) {
            cmd[cmd_i++]=in_ch;
            cmd[cmd_i]=0;
          }
      }
      if (in_ch > 0) {
        if (cmd_i > 0) {
          printf("CMD: \"%s\"\n",cmd);
          if (strcasecmp(cmd,"QUIT")==0) {
            writes(ns,"DONE\r\n");
            close(ns);
            ns=-1;
            connectionOpen=0;
            printf("Closed connection\n");
          } else if (strcasecmp(cmd,"STOP")==0) {
            writes(ns,"STOP\n");
            for (i=0;i<npids;i++) {
              if (ioctl(fd[i], DMX_STOP) < 0)  {
                 perror("DMX_STOP");
              }
            }
            for (i=0;i<8192;i++) {
              hi_mappids[i]=(i >> 8);
              lo_mappids[i]=(i&0xff);
            }
            writes(ns,"DONE\r\n");
          } else if (strncasecmp(cmd,"ADD",3)==0) {
            i=4;
            if ((cmd[3]=='V') || (cmd[3]=='v')) pestype=DMX_PES_VIDEO;
            else if ((cmd[3]=='A') || (cmd[3]=='a'))
pestype=DMX_PES_AUDIO;
            else if ((cmd[3]=='T') || (cmd[3]=='t'))
pestype=DMX_PES_TELETEXT;
            else { pestype=DMX_PES_OTHER; i=3; }
            while (cmd[i]==' ') i++;
```

```
            if ((ch=(char*)strstr(&cmd[i],":"))!=NULL) {
              pid2=atoi(&ch[1]);
              ch[0]=0;
            } else {
              pid2=-1;
            }
            pid=atoi(&cmd[i]);
            if (pid) {
              if (npids == MAX_CHANNELS) {
                fprintf(stderr,"\nsorry, you can only set up to 8
filters.\n\n");
                return(-1);
              } else {
                pestypes[npids]=pestype;
                pestype=DMX_PES_OTHER;
                pids[npids]=pid;
                if (pid2!=-1) {
                  hi_mappids[pid]=pid2>>8;
                  lo_mappids[pid]=pid2&0xff;
                  fprintf(stderr,"Mapping %d to %d\n",pid,pid2);
                }

                if((fd[npids] = open(demuxdev[card],O_RDWR)) < 0){
                  fprintf(stderr,"FD %i: ",i);
                  perror("DEMUX DEVICE: ");
                } else {
                  set_ts_filt(fd[npids],pids[npids],pestypes[npids]);
                  npids++;
                }
              }
            }
            writes(ns,"DONE\r\n");
          } else if (strcasecmp(cmd,"START")==0) {
            writes(ns,"START\n");
            for (i=0;i<npids;i++) {
              set_ts_filt(fd[i],pids[i],pestypes[i]);
            }
            writes(ns,"DONE\r\n");
          } else if (strncasecmp(cmd,"TUNE",4)==0) {
            for (i=0;i<8192;i++) {
              hi_mappids[i]=(i >> 8);
              lo_mappids[i]=(i&0xff);
            }
            for (i=0;i<npids;i++) {
              if (ioctl(fd[i], DMX_STOP) < 0)  {
                 perror("DMX_STOP");
                 close(fd[i]);
              }
            }
            npids=0;
            i=4;
            while (cmd[i]==' ') i++;
            freq=atoi(&cmd[i])*1000UL;
            while ((cmd[i]!=' ') && (cmd[i]!=0)) i++;
            if (cmd[i]!=0) {
              while (cmd[i]==' ') i++;
              pol=cmd[i];
              while ((cmd[i]!=' ') && (cmd[i]!=0)) i++;
              if (cmd[i]!=0) {
                while (cmd[i]==' ') i++;
                srate=atoi(&cmd[i])*1000UL;
```

```
                fprintf(stderr,"Tuning to
%ld,%ld,%c\n",freq,srate,pol);

tune_it(fd_frontend,fd_sec,freq,srate,pol,tone,specInv,diseqc,modulati
on,HP_CodeRate,TransmissionMode,guardInterval,bandWidth);
              }
            }
          }
          cmd_i=0;
          cmd[0]=0;
          writes(ns,"DONE\r\n");
        }
      }
    }
    return(0);
}


/* The output routine for sending a PS */
void my_write_out(uint8_t *buf, int count,void  *p)
{
  /* to fix: change this buffer size and check for overflow */
  static uint8_t out_buffer[1000000];
  static int out_buffer_n=0;
  int i;

  if (to_stdout) {
    /* This one is easy. */

    write(STDOUT_FILENO, buf, count);
  } else { /* We are streaming it. */
    /* Copy data to write to the end of out_buffer */

    memcpy(&out_buffer[out_buffer_n],buf,count);
    out_buffer_n+=count;

    /* Send as many full packets as possible */

    i=0;
    while ((i + MAX_RTP_SIZE) < out_buffer_n) {
       hdr.timestamp = getmsec()*90;
       sendrtp2(socketOut,&sOut,&hdr,&out_buffer[i],MAX_RTP_SIZE);
       i+=MAX_RTP_SIZE;
    }

    /* Move whatever data is left to the start of the buffer */

    memmove(&out_buffer[0],&out_buffer[i],out_buffer_n-i);
    out_buffer_n-=i;
  }
}

void my_ts_to_ps( uint8_t* buf, uint16_t pida, uint16_t pidv)
{
  uint16_t pid;
  ipack *p;
  uint8_t off = 0;

  pid = get_pid(buf+1);
  if (!(buf[3]&0x10)) // no payload?
    return;
```

```
  if (pid == pidv){
    p = &pv;
  } else {
    if (pid == pida){
      p = &pa;
    } else return;
  }

  if ( buf[1]&0x40) {
    if (p->plength == MMAX_PLENGTH-6){
      p->plength = p->found-6;
      p->found = 0;
      send_ipack(p);
      reset_ipack(p);
    }
  }

  if ( buf[3] & 0x20) {  // adaptation field?
    off = buf[4] + 1;
  }

  instant_repack(buf+4+off, TS_SIZE-4-off, p);
}


typedef struct {
  char *filename;
  int fd;
  int pids[MAX_CHANNELS];
  int num;
} pids_map_t;

pids_map_t *pids_map;
int map_cnt;


int main(int argc, char **argv)
{
  //  state_t state=STREAM_OFF;
  unsigned short int port=DEFAULT_PORT;
  int fd_dvr;
  int i;
  int j;
  int k;
  int indi;
  int posicion;
  unsigned char buf[MTU];
  struct pollfd pfds[2];  // DVR device and Telnet connection
  unsigned int secs = 0;
  unsigned long freq=0;
  unsigned long srate=0;
  int count;
  char* ch;
  dmx_pes_type_t pestype;
  int bytes_read;
  int do_analyse=0;
  unsigned char* free_bytes;
  unsigned char* puntero;
  int output_type=RTP_TS;
  int64_t counts[8192];
  double f;
```

```
  unsigned char msg[188];
  unsigned char codeword[204];
  unsigned char codeword2[1428];
  unsigned char entrelazador [612][204];
  unsigned char salida[1428];
  int cont7 = 0;
  /* Output: {uni,multi,broad}cast socket */
  char ipOut[20];
  int portOut;
  int ttl;

  pids_map = NULL;
  map_cnt = 0;

  fprintf(stderr,"dvbstream v0.4pre3 - (C) Dave Chapman 2001\n");
  fprintf(stderr,"Released under the GPL.\n");
  fprintf(stderr,"Latest version available from
http://www.linuxstb.org/\n");

  /* Initialise PID map */
  for (i=0;i<8192;i++) {
    hi_mappids[i]=(i >> 8);
    lo_mappids[i]=(i&0xff);
    counts[i]=0;
  }

  /* Set default IP and port */
  strcpy(ipOut,"224.0.1.2");
  portOut = 5004;

  if (argc==1) {
    fprintf(stderr,"Usage: dvbtune [OPTIONS] pid1 pid2 ... pid8\n\n");
    fprintf(stderr,"-i          IP multicast address\n");
    fprintf(stderr,"-r          IP multicast port\n");
    fprintf(stderr,"-o          Stream to stdout instead of
network\n");
    fprintf(stderr,"-o:file.ts  Stream to named file instead of
network\n");
    fprintf(stderr,"-n secs     Stop after secs seconds\n");
    fprintf(stderr,"-ps         Convert stream to Program Stream
format (needs exactly 2 pids)\n");
    fprintf(stderr,"-v vpid     Decode video PID (full cards
only)\n");
    fprintf(stderr,"-a apid     Decode audio PID (full cards
only)\n");
    fprintf(stderr,"-t ttpid    Decode teletext PID (full cards
only)\n");
    fprintf(stderr,"\nEstándar tuning options:\n\n");
    fprintf(stderr,"-f freq     absolute Frequency (DVB-S in Hz or
DVB-T in Hz)\n");
    fprintf(stderr,"            or L-band Frequency (DVB-S in Hz or
DVB-T in Hz)\n");
    fprintf(stderr,"-p [H,V]    Polarity (DVB-S only)\n");
    fprintf(stderr,"-s N        Symbol rate (DVB-S or DVB-C)\n");

    fprintf(stderr,"\nAdvanced tuning options:\n\n");
    fprintf(stderr,"-c [0-3]    Use DVB card #[0-3]\n");
    fprintf(stderr,"-qam X      DVB-T modulation - 16%s, 32%s, 64%s,
128%s or 256%s\n",(CONSTELLATION_DEFAULT==QAM_16 ? " (default)" :
""),(CONSTELLATION_DEFAULT==QAM_32 ? " (default)" :
""),(CONSTELLATION_DEFAULT==QAM_64 ? " (default)" :
```

```
""),(CONSTELLATION_DEFAULT==QAM_128 ? " (default)" :
""),(CONSTELLATION_DEFAULT==QAM_256 ? " (default)" : ""));
    fprintf(stderr,"-gi N       DVB-T guard interval 1_N (N=32%s,
16%s, 8%s or 4%s)\n",(GUARD_INTERVAL_DEFAULT==GUARD_INTERVAL_1_32 ? "
(default)" : ""),(GUARD_INTERVAL_DEFAULT==GUARD_INTERVAL_1_16 ? "
(default)" : ""),(GUARD_INTERVAL_DEFAULT==GUARD_INTERVAL_1_8 ? "
(default)" : ""),(GUARD_INTERVAL_DEFAULT==GUARD_INTERVAL_1_4 ? "
(default)" : ""));
    fprintf(stderr,"-cr N       DVB-T code rate. N=AUTO%s, 1_2%s,
2_3%s, 3_4%s, 5_6%s, 7_8%s\n",(HP_CODERATE_DEFAULT==FEC_AUTO ? "
(default)" : ""),(HP_CODERATE_DEFAULT==FEC_1_2 ? " (default)" :
""),(HP_CODERATE_DEFAULT==FEC_2_3 ? " (default)" :
""),(HP_CODERATE_DEFAULT==FEC_3_4 ? " (default)" :
""),(HP_CODERATE_DEFAULT==FEC_5_6 ? " (default)" :
""),(HP_CODERATE_DEFAULT==FEC_7_8 ? " (default)" : ""));
    fprintf(stderr,"-bw N       DVB-T bandwidth (Mhz) - N=6%s, 7%s or
8%s\n",(BANDWIDTH_DEFAULT==BANDWIDTH_6_MHZ ? " (default)" :
""),(BANDWIDTH_DEFAULT==BANDWIDTH_7_MHZ ? " (default)" :
""),(BANDWIDTH_DEFAULT==BANDWIDTH_8_MHZ ? " (default)" : ""));
    fprintf(stderr,"-tm N       DVB-T transmission mode - N=2%s or
8%s\n",(TRANSMISSION_MODE_DEFAULT==TRANSMISSION_MODE_2K ? " (default)"
: ""),(TRANSMISSION_MODE_DEFAULT==TRANSMISSION_MODE_8K ? " (default)"
: ""));

    fprintf(stderr,"\n-analyse    Perform a simple analysis of the
bitrates of the PIDs in the transport stream\n");

    fprintf(stderr,"\n");
    fprintf(stderr,"NOTE: Use pid1=8192 to broadcast whole TS stream
from a budget card\n");
    return(-1);
  } else {
    npids=0;
    pestype=DMX_PES_OTHER;  // Default PES type
    for (i=1;i<argc;i++) {
      if (strcmp(argv[i],"-ps")==0) {
        output_type=RTP_PS;
      } else if (strcmp(argv[i],"-analyse")==0) {
        do_analyse=1;
        output_type=RTP_NONE;
        if (secs==0) { secs=10; }
      } else if (strcmp(argv[i],"-i")==0) {
        i++;
        strcpy(ipOut,argv[i]);
      } else if (strcmp(argv[i],"-r")==0) {
        i++;
        portOut=atoi(argv[i]);
      } else if (strcmp(argv[i],"-f")==0) {
        i++;
        freq=atoi(argv[i])*1000UL;
      } else if (strcmp(argv[i],"-p")==0) {
        i++;
        if (argv[i][1]==0) {
    if (tolower(argv[i][0])=='v') {
            pol='V';
          } else if (tolower(argv[i][0])=='h') {
            pol='H';
          }
        }
      }
      else if (strcmp(argv[i],"-s")==0) {
```

```
        i++;
        srate=atoi(argv[i])*1000UL;
      }
      else if (strcmp(argv[i],"-D")==0)
      {
        i++;
        diseqc=atoi(argv[i]);
        if(diseqc < 0 || diseqc > 4) diseqc = 0;
      }
      else if (strcmp(argv[i],"-o")==0) {
        to_stdout = 1;
      } else if (strcmp(argv[i],"-n")==0) {
        i++;
        secs=atoi(argv[i]);
      } else if (strcmp(argv[i],"-c")==0) {
        i++;
        card=atoi(argv[i]);
        if ((card < 0) || (card > 3)) {
          fprintf(stderr,"ERROR: card parameter must be between 0 and
4\n");
        }
      } else if (strcmp(argv[i],"-v")==0) {
        pestype=DMX_PES_VIDEO;
      } else if (strcmp(argv[i],"-a")==0) {
        pestype=DMX_PES_AUDIO;
      } else if (strcmp(argv[i],"-t")==0) {
        pestype=DMX_PES_TELETEXT;
      } else if (strcmp(argv[i],"-qam")==0) {
        i++;
        switch(atoi(argv[i])) {
          case 16:  modulation=QAM_16; break;
          case 32:  modulation=QAM_32; break;
          case 64:  modulation=QAM_64; break;
          case 128: modulation=QAM_128; break;
          case 256: modulation=QAM_256; break;
          default:
            fprintf(stderr,"Invalid QAM rate: %s\n",argv[i]);
            exit(0);
        }
      } else if (strcmp(argv[i],"-gi")==0) {
        i++;
        switch(atoi(argv[i])) {
          case 32:  guardInterval=GUARD_INTERVAL_1_32; break;
          case 16:  guardInterval=GUARD_INTERVAL_1_16; break;
          case 8:   guardInterval=GUARD_INTERVAL_1_8; break;
          case 4:   guardInterval=GUARD_INTERVAL_1_4; break;
          default:
            fprintf(stderr,"Invalid Guard Interval: %s\n",argv[i]);
            exit(0);
        }
      } else if (strcmp(argv[i],"-tm")==0) {
        i++;
        switch(atoi(argv[i])) {
          case 8:   TransmissionMode=TRANSMISSION_MODE_8K; break;
          case 2:   TransmissionMode=TRANSMISSION_MODE_2K; break;
          default:
            fprintf(stderr,"Invalid Transmission Mode: %s\n",argv[i]);
            exit(0);
        }
      } else if (strcmp(argv[i],"-bw")==0) {
        i++;
```

```
        switch(atoi(argv[i])) {
          case 8:   bandWidth=BANDWIDTH_8_MHZ; break;
          case 7:   bandWidth=BANDWIDTH_7_MHZ; break;
          case 6:   bandWidth=BANDWIDTH_6_MHZ; break;
          default:
            fprintf(stderr,"Invalid DVB-T bandwidth: %s\n",argv[i]);
            exit(0);
        }
      } else if (strcmp(argv[i],"-cr")==0) {
        i++;
        if (!strcmp(argv[i],"AUTO")) {
          HP_CodeRate=FEC_AUTO;
        } else if (!strcmp(argv[i],"1_2")) {
          HP_CodeRate=FEC_1_2;
        } else if (!strcmp(argv[i],"2_3")) {
          HP_CodeRate=FEC_2_3;
        } else if (!strcmp(argv[i],"3_4")) {
          HP_CodeRate=FEC_3_4;
        } else if (!strcmp(argv[i],"5_6")) {
          HP_CodeRate=FEC_5_6;
        } else if (!strcmp(argv[i],"7_8")) {
          HP_CodeRate=FEC_7_8;
        } else {
          fprintf(stderr,"Invalid Code Rate: %s\n",argv[i]);
          exit(0);
        }
      } else if (strstr(argv[i], "-o:")==argv[i]) {
        if (strlen(argv[i]) > 3) {
          int pid_cnt = 0, pid, j;

          map_cnt++;
          pids_map = (pids_map_t*) realloc(pids_map,
sizeof(pids_map_t) * map_cnt);
          for(j=0; j < MAX_CHANNELS; j++) pids_map[map_cnt-1].pids[j]
= -1;
          pids_map[map_cnt-1].filename = (char *)
malloc(strlen(argv[i]) - 2);
          strcpy(pids_map[map_cnt-1].filename, &argv[i][3]);
          i++;

          while(i < argc) {
            int found;
            if (sscanf(argv[i], "%d", &pid) == 0) {
              i--;
              break;
            }

            // block for the map
            found = 0;
            for (j=0;j<MAX_CHANNELS;j++) {
              if(pids_map[map_cnt-1].pids[j] == pid) found = 1;
            }
            if (found == 0) {
              pids_map[map_cnt-1].pids[pid_cnt] = pid;
              pid_cnt++;
            }

            // block for the list of pids to demux
            found = 0;
            for (j=0;j<npids;j++) {
              if(pids[j] == pid) found = 1;
```

```
            }
            if (found==0) {
              pestypes[npids] = DMX_PES_OTHER;
              pids[npids++] = pid;
            }
            i++;
          }

          if (pid_cnt > 0) {
             FILE *f;
             f = fopen(pids_map[map_cnt-1].filename, "w+b");
             if (f != NULL) {
               pids_map[map_cnt-1].fd = fileno(f);
               make_nonblock(pids_map[map_cnt-1].fd);
               fprintf(stderr, "Open file %s\n", pids_map[map_cnt-
1].filename);
             } else {
               pids_map[map_cnt-1].fd = -1;
               fprintf(stderr, "Couldn't open file %s, errno:%d\n",
pids_map[map_cnt-1].filename, errno);
             }
         }
           output_type = MAP_TS;
          }
        } else {
          if ((ch=(char*)strstr(argv[i],":"))!=NULL) {
            pid2=atoi(&ch[1]);
            ch[0]=0;
          } else {
            pid2=-1;
          }
          pid=atoi(argv[i]);
          if (pid) {
            if (npids == MAX_CHANNELS) {
              fprintf(stderr,"\nSorry, you can only set up to %d
filters.\n\n",MAX_CHANNELS);
              return(-1);
            } else {
              pestypes[npids]=pestype;
              pestype=DMX_PES_OTHER;
              pids[npids++]=pid;
              if (pid2!=-1) {
                hi_mappids[pid]=pid2>>8;
                lo_mappids[pid]=pid2&0xff;
                fprintf(stderr,"Mapping %d to %d\n",pid,pid2);
              }
            }
          }
        }
      }
    }

  if ((output_type==RTP_PS) && (npids!=2)) {
    fprintf(stderr,"ERROR: PS requires exactly two PIDS - video and
audio.\n");
    exit;
  }


  if (signal(SIGHUP, SignalHandler) == SIG_IGN) signal(SIGHUP,
SIG_IGN);
```

```
  if (signal(SIGINT, SignalHandler) == SIG_IGN) signal(SIGINT,
SIG_IGN);
  if (signal(SIGTERM, SignalHandler) == SIG_IGN) signal(SIGTERM,
SIG_IGN);
  if (signal(SIGALRM, SignalHandler) == SIG_IGN) signal(SIGALRM,
SIG_IGN);

  if ( (freq>100000000)) {
    if (open_fe(&fd_frontend,0)) {

i=tune_it(fd_frontend,0,freq,srate,0,tone,specInv,diseqc,modulation,HP
_CodeRate,TransmissionMode,guardInterval,bandWidth);
    }
  } else if ((freq!=0) && (pol!=0) && (srate!=0)) {
    if (open_fe(&fd_frontend,&fd_sec)) {

i=tune_it(fd_frontend,fd_sec,freq,srate,pol,tone,specInv,diseqc,modula
tion,HP_CodeRate,TransmissionMode,guardInterval,bandWidth);
    }
  }

  if (i<0) { exit(i); }

  for (i=0;i<npids;i++) {
    if((fd[i] = open(demuxdev[card],O_RDWR)) < 0){
      fprintf(stderr,"FD %i: ",i);
      perror("DEMUX DEVICE: ");
      return -1;
    }
  }

  if((fd_dvr = open(dvrdev[card],O_RDONLY|O_NONBLOCK)) < 0){
    perror("DVR DEVICE: ");
    return -1;
  }

  /* Now we set the filters */
  for (i=0;i<npids;i++) set_ts_filt(fd[i],pids[i],pestypes[i]);

  if (do_analyse) {
    fprintf(stderr,"Analysing PIDS\n");
  } else {
    if (to_stdout) {
      fprintf(stderr,"Output to stdout\n");
    }
    else {
      ttl = 2;
      fprintf(stderr,"Using %s:%d:%d\n",ipOut,portOut,ttl);

      /* Init RTP */
      socketOut = makesocket(ipOut,portOut,ttl,&sOut);
      #warning WHAT SHOULD THE PAYLOAD TYPE BE FOR "MPEG-2 PS" ?
      initrtp(&hdr,(output_type==RTP_TS ? 33 : 34));
      fprintf(stderr,"version=%X\n",hdr.b.v);
      unsigned char tos;
//asignamos la calidad de servicio
      socklen_t optlen;
        tos=IPTOS_PREC_CRITIC_ECP;
          optlen=sizeof(unsigned char);
            setsockopt(socketOut,SOL_IP,IP_TOS,&tos,optlen);
```

```
    }
    fprintf(stderr,"Streaming %d stream%s\n",npids,(npids==1 ? "" :
"s"));
  }

  if (output_type==RTP_PS) {
    init_ipack(&pa, IPACKS,my_write_out, 1);
    init_ipack(&pv, IPACKS,my_write_out, 1);
  }

  /* Read packets */
  free_bytes = buf;

  /* Setup socket to accept input from a client */
  gethostname(hostname, sizeof(hostname));
  if ((hp = gethostbyname(hostname)) == NULL) {
    fprintf(stderr, "%s: host unknown.\n", hostname);
    exit(1);
  }
  if ((socketIn = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("server: socket");
    exit(1);
  }

setsockopt(socketIn,SOL_SOCKET,SO_REUSEADDR,&ReUseAddr,sizeof(ReUseAdd
r));
  name.sin_family = AF_INET;
  name.sin_port = htons(port);
  name.sin_addr.s_addr = htonl(INADDR_ANY);
  if (bind(socketIn,(struct sockaddr* )&name,sizeof(name)) < 0) {
    perror("server: bind");
    exit(1);
  }

  make_nonblock(socketIn);

  if (listen(socketIn, 1) < 0) {
    perror("server: listen");
    exit(1);
  }

  connectionOpen=0;
  ns=-1;
  pfds[0].fd=fd_dvr;
  pfds[0].events=POLLIN|POLLPRI;
  pfds[1].events=POLLIN|POLLPRI;

  /* Set up timer */
  if (secs > 0) alarm(secs);
  while ( !Interrupted) {
    /* Poll the open file descriptors */
    if (ns==-1) {
        poll(pfds,1,500);
    } else {
        pfds[1].fd=ns;  // This can change
        poll(pfds,2,500);
    }

    process_telnet();  // See if there is an incoming telnet
connection
```

```
    if (output_type==RTP_TS) {
      /* Attempt to read 188 bytes from /dev/ost/dvr */
// Reed solomon
      puntero=msg;
      posicion=free_bytes;
      if ((bytes_read = read(fd_dvr,puntero,PACKET_SIZE)) > 0) {
        if (bytes_read!=PACKET_SIZE) {
          fprintf(stderr,"No bytes left to read - aborting\n");
          break;
        }

        pid=((free_bytes[1]&0x1f) << 8) | (free_bytes[2]);
        free_bytes[1]=(free_bytes[1]&0xe0)|hi_mappids[pid];
        free_bytes[2]=lo_mappids[pid];
      encode_data(msg, sizeof(msg), codeword);
      free_bytes+=bytes_read+PARIDAD;
      for (j = (0+204*cont7); j < (204 + 204*cont7); j++) codeword2[j]
= codeword[j-(204*cont7)];
// entrelazador
        if (cont7==7) {
            indi = 0;
        while (indi<1428) {
            for (j = 0; j<204 ;j++) {
            entrelazador[2*j][j] = codeword2 [indi];
            salida[indi] = entrelazador [0][j];
            for (k = 1 ; k < 204 ; k++) entrelazador [k-1][j] =
entrelazador [k][j];
            indi++;
        }
        }

          hdr.timestamp = getmsec()*90;
          if (to_stdout) {
            write(1, salida,1428);
          } else {
            sendrtp2(socketOut,&sOut,&hdr,salida,1428);

          }
        cont7=0;
          free_bytes = buf;
        }
        count++;
      cont7++;
      }
    } else if (output_type==RTP_PS) {
       if (read(fd_dvr,buf,TS_SIZE) > 0) {
         my_ts_to_ps((uint8_t*)buf, pids[0], pids[1]);
       }
    } else if(output_type==MAP_TS) {
       if(read(fd_dvr, buf, TS_SIZE) > 0) {
         if(buf[0] == 0x47) {
           int pid, i, j;

           pid = ((buf[1] & 0x1f) << 8) | buf[2];
           if (pids_map != NULL)        {
             for (i = 0; i < map_cnt; i++) {
               for (j = 0; j < MAX_CHANNELS; j++) {
                 if (pids_map[i].pids[j] == pid) {
                   errno = 0;
                   write(pids_map[i].fd, buf, TS_SIZE);
                 }
```

```
                }
              }
            }
          } else {
            fprintf(stderr, "NON 0X47\n");
          }
        }
    } else {
      if (do_analyse) {
        if (read(fd_dvr,buf,TS_SIZE) > 0) {
          pid=((buf[1]&0x1f) << 8) | (buf[2]);
          counts[pid]++;
        }
      }
    }
  }

  if (Interrupted) {
    fprintf(stderr,"Caught signal %d - closing
cleanly.\n",Interrupted);
  }

  if (ns!=-1) close(ns);
  close(socketIn);

  if (!to_stdout) close(socketOut);
  for (i=0;i<npids;i++) close(fd[i]);
  close(fd_dvr);
  close(fd_frontend);
  if (fd_sec) close(fd_sec);

  if (do_analyse) {
    for (i=0;i<8192;i++) {
      if (counts[i]) {
        f=(counts[i]*184.0*8.0)/(secs*1024.0*1024.0);
        if (f >= 1.0) {
          fprintf(stdout,"%d,%.3f Mbit/s\n",i,f);
        } else {
          fprintf(stdout,"%d,%.0f kbit/s\n",i,f*1024);
        }
      }
    }
  }
  return(0);
}
```