TREBALL DE FI DE CARRERA

TÍTOL: Multiplexació de senyals DVB: anàlisi i millora del *software* Mplex 13818

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Sistemes de Telecomunicació

AUTOR: Marc Molina Pena

DIRECTOR: David Rincón Rivera

DATA: 20 de gener de 2010

Títol: Multiplexació de senyals DVB: anàlisi i millora del *software* Mplex 13818

Autor: Marc Molina Pena

Director: David Rincón Rivera

Data: 20 de gener de 2010

Resum

Aquest TFC tracta sobre la multiplexació de senyals de vídeo i àudio MPEG-2, estàndard en què es basen la major part d'aplicacions de difusió i emmagatzematge de vídeo digital. Concretament, se centra en els *Transport Streams*, adequats per a difondre continguts audiovisuals en format digital a través de multitud de plataformes com poden ser la televisió digital terrestre (TDT), la televisió per satèl·lit i per cable, i la difusió de continguts audiovisuals a través d'Internet, entre altres.

L'objectiu final del treball era l'obtenció d'una peça de *software* de codi obert que permetés generar aquest tipus de fluxos de transport. Per aconseguir-ho, s'ha partit d'un programa ja existent, l'Mplex 13818, del qual se n'ha modificat el codi per millorar les seves prestacions, ja que les proves inicials van descobrir deficiències en la seva implementació que n'impossibilitaven l'ús.

Després de realitzar una explicació teòrica sobre els estàndards que intervenen en la generació dels fluxos de transport, una part important del treball s'ha centrat en l'anàlisi del Mplex 13818, amb la finalitat de descriure l'algoritme de multiplexació que utilitza i buscar els problemes d'implementació que s'havien de tractar. Concretament s'han detectat dos aspectes a corregir, relacionats amb el sistema de temporització, que impossibilitaven la correcta reproducció dels *streams* generats amb Mplex.

Per avaluar el resultat de les millores proposades s'han realitzat proves a partir de captures de diferents múltiplex, combinant-les tal d'obtenir nous fluxos de transport barrejant programes de les fluxos originals. Els resultats han estat satisfactoris, aconseguint que els *Transport Streams* generats amb Mplex compleixin l'estàndard ISO 13818-1 (capa de sistema MPEG-2).

Title: Multiplexing of DVB signals: analysis and improvement of Mplex 13818

Author: Marc Molina Pena

Director: David Rincón Rivera

Date: Januray, 20th 2010

Overview

This TFC focuses in the multiplexing of MPEG-2 video and audio streams. MPEG-2 is the standard followed by the majority of applications of digital video storage and broadcast. Specifically, our work focuses on the MPEG-2 Transport Stream, used for broadcasting digital audiovisual contents through many different platforms such as digital terrestrial television (DTT), satellite and cable television, and the dissemination of audiovisual contents via Internet, among others.

The goal of our work was to obtain a piece of open source software able to correctly generate Transport Streams. We started from an existing program, Mplex 13818, whose code has been modified in order to improve its performance, since the initial tests found shortcomings in its implementation that precluded its use.

After an introduction to the standards involved in the generation of Transport Streams, an important part of the work has focused on the analysis of Mplex 13818, the description of the multiplexing algorithm, and the identification of the implementation problems. We found two issues related to the timing system, that made impossible the correct playback of streams generated by Mplex.

To evaluate the results of the proposed improvements, some tests combining different input streams to obtain new transport streams were carried out. The results have been satisfactory, and we claim to have obtained a truly ISO standard 13818-1 –compliant version of Mplex 13818.

ÍNDEX

CAPITOL 1. INTRODUCCIÓ I OBJECTIUS

En els darrers anys s'ha produït un gran progrés en les tecnologies de processat, emmagatzematge i transmissió de vídeo i àudio digital. Els sistemes d'emmagatzematge *Digital Video Disc* (DVD), la televisió digital de pagament per satèl·lit i per cable, la difusió de vídeo i ràdio a través d'Internet, i l'actual desplegament de la televisió digital terrestre amb la imminent apagada analògica, són alguns exemples de com els avenços en aquest camp estan molt presents en la nostra societat.

Aquesta revolució digital ha estat afavorida pel gran nombre d'avantatges que presenta la televisió digital respecte l'analògica, tan per als proveïdors de continguts audiovisuals com per a l'usuari final. D'entrada, l'ús del format digital incrementa la qualitat del vídeo i àudio, i permet a l'usuari interactuar amb la informació que rep. A més, la compressió digital permet difondre diversos programes en un mateix flux de transport, reduint els costos de distribució de continguts audiovisuals i ocupant menys ample de banda del que ocupava un únic programa analògic, la qual cosa suposa un millor aprofitament de l'espectre radioelèctric, i la possibilitat d'oferir televisió d'alta definició.

Els principals responsables de que totes aquestes aplicacions siguin possibles són els sistemes de codificació i transmissió de la família *Moving Pictures Experts Group* (MPEG) que han permès l'estandardització dels mecanismes de codificació digital de senyals audiovisuals. MPEG ha esdevingut l'estàndard de compressió, emmagatzematge i transmissió de vídeo i àudio més utilitzat, i en l'actualitat s'hi basen la majoria d'aplicacions de difusió digital. El gran esforç d'aquesta organització en el disseny de tecnologies de compressió eficients, que permeten reduir significativament l'ample de banda necessari per transmetre aquest tipus de senyals, ha estat l'avenç més important que ha permès que la radiodifusió massiva de vídeo i àudio digital sigui possible avui en dia.

D'altra banda, la ETSI (*European Telecommunication Standards Institute*) ha promogut el projecte DVB, amb l'objectiu de d'estandarditzar les especificacions per a la difusió d'aquest tipus de senyals a nivell físic, definint el tipus de modulació, per exemple, en funció del sistema de radiodifusió.

A la vista de la gran varietat d'aplicacions que té en l'actualitat la transmissió de vídeo digital en format MPEG, en aquest TFC es pretén obtenir un *software* de codi obert que actuï com a multiplexor de vídeo i àudio digital, capaç de generar els fluxos adequats per a la transmissió de diversos programes audiovisuals a través de canals de comunicacions sorollosos, seguint l'estàndard MPEG-2 (ISO 13818). Concretament, es vol aconseguir un multiplexor de *Transport Streams*, el tipus de flux MPEG-2 que compleix aquestes característiques. Per fer-ho es partirà d'un peça de codi existent, l'"Mplex 13818", de codi obert i ús lliure, i es modificarà per tal de millorar la seva funcionalitat.

A l'inici del TFC, en les diverses proves realitzades es va determinar que cap reproductor MPEG era capaç de reproduir correctament els fluxos de transport generats amb aquest programa. Així doncs, s'han hagut de realitzar les modificacions necessàries per tal que els fluxos de sortida compleixin les especificacions de l'estàndard ISO 13818.

Una part molt important del treball s'ha centrat en l'anàlisi del Mplex 13818, entrant a fons en el codi per tal de descriure'l a nivell funcional, i amb la finalitat de trobar deficiències en la seva implementació que s'havien de millorar. Tal i com reconeix el propi desenvolupador del *software*, Oskar Schirmer, el codi és llarg (amb més de 20 fitxers de codi, i les corresponents capçaleres) i confús, i té molts algoritmes difícils d'entendre en gran part a causa de la poca informació que es dóna sobre les variables i funcions en el propi codi. És per aquest motiu que ha sigui una tasca especialment difícil arribar a un nivell de comprensió del codi suficient com per descriure l'algoritme de multiplexació, per una banda, i trobar els punts que s'havien de modificar per aconseguir que funcionés correctament.

Aquest TFC s'emmarca dins una sèrie de TFCs futurs que tenen com a objectiu muntar un escenari complet de multiplexació estadística, on es provaran diversos algoritmes de multiplexació. L'anàlisi del codi d'Mplex que s'ha realitzat en aquest treball servirà com a base per a la resta.

La resta dels capítols d'aquest document s'estructuren de la següent manera: en el capítol 2 es fa una explicació teòrica dels estàndards que defineixen la codificació de senyals de vídeo i àudio digital; el capítol 3 proporciona una primera introducció al *software* del qual es partirà: Mplex 13818; en el capítol 4 s'analitza a fons el codi i l'algoritme de multiplexació del programa amb la finalitat de trobar les causes per les quals els fluxos de transport no es reprodueixen correctament; el capítol 5 explica les millores que s'han realitzat en el codi, i presenta els resultats obtinguts; finalment, en el capítol 6 es presenten les conclusions que s'han extret de la realització del treball, i es proposen línies futures d'investigació.

CAPITOL 2. DIGITAL VIDEO BROADCASTING (DVB)

2.1 Digital Video Broadcasting (DVB)

DVB [W6] és un organisme format per més de 250 empreses i organitzacions de tot el món que s'encarrega d'estandarditzar els sistemes de transmissió i difusió de vídeo digital amb la finalitat d'aconseguir una televisió digital compatible. Està liderat per les següents organitzacions:

- European Committee for Electrotechnical Standardization (CENELEC)
- European Telecommunications Standards Institute (ETSI)
- European Broadcasting Union (EBU)

Els seus estàndards permeten la difusió de continguts audiovisuals a través de diferents plataformes i canals de comunicació, i defineixen tant la capa de transmissió del senyal, a nivell de codificació digital dels fluxos adequats per transportar el senyal, com la capa física, a nivell de medi de difusió. Els estàndards de DVB es diferencien precisament en la capa física (modulació, etc), que es defineix en funció de les característiques del sistema de radiodifusió o transport.

- Terrestre: DVB-T
- Satèl·lit: DVB-S
- Cable: DVB-C
- Terrestre per a dispositius portàtils: DVB-H
- Satèl·lit per a dispositius portàtils: DVB-SH
- Xarxa IP: DVB-IP

Per a tots ells, la capa de transmissió es basa en els estàndards definits per MPEG. Concretament, el tipus de flux utilitzat en DVB és el *Transport Stream*, definit en l'estàndard ISO 13818-1 (Capa de sistema MPEG-2) i adequat per al transport de senyals de vídeo i àudio comprimits seguint l'ISO 13818-2 (Capa de compressió MPEG-2), a través de canals de comunicacions que introdueixen errors i pèrdues d'informació. Els fluxos de transport MPEG-2 es complementen amb la "Informació de Servei" o "*Service Information*" (*SI*), definida per DVB-SI en la norma ETSI EN 300 468.

DVB no és l'únic estàndard de televisió digital al món. Existeixen dos grans estàndards més: l'ISDB [W13] al Japó, i l'ATSC [W14] a Nord Amèrica. Igual que en DVB, tots dos estan basats en MPEG-2, i per tant són similars pel que fa al sistema de codificació que utilitzen; bàsicament es diferencien en la seva capa física.

2.2 Moving Pictures Experts Group 2 (MPEG-2)

2.2.1 Introducció

MPEG-2 és un estàndard de codificació i transmissió de programes audiovisuals, creat per MPEG i publicat com ISO/IEC 13818 [E1, E2]. Defineix tant la codificació font de vídeo i àudio comprimit com la generació dels fluxos adequats per tal d'emmagatzemar o transmetre la informació. Entre les nombroses aplicacions que té destaquen els sistemes d'emmagatzematge DVD, i els sistemes de difusió DVB.

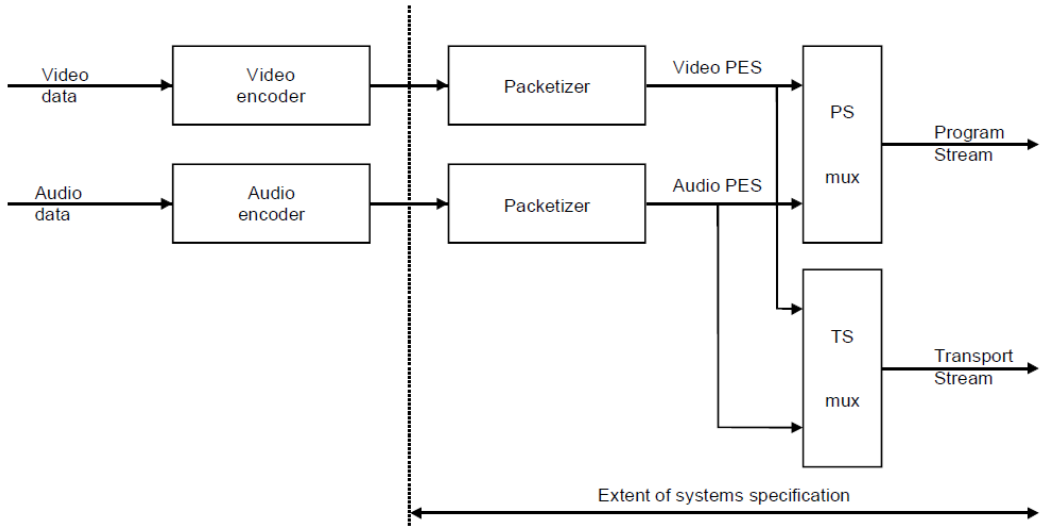


Figura 2.1 Esquema del procés de codificació MPEG-2. Extret de [E1]

Les diferents etapes del procés de codificació d'MPEG-2 es poden dividir en dues capes: la capa de compressió (ISO/IEC 13818-2) i la capa de sistema (ISO/IEC 13818-1).

2.2.2 Capa de compressió

En la capa de compressió es realitzen les operacions de codificació font, basada en diferents mecanismes de compressió de dades aplicats als fluxos de vídeo i àudio d'entrada. Com a font d'informació, es disposa d'un programa digitalitzat sense compressió, format per un vídeo, un o més àudios, i altres dades. La informació de cadascun d'aquests components s'organitza en "Unitats de Presentació" (UP), que un cop comprimides passen a anomenar-se "Unitats d'Accés" (UA). A cadascun dels fluxos de sortida resultants del procés de compressió se l'anomena *Elementary Stream (ES)*.

- Vídeo

En la component de vídeo, es parteix d'un senyal digital sense comprimir, en format CCIR 601, codificat en 4:2:2 i quantificat amb 8 bits, que dóna com a resultat imatges de 830 Kbytes (en sistemes de 625 línies). Cadascuna d'aquestes imatges correspon a una Unitat de Presentació.

El procés de compressió consisteix en la reducció de la mida d'aquestes imatges utilitzant diferents tècniques. Cada una de les imatges comprimides correspon a una "Unitat d'Accés" (UA). En vídeo MPEG-2, es defineixen tres tipus d'UA:

- I (*Intra*): Es codifiquen sense referències a cap altra imatge. Contenen tota la informació necessària per reconstruir la imatge. Són, essencialment, imatges JPEG.
- P (Previstes): Es codifiquen fent referència a una imatge I o P anterior, utilitzant tècniques de compensació de moviment.
- B (Bidireccionals): Es codifiquen per interpol·lació entre les dues imatges P o I anterior i posterior.

Les tècniques de compensació de moviment aprofiten la redundància temporal entre imatges consecutives, transmetent les diferències entre aquestes. Les imatges es divideixen en blocs (anomenats macroblocs), i es busca el desplaçament de cadascun d'aquests macroblocs entre imatges successives; aquesta diferència es representa i es transmet mitjançant els anomenats "vectors de desplaçament". L'error comès entre la imatge obtinguda a partir de la compensació de moviment i la imatge real es corregeix mitjançant la "imatge diferència". Periòdicament, es codifica una imatge del tipus I que serveix de referència clau per a la codificació de les imatges P posteriors, i les B anteriors i posteriors. És important que es transmetin imatges d'aquest tipus, per tal que els errors es propaguin el mínim possible.

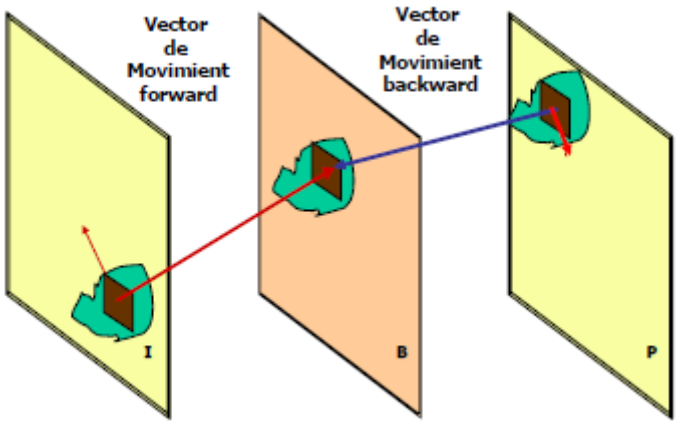


Figura 2.2 Tipus d'imatges comprimides mitjançant compensació de moviment. Extret de [M1]

La informació de l'error o diferència es passa per un procés pràcticament idèntic al de JPEG (eliminació d'altres freqüències) i es codifica entròpicament, amb codis de *Huffman*, assignant codis de menor o major longitud a cada símbol en funció de la probabilitat de que apareguin en el flux.

Al grup d'imatges relacionades entre elles, delimitat per dues imatges del tipus *Intra*, se l'anomena GOP (*Group of Pictures*). Hi ha moltes estructures possibles, però s'acostuma a utilitzar següent:

I	B	B	P	B	B	P	B	B	P	B	B	P	B	B	I

Les imatges P del GOP depenen de la imatge I o P immediatament anterior, mentre que les imatges B estan referides a la imatge I o P immediatament anterior i a la immediatament posterior. Així doncs, tan en el procés de codificació com en el de descodificació, abans de processar una imatge tipus B s'haurà de codificar/descodificar la imatge I o P posterior de la qual en depèn. El flux elemental de vídeo resultant del procés de compressió contindrà les imatges disposades en l'ordre que s'han de processar, no en ordre de presentació, de manera que serà necessari algun mecanisme per reordenar la presentació de les imatges; MPEG-2 tracta aquest aspecte en la capa de sistema.

Amb un vídeo comprimit en CBR (*Constant Bit Rate*), amb una velocitat binària de 5 Mbps, les unitats d'accés tindran una mida mitja (dependrà de la complexitat de la imatge) de 100, 33 i 12 kbytes per a imatges del tipus I, P i B, respectivament.

- Àudio

Pel que fa l'àudio, el senyal d'entrada és una trama AES/EBU, que s'organitza en blocs de 192 trames, cadascuna de les quals conté 32 bits de dades de cadascun dels dos canals estèreo, i correspon a una unitat de presentació. El procés de compressió es basa en la utilització de models psicoacústics, reduint el nombre d'informació associada a les freqüències per a les que l'oïda humana té menys sensibilitat.

2.2.3 Capa de sistema

La capa de sistema especifica la conformació dels anomenats fluxos de programa (*Program Stream (PS)*) i fluxos de transport (*Transport Stream (TS)*), adequant les dades generades en la capa de compressió per tal que puguin ser emmagatzemades o transportades, respectivament. En aplicació a DVB s'utilitza el *Transport Stream*, que és l'adequat per tal de transmetre la informació audiovisual a través d'un canal de comunicacions sorollós.

2.2.3.1 Empaquetatge PES

En la primera etapa del procés de generació dels fluxos, cadascun dels ES generats en la capa de compressió s'empaqueten, formant l'anomenat *Packetized Elementary Stream (PES)*. Un paquet PES està format per una capçalera i una càrrega útil (*payload*), que correspon a una porció de dades del ES. Els paquets PES tenen una longitud indefinida i que pot ser variable, i no hi ha restriccions en quant a la correspondència entre UA i paquets PES. No obstant, s'acostuma a generar fluxos en què cada paquet PES encapsula una única UA completa.

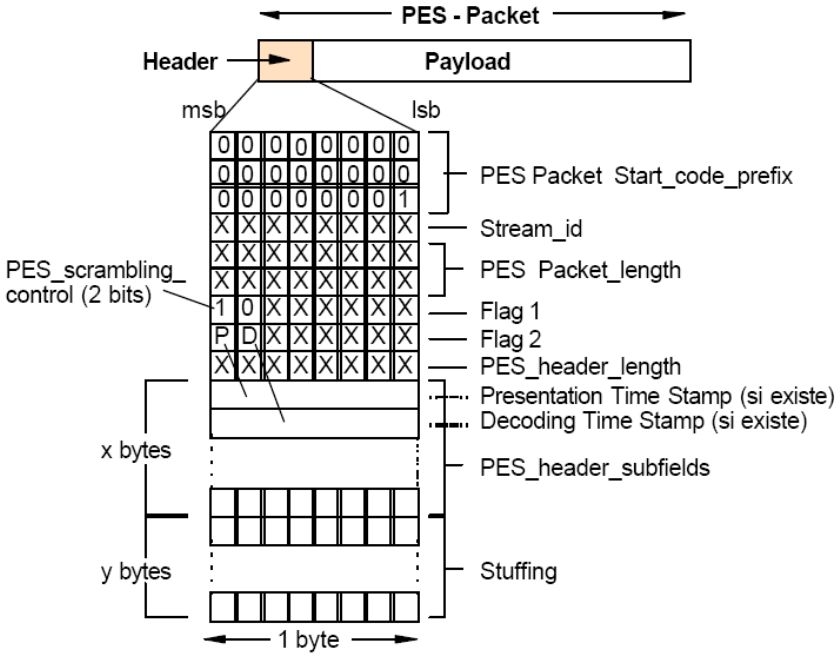


Figura 2.3 Estructura de la capçalera d'un paquet PES. Extret de [M2]

Com es pot observar en la figura 2.3, la capçalera està formada per 9 *bytes* fixes i un número variable de *bytes* opcionals. A continuació s'expliquen els diferents camps que la formen:

- Start_code_prefix: Indica l'inici d'un paquet PES. Està format per tres *bytes* amb valor 0x00, 0x00, 0x01. Aquest valor no es pot repetir a l'interior del flux PES.
- Stream_id: Identifica el flux elemental.
- Packet_length: Indica la longitud del paquet PES sencer.
- Flag 1 i Flag 2: Indiquen la presencia dels diferents camps opcionals que es poden trobar en la capçalera.
- PES_header_length: Indica la longitud dels camps opcionals de la capçalera. És el darrer *byte* obligatori.

Existeixen 25 camps opcionals que es poden trobar en la capçalera d'un paquet PES, els quals porten informació addicional sobre el flux elemental com per exemple la seva prioritat, si està xifrat, si té *copyright*, etc. Un dels camps opcionals més importants, present en tots els fluxos PES que requereixen sincronisme (no en tots els paquets) són els *timestamps*. Els *timestamps* són el mecanisme que proporciona MPEG per sincronitzar els diferents fluxos que formen part d'un mateix programa, indicant en quin moment s'ha de presentar i/o descodificar cada unitat d'accés. Existeixen dos tipus de *timestamp*:

- *Presentation Time Stamp (PTS)*: Indica l'instant en què s'ha de presentar la UA associada al paquet PES.
- *Decoding Time Stamp (DTS)*: Indica en quin moment s'ha de descodificar la UA associada al paquet PES.

Tan el *timestamp* de presentació com el de descodificació fan referència a la primera UA que comença dins el paquet PES. Alguns fluxos es poden codificar utilitzant únicament PTS's, si les UA es descodifiquen en ordre de presentació; en aquest cas, el descodificador interpretarà que la UA s'ha de descodificar i presentar en el moment que indica el *timestamp* de presentació. L'estàndard no especifica com solucionar la problemàtica que comporta el retard de descodificació; aquest aspecte es deixa en mans de la implementació del descodificador. Altres fluxos, com el vídeo comprimit en MPEG-2, requereixen la utilització de marques de descodificació, ja que com s'ha explicat anteriorment algunes unitats d'accés s'han de descodificar una estona abans de ser presentades. Per exemple, una imatge del tipus I o P s'haurà de descodificar abans de descodificar i presentar les imatges B anteriors que en depenen, però es presentarà després d'aquestes. En aquests casos, les marques de descodificació hauran d'anar acompanyades sempre d'una marca de presentació.

No és necessari incloure un *timestamp* per a totes les UA, ja que el descodificador coneix la cadència amb què ha de presentar-les. MPEG-2 especifica que ha d'aparèixer una marca temporal almenys cada 0.7 segons, en els fluxos de vídeo i àudio.

En quant a la seva estructura, són números de 33 bits referits al rellotge de sistema (*System Clock Reference (SCR)*), present en el codificador, empaquetador PES i multiplexor per sincronitzar els diferents fluxos que formen un programa, amb una resolució de 90 KHz. Per sincronitzar el rellotge del descodificador al rellotge de sistema d'un determinat programa s'utilitzaran els mecanismes que proporciona MPEG-2 en l'etapa de generació del *Transport Stream* o *Program Stream*.

A partir d'aquest punt, es poden generar els fluxos de programa (PS) i els fluxos de transport (TS). Aquests dos tipus de multiplexació estan dissenyats per a dos escenaris completament diferents. El *Transport Stream* és adequat en aplicacions on els errors i les pèrdues d'informació són presents, com per exemple els canals de comunicació sorollosos utilitzats per transmetre senyals en DVB. En canvi, els *Program Streams* estan dissenyats per a mitjans lliures d'errors, com els sistemes d'emmagatzematge DVD o medis de transmissió

fiables. A més, mentre els fluxos de transport permeten la multiplexació de diferents programes en un mateix *stream*, els fluxos de programa només poden contenir un únic programa, entenent per programa un únic flux de vídeo acompanyat d'un o més àudios, subtítols, teletext, etc.

2.2.3.2 Program Stream

Un *Program Stream*, com el seu nom indica, només pot contenir fluxos elementals d'un únic programa (amb un únic flux de vídeo), tots amb el mateix rellotge de referència. Els paquets PES dels diferents fluxos elementals s'agrupen formant els anomenats "*packs*".

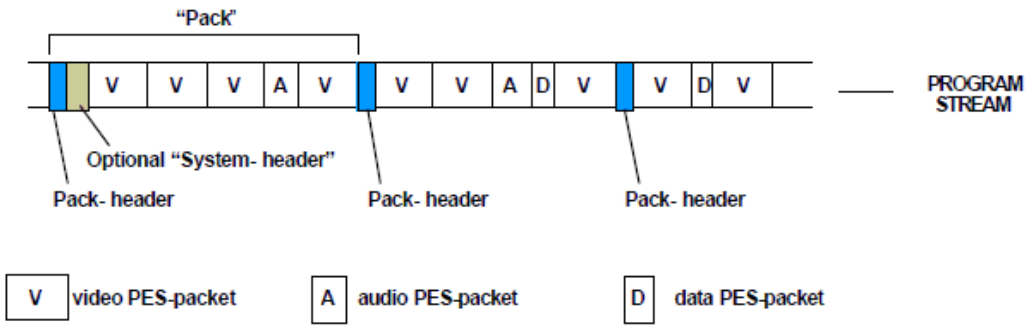


Figura 2.4 Estructura del *Program Stream*. Extret de [M2]

Com es pot observar en la figura 2.4, cada *pack* està format per una capçalera, una capçalera de sistema opcional, i un número indefinit de paquets PES procedents dels diferents fluxos elementals del programa. Així doncs, la longitud de cada *pack* dependrà el nombre de paquets PES que s'hi encapsulin; l'únic requeriment en quant a la seva mida és que ha d'aparèixer una capçalera almenys cada 0,7 s. Aquesta restricció és deguda a que en la capçalera hi viatja la referència del SCR (*System Clock Reference*), que permet al descodificador sincronitzar el seu rellotge intern amb el rellotge del flux, al que estan referits els *timestamps* de descodificació i presentació. Aquestes referències del rellotge de sistema són marques amb una resolució temporal de 27 MHz i expressades en números de 42 bits; els primers 33 bits corresponen al SCR_base i els altres 9 al SCR_extension.

D'altra banda, en la capçalera de sistema hi viatja informació relacionada amb les característiques del flux de programa, com per exemple el nombre de fluxos que el formen, el *bit rate*, etc.

Com ja s'ha dit, el *Program Stream* és adequat per escenaris lliures d'errors. Un dels principals motius és que cada *pack* conté en la seva capçalera informació de sincronisme relacionada amb tots els paquets PES que inclou. El fet d'utilitzar *packs* d'una longitud relativament gran, en què s'inclou un nombre indeterminat de paquet PES, implica que un error en la capçalera del paquet

afectarà a una quantitat de dades important. A més, com que els *packs* tenen una longitud variable, un error en el camp de la capçalera que indica la seva longitud provocarà la pèrdua de sincronisme amb el flux. Finalment, per poder adaptar el senyal a una determinada tecnologia de transmissió utilitzant un mecanisme de correcció d'errors eficaç és preferible disposar d'un flux amb paquets de mida constant i coneguda.

2.2.3.3 Transport Stream

Els *Transport Streams* han estat dissenyats específicament per transmetre un senyal MPEG-2 a través d'un canal de comunicacions que introdueix errors. A diferència del *Program Stream*, permet multiplexar diferents programes en un únic flux de transport, cadascun amb el seu propi rellotge de sistema, a partir d'ara rellotge de programa (*Program Clock Reference (PCR)*).

Els paquets PES s'encapsulen en paquets més petits, anomenats "paquets de transport" o "paquets TS". Aquests tenen una longitud fixa de 188 *bytes*, i estan formats per una capçalera, de 4 *bytes,* un camp d'adaptació de longitud variable i la càrrega útil. A l'hora de fragmentar els paquets PES i encapsular-los en paquets TS existeixen dues restriccions:

- El primer *byte* d'un paquet PES ha de correspondre al primer *byte* de la càrrega útil d'un paquet de transport.
- Cada paquet TS pot contenir informació d'un únic paquet PES.

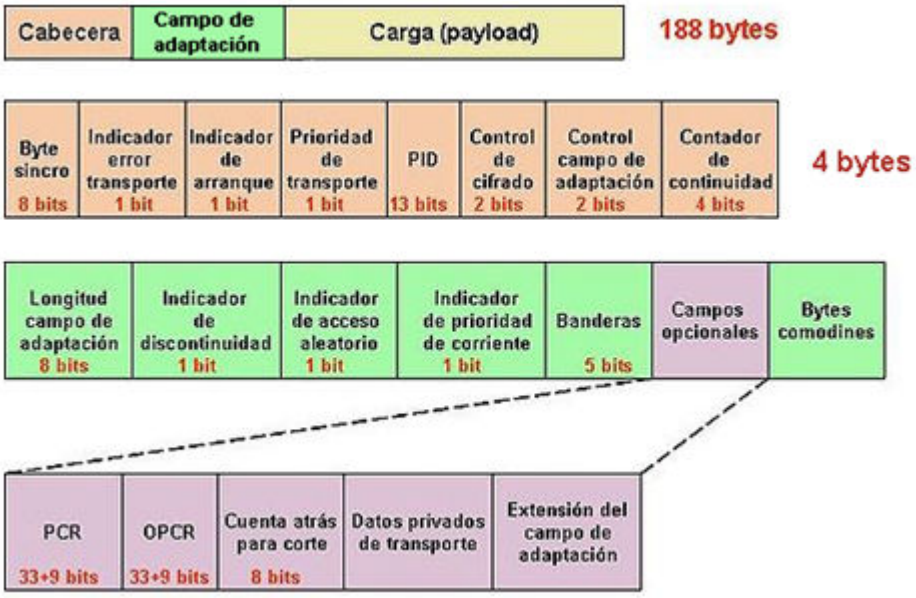


Figura 2.5 Estructura d'un paquet de transport. Extret de [W1]

Taula 2.1 Camps de la capçalera d'un paquet de transport

CAMP	DESCRIPCIÓ	BITS
sync_byte	*Byte* d'inici de paquet de transport. Té per valor 0x47. Es pot repetir dins el flux de transport, gràcies a la mida fixa dels paquets TS.	8
transport_error_indicator	Indica si s'ha produït algun error en el flux de transport, detectat més enrere.	1
payload_unit_start	Indica que el primer *byte* de càrrega útil correspon a l'inici d'un paquet PES.	1
transport_priority	Indicador de prioritat.	1
PID	Identificador del paquet de transport.	13
transport_scrambling_control	Tipus de xifrat de transport.	2
adaption_field_control	Indica si hi ha camp d'adaptació.	2
continuity_counter	Comptador de continuïtat, entre paquets amb el mateix PID.	4

Taula 2.2 Possibles valors del camp *adaption_field_control*

00	Reservat per a ús futur.
01	No hi ha camp d'adaptació, només càrrega útil
10	No hi ha càrrega útil, només camp d'adaptació
11	Hi ha camp d'adaptació seguit de càrrega útil

La figura 2.5 mostra l'estructura del camp d'adaptació. Com es pot observar, està format per dos *bytes* obligatoris i una sèrie de camps opcionals, amb informació addicional sobre el flux de transport associat a un determinat flux elemental. A més, també s'utilitza per omplir els paquets TS en cas que no hi hagi prou informació disponible. Com s'ha dit anteriorment, els paquets TS només poden contenir dades d'un únic paquet PES, i és poc probable que un paquet PES es pugui dividir en un número enter de fragments de manera que s'ompli el *payload* de tots els paquets TS. Una de les finalitats del camp d'adaptació és omplir amb *bytes* de farciment el paquet de transport per tal d'arribar als 188 *bytes* quan s'empaqueti el darrer fragment d'un paquet PES i no hi hagi prou dades. Aquest farciment es trobarà a continuació dels camps opcionals del camp d'adaptació.

Un dels camps opcionals més importants és el *PCR (Program Clock Reference*), que és el mecanisme que proporciona el flux de transport per tal de sincronitzar el rellotge del descodificador amb el rellotge de programa. Aquesta referència està formada per dos camps: *PCR Base* i *PCR extension*.

- PCR Base: Marca temporal del rellotge de programa amb una resolució de 90 KHz i expressada amb un número de 33 bits.
- PCR extension: Marca temporal del rellotge de programa, amb una resolució de 27 MHz i expressada amb 9 bits.

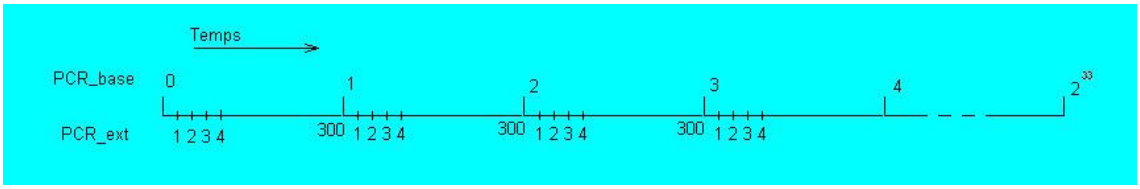


Figura 2.6 Relació entre PCR Base i PCR extension. Extret de [M4]

Les marques PCR indiquen l'instant d'arribada al descodificador del *byte* que conté el darrer bit d'aquesta referència. La figura 2.6 mostra com l'extensió del PCR s'utilitza per augmentar la resolució d'aquesta referència. En cada increment del PCR_base passaran 300 cicles del PCR_ext, reiniciant-se al canviar el valor del primer. El factor 300 correspon a la relació entre les resolucions de la base i de l'extensió. Les marques PTS/DTS dels fluxos PES tenen una resolució de 90 KHz, de manera que l'ús de l'extensió del PCR no és estrictament necessari.

Un *Transport Stream* pot contenir diferents programes, cadascun amb el seu propi rellotge de referència. Així doncs, cadascun dels programes haurà de tenir un flux en què hi viatgin les seves pròpies referències PCR, definit com a *PCR_PID*. Habitualment les marques PCR de cada programa viatgen en el camp d'adaptació dels paquets de transport que contenen el flux vídeo. Quan el TS arribi al descodificador, les marques PCR del programa seleccionat seran separades del flux sencer per tal de posar en fase el rellotge del descodificador amb el rellotge de programa, permetent una correcta sincronització entre el flux de vídeo i els fluxos d'àudio, subtítols, etc. El *transport rate* d'entrada d'un programa desmultiplexat al descodificador és constant entre dues marques PCR, però varia en cada punt del flux de transport en què apareix una referència PCR, ja que el temps transcorregut entre la posició de les dues marques ve determinat per la diferència entre els seus valors. D'altra banda, en un determinat punt del *Transport Stream* cada programa tindrà un *transport rate* diferent, tenint en compte que les marques de temps són independents per a cadascun dels diferents programes. Sota aquestes consideracions, l'estàndard explica que només es pot construir un *transport stream* amb un sistema de temporització consistent si aquest té un *bit rate* constant.

Es pot calcular l'instant d'arribada de qualsevol *byte* que es trobi entre dues referències PCR, utilitzant les següents expressions:

$$t(i) = \frac{PCR(i'')}{system\_clock\_frequency} + \frac{i - i''}{transport\_rate\,(i)} \qquad \textbf{(2.1)}$$

$$transport\_rate(i) = \frac{((i' - i'') \times system\_clock\_frequency)}{PCR(i') - PCR(i'')} \qquad \textbf{(2.2)}$$

on:
i: índex del *byte* del qual se'n vol calcular l'instant d'arribada
i'': índex del *byte* que conté el darrer bit de la referència PCR anterior
i': índex del *byte* que conté el darrer bit de la referència PCR posterior
PCR(i''): referència PCR immediatament anterior
PCR(i'): referència PCR immediatament posterior

$$i'' < i < i$$

Per calcular l'instant d'arribada del *byte* 'i', t(i), es sumarà a la referència PCR anterior (*PCR(i')*) el temps transcorregut fins l'arribada del *byte* en qüestió, utilitzant el *transport rate* del programa que s'està considerant, entre les dues marques PCR que emmarquen el *byte* del que se'n vol calcular el temps d'arribada. La velocitat de transport d'un programa dins el flux sencer entre dues marques PCR es calcula com el número de *bytes* que hi ha entre les dues referències (tenint en compte els *bytes* de tots els programes), entre la diferència dels seus valors; és a dir: número de *bytes* dividit entre la distància temporal entre ells, segons el rellotge del programa.

En quant a l'ordenació interna dels paquets de transport, no existeix cap restricció excepte que tots els paquets d'un mateix flux elemental apareguin en ordre cronològic.

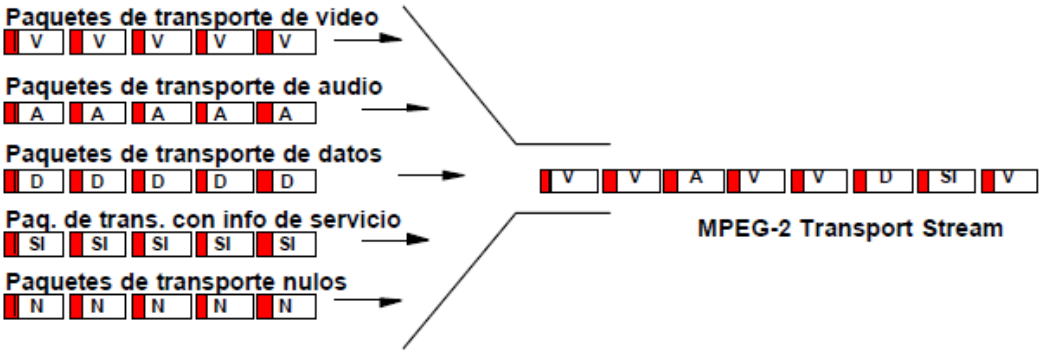


Figura 2.7 Entrellaçat dels paquets de transport. Extret de [M5]

L'estàndard ISO 13818 defineix l'STD (*System Target Decoder*) com a base per al disseny de descodificadors compatibles, i l'utilitza per descriure el sistema de temporització dels fluxos i especificar les restriccions que han de complir els *buffers* del descodificador per tal d'evitar situacions d'excés o manca d'informació. És responsabilitat del dissenyador del multiplexor que el *Transport Stream* de sortida no violi aquestes restriccions, detallades en l'annex B del present document.

A més dels paquets PES procedents dels diferents fluxos elementals, el *Transport Stream* contindrà l'anomenada *Program Specific Information (PSI)*, o Informació Específica de Programa, especificada per MPEG2 en l'estàndard ISO 13818-1. Aquesta informació, organitzada en taules, és necessària per tal que el descodificador pugui relacionar els diferents fluxos elementals amb el

programa o programes que composen. Les taules PSI es divideixen en seccions i s'encapsulen en paquets de transport, sense la necessitat d'encapsular-les prèviament en paquets PES. Existeixen quatre tipus de taules, dues de les quals dues són obligatòries i imprescindibles per a una correcta descodificació dels programes:

- *Program Association Table (PAT)*

És tracta d'una taula obligatòria i viatja en el PID 0x00. Conté la relació entre els programes que formen part del múltiplex i el PID en què viatja la taula PMT de cadascun d'ells.

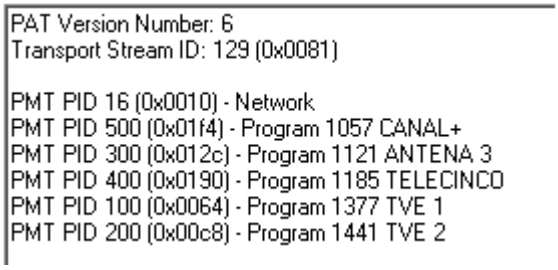


Figura 2.8 Exemple de taula PAT, extreta amb el TS Reader

.

- *Program Map Table (PMT)*

Cada programa del flux de transport ha de tenir associada una taula PMT. Aquestes poden viatjar en PIDs arbitraris, exceptuant els reservats per a algunes de les taules PSI/SI. Conté informació imprescindible per tal de descodificar un programa correctament:

- PID dels paquets de transport que contenen el PCR del programa
- PID dels paquets de transport que contenen el vídeo del programa
- PID's dels paquets de transport que contenen els àudios del programa
- PID's dels paquets de transport que contenen altres dades associades al programa (teletext, subtítols, etc).

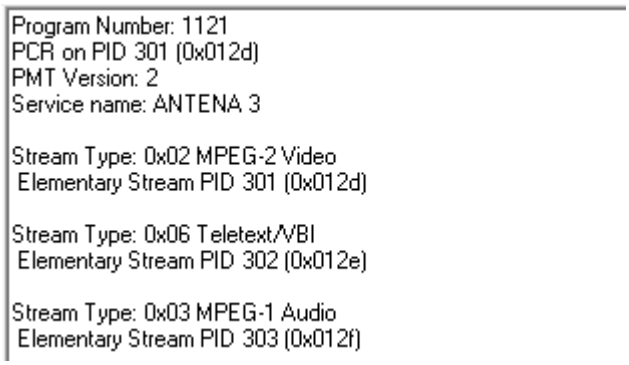


Figura 2.9 Exemple de taula PMT, extreta amb el TS Reader.

- *Conditional Acces Table (CAT)*

Ha d'estar present en el flux de transport si almenys un dels fluxos elementals és d'accés condicional. Si hi és, viatjarà en el PID 0x0001. Bàsicament conté informació sobre els mecanismes d'encriptació utilitzats, i indica els PIDs en què viatja la informació de control sobre l'accés condicional.

Com s'ha dit abans, en aplicació a DVB la informació PSI es complementa amb "Informació del Servei" (SI), definida per DVB-SI. De la mateixa manera que la informació específica de programa, la informació de servei també s'organitza en taules que es divideixen en seccions i viatgen encapsulades en paquets del flux de transport. Existeixen les següents taules:

Obligatòries:

- *Network Information Table (NIT)*

Conté informació sobre la xarxa física utilitzada en la difusió del *Transport Stream*, com per exemple la freqüència del canal, la modulació utilitzada, etc.

- *Service Description Table (SDT)*

Conté informació que descriu els serveis que viatgen en el *Transport Stream* (nom dels serveis, noms dels proveïdors, etc.)

- *Event Information Table (EIT)*

Conté informació relativa als esdeveniments que es produiran en el múltiplex; hi viatja la programació dels diferents programes.

- *Time & Date Table (TDT)*

Conté informació sobre l'hora i la data i s'utilitza per posar en hora el rellotge del descodificador.

Opcionals:

- *Running Status Table (RST)*

S'utilitza per actualitzar de forma ràpida la informació relativa a un determinat esdeveniment que està succeint en el múltiplex.

- *Time Offset Table (TOT)*

Proporciona informació sobre la data i l'hora real, així com la diferència horària local (*"local time offset"*).

- *Bouquet Association Table (BAT)*

S'utilitza per associar serveis proporcionats per un únic proveïdor i repartits en diversos fluxos de transport.

- *Stuffing Tables (ST)*

S'utilitzen per a invalidar taules que han quedat obsoletes. Viatgen en el PID reservat per la taula que cancel·len.

Taula 2.3 PID's reservats per a les taules PSI/SI

TAULA	PID
PAT	0x0000
CAT	0x0001
TSDT	0x0002
Reservats	0x0003 – 0x000F
NIT, ST	0x0010
SDT, BAT, ST	0x0011
EIT, ST	0x0012
RST, ST	0x0013
TDT, TOT, ST	0x0014
Sincronització de xarxa	0x0015
Reservats per futurs usos	0x0016 – 0x001D

CAPITOL 3. PRESENTACIÓ DEL MPLEX 13818

L'objectiu d'aquest TFC és obtenir una aplicació informàtica de codi obert capaç de generar fluxos de transport ISO 13818 que permetin la transmissió de programes audiovisuals. Per fer-ho, es parteix d'una aplicació ja existent: l'Mplex 13818 [W12], que en el moment de començar el treball es va considerar el millor candidat. Més endavant es va trobar el multiplexor *JustDVB-it*, també de codi obert i amb unes prestacions més avançades que l'Mplex. No obstant, com que ja s'havia començat a treballar amb el primer, es va decidir deixar el *JustDVB-it* com a base per a un segon TFC. La versió d'Mplex sobre la que es treballarà, la més recent a l'inici del treball, és la 1.1.3, publicada al febrer de 2005.

Mplex 13818 és un *software* de codi obert implementat per Oskar Schirmer, programador de sistemes de control i processos multi tasca, i de processos de tractament d'imatge en temps real. Està dissenyat per a treballar de forma ininterrompuda en un servidor destinat a difondre continguts audiovisuals. La primera versió del programa va ser escrita l'any 2001 per a *Convergence Integrated Media GmbH*, i finalment es va publicar com a codi obert sota una llicència pública general (GPL), el 2003. Aquest tipus de llicència, creada per la fundació *Free Software Foundation* l'any 1989, té com a objectiu promoure la lliure distribució, ús i modificació de *software*. Així doncs, amb Mplex 13818 com a base és possible aconseguir un programa d'ús lliure, modificant el seu codi sense problemes. Segons l'autor, el *software* té les següents característiques:

- Pot generar fluxos de transport i fluxos de programa seguint l'estàndard ISO 13818.
- Els fluxos d'entrada poden ser *program streams, packetized elementary streams* i *transport streams*, en qualsevol combinació.
- Els *streams* es mantenen sincronitzats durant la multiplexació, quan sigui possible.
- El programa pot operar en temps real basant-se en el sistema temporització dels *streams*, però també permet treballar en mode *untimed* per a realitzar conversions en sistemes de fitxers.
- La informació de programa pot canviar "al vol" durant el procés de multiplexat, sense interrompre el flux de sortida.
- Les taules de programa es poden manipular, afegint i traient descriptors en temps real.
- Inclou una eina addicional per generar taules SI (EN 300468).

Aquest TFC es centrarà principalment en les funcionalitats relacionades amb la multiplexació de fluxos de vídeo i àudio, i no s'entrarà a fons en l'anàlisi del tractament de la informació PSI/SI que, d'altra banda, s'ha comprovat que es genera correctament. Concretament, es tractarà el cas en que els fluxos d'entrada i de sortida són *Transport Streams*. En aquest sentit, s'ha comprovat que cap reproductor MPEG és capaç de reproduir correctament un flux de transport generat per la versió publicada d'Mplex 13818. Els esforços d'aquest treball es centraran en analitzar el codi d'Mplex i modificar-lo per a que funcioni correctament.

L'estudiant de la EPSC Daniel Guerrero va realitzar una activitat de suport a la docència per al departament d'Enginyeria Telemàtica de la UPC l'any 2005 [M3], en què s'estudia el comportament d'Mplex a partir de l'anàlisi de fluxos de transport generats amb aquesta aplicació. D'aquest document se n'extreuen dues conclusions importants que poden donar una primera visió de l'algoritme de multiplexació, i dels possibles problemes que presenta. Les proves es van realitzar amb el flux "*rete.ts*", capturat del múltiplex de Retevisión emés des del centre emissor de Collserola anomenat "RGN" (*Red Global de Cobertura Nacional*), a la freqüència de 770 MHz, al novembre de 2003, que en aquell temps transportava els programes de TVE i les 3 TV privades (A3, T5 i Canal+). Concretament, es va generar un flux de transport amb dos dels quatre programes d'aquesta captura: Telecinco (1185) i Antena 3 (1121).

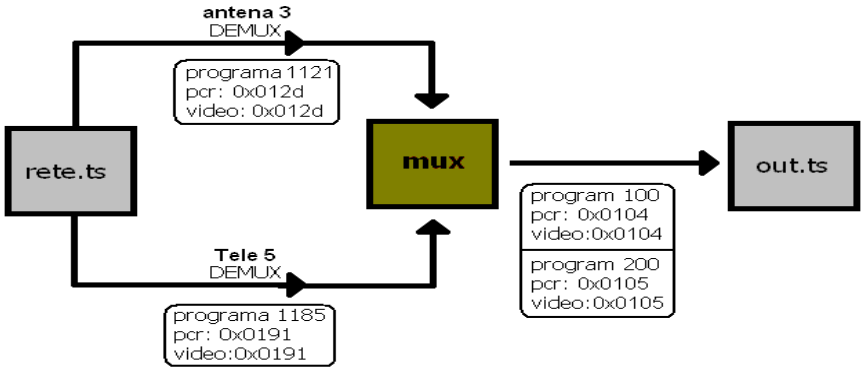


Figura 3.1 Escenari de proves. Extret de [M3]

En primer lloc es va analitzar l'estructura dels fluxos de sortida d'Mplex, concloent que tots els paquets de transport corresponents a un mateix paquet PES apareixen de forma consecutiva, generant ràfegues de tràfic per a un determinat ES. El primer paquet de transport de cadascun d'aquests blocs, que conté l'inici d'un paquet PES i per tant, generalment, d'una UA, inclou una referència al rellotge de programa, en el cas del *PCR_PID* (que en les proves realitzades correspon al flux de vídeo). A més, si un paquet PES d'entrada està marcat amb un *timestamp*, també ho està a la sortida. La informació associada a les taules PSI/SI també apareix en blocs.

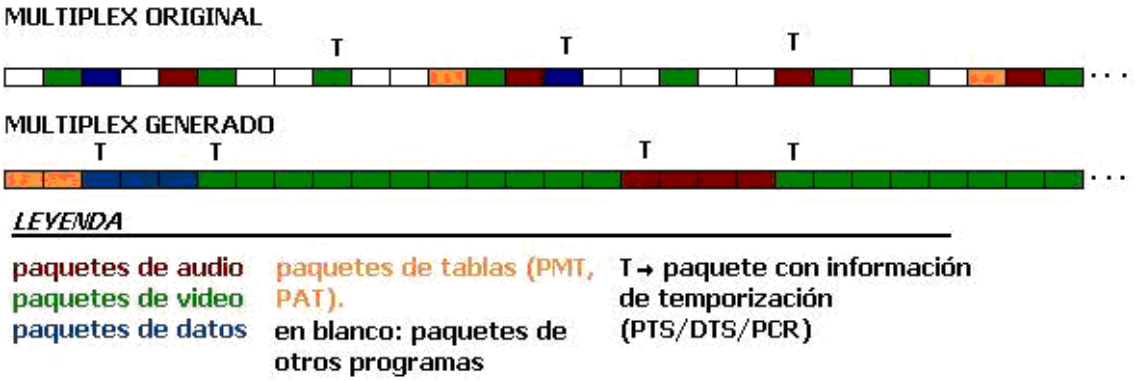


Figura 3.2 Estructura dels paquets TS en un flux d'Mplex. Extret de [M3]

D'altra banda, es va intentar descriure l'algoritme de temporització utilitzat pel programa. A causa de la seva complexitat, i de la relativament poca informació que pot proporcionar un flux de sortida, no es va aconseguir determinar quin mecanisme se segueix per a triar en quin moment s'ha de treure cada paquet PES a la sortida. No obstant, aquest anàlisi va servir per trobar un aspecte que des d'un principi s'ha tractat com un dels principals problemes a resoldre. Les figures 3.3 i 3.4 mostren informació de temporització (marques PTS/DTS i marques PCR) del flux amb PID 191, en el flux d'entrada, i del PID 102, en el flux de sortida. Tots dos corresponen al flux de vídeo del programa Telecinco, on hi viatja el PCR. Es mostren únicament els paquets de transport amb informació temporal, és a dir, paquets TS amb marques PCR i paquets TS on hi comença un paquet PES marcat amb un *timestamp*. Com es pot observar, el rellotge utilitzat com a referència per al programa en el flux de sortida és diferent del que tenia a l'entrada, però els *timestamps* associats a les UA d'aquests fluxos no es modifiquen i queden referits al rellotge vell. Així doncs, les marques que determinen l'instant en què s'han de descodificar/presentar les UA no estaran referides al rellotge de programa, la qual cosa provocarà problemes de sincronisme.

Paquete	PID	PCR	OPCR	PTS	DTS
1	191	2339370017913	0	0	0
130	191	0	0	2339376539400	0
458	191	0	0	2339380859400	2339371029000
490	191	2339371015471	0	0	0
980	191	2339372015073	0	0	0
1467	191	2339373008551	0	0	0
1499	191	0	0	2339378699400	0
1760	191	0	0	2339379779400	0
1960	191	2339374014269	0	0	0
2042	191	0	0	2339384099400	2339384099400
2448	191	2339375009789	0	0	0
2937	191	2339376007348	0	0	0
3094	191	0	0	2339381939400	0
3314	191	0	0	2339383019400	0
3431	191	2339377015108	0	0	0
3578	191	0	0	2339387339400	2339387339400
3919	191	2339378010626	0	0	0
4290	191	0	0	2339385179400	0
4407	191	2339379006146	0	0	0
4546	191	0	0	2339386259400	0
4824	191	2339379856824	0	2339390579400	2339380749000
4901	191	2339380013904	0	0	0

Figura 3.3 Temporització del flux amb PID 401 (vídeo del programa 1185), en el flux d'entrada. Extret de [M3]

Paquete	PID	PCR	OPCR	PTS	DTS
9	102	6156000	0	2339378699400	0
74	102	7182000	2339374014269	2339379779400	0
136	102	9153000	2339376007348	2339384099400	2339384099400
315	102	0	0	2339381939400	0
366	102	10179000	2339377015108	2339383019400	0
418	102	11178000	2339378010626	2339387339400	2339387339400
546	102	12150000	2339379006146	2339385179400	0
590	102	0	0	2339386259400	0
678	102	16173000	2339383008624	2339390579400	2339380749000
1027	102	17172000	2339384016382	2339388419400	0
1075	102	0	0	2339389499400	0
1139	102	19170000	2339386005380	2339393819400	2339393819400
1312	102	20169000	2339387004980	2339391659400	0
1354	102	0	0	2339392739400	0
1425	102	22167000	2339389014379	2339397059400	2339397059400
1569	102	23166000	2339390007857	2339394899400	0
1606	102	0	0	2339395979400	0
1671	102	25164000	2339392017256	2339400299400	2339390469000
1846	102	26163000	2339393012775	2339398139400	0
1893	102	0	0	2339399219400	0
1976	102	29160000	2339396007493	2339403539400	2339403539400

Figura 3.4 Temporització del flux amb PID 401 (vídeo del programa 1185), en el flux de sortida. Extret de [M3]

CAPÍTOL 4. ANÀLISI DE L'ALGORITME D'MPLEX 13818

4.1 Introducció

En aquest punt s'analitza el codi del *software* Mplex 13818 amb la finalitat de descriure l'algoritme de multiplexació que utilitza i treure conclusions sobre aspectes de temporització i d'ordenació de paquets. L'objectiu final d'aquest estudi és identificar les deficiències en la implementació del multiplexor i proposar possibles millores.

El codi d'Mplex està escrit en llenguatge C, i és adequat per a la seva compilació en màquines amb sistema operatiu Linux/Unix. S'organitza en diferents mòduls, cadascun dels quals és un bloc de codi escrit en un arxiu *.c* diferent, amb la corresponent capçalera (arxiu *.h*), i proporciona una sèrie de funcions destinades a implementar cada una de les diferents etapes del procés de multiplexat. A més dels mòduls, també hi ha altres arxius de codi on s'hi defineixen variables i constants simbòliques, que són usades en la resta de codi. Per exemple, a l'arxiu de capçalera *ts.h* s'hi defineixen, entre altres, els PID's estandarditzats en l'iso13818 (taula PAT, CAT, paquet nul, etc), mitjançant ordres del tipus *#define* per al preprocessador de C. El fitxer *global.h* conté la definició de totes les estructures de dades que s'utilitzen en el programa; *global.c* proporciona funcions de caràcter general. En total, el codi està format per 22 fitxers de codi C i 23 fitxers de capçalera.

Per començar l'anàlisi, es farà una breu descripció dels diferents mòduls, explicant el paper de cadascun d'ells dins el procés de multiplexació; a continuació, es descriuran les estructures de dades més importants que utilitza el programa, i es mostrarà l'esquema de *buffers*; finalment es presentaran les conclusions que s'han obtingut a partir de l'anàlisi a fons del codi i de les proves realitzades, que es descriuen detalladament als annexos C i D.

4.2 Descripció dels Mòduls

El codi del Mplex 13818 està dividit en sis mòduls: *Command, Dispatch, Input, Split (TS, PS, i PES), Splice (TS i PS)* i *Output*. Més endavant, en l'apartat 4.4, s'explica la relació funcional entre ells. Els mòduls *Split* i *Splice* estan formats per més d'un arxiu de codi (amb les corresponents capçaleres), però es consideren un únic mòdul. S'utilitzen les funcions d'un fitxer o d'un altre segons el tipus de flux d'entrada (*Transport Stream, Program Stream* o flux PES), i del flux de sortida (TS o PS). A continuació es descriuen els diferents mòduls, explicant la finalitat amb què s'ha dissenyat cadascun d'ells.

4.2.1 Command

Aquest mòdul proporciona la interfície entre l'usuari i el *software*. S'encarrega de recollir les comandes introduïdes pel teclat i passar-les a la resta de mòduls

per tal que actuïn en conseqüència. L'objectiu d'aquest estudi és entendre l'algoritme de multiplexació que utilitza Mplex, i no tan com interactua amb l'usuari, de manera que no s'entrarà en l'anàlisi de *Command* a nivell de codi.

4.2.2 Dispatch

És el mòdul que conté el bucle principal del programa, i s'encarrega de repartir la feina entre la resta de mòduls. La seva tasca consisteix en determinar en quins punts es criden les funcions dels altres mòduls, sempre amb l'objectiu que les dades avancin, des dels *buffers* d'entrada fins al de sortida, passant pel procés d'extracció i d'empalmament. En general, la seqüència que se segueix en el bucle és la següent: cada cop que es vol moure dades d'un *buffer* cap al següent, es comprova la disponibilitat de dades en el *buffer* origen i d'espai en el *buffer* destí, mitjançant funcions específiques del mòdul que conté aquests *buffers*, i es crida la funció que s'encarrega d'agafar les dades del primer, processar-les, i emmagatzemar-les en el segon. El bucle principal continua mentre hi hagi dades per ser transmeses al flux de sortida.

4.2.3 Input

S'encarrega d'adquirir i organitzar la informació procedent de les diferents fonts d'entrada, i preparar-la per ser encadenada al flux de sortida. Està format per dues estructures principals: una llista de fitxers oberts, i una llista *d'elementary streams* oberts (ES procedents de les diferents fonts d'entrada i que s'estan traient en el TS de sortida). La llista de fitxers consisteix en un punter a una estructura '*file_descr*', la qual conté un *buffer* on s'hi emmagatzemen les dades llegides i una sèrie de camps on es guarda informació sobre la font. Aquests són els anomenats *raw buffers*, que contenen la informació tal i com es llegeix, sense processar-la. D'altra banda, la llista d'ESs és un punter a una estructura '*stream_descr*', que conté les dades corresponents a un determinat flux elemental, en format PES; són els anomenats *input buffers*. Més endavant s'explicaran amb més profunditat les principals estructures de dades que utilitza el programa.

El mòdul *Input* conté funcions per obrir i tancar fonts d'entrada i ESs, per comprovar la disponibilitat d'espai i d'informació als diferents *buffers* que en formen part, i per llegir dades de les fonts d'entrada i desar-les en els *raw buffers*. Per extreure informació d'un determinat PID d'un *raw buffer* cap al *input buffer* corresponent, s'ajuda del mòdul *Split* (TS, PS o PES, segons el tipus de flux d'entrada). D'altra banda, també s'encarrega de comprovar si la informació continguda en els *input buffers* dels diferents ES està preparada per ser empalmada en el flux de sortida, segons les condicions internes del *buffer* (bàsicament, disponibilitat d'una mínima quantitat d'informació), i de determinar en quin moment es traurà cada paquet PES de cada flux elemental. Com s'ha explicat anteriorment, tot i no existir cap restricció en quant a l'encapsulament d'unitats d'accés en paquets PES, habitualment un paquet PES correspondrà a una unitat d'accés: una imatge de tipus I, P o B en el cas de vídeo, o un fragment d'àudio. L'algoritme que s'utilitza per determinar quan s'ha de treure

cada paquet PES es basa en que la distància temporal entre els paquets PES d'un determinat flux elemental sigui la mateixa en el TS de sortida que en el d'entrada.

4.2.4 Split (TS, PS, PES)

És el mòdul encarregat d'extreure la informació dels diferents fluxos elementals i disposar-los, en coordinació amb el mòdul *Input*, en els *input buffers* corresponents. Les dades que es copien als *buffers* dels ES corresponen al *payload* dels paquets del flux original, és a dir, són fragments de paquets PES; no es copien les capçaleres dels paquets TS ni PS, ni el camp d'adaptació. Tota la informació de les capçaleres originals es disposa en diferents camps de l'estructura '*stream_descr'* que conté l'ES. Segons el tipus de flux origen, s'utilitzen funcions del fitxer *SplitTS, SplitPS* o *SplitPES*. Aquest estudi se centrarà en el cas en què els fluxos d'entrada són *Transport Streams*. D'altra banda, també extreu, emmagatzema i analitza la informació de les taules PSI (PAT i PMT, no processa taules CAT) i SI.

4.2.5 Splice (TS, PS)

Aquest mòdul és l'encarregat de generar el flux de sortida empalmant els paquets PES dels diferents fluxos elementals d'entrada. Conté funcions que es poden utilitzar independentment del tipus de flux desitjat, ja que internament crida les funcions del fitxer *SpliceTS* o *SplicePS*, segons calgui. Aquest estudi es centrarà en el comportament del multiplexor quan el flux de sortida és un *Transport Stream*, de manera que únicament s'analitzarà el codi del fitxer *SpliceTS*.

Bàsicament, *SpliceTS* conté funcions per encapsular els paquets PES continguts en els diferents *input buffers* en paquets de transport, generant la capçalera i el camp d'adaptació d'aquests. Els paquets TS els disposa, en coordinació amb el mòdul *Output*, en el *buffer* de sortida. Com ja s'ha dit, l'encarregat de seleccionar en quin moment s'encapsula cada paquet PES és el mòdul *Input*. En el moment en què *Input* determina que ha arribat el moment de treure un determinat paquet, passa la referència de l'estructura *'stream_descr'* que conté aquest paquet al mòdul *Dispatch*, i aquest la passa a *Splice*. D'altra banda, també s'encarrega de manejar els programes de sortida, lligant els fluxos elementals al programa corresponent, i de generar les taules PSI (PAT i PMT dels diferents programes).

4.2.6 Output

Aquest mòdul té la funció d'emmagatzemar el flux generat per *Splice*, en l'anomenat *Output buffer*, i de treure'l a la sortida del multiplexor. Conté funcions per comprovar la disponibilitat d'espai en el *buffer*, per reservar-hi l'espai necessari, quan el mòdul *Splice* ho demani, i de comprovar la disponibilitat de dades i treure-les a la sortida.

4.3 Descripció de les principals estructures de dades

A continuació es descriuen les principals estructures de dades amb què treballa el programa, definides al fitxer *global.h*. Les més importants són *file_descr*, *stream_descr* i *prog_descr*, que descriuen fonts d'entrada, fluxos elementals, i programes de sortida, respectivament.

4.3.1 Descriptor d'un fitxer (*file_desc*)

```
/* Source file */
typedef struct {
  refr_data data;
  int handle;
  char *name;
  int filerefnum;
  int st_mode;
  struct pollfd *ufds;
  int skipped; /* undesired bytes skipped, total */
  int payload; /* split payload used total */
  int total; /* split total (skipped, used, wasted) */
  int sequence; /* source counter for PES sequence */
  short openstreams[number_sd];
  [...]
  content_type content;
  union {
    struct {
      struct streamdescr *stream;
    } pes;
    struct {
      struct streamdescr *stream;
      [...]
    } ps;
    struct {
      struct streamdescr *stream;
      [...]
    } ts;
  } u;
} file_descr;
```

Com s'ha comentat anteriorment, l'estructura *file_descr* és l'encarregada d'emmagatzemar la informació de cada una de les diferents fonts d'informació obertes. Està declarada al mòdul *Input*, on hi ha una llista de fitxer oberts, que correspon a un vector d'estructures d'aquest tipus.

L'estructura està formada per l'anomenat *raw buffer*, on emmagatzema les dades tal i com es llegeixen, i conté una sèrie de camps amb informació sobre la font i sobre el flux original. El *buffer* de dades és una estructura del tipus *refr_data*, definida de la següent manera:

```
/* Data buffer */
typedef struct {
  byte *ptr;
  int in;
  int out;
  int mask;
} refr_data;
```

Com es pot veure, conté un punter **ptr*, que apunta a la posició de memòria on comença el *buffer*, i dues variables *in* i *out* que indiquen la posició d'entrada i de sortida, respectivament, entenent que la sortida del *buffer* conté la informació més antiga, la que fa més estona que ha entrat. La variable *mask* agafa com a valor la mida màxima del *buffer*, especificada en el moment de crear-lo en el mòdul *Input* (quan s'obre la font) i s'utilitza, entre altres coses, per calcular l'espai disponible. Tots els *raw buffers* tenen una mida fixa de 262144 bytes (2^{18}).

El programa pot treballar amb diferents fonts d'informació obertes simultàniament; l''estructura *file_descr* conté variables per identificar-les.

```
int handle;
char *name;
int filerefnum;
```

Per referir-se internament a una determinada font, s'utilitza la variable *filerefnum*, un enter assignat aleatòriament en el moment d'obrir el fitxer. Aquesta variable identifica cada una de les estructures *file_desc*, però no la font en sí. La variable *handle* és el descriptor de fitxer, que identifica la font d'informació associada a una estructura *file_desc*. La cadena de caràcters *name* s'utilitza per desar el nom del fitxer.

D'altra banda, conté una estructura del tipus *pollfd,* anomenada *ufds*. Aquest tipus d'estructura, definida a la llibreria *poll.h*, defineix un descriptor de fitxer mitjançant un identificador, que correspon amb el valor de *handle*, i conté una sèrie de màscares utilitzades per especificar condicions de lectura i escriptura, utilitzades en la funció *poll()*. La llibreria *poll.h* s'explica en l'apartat 5.1 de l'annex C.

Les variables *total, payload* i *skipped*, de tipus enter, s'utilitzen per comptar els *bytes* totals que han estat llegits, els que han estat extrets, i els descartats, respectivament. Els *bytes* poden ser descartats, per exemple, a l'inici del procés d'extracció d'un determinada font d'entrada, en que s'ignoraran tots els paquets TS fins que es trobi l'inici d'un paquet PES. La variable *sequence* és un comptador dels paquets PES que s'han extret d'aquest flux d'entrada; el comptador és únic per a tots els PIDs, de manera que els número de seqüència dels paquets PES d'un determinat flux elemental no seran consecutius.

La variable *openstreams* indica el nombre de fluxos elementals procedents d'aquesta font "oberts", és a dir, el nombre d'ESs que s'estan extraient d'aquest flux d'entrada, i que són usats en el flux de sortida.

```
short openstreams[number_sd];
```

Number_sd especifica el tipus de contingut d'un *stream_desc* (dades, taules PSI o taules SI), d'acord amb l'enumeració *streamdata_type* definida a *global.h*. Cadascuna de les tres posicions del vector *openstreams* indica el numero d'estructures *stream_descr* d'un determinat tipus associades a aquesta font d'entrada.

La variable *content*, del tipus *content_type*, que també és una enumeració definida a *global.h*, determina el tipus de flux origen. Els valors possibles són *ct_packetized, ct_program* i *ct_transport,* indicant que es tracta d'un flux PES, PS o TS, respectivament.

Fins ara, les variables que s'han explicat són comunes sigui quin sigui el tipus de flux d'entrada. L'estructura *file_descr* conté tres estructures dins seu, anomenades *pes, ps* i *ts,* amb camps específics que s'utilitzen segons el tipus de flux. Totes tres tenen en comú un vector que apunta a les estructures *stream_desc* que contenen els fluxos elementals extrets d'aquesta font.

```
streamdescr *stream[MAX_STRPERPS];
```

Cada un dels fluxos elementals oberts i procedents d'aquesta font *file_desc*, estarà apuntat per la posició *sid* del vector **stream[]*, on *sid* serà el PID del ES en el flux original. Si el flux d'entrada és un TS, la posició zero del vector apuntarà al *stream_descr* que conté la taula PAT d'entrada, i les taules PMT dels diferents programes estaran apuntades per l'*sid* corresponent al PID on viatgin en el flux original. A més, la posició *TS_UNPARSED_SI,* definida a *ts.h*, apunta a una estructura *stream_descr* on s'emmagatzema la informació SI del flux d'entrada que encara no ha estat processada.

4.3.2 Descriptor d'un flux elemental (*stream_descr*)

```
/* Single data or map stream */
typedef struct streamdescr {
  refr_ctrl ctrl;
  refr_data data;
  file_descr *fdescr;
  short sourceid;   /* index into fdescr->u.xx.stream[] */
  byte stream_id;   /* elementary stream id, table 2-35, etc */
  byte stream_type;  /* table 2-29 */
  [...]
  streamdata_type streamdata;
  union {
    struct {
      short pid; /* splicets: 0010..1FFE, spliceps: ...FF */
      boolean discontinuity;
      boolean trigger;
      boolean mention;
      boolean has_clockref; /* in output */
      boolean has_opcr; /* in input */
      struct streamdescr *mapstream;
```

```
      t_msec next_clockref;
      t_msec delta;
      conversion_base conv;
      t_msec lasttime;
      short progs;
      prog_descr *pdescr[MAX_PRGFORSTR];
    } d;
    struct {
      t_msec msectime;
      conversion_base conv;
      int psi_length;
      byte psi_data[MAX_PSI_SIZE+TS_PACKET_SIZE];
    } m;
    struct {
    } usi;
  } u;
} stream_descr;
```

L'estructura *stream_descr* té com a finalitat emmagatzemar informació d'un determinat *elementary stream*. De la mateixa manera que l'anterior, està declarada en el mòdul *Input*, que conté un vector amb els diferents fluxos elementals oberts. Està formada per un *buffer*, anomenat *Input buffer*, i una sèrie de camps que contenen informació sobre el flux elemental.

En aquest cas, el *buffer* no és únicament un punter a la posició de memòria amb els corresponents índexs, sinó que a més, existeix un *buffer* de control utilitzat bàsicament per dividir la informació del *buffer* de dades en blocs, cadascun dels quals correspondrà a un paquet PES, i per emmagatzemar informació associada a cadascun d'aquests paquets.

```
/* Control buffer */
typedef struct {
  ctrl_buffer *ptr;
  int in;
  int out;
  int mask;
} refr_ctrl;
```

```
typedef struct {
  int index;
  int length;
  int sequence;
  t_msec msecread;
  t_msec msecpush;
  clockref pcr;
  clockref opcr;
  byte scramble;
} ctrl_buffer;
```

El *buffer* de control té una estructura similar al de dades, amb la diferència de que cadascuna de les posicions del vector **ptr* no és simplement un *byte*, sinó una altra estructura anomenada *ctrl_buffer*, que és la que conté la informació relacionada amb un determinat bloc del *buffer* de dades. Aquesta estructura de *buffers* (*buffer* de dades i *buffer* de control) no només es troba en els *stream_descr*, sinó que també és utilitzada en el mòdul *Output*, en què es declara directament una variable del tipus *refr_data*, i una del tipus *refr_ctrl*, les quals formen el *buffer* de sortida.

La variable *index* indica la posició del *buffer* de dades on comença el bloc al que fa referència, essent length la longitud del bloc. *Sequence* indica el número de seqüència del paquet PES. A cada paquet PES extret d'una font d'entrada se li assigna un número de seqüència consecutiu, independentment de quin sigui el seu PID, de manera que els números de seqüència de cada flux elemental no seran consecutius. La taula 4.1 il·lustra aquest procés amb exemple senzill.

Taula 4.1 Assignació dels números de seqüència.

PID	Núm. Seqüència PES
301	1
203	2
101	3
402	4
301	5

La mida dels *input buffers* és fixa i es defineix en funció del tipus de dades que contenen. Per als fluxos de vídeo, el *buffer* de dades pot emmagatzemar fins a 1048576 bytes (2^{20}). En la resta de fluxos d'entrada (àudio, taules PSI, i altres), la mida del *buffer* de dades és de 524288 bytes (2^{19}). En tots els casos, el *buffer* de control pot desar informació associada a 1024 blocs (paquets PES). Curiosament, la mida dels *input buffers* dels diferents PIDs és més gran que la del *raw buffer* (que conté dades de diferents fluxos elementals), que només pot admetre 262144 bytes. D'això se'n pot deduir que el procés d'extracció de la informació és força ràpid, mentre que les dades han d'estar-se en l'*input buffer* fins que arribi el moment de treure-les a la sortida.

D'altra banda, apareixen variables que fan referència a aspectes de temporització. *PCR* i *OPCR* contenen la marca PCR del paquet de transport que conté l'inici del paquet PES en el flux de sortida i en el d'entrada, respectivament. *Msecread* i *msecpush* són dues variables clau per determinar en quin moment s'ha de treure cada paquet PES a la sortida, i seran explicades en detall més endavant. De moment, mencionar que a *msecread* se li assigna el valor del rellotge intern del multiplexor quan s'ha acabat d'extreure el paquet PES, en milisegons, i *msecpush* agafa el valor de la darrera mostra PCR extreta del programa al que pertany l'ES en el flux original, també en milisegons. Finalment, la variable *scramble* s'utilitza per indicar si la informació que conté el paquet PES està encriptada.

Per entendre una mica millor la relació entre el *buffer* de dades i el de control, es presenta el següent exemple. Sigui *stream_descr* *s. Es vol apuntar a l'inici del paquet PES més antic del *buffer* de dades. Per fer-ho, és necessari conèixer la posició del *byte* que conté l'inici d'aquest paquet, i aquesta informació la proporciona la variable *index* del bloc de control associat al paquet PES al que es vol accedir. Interessa apuntar al paquet més antic, i la posició del *buffer* de control que fa referència a aquest bloc de dades és *ctrl.out*. Per tant, per accedir al paquet PES més antic del *buffer* de dades, s'ha d'escriure:

```
s->data.ptr[s->ctrl.ptr[s->ctrl.out].index]
```

on *s->ctrl.ptr[s->ctrl.out].index* és l'índex que apunta al primer *byte* del bloc més antic del *buffer* de dades.

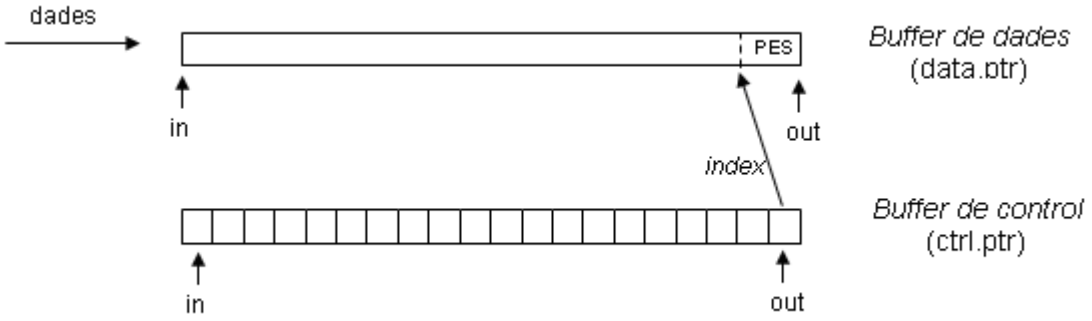


Figura 4.1. Relació entre el *buffer* de control i el *buffer* de dades

Un cop s'ha entès l'estructura i el funcionament dels *buffers* de control i de dades que defineixen els i*nput buffers* (i el *buffer* de sortida), s'explicaran els principals camps de l'estructura *stream_desc*. De la mateixa manera que cada *file_descr* apunta a tots els *stream_descr* (fluxos elementals) que té oberts, cada *stream_descr* apunta al *file_descr* al que pertany, mitjançant el punter **fdescr*. Com s'ha explicat anteriorment, el *file_descr* apunta als diferents *stream_descr* mitjançant el vector de punters **stream[sid]*. En l'estructura *stream_descr* hi ha una variable anomenada *sourceid*, que indica la posició que ocupa l'*stream* dins el vector del *file_descr* al que pertany (**stream[sourceid]*). El *sourceid* correspon al PID en el flux d'entrada.

D'altra banda, la variable *streamdata*, del tipus *streamdata_type*, que és una enumeració definida a *global.h*, indica si l'*stream_desc* conté informació encapsulada en un flux PES (dades de vídeo, àudio, teletext, etc), taules PSI, o taules SI. La variable *stream_id* identifica el flux elemental, i *stream_type* defineix el tipus d'informació que viatja encapsulada dins el flux PES (vídeo MPEG2, àudio, etc), d'acord amb la taula 2-18 del iso13818-1.

Dins de l'estructura *stream_descr*, hi ha tres estructures anomenades *d*, *m* i *usi*. La primera conté camps que s'utilitzen en cas que l'*stream* contingui dades (*data*), és a dir, informació encapsulada en paquets PES. La segona, en cas que contingui informació PSI (*mapstream*), i la tercera si conté informació SI no analitzada (*unparsed SI*). Aquesta última està buida, i no s'hi fa referència en cap altra part del codi; possiblement està preparada per a futures ampliacions. A continuació es descriuen els diferents camps de les estructures *d* i *m*.

- **data**

struct {

short pid;	PID del *elementary stream*, al flux de sortida
boolean discontinuity;	Indicador de discontinuïtat
boolean trigger; /* */	Indica que l'ES està preparat per ser empalmat a la sortida.
boolean mention;	Indica que l'ES ha estat preparat per ser empalmat a la sortida en algun moment.
boolean has_clockref;	Indica que l'ES porta marques de rellotge en el flux de sortida.
boolean has_opcr;	Indica que l'ES porta marques de rellotge en el flux d'entrada.
struct streamdescr *mapstream;	Punter al *stream_descr* que conté la taula PMT del programa del flux original al que pertany aquest ES.
t_msec next_clockref;	Si *has_clockref,* indica el proper cop que, com a molt tard, s'ha de treure una referència de temps a la sortida, tenint en compte la màxima distància entre referències PCR segons l'estàndard.
t_msec delta;	Diferència temporal entre el rellotge intern i el rellotge del programa d'entrada, en ms.
conversion_base conv;	Base de conversió usada per passar d'unitat de temps a referència de 90KHz.
t_msec lasttime;	*Timestamp* de sortida del proper paquet PES que s'ha de treure (el més antic).
short progs;	Nombre de programes de sortida que contenen aquest ES.
prog_descr *pdescr[];	Punters a les estructures *prog_descr* dels programes on apareix l'ES.

} d;

- **mapstream**

struct {

t_msec msectime;	Darrera mostra PCR extreta del programa d'entrada, en milisegons.
conversion_base conv;	Base de conversió usada per passar d'unitat de temps a referència de 90KHz.
int psi_length;	Longitud de la informació PSI del programa.
byte psi_data[];	Taula PMT del programa (del flux d'entrada)

} m;

Cada cop que s'acaba d'extreure un paquet PES d'un determinat PID i d'un determinat flux d'entrada, es comprova a quin programa d'entrada pertany, mitjançant el punter **mapstream* de l'estructura *d* del *stream_descr* que el conté. A la variable *msecpush* del paquet que s'acaba d'extreure, se li assigna el valor de *msectime* del *mapstream* en qüestió.

4.3.3 Descriptor d'un programa (prog_descr)

```
/* Target program */
typedef struct {
  int program_number;
  short pcr_pid;
  short pmt_pid;
  byte pmt_conticnt;
  byte pmt_version;
  boolean changed; /* must generate new psi due to change */
  boolean unchanged; /* must generate new psi due to timing */
  short pat_section;
  short streams;
  struct streamdescr *stream[MAX_STRPERPRG];
  stump_descr *stump;
  descr_descr manudescr;
} prog_descr;
```

El TS de sortida del multiplexor estarà format per diversos fluxos elementals, cadascun dels quals formarà part d'un o més programes. A més, cadascun dels programes contindran un o més fluxos elementals. Per manejar els programes de sortida, Mplex utilitza estructures del tipus *prog_descr*. Aquesta estructura conté informació específica del programa, i el lliga als ES que el formen mitjançant el vector de punters **stream[]*. Està declarada en el mòdul *Splice*, que conté la llista de programes de sortida.

L'estructura conté variables que proporcionen informació com el número de programa dins el TS de sortida, el PID on viatgen el PCR i la taula PMT del programa, el número de versió de la taula PMT, i el nombre d'ES que conté, entre altres.

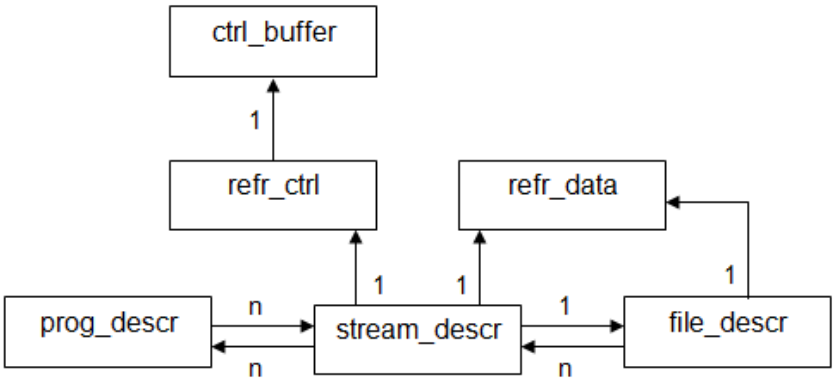


Figura 4.2. Relació entre les principals estructures de dades

4.4 Esquema funcional. Funcions principals

En aquest apartat es pretén descriure l'estructura de *buffers* del multiplexor, així com el punt en què actuen les principals funcions dels diferents mòduls, basant-se en la descripció dels mòduls i de les estructures de dades, i en l'anàlisi de les funcions que es presenta en l'annex C.

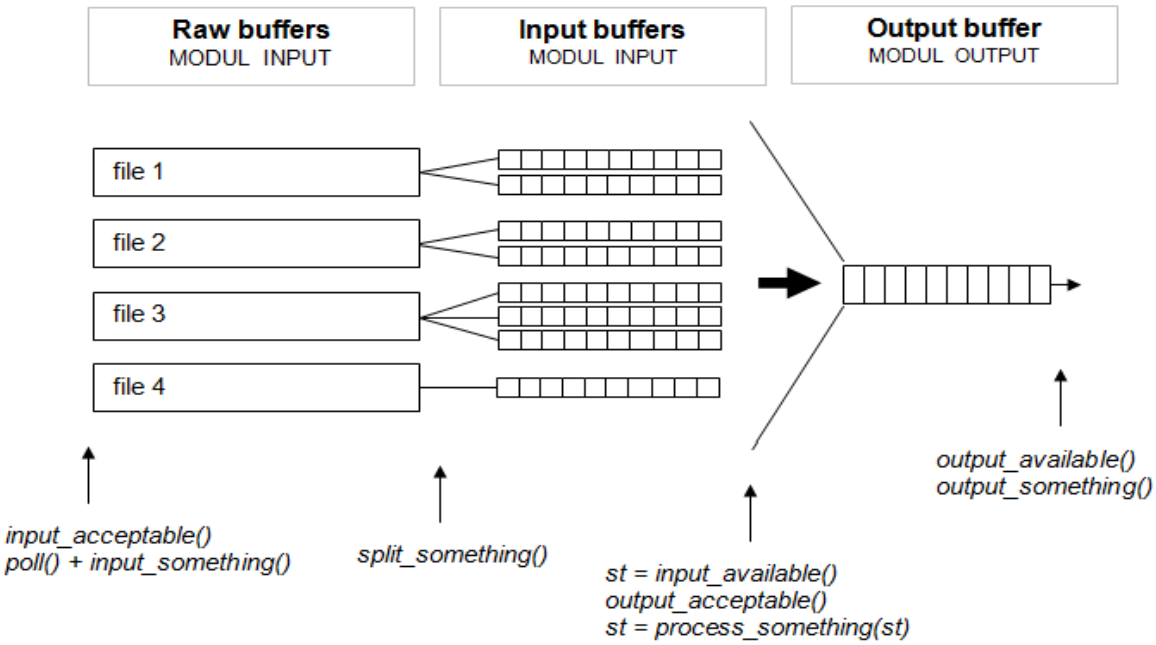


Figura 4.3. Esquema funcional del multiplexor

En la figura 4.3 es poden observar els *buffers* amb què treballa el programa, dibuixats d'esquerra a dreta en l'ordre en què viatgen les dades, i algunes de les funcions utilitzades en cada punt. Aquest diagrama no mostra l'ordre en que es realitzen les diferents etapes del multiplexat en el bucle principal, o dit d'una altra manera, les funcions no es criden en l'ordre en que estan representades al diagrama; l'esquema únicament pretén mostrar de forma visual l'estructura de *buffers* i sobre quin punt actua cada funció i mòdul. En l'annex C s'analitza el diagrama del flux del bucle principal, on es veu la seqüència de passos del procés de multiplexació.

Com es pot observar, el nom de les funcions de cada mòdul comença precisament amb el nom del mòdul al que pertanyen, fet que ajuda a identificar ràpidament sobre quin *buffer* estan actuant, amb excepció de la funció *process_something()* que pertany al mòdul *Splice*. Hi ha una sèrie de funcions comuns en els diferents mòduls: *_acceptable* i *_available*, en els mòduls *Input* i *Output*, i *_something* en tots ells (*Input, Split, Splice* i *Output*). La primera, s'utilitza per comprovar la disponibilitat d'espai en els *buffers*, ja sigui en el *raw buffer* en el cas del mòdul *Input* (*input_acceptable)*, o en el *buffer* de sortida, en el mòdul *Output* (*output_acceptable)*. La segona té com a funció determinar si hi ha informació disponible al *buffer* (*input buffer* o *output buffer*). Noti's que en

el cas d'*Input*, es comprova la disponibilitat d'espai a "l'entrada" del mòdul (*raw buffer*), i la disponibilitat d'informació a la sortida (*input buffer*). És el mòdul *Split* qui s'encarregarà de comprovar la disponibilitat d'informació als diferents *raw buffers*, i d'espai als *input_buffers*, en el moment en que intenti extreure dades d'un determinat PID. Cal tenir en compte que la disponibilitat d'informació en un *input buffer* no és pot determinar únicament comprovant si hi ha dades, ja que els paquets PES només es poden treure en un *timestamp* determinat. La funció *input_available()* tornarà un punter al ES que conté el paquet PES que s'ha de treure en cada moment. Tornarà un punter nul en cas que no hi hagi cap paquet per treure en una determinada passada de bucle.

La tercera, *_something*, s'encarrega de fer anar les dades "endavant", cap al següent *buffer*, després de tractar-les adequadament. La funció *input_something()* llegeix informació de les diferents fonts d'entrada i l'emmagatzema en els *raw buffers*. S*plit_something()* extreu el *payload* dels paquets dels fluxos d'entrada i disposa les dades en l'*input_buffer* corresponent. *Process_something()* s'encarrega d'encapsular la informació dels *input_buffers* en paquets TS (o PS, segons el flux de sortida seleccionat), i disposar-los en el *buffer* de sortida. I finalment, *output_something()* s'encarrega de treure la informació del *buffer* de sortida cap a la sortida del multiplexor.

En general, abans de fer anar les dades endavant, es comprova tant la disponibilitat de dades en el *buffer* origen com la disponibilitat d'espai en el destí, excepte en el procés d'extracció d'informació d'un PID, del *raw buffer* cap al *input_buffer*, en què com ja s'ha dit és la funció *split_something()* qui s'encarrega de realitzar aquestes comprovacions. El mòdul *Dispatch* determina en quints punts del programa es realitzen aquestes comprovacions d'estat dels *buffers*, i crida les funcions per processar les dades d'un determinat *buffer* i fer-les passar cap al següent.

Els annexos B i C mostren en detall l'anàlisi a fons del codi i les proves que s'han realitzat per tal de definir l'algoritme de multiplexació, respectivament. El següent apartat presenta les conclusions que se n'han extret.

4.5 Conclusions

La finalitat d'aquest informe era descriure de la manera més acurada i precisa possible l'algoritme de multiplexació d'Mplex. A continuació es presenten de forma resumida les idees que s'extreuen de l'estudi realitzat, que serviran com a base per a determinar els punts que s'han de millorar.

EXTRACCIÓ DE LA INFORMACIÓ

- Quan s'extreu informació d'un flux d'entrada, es copia el *payload* dels paquets (fragments PES) al *input buffer* corresponent. La informació que proporciona la capçalera dels paquets TS/PS s'emmagatzema en diferents camps de l'estructura de dades que conté l'ES.

- Els *input buffers* estan formats per un *buffer* de dades i un *buffer* de control. Les dades s'organitzen en blocs, cadascun dels quals correspon a un paquet PES. El *buffer* de control guarda informació sobre l'estructura del *buffer* de dades i sobre els paquets PES.
- A cada paquet PES se li assigna com a valor de lectura l'instant de temps en que s'ha acabat d'extreure (*msecread*). També es marca amb la darrera referència de PCR extreta del programa al que pertany l'ES en el flux original (*msecpush*).

GENERACIÓ DEL TS DE SORTIDA

- Es considera que un ES està preparat per ser empalmat a la sortida quan el *buffer* de dades té una determinada ocupació. En aquest moment, es marca amb el *flag* de *trigger* i s'assigna com a *timestamp* de sortida del paquet PES més antic l'instant actual.
- Un cop assignat el *trigger*, es comencen a treure els paquets PES quan ho determinen els seus *timestamps*, intentant que la distància temporal entre els paquets PES de cada ES sigui la mateixa en el flux de sortida que en el d'entrada. S'entén que la distància entre paquets PES (unitats d'accés) en el flux d'entrada és la correcta per tal que es compleixin les restriccions dels *buffers* imposades per l'STD. Així doncs, s'intenta mantenir aquesta distància per a que el flux de sortida també compleixi aquestes restriccions. Per calcular el *timestamp* de sortida, s'utilitza la referència PCR d'entrada (*msecpush*) i la diferència entre el rellotge de programa a l'entrada i el rellotge intern (*delta*).
- Tots els paquets TS en què s'encapsula cada paquet PES es treuen a la sortida de forma consecutiva, generant ràfegues de paquets de transport amb un mateix PID. Fins que no s'acaba de treure un paquet PES sencer, no es tornen a comprovar els *timestamps* dels paquets PES dels diferents fluxos. En cada passada de bucle, es genera un únic paquet de transport.
- Tots els paquets TS que contenen l'inici d'un paquet PES, porten com a referència PCR en el flux de sortida el *timestamp* associat al paquet PES.
- Tots els programes de sortida tenen el mateix rellotge de programa, corresponent al rellotge intern del multiplexor.
- En cap cas es modifica ni processa la informació original, és a dir, no es canvia ni la capçalera ni el contingut dels paquets PES. Això implica que no es regeneren les marques de presentació ni de descodificació (PTS/DTS).

TRACTAMENT DE LA INFORMACIÓ PSI D'ENTRADA

- Bàsicament s'analitza per tal de relacionar els ES amb el *mapstream* del programa al que pertanyen en el flux d'entrada, amb la finalitat de tenir associada la referència PCR d'un programa d'entrada amb tots els ES que utilitzen aquesta mateixa referència.

CAPÍTOL 5. Millores proposades per al Mplex 13818

L'objectiu final de l'anàlisi de l'algoritme d'Mplex era trobar deficiències en la implementació del multiplexor que fan que el *Transport Stream* de sortida presenti problemes a l'hora de ser reproduït. En aquest punt es presenten els problemes que s'han detectat i es proposen solucions. La seva implementació s'ha realitzat utilitzant l'entorn de programació Anjuta, descrit en l'annex E, en una màquina amb sistema operatiu Linux/Unix.

5.1 Problemes detectats en l'anàlisi del *software*

5.1.1 Marques PTS/DTS

Tal i com s'ha explicat en apartats anteriors, durant el procés de multiplexació es canvia el rellotge de referència amb que es sincronitza cada flux elemental (PCR) a la sortida, utilitzant com a nova referència el rellotge intern del sistema, però no es canvien les marques de presentació ni de descodificació (PTS/DTS) dels paquets PES. Això provoca que el descodificador no presenti/descodifiqui les unitats d'accés en el moment adequat, ja que el seu rellotge intern s'enganxa al rellotge del flux mitjançant les marques PCR, mentre que les marques PTS/DTS estan referides a un rellotge diferent. Aquest és el problema base, pel qual és impossible que cap descodificador sigui capaç de presentar correctament cap programa d'un flux de transport generat amb Mplex. En les proves realitzades, s'ha comprovat que quan s'intenta reproduir un flux de sortida d'Mplex amb el reproductor VLC, la pantalla es queda negra i no es reprodueix l'àudio, per a cap programa.

5.1.2 Distància entre paquets PES

El *timestamp* de sortida de cada paquet PES es calcula tenint en compte l'instant d'entrada del paquet (*msecpush*) i la diferència entre el rellotge del programa en el flux d'entrada i el rellotge del multiplexor (*delta*), amb l'objectiu de que la distància entre els paquets sigui la mateixa a la sortida que a l'entrada del multiplexor. D'aquesta manera, assumint que en el flux d'entrada la distància entre els paquets compleix les restriccions imposades pel T-STD, el flux de sortida també complirà aquestes condicions. No obstant, la marca que s'utilitza com a referència per saber a quin instant ha entrat un paquet no correspon al moment en què ha entrat el primer *byte* del paquet en qüestió, sinó que s'agafa la darrera referència PCR associada amb l'ES. Així doncs, el *timestamp* d'entrada (*msecpush*) tindrà una petita desviació respecte la posició real que ocupa l'inici del paquet dins el flux d'entrada, i aquesta desviació es traslladarà al flux de sortida, fent que la distància entre els paquets no sigui exactament la mateixa a la sortida que a l'entrada. Tot i que aquesta desviació és relativament petita, podria provocar una violació de les restriccions del T-STD, podent-se produir un excés d'informació en els *buffers* d'entrada del descodificador si la distància entre dos paquets és menor en el flux de sortida

que en el d'entrada, o bé una manca d'informació si és major, sempre assumint que la distància entre els paquets en el flux d'entrada compleix aquestes restriccions.

5.1.3 Entrellaçat dels paquets TS

D'altra banda, com ja s'havia observat en l'estudi dels fluxos de sortida d'Mplex, s'ha comprovat que per la manera de treballar del mòdul *Dispatch* tots els paquets de transport que contenen un paquet PES es disposen de forma consecutiva. Aquest aspecte no ha d'influir necessariament a la correcta reproducció del flux. D'una banda, els *buffers* del descodificador estan preparats per contenir més d'una unitat d'accés, i per tant més d'un paquet PES. Així, el fet que tots els fragments d'un paquet PES arribin al *buffer* en bloc no hauria de ser motiu perquè es produeixi un desbordament. D'altra banda, aquesta disposició dels paquets tampoc hauria de provocar una situació de falta de dades, tenint en compte que la distància entre l'inici dels paquets PES és la mateixa, i que en aquest cas arriben en bloc, amb menys retard entre l'inici i el final del paquet. La conseqüència d'aquesta forma d'entrellaçar els paquets és que si en algun moment es produeix una ràfega d'errors causada per les condicions del canal en què viatja la informació, que afecti a una sèrie de paquets TS consecutius, els errors afectaran a menys paquets PES que si el flux estès format per paquets TS de diferents paquets PES entrellaçats. Això pot ser una avantatja si no s'utilitza cap mecanisme de correcció d'errors, ja que es perdrà menys informació, però si se n'utilitza pot ser un aspecte negatiu ja que els errors estaran menys repartits entre els paquets i per tant serà menys probable que es puguin corregir.

En aquest treball, la millora del *software* se centrarà en els dos primers punts que s'han presentat:

- Les marques PTS/DTS no queden referides a la base de temps del programa.
- La distància entre els paquets PES presenta una petita desviació en relació a la distància en el flux d'entrada.

5.2 Solucions proposades

5.2.1 Marques PTS/DTS

5.2.1.1 Descripció de la solució

Per tal de solucionar el problema de les marques PTS/DTS, s'ha decidit regenerar-les de manera que quedin referides al nou rellotge de programa (el rellotge intern del multiplexor).

Tasques a realitzar

1. Determinar el procediment per recalcular els *timestamps*.
2. Definir en quin punt del procés de multiplexació es realitza.
3. Implementar el codi necessari per extreure les marques PTS/DTS.
4. Implementar el codi necessari per modificar els *timestamps* en la capçalera dels paquets PES.

Les marques de descodificació/presentació originals estan referides a una base temporal coneguda, que correspon a la referència PCR del flux d'entrada. Coneixent la diferència temporal entre el rellotge amb què s'ha marcat originalment el flux PES, i el rellotge amb què s'està sincronitzant el flux de sortida (l'intern del multiplexor), s'ajustaran les marques PTS/DTS sumant aquesta diferència, en milisegons, a les marques PTS/DTS originals, també en milisegons, i tornant a referir el resultat a una marca de rellotge de 90 KHz. El decalatge entre els dos rellotges serà constant.

En quant al punt en què s'ha de realitzar el *restamping*, s'ha decidit que el més adient és fer aquesta modificació just en el moment en què la capçalera del paquet PES s'encapsula en un paquet TS i es treu al *buffer* de sortida.

5.2.1.2 Implementació

Per regenerar les marques de presentació i descodificació dels paquets PES, s'ha creat una funció anomenada *procdata_restamp*, en el mòdul *Splice*. Aquesta s'encarrega d'extreure les marques PTS/DTS d'un paquet PES, regenerar-les, i modificar els *bytes* del *buffer* on hi ha el paquet per tal que els canvis es vegin reflectits. La funció es crida cada cop que es s'empaqueta la capçalera d'un paquet PES en un paquet TS de sortida, just abans de copiar el fragment de paquet PES en el *payload* del paquet de transport. La crida a la funció es realitza dins la funció *process_something*, del mòdul *Splice*, que és l'encarregada d'empaquetar els paquets PES en paquets TS.

```
stream_descr *process_something (stream_descr *s)
{
  [...]
  switch (s->streamdata) {
   case sd_data:
     c = &s->ctrl.ptr[s->ctrl.out];
     procdata_check_psi (&pid, &scramble, &size, s, c);
     d = output_pushdata (TS_PACKET_SIZE, TRUE, c->msecpush + s->u.d.delta);
     if (d == NULL) {
       return (s);
     }
     space = procdata_adaptfield_flags (&adapt_flags1, &adapt_flags2,
        &adapt_ext_len, s, c);
     payload = procdata_adaptfield_frame (space, adapt_field_ctrl, adapt_flags1, size);
     d = procdata_syn_head (d, pid, scramble, adapt_field_ctrl);
     d = procdata_syn_adaptfield (s, c, d, space-payload, payload,
```

```
        adapt_field_ctrl, adapt_flags1, adapt_flags2, adapt_ext_len);
    if (unit_start == TS_UNIT_START && psi_size<=0) {
      procdata_restamp (s, c);
    }
    return (procdata_syn_payload (s, c, d, payload));
    break;
  [...]
}
```

Aquest és un fragment del codi de la funció *process_something*, en què es realitza la crida a la funció *procdata_restamp*. Tal i com es pot observar, cada cop que es crida la funció *process_something* per generar un paquet TS, i que les dades que conté l'*stream_descr *s* corresponen a un flux de dades en format PES (*s->streamdata == sd_data*), es comprova que la variable *unit_start* sigui igual a TS_UNIT_START, i que la variable *psi_size* sigui menor o igual a zero. La primera indicarà que la informació que s'ha d'encapsular en aquest paquet TS correspon a l'inici del bloc de dades (paquet PES més antic del *stream descr *s*), o bé al primer fragment d'una taula si hi ha informació PSI de sortida disponible. La segona comprovació indica que no hi ha informació PSI de sortida pendent de ser transmesa; en cas que n'hi hagués, primer es transmetrien les taules disponibles, i no les dades de l'*stream_descr *s*. Així doncs, només es cridarà la nova funció en el punt en què s'hagi d'empaquetar la capçalera d'un paquet PES. A continuació es presenta el codi de la funció *procdata_restamp*.

Implementació de la funció *procdata_restamp*

```
void procdata_restamp (stream_descr *s, ctrl_buffer *c)
{
   int i, j, k;
   clockref pts, dts;
   t_msec msecpts, msecdts;
   i = c->index;
   if((s->data.ptr[i]==0x00) && (s->data.ptr[++i]==0x00) && (s->data.ptr[++i]==0x01)) {
      i+=5;
      if ((s->data.ptr[i] >> 7) & 1) {
         j = i+2;
         k = j;
         //Extraccio PTS
         pts.ba33 = (s->data.ptr[j] >>3) & 1;
         pts.base = (s->data.ptr[j] >>1) & 3;
         pts.base = (pts.base << 8) | s->data.ptr[++j];
         pts.base = (pts.base << 7) | ((s->data.ptr[++j]>>1) & 127);
         pts.base = (pts.base << 8) | s->data.ptr[++j];
         pts.base = (pts.base << 7) | ((s->data.ptr[++j]>>1) & 127);
         pts.ext = 0;
         pts.valid = TRUE;
         //Calcul nou PTS
         cref2msec (&s->u.d.conv, pts, &msecpts);
         msec2cref (&s->u.d.conv, msecpts + s->u.d.delta, &pts);
```

```
            //Modificació Capçalera PES
            s->data.ptr[k] = ((s->data.ptr[i]>>6) & 1) ? 0x30 : 0x20;
            s->data.ptr[k] |= (pts.ba33 << 3) & 8;
            s->data.ptr[k] |= (pts.base >> 29) & 6;
            s->data.ptr[k] |= 1;
            s->data.ptr[++k] = (pts.base >> 22) & 255;
            s->data.ptr[++k] = (pts.base >> 14) & 254;
            s->data.ptr[k] |= 1;
            s->data.ptr[++k] = (pts.base >> 7) & 255;
            s->data.ptr[++k] = (pts.base << 1) & 254;
            s->data.ptr[k] |= 1;
        }
        if ((s->data.ptr[i] >> 6) & 1) {
            //Extracció DTS
            dts.ba33 = (s->data.ptr[++j] >>3) & 1;
            dts.base = (s->data.ptr[j] >>1) & 3;
            dts.base = (dts.base << 8) | s->data.ptr[++j];
            dts.base = (dts.base << 7) | ((s->data.ptr[++j]>>1) & 127);
            dts.base = (dts.base << 8) | s->data.ptr[++j];
            dts.base = (dts.base << 7) | ((s->data.ptr[++j]>>1) & 127);
            dts.ext = 0;
            dts.valid = TRUE;
            //Calcul nou DTS
            cref2msec (&s->u.d.mapstream->u.m.conv, dts, &msecdts);
            msec2cref (&s->u.d.conv, msecdts + s->u.d.delta, &dts);
            //Modificació Capçalera PES
            s->data.ptr[++k] = 0x10;
            s->data.ptr[k] |= (dts.ba33 << 3) & 8;
            s->data.ptr[k] |= (dts.base >> 29) & 6;
            s->data.ptr[k] |= 1;
            s->data.ptr[++k] = (dts.base >> 22);
            s->data.ptr[++k] = (dts.base >> 14) & 254;
            s->data.ptr[k] |= 1;
            s->data.ptr[++k] = (dts.base >> 7);
            s->data.ptr[++k] = (dts.base << 1) & 254;
            s->data.ptr[k] |= 1;
        }
    }
}
```

- Paràmetres formals

  - stream_descr *s: ES del qual es volen regenerar les marques PTS/DTS (del paquet PES més antic).
  - ctrl_buffer *c: *buffer* de control del paquet PES més antic

- Valor de retorn

  - No té valor de retorn.

- Algoritme

1. Es comprova que els tres primers *bytes* del bloc de dades corresponguin a l'inici d'un paquet PES (0x00, 0x00, 0x01). La variable *i* apunta al primer *byte* del paquet. Si els tres *bytes* tenen el valor que han de tenir, es procedeix a realitzar la regeneració de les marques PTS/DTS.

Taula 5.1. Estructura de la capçalera d'un paquet PES. Extret de [E1]

Syntax	No. of bits	Mnemonic
PES_packet() {		
packet_start_code_prefix	**24**	**bslbf**
stream_id	**8**	**uimsbf**
PES_packet_length	**16**	**uimsbf**
if (stream_id != program_stream_map		
&& stream_id != padding_stream		
&& stream_id != private_stream_2		
&& stream_id != ECM		
&& stream_id != EMM		
&& stream_id != program_stream_directory		
&& stream_id != DSMCC_stream		
&& stream_id != ITU-T Rec. H.222.1 type E stream) {		
'10'	**2**	**bslbf**
PES_scrambling_control	**2**	**bslbf**
PES_priority	**1**	**bslbf**
data_alignment_indicator	**1**	**bslbf**
copyright	**1**	**bslbf**
original_or_copy	**1**	**bslbf**
PTS_DTS_flags	**2**	**bslbf**
ESCR_flag	**1**	**bslbf**
ES_rate_flag	**1**	**bslbf**
DSM_trick_mode_flag	**1**	**bslbf**
additional_copy_info_flag	**1**	**bslbf**
PES_CRC_flag	**1**	**bslbf**
PES_extension_flag	**1**	**bslbf**
PES_header_data_length	**8**	**uimsbf**
if (PTS_DTS_flags == '10') {		
'0010'	**4**	**bslbf**
PTS [32..30]	**3**	**bslbf**
marker_bit	**1**	**bslbf**
PTS [29..15]	**15**	**bslbf**
marker_bit	**1**	**bslbf**
PTS [14..0]	**15**	**bslbf**
marker_bit	**1**	**bslbf**
}		

Syntax	No. of bits	Mnemonic
if (PTS_DTS_flags == '11') {		
'0011'	**4**	**bslbf**
PTS [32..30]	**3**	**bslbf**
marker_bit	**1**	**bslbf**
PTS [29..15]	**15**	**bslbf**
marker_bit	**1**	**bslbf**
PTS [14..0]	**15**	**bslbf**
marker_bit	**1**	**bslbf**
'0001'	**4**	**bslbf**
DTS [32..30]	**3**	**bslbf**
marker_bit	**1**	**bslbf**
DTS [29..15]	**15**	**bslbf**
marker_bit	**1**	**bslbf**
DTS [14..0]	**15**	**bslbf**
marker_bit	**1**	**bslbf**
}		

2. Es fa apuntar la variable *i* al vuitè *byte* del paquet PES, en què els dos bits de més pes corresponen als *PTS_DTS_flags*

Taula 5.2. Possibles valors dels *PTS_DTS_flags*

PTS_DTS_flags	Descripció
00	El paquet no conté cap marca (PTS ni DTS).
01	Valor prohibit.
10	El paquet conté marca de presentació (PTS).
11	El paquet conté marca PTS i DTS.

3. Es comprova que el bit més significatiu d'aquest *byte* sigui "1". Si ho és, es procedirà a regenerar la marca PTS. En cas que sigui zero, indicarà que el paquet PES no té cap marca de temps, i per tant acabarà la funció. Un cop regenerada la marca PTS, es comprovarà el segon bit de més pes per determinar si hi ha marca DTS, i es processarà en cas afirmatiu.

   if ((s->data.ptr[i] >> 7) & 1) → Comprovació PTS
   if ((s->data.ptr[i] >> 6) & 1) → Comprovació DTS

4. Es fan apuntar les variables *j* i *k* al primer *byte* que conté la marca PTS. La variable *i* continuarà apuntant al *PTS_DTS_flag*, per tal de comprovar-ne el segon bit més endavant, un cop processada la marca PTS. La variable *j* s'utilitzarà per recórrer els *bytes* que contenen les marques en el procés d'extracció d'aquestes, i la variable *k* s'utilitzarà per reescriure els *bytes*.

5. Es procedeix a l'extracció de la marca PTS. Aquesta es desa en la variable *pts*, del tipus *clock_ref* (definit a *global.h*). Aquest tipus d'estructura s'utilitza per emmagatzemar referències de rellotge en el format especificat en l'ISO 13818-1.

```
/* ISO 13818 clock reference 90kHz (33 bit) with 27MHz extension (9 bit).
 * ba33 holds the high bit of base.
 */
typedef struct {
  uint32_t base;
  uint16_t ext;
  byte ba33;
  boolean valid;
} clockref;
```

Tal i com explica l'autor del codi, conté una variable anomenada *base*, en què s'emmagatzemen els 32 bits de menys pes de la referència de 90KHz (*base*). *Ba33* conté el bit de més pes d'aquesta referència. La variable *ext* conté els 9 bits de la referència de 27MHz (*extension*). Les marques PTS i DTS estan

formades únicament per la referència de 90KHz, de manera que únicament s'utilitzaran les variables *base* i *ba33*.

A la taula 5.1 es pot observar l'estructura de la capçalera dels paquets PES. Un cop s'està apuntant al *byte* que conté l'inici de la marca PTS, se n'extreu el quart bit que correspon al bit més significatiu de la referència i s'emmagatzema en la variable *ba33*.

A partir d'aquí, tots els bits restants s'emmagatzemen en la variable *base*. Per a cada *byte* que conté un fragment de la marca PTS, es desplacen els bits de manera que quedi la informació a extreure a la dreta, i es fa una operació AND, bit a bit, per extreure únicament els bits desitjats. A continuació, es desplacen els bits de la variable *base* cap a l'esquerra, tantes posicions com els bits que s'hi afegiran a continuació, i es realitza una operació OR, bit a bit, amb els nous bits. Així doncs, els bits que es van extraient, es van deixant en les posicions de menys pes de la variable *base*, i es van desplaçant cap a l'esquerra a mesura que en van entrant de nous. La taula 5.3 mostra els bits que s'extreuen en cada una de les instruccions del codi.

Taula 5.3. Procés d'extracció de la marca PTS

```
//Extracció PTS
```

	BITS EXTRETS	BYTE
pts.ba33 = (s->data.ptr[j] >>3) & 1;	[_ _ _ _ x _ _ _]	1
pts.base = (s->data.ptr[j] >>1) & 3;	[_ _ _ _ _ x x _]	1
pts.base = (pts.base << 8) | s->data.ptr[++j];	[x x x x x x x x]	2
pts.base = (pts.base << 7) | ((s->data.ptr[++j]>>1) & 127);	[x x x x x x x _]	3
pts.base = (pts.base << 8) | s->data.ptr[++j];	[x x x x x x x x]	4
pts.base = (pts.base << 7) | ((s->data.ptr[++j]>>1) & 127);	[x x x x x x x _]	5

6. Un cop extreta la marca PTS, es converteix a milisegons, mitjançant la funció *cref2msec*, implementada en el fitxer *global.c*. Tal i com s'ha explicat en la descripció de la solució, es recalcula la marca PTS sumant-li al valor original el valor de la variable *delta*, que correspon a la diferència entre el rellotge intern i el rellotge del programa en el flux d'entrada. El resultat d'aquesta operació, que correspon a l'instant en què s'ha de presentar la imatge, en milisegons i referit al rellotge intern del multiplexor, es torna a passar a marca de 90KHz, mitjançant la funció *msec2cref*.

7. Després de recalcular la marca, es modifiquen els *bytes* dels que s'havia extret, per tal que emmagatzemin el nou valor. Com es pot observar a la taula 5.1, el primer *byte* que conté la marca PTS, comença amb quatre bits concrets, que depenen de si el paquet conté marca PTS, o bé marca PTS i DTS. Així doncs, en el moment de reescriure el primer *byte* de la marca PTS, es comprova si el paquet conté també marca DTS, i en funció d'això s'escriu la seqüència '0010' (0x20) ó '0011' (0x30). Complint amb l'estàndard, es posen a 1 tots els *marker bits*.

Taula 5.4. Procés d'escriptura de la marca PTS

//Modificació Capçalera PES

	BITS ESCRITS	BYTE
s->data.ptr[k] = ((s->data.ptr[i]>>6) & 1) ? 0x30 : 0x20;	[x x x x _ _ _ _]	1
s->data.ptr[k] |= (pts.ba33 << 3) & 8;	[_ _ _ _ x _ _ _]	1
s->data.ptr[k] |= (pts.base >> 29) & 6;	[_ _ _ _ _ x x _]	1
s->data.ptr[k] |= 1;	[_ _ _ _ _ _ _ x]	1
s->data.ptr[++k] = (pts.base >> 22) & 255;	[x x x x x x x x]	2
s->data.ptr[++k] = (pts.base >> 14) & 254;	[x x x x x x x _]	3
s->data.ptr[k] |= 1;	[_ _ _ _ _ _ _ x]	3
s->data.ptr[++k] = (pts.base >> 7) & 255;	[x x x x x x x x]	4
s->data.ptr[++k] = (pts.base << 1) & 254;	[x x x x x x x _]	5
s->data.ptr[k] |= 1;	[_ _ _ _ _ _ _ x]	5

Com es pot observar, en el moment d'escriure els primers quatre bits del primer *byte* que conté la marca PTS, es comprova si hi ha també marca DTS, i en funció d'això, s'escriu el valor '0010' o '0011'. A partir d'aquí, es van escrivint els bits de PCR en les posicions adequades, i es posen els *marker bits* a '1'.

8. Es comprova si el paquet porta marca DTS. En cas afirmatiu es realitza el mateix procés d'extracció, càlcul i escriptura.

5.2.1.3 Proves realitzades. Resultats.

Per valorar les millores d'aquesta solució, s'han realitzat proves amb les captures "rete.ts" i "transponder10818.ts". La primera, presentada en el capítol 3, correspon a una captura de DVB-T. La segona, és tracta d'un flux de DVB-S, capturada del satèl·lit ASTRA al novembre de 2003, i que inclou següents programes:

29950 CINEMANÍA
29951 CINEMANÍA 2
29952 DCINE ESPAÑOL
29953 CANAL COCINA
29954 TVV INT.
29955 FOX
29956 AXN

Totes les proves s'han realitzat amb un PC de sobretaula amb processador Intel Q9550 (quatre nuclis a 2,83 GHz, 12MB L2, bus de 1333 MHz), 8 Gigues de memòria RAM DDR2 i placa base Asus amb FSB de 1600 MHz.

La taula 5.5 mostra les marques PTS i DTS dels paquets PES en el flux de sortida de Mplex resultat de la remultiplexació de la captura "rete.ts", abans d'implementar la millora, en la versió inicial del *software*.

Taula 5.5 Marques temporals en la versió inicial d'Mplex

num. Seq.	PID	PCR	PTS	DTS
3	259	4410	3417409608	-
41	266	-	3417402250	-
44	260	4410	1411153188	-
85	257	-	1411138630	-
92	265	-	1411139087	-
106	259	4500	3417413208	-
145	266	-	3417405850	-
151	266	-	3417409450	-
152	264	-	3417398958	-
166	273	4500	632935428	-
204	272	-	632914923	-
234	263	4770	4180033264	-
289	262	-	4180018716	-
296	269	-	4180020838	-
309	274	-	4180022949	-
339	268	4860	207469778	-
376	275	-	207455227	-
385	268	4950	207484178	207473378
518	275	-	207458827	-
525	268	*5775	207476978	-
565	270	-	207449373	-
578	276	-	207458007	-
607	257	-	1411142230	-
614	260	7740	1411167588	1411156788
767	259	7740	3417427608	3417416808

Com es pot observar, les marques de presentació i descodificació tenen un valor completament incoherent amb les referències PCR. Les marques DTS/PTS estan referides al rellotge amb què s'havia marcat cada un dels programes originalment, mentre que les marques PCR de tots els programes han quedat referides al rellotge intern del multiplexor. Per exemple, quan arribi el paquet TS amb número de seqüència 3, el rellotge del descodificador, enganxat al del flux d'entrada, tindrà un valor de 4410. El paquet PES (unitat d'accés) associat a aquest paquet té una marca de presentació igual a 3417409608. La diferencia entre els dos valors és de 3417405198 que, amb una resolució de 90KHz, correspon a 37971 segons aproximadament (més de 10 hores), de manera que les imatges no es presentaran mai.

La taula 5.6 mostra la mateixa informació del flux generat amb Mplex com a resultat de la remultiplexació de "rete.ts", però amb la regeneració de les marques PTS/DTS implementada. En aquest cas, les marques tenen un valor molt més proper al rellotge de programa. Per exemple, en el paquet amb número de seqüència 44, la diferència entre la marca PCR i la marca de presentació és de 20880, que equival a uns 230 ms. S'ha agafat uns quants paquets a l'atzar d'un determinat PID, en la captura "rete.ts" original i en el flux remultiplexat, i se n'ha calculat la distància temporal entre les marques PCR i les marques PTS. Els resultats es mostren en les taules 5.7 i 5.8.

Taula 5.6 Marques temporals regenerades

num. Seq.	PID	PCR	PTS	DTS
41	302	-	45630	-
44	401	36990	57870	-
85	402	-	43380	-
92	403	-	30510	-
106	301	38160	58050	-
145	302	-	44010	-
151	302	-	47610	-
152	303	-	33930	-
166	201	38340	61740	-
221	202	-	43920	-
228	204	-	29340	-
241	203	-	28170	-
308	102	-	47070	-
317	501	39780	65790	-
355	503	-	28620	-
384	402	-	46980	-
391	401	40320	72270	61470
545	101	41040	74250	63450
678	102	-	48870	-
685	101	*41982	67050	-
725	104	-	32760	-
738	103	-	27990	-
767	301	41400	72450	61650
893	302	-	51210	-

Taula 5.7 Distància entre marques PTS i rellotge intern, en la captura original

num. Seq.	PID	PCR	PTS	DTS	PTS-PCR (ms)
236	101	207434881	207455378	-	227,74
436	101	207436241	207458978	-	252,63
640	101	207437628	207473378	207462578	397,22
1464	101	207443232	207466178	-	254,96
1678	101	207444687	207469778	-	278,79
1908	101	207446251	207484178	207473378	421,41
2746	101	207451949	207476978	-	278,10
2992	101	207453622	207480578	-	299,51
3531	101	207457287	207494978	207484178	418,79

Taula 5.8 Distància entre marques PTS i rellotge intern, en el flux de sortida

num. Seq.	PID	PCR	PTS	DTS	PTS-PCR (ms)
545	101	41040	74250	63450	369,00
685	101	41982	67050	-	278,53
1053	101	44460	70650	-	291,00
1798	101	51120	85050	74250	377,00
1897	101	51675	77850	-	290,83
2392	101	54450	81450	-	300,00
3154	101	64350	95850	85050	350,00

Es pot veure que en els dos casos la distància entre les marques de presentació i el rellotge de programa tenen valors del mateix ordre de magnitud. No es disposen de mitjans per conèixer la correspondència entre un determinat paquet PES en el flux de sortida i en el flux d'entrada, de manera que no es pot comprovar que la distància sigui la mateixa en els dos fluxos per a un determinat paquet. La única manera d'associar un paquet PES d'entrada amb un de sortida seria analitzant-ne el contingut, corresponent al flux elemental comprimit seguint la capa de compressió MPEG-2. En aquest TFC no s'ha estudiat a fons l'estructura dels fluxos ES, i per tant no es pot determinar aquesta correspondència.

Finalment, s'ha generat un TS amb el programa TVE1 de la captura "rete.ts" (DVB-T), i el programa TVV int. de la captura "transponder10818.ts" (DVB-S). A la taula 5.9 es mostren les marques PTS/DTS i les marques PCR del flux de sortida. Com es pot veure també hi ha coherència entre aquestes, essent la distància suficientment petita per tal que les imatges es reprodueixin.

Taula 5.9 Marques temporals regenerades

num. Seq.	PID	PCR	PTS	DTS
2681	101	115830	139410	-
2796	169	116100	143550	136350
2902	116	-	102330	-
2919	169	117990	139950	-
2972	102	-	124830	-
2979	102	-	128430	-
2986	101	125820	153810	143010
3221	102	-	132030	-
3228	101	*128240	146610	-
3258	101	*128540	150210	-
3293	104	-	116640	-
3306	169	127080	150750	143550

A nivell de reproducció, els dos fluxos es veuen correctament amb l'VLC, sense pixelacions i amb una bona sincronització entre àudio i vídeo. A continuació es mostren els *logs* que proporciona el VLC amb informació sobre problemes detectats en el flux d'entrada, per a la captura "rete.ts".

- Log de la captura original (rete.ts)

```
main warning: output date isn't PTS date, requesting resampling (55362)
main warning: dts != current_pts (-175809)
main warning: decoder synchro warning: pts != current_date (-40000)
main warning: buffer is 55373 late, triggering upsampling
main warning: output date isn't PTS date, requesting resampling (87256)
main warning: audio drift is too big (142430), dropping buffer
main warning: timing screwed, stopping resampling
main warning: buffer is 118608 late, triggering upsampling
```

```
main warning: output date isn't PTS date, requesting resampling (63238)
main warning: audio drift is too big (179867), dropping buffer
main warning: audio drift is too big (155867), dropping buffer
main warning: audio drift is too big (131867), dropping buffer
ts warning: discontinuity indicator (pid=201)
ts warning: discontinuity received 0x5 instead of 0x3 (pid=501)
ts warning: discontinuity received 0x0 instead of 0xf (pid=201)
ts warning: discontinuity received 0x8 instead of 0x5 (pid=301)
ts warning: discontinuity received 0xb instead of 0xa (pid=101)
ts warning: discontinuity received 0xa instead of 0x8 (pid=401)
ts warning: discontinuity received 0xc instead of 0xb (pid=203)
ts warning: discontinuity received 0x0 instead of 0xe (pid=402)
ts warning: discontinuity received 0x1 instead of 0x0 (pid=503)
ts warning: discontinuity received 0x9 instead of 0x8 (pid=303)
ts warning: discontinuity received 0xd instead of 0xc (pid=204)
ts warning: discontinuity received 0x2 instead of 0x1 (pid=102)
ts warning: discontinuity received 0x2 instead of 0x0 (pid=202)
ts warning: discontinuity received 0x9 instead of 0x7 (pid=103)
ts warning: discontinuity received 0x9 instead of 0x8 (pid=302)
ts warning: discontinuity received 0x1 instead of 0x0 (pid=403)
main warning: audio drift is too big (-135779), clearing out
main warning: timing screwed, stopping resampling
main warning: mixer start isn't output start (-54067)
main warning: output date isn't PTS date, requesting resampling (54359)
main warning: buffer is 54514 late, triggering upsampling
main warning: resampling stopped after 11365546 usec (drift: 507)
```

- Log de la captura remultiplexada amb la versió inicial

```
[...]
main warning: early picture skipped
main warning: early picture skipped
main warning: early picture skipped
main warning: early picture skipped
main warning: early picture skipped
main warning: early picture skipped
main warning: received buffer in the future
main warning: received buffer in the future
main warning: received buffer in the future
main warning: received buffer in the future
main warning: early picture skipped
main warning: received buffer in the future
main warning: received buffer in the future
main warning: early picture skipped
main warning: early picture skipped
main warning: early picture skipped
[...]
```

En aquest cas es mostra només un fragment del *log*; en total conté 2284 entrades, la majoria del tipus "*received buffer in the future*" i "*early picture skipped*".

Log de la captura remultiplexada amb la versió millorada

```
ts warning: discontinuity indicator (pid=259)
ts warning: discontinuity indicator (pid=268)
main warning: output date isn't PTS date, requesting resampling (44393)
main warning: buffer is 44393 late, triggering upsampling
main warning: output date isn't PTS date, requesting resampling (107931)
main warning: dts != current_pts (-87639)
main warning: decoder synchro warning: pts != current_date (-40000)
main warning: audio drift is too big (152136), dropping buffer
main warning: audio drift is too big (128136), dropping buffer
main warning: timing screwed, stopping resampling
main warning: buffer is 104303 late, triggering upsampling
main warning: output date isn't PTS date, requesting resampling (93770)
main warning: audio drift is too big (195906), dropping buffer
main warning: audio drift is too big (171906), dropping buffer
main warning: audio drift is too big (147906), dropping buffer
main warning: audio drift is too big (123906), dropping buffer
main warning: output date isn't PTS date, requesting resampling (65143)
main warning: audio drift is too big (155028), dropping buffer
main warning: audio drift is too big (131028), dropping buffer
main warning: output PTS is out of range (4393), clearing out
main warning: audio drift is too big (-226076), clearing out
main warning: output date isn't PTS date, requesting resampling (60036)
```

D'entrada, s'observa que els errors de discontinuïtat que apareixen en el *log* de la captura original, causats per una pèrdua de paquets de transport, no es propaguen cap a flux de sortida ja que Mplex regenera el número de seqüència dels paquets de sortida. El *log* obtingut a partir de la reproducció del flux de transport generat amb la versió inicial d'Mplex mostra que el reproductor es queixa contínuament de que les imatges arriben massa tard o bé massa aviat, amb un *log* de 2284 la majoria de les quals fan referència a aquest problema, causat per la incoherència entre els *timestamps* de descodificació/presentació i el rellotge de programa. En canvi, en el *log* corresponent a la captura remultiplexada amb la versió millorada del *software* bàsicament apareixen missatges de *buffer is late*, *audio drift is too big*, i d'altres que fan referència directa als *timestamps* com per exemple *output date isn't PTS date* o *dts != current_pts*. Aquests errors en els *timestamps* d'algunes UA ja apareixen en el flux original, i es propaguen cap al flux de sortida. A més, alguns d'aquests *warnings* són inevitables ja que les captures de què es disposa comencen en un punt indeterminat el flux de transport i és probable que faltin les imatges clau necessàries per a descodificar les següents, fins l'arribada de la primera imatge *Intra*. En cap cas els *warnings* que apareixen en el *log* són crítics i subjectivament no afecten a la correcta reproducció del vídeo ni l'àudio.

Així doncs, queda demostrat que el problema principal d'Mplex era que no es regeneraven les marques PTS/DTS. El tema de la distància entre paquets PES també es corregirà a continuació, però amb aquesta implementació ja s'ha aconseguit una gran millora.

5.2.2 Distància entre paquets PES

5.2.2.1 Descripció de la solució

S'ha realitzat una modificació del codi per tal que en el moment en què s'assigna la variable *msecpush* es tingui en compte el temps transcorregut, referit al rellotge del flux d'entrada, des de la darrera extracció d'una marca PCR.

Tasques a realitzar

1. Determinar el procediment per calcular el valor que s'assigna a la variable *msecpush*.
2. Modificar la funció *ts_data_stream*, del mòdul *Split*, per tal que assigni a la variable *msecpush* el nou valor.

Tal i com descriu l'estàndard iso13818-1, es pot calcular el temps d'arribada de qualsevol *byte* d'un flux TS coneixent la marca PCR immediatament anterior, la immediatament posterior, i el número de *byte* en què viatgen aquestes dues referències. Així doncs, es pot conèixer el temps d'arribada de qualsevol paquet PES (del primer *byte* d'aquest), referit al rellotge d'entrada. Caldrà doncs implementar el codi necessari per tal que en tot moment es coneguin les marques PCR anterior i posterior als paquets PES que s'estan extraient, i així poder realitzar aquest càlcul.

Les expressions 5.1 i 5.2, extretes de l'estàndard ISO 13818-1, indiquen com calcular l'instant d'arribada de qualsevol *byte*.

$$t(i) = \frac{PCR(i'')}{system\_clock\_frequency} + \frac{i - i''}{transport\_rate\,(i)} \qquad \textbf{(5.1)}$$

$$transport\_rate(i) = \frac{((i' - i'') \times system\_clock\_frequency)}{PCR(i') - PCR(i'')} \qquad \textbf{(5.2)}$$

on:
i: índex del *byte* del qual se'n vol calcular l'instant d'arribada
i'': índex del *byte* que conté el darrer bit de la referència PCR anterior
i': índex del *byte* que conté el darrer bit de la referència PCR posterior
PCR(i''): referència PCR immediatament anterior
PCR(i'): referència PCR immediatament posterior

$$i'' < i < i'$$

L'operació "*PCR / system_clock_frequency*" equival a utilitzar la funció *cref2msec*, que passa de referència de 90KHz a temps en milisegons.

Mitjançant aquestes expressions, es calcularà el valor de la variable *msecpush* com l'instant d'arribada del primer *byte* del paquet PES.

Tal i com s'explica a l'estàndard iso13818, aquestes equacions són vàlides sempre i quan les marques PCR anterior i posterior al *byte* del qual es vol calcular el temps d'arribada estiguin referides a la mateixa base de temps. Si es produeix una discontinuïtat en el rellotge amb el que s'ha marcat el programa, indicada amb el *flag* del camp d'adaptació *discontinuity_indicator*, les expressions 5.1 i 5.2 no es poden aplicar entre la darrera marca PCR de la base de temps antiga i la marca PCR referenciada a la nova base de temps. Complint amb les especificacions de l'estàndard, per calcular el temps d'arribada dels *bytes* entre aquestes dues marques s'utilitzarà la formula 5.2, amb el darrer valor de *transport rate* calculat amb la base de temps antiga.

Per realitzar el càlcul, serà necessari conèixer en tot moment la marca PCR anterior i la posterior als *bytes* que s'estan processant. Per aconseguir-ho, s'ha dissenyat un algoritme que permet que el procés de multiplexació d'Mplex segueixi sent progressiu, realitzant tots els passos en temps real, sense fer salts enrere i sense la necessitat de fer un anàlisi previ de les captures per tal de conèixer les marques PCR futures. Cada cop que es troba i s'extreu una marca PCR d'un determinat programa, es busca la següent marca PCR associada al mateix programa en el flux d'entrada. El procés d'extracció de la marca PCR futura és independent del fil d'extracció principal; no es modifiquen els punters ni variables relacionats amb aquest procés. La marca futura es desa en l'estructura que conté el *mapstream* del programa, igual que es fa amb l'anterior. Tant en l'extracció de la marca PCR actual com en l'extracció de la marca futura, es guarda el *byte* del flux d'entrada en què viatgen. D'aquesta manera, cada cop que s'extregui un paquet PES, es disposarà de la informació necessària per tal de calcular, mitjançant el procediment definit en l'estàndard, l'instant d'arribada del primer *byte* del paquet PES segons el rellotge de programa, i s'assignarà aquest valor a la variable *msecpush*.

Per a que aquest algoritme funcioni correctament, és necessari que cada cop que s'extreu una marca PCR, la marca PCR futura ja hagi entrat en el *raw buffer*. Tenint en compte la mida dels *raw buffers*, de 262144 bytes, i la restricció que ha d'aparèixer una marca PCR almenys cada 0,7 segons per a cada programa, es pot determinar la velocitat mitja d'entrada de dades (V_E) mínima per a que es compleixi aquesta condició.

$$V_E = \frac{\text{dades llegides (bits)}}{\text{t (segons)}}[\text{bps}] \qquad \textbf{(5.3)}$$

$$V_E^{MIN} = \frac{8 \cdot 262144}{0.7} = 3.00\text{Mbps}$$

Es necessita una velocitat de lectura del *Transport Stream* d'entrada d'almenys 3 Mbps, que s'hauria complir de sobres tenint en compte que generalment un únic flux de vídeo ja té una velocitat binària major. Un cop implementada la solució es comprovarà que en tots els casos la marca PCR futura està disponible en el *buffer*.

5.2.2.2 Implementació

A continuació s'expliquen en detall els passos que s'han seguit en la implementació d'aquesta solució.

1. Afegir a l'estructura que conté el *mapstream* les variables necessàries per emmagatzemar la mostra PCR futura, així com els índexs dels *bytes* que contenen la marca anterior i la posterior. Afegir també una variable per indicar si les dues marques tenen la mateixa base de temps.

Amb aquest objectiu, s'ha modificat l'estructura *'u.m'*, de l'estructura *stream_descr*, afegint-hi les variables destacades en negreta. Aquesta estructura és la que conté els camps que s'utilitzen quan s'emmagatzema un *mapstream*.

```
struct {
  t_msec msectime;
  t_msec next_msectime;          Marca PCR futura, en ms.
  int PCR_byte;                  Índex del byte que conté el darrer bit de la marca PCR anterior.
  int next_PCR_byte;             Índex del byte que conté el darrer bit de la marca PCR futura.
  boolean discontinuity_indicator  Indica una discontinuïtat en la base de temps amb què es marca el programa d'entrada.
  conversion_base conv;
  int psi_length;
  byte psi_data[MAX];
} m;
```

2. Afegir una variable al *buffer* de control per guardar el número de *byte* que conté l'inici del paquet PES:

```
typedef struct {
  int index;
  int length;
  int sequence;
  t_msec msecread;
  t_msec msecpush;
  int PES_byte;                  Índex del byte que conté l'inici del paquet PES.
  clockref pcr;
  clockref opcr;
  byte scramble;
} ctrl_buffer;
```

3. Modificar la funció *ts_adaption_field*, del mòdul *Split*, per tal que cada cop que s'extreu una marca PCR es guardi el número de *byte* en què viatja el darrer bit d'aquesta, i es cridi la funció que s'encarrega de buscar la següent marca PCR:

```
static void ts_adaption_field (file_descr *f, int adf, int pid,
                               stream_descr *s, stream_descr *m)
{
    [...]
    cref2msec (&m->u.m.conv, *pcr, &m->u.m.msectime);
    m->u.m.PCR_byte = f->total + TS_PACKET_PCR;
    next_PCR_ref (f,pid,m);
   [...]
}
```

Després d'emmagatzemar el valor de la marca PCR en la variable *msectime*, en milisegons, s'assigna l'índex del *byte* que conté l'últim bit d'aquesta a la variable *PCR_byte* del *mapstream* corresponent. Per aconseguir aquest valor, s'agafa com a referència la variable *total* del *file_descr* que conté el TS del qual se n'està extraient informació. Tal i com s'ha explicat en apartats anteriors, aquesta variable és un comptador de *bytes* processats del flux d'entrada. En el moment d'avaluar el camp d'adaptació (crida a la funció *ts_adaption_field*), aquesta variable apunta al primer *byte* del paquet TS (s'actualitza quan s'acaba de processar el paquet). A aquest valor se li ha de sumar el número de *bytes* entre l'inici del paquet TS i el *byte* que conté el darrer bit de la marca PCR, tal i com diu l'estàndard. Per aquest motiu, i tenint en compte que a l'estructura *ts.h* hi havia altres definicions realitzades a tal efecte (com per exemple *TS_PACKET_HEADSIZE*, indicant el número de *bytes* entre l'inici d'un paquet TS i el *byte* que emmagatzema la longitud de la capçalera), s'ha definit la constant simbòlica *TS_PACKET_PCR*, indicant la distància entre l'inici del paquet TS i el *byte* que conté el final de la marca PCR. A continuació, es realitza la crida a la funció *next_PCR_ref*, que s'encarregarà de buscar, extreure i emmagatzemar la propera marca PCR.

4. Implementar la funció que s'encarrega de buscar i extreure la següent marca PCR associada al mateix programa.

```
void next_PCR_ref (file_descr *f, int pid, stream_descr *m)
{
    int i, l, size, k, epid;
    boolean found;
    byte flags;
    l = size = list_size (f->data);
    k = f->data.out;
    found = FALSE;
    while (!found && l>TS_PACKET_SIZE) {
      /* Búsqueda inici paquet TS*/
       do  {
         list_incr (k,f->data,1);
         l --;
       }while ((l > 0) && (f->data.ptr[k] != TS_SYNC_BYTE));
       /* Avaluació capçalera paquet TS + flags del camp d'adaptació*/
       if (l>TS_PACKET_SIZE) {
          i = k;
          flags = 0x00;
```

```
            list_incr (i,f->data,TS_PACKET_PID);
            epid = f->data.ptr[i] & 0x1F;
            list_incr (i,f->data,1);
            epid = (epid << 8) | f->data.ptr[i];
            list_incr (i,f->data,1);
            if (!(f->data.ptr[i] & TS_AFC_BOTH)) {
               epid = TS_PID_NULL;
            } else {
                 if (f->data.ptr[i] & TS_AFC_ADAPT) {
                   list_incr (i,f->data,1);
                   if (f->data.ptr[i] != 0) {
                     list_incr (i,f->data,1);
                     flags = f->data.ptr[i];
                   }
                 }
            }
            /* Extracció del PCR futur
            if ((epid == pid) && (flags & TS_ADAPT_PCRFLAG)) {
               list_incr (i,f->data,1);
               clockref pcr;
               long b;
               byte a;
               a = f->data.ptr[i];
               list_incr (i,f->data,1);
               pcr.ba33 = (a >> 7) & 1;
               b = (a << 8) | f->data.ptr[i];
               list_incr (i,f->data,1);
               b = (b << 8) | f->data.ptr[i];
               list_incr (i,f->data,1);
               b = (b << 8) | f->data.ptr[i];
               list_incr (i,f->data,1);
               a = f->data.ptr[i];
               pcr.base = (b << 1) | ((a >> 7) & 1);
               marker_check (a,0x7E,0x7E);
               list_incr (i,f->data,1);
               pcr.ext = ((a & 1) << 8) | f->data.ptr[i];
               pcr.valid = TRUE;
               cref2msec (&m->u.m.conv, pcr, &m->u.m.next_msectime);
               m->u.m.next_PCR_byte = f->total + size - l + TS_PACKET_PCR;
               m->u.m.discontinuity_indicator = (flags >> 7) & 1;
               found = TRUE;
            }
         }
      }
   }
```

- Paràmetres formals

  - file_descr *f: punter al fitxer en el qual es buscarà la referència PCR.
  - int pid: PID del qual s'extraurà la marca PCR.
  - stream_descr *m: punter al *mapstream* del programa associat a la referència PCR.

- Valor de retorn

  - No té valor de retorn.

- Algoritme

El bucle principal de la funció recorre el TS contingut en el *raw buffer* del *file_descr *f*, començant pel *byte* més antic i anant cap a la informació que ha entrat més recentment. El bucle s'acabarà quan es trobi la marca PCR que s'està buscant, o bé quan s'hagin analitzat totes les dades del *buffer*. La marca es buscarà en el flux amb el PID especificat en els paràmetres formals de la funció (que correspondrà al PID del què s'ha extret la marca PCR en la funció *ts_adaption_field*).

```
while (!found && l>TS_PACKET_SIZE)
```

En cada passada del bucle principal, en primer lloc es busca l'inici d'un paquet de transport:

```
do  {
   list_incr (k,f->data,1);
   l --;
}while ((l > TS_PACKET_SIZE) && (f->data.ptr[k] != TS_SYNC_BYTE));
```

El motiu pel qual s'incrementa la variable *k* abans de comprovar si el *byte* al que apunta correspon a l'inici d'un paquet TS és que cada cop que comença una passada en el bucle principal, la variable *k* està apuntant al primer *byte* del paquet TS avaluat en la passada anterior. Si no s'incrementés abans de fer la comprovació, *k* sempre apuntaria al mateix paquet TS.

Un cop trobat l'inici del següent paquet TS a ser avaluat, se n'extreu el PID i els *flags* del camp d'adaptació. Si el PID del paquet coincideix amb el PID especificat en la capçalera, i el *flag* indica que el paquet conté marca PCR, s'extreu. A continuació, es passa a milisegons mitjançant la funció *cref2msec*, i s'emmagatzema en la variable *next_msectime*. Tot seguit, es calcula l'índex del *byte* que conté el darrer bit de la marca PCR. Per fer-ho, es determina el numero de *bytes* que hi ha entre el paquet que s'està processant en el fil d'extracció principal i el paquet del qual s'ha extret la marca PCR futura (*size – l*), i es suma al valor de la variable *total*. També es té en compte la distància entre l'inici del paquet i el final de la marca PCR (*TS_PACKET_PCR*).

```
m->u.m.next_PCR_byte = f->total + size - l + TS_PACKET_PCR;
```

Finalment, es desa el valor del *flag discontinuity_indicator* en la variable del *mapstream* creada a tal efecte. Així, si aquesta variable és certa, indicarà que entre les marques *msectime* i *next_mesectime* s'ha produït una discontinuïtat en la base de temps.

5. Implementar la funció que s'encarrega de calcular el valor de la variable *msecpush*, cada cop que s'extreu un paquet PES.

```
void calc_msecpush (stream_descr *m, stream_descr *s, ,ctrl_buffer *c)
{
   if (m->u.m.next_msectime != m->u.m.msectime
      && m->u.m.next_PCR_byte > m->u.m.PCR_byte) {
       if (!m->u.m.discontinuity_indicator) {
          m->u.m.transport_rate = (m->u.m.next_PCR_byte - m->u.m.PCR_byte)
                                    / (m->u.m.next_msectime - m->u.m.msectime);
       }
       c->msecpush = m->u.m.msectime
                       + ((c->PES_byte - m->u.m.PCR_byte) / m->u.m.transport_rate);
   }
   else {
      c->msecpush = m->u.m.msectime;
   }
}
```

- Paràmetres formals

  - stream_descr *m: punter al *mapstream* associat al paquet PES del qual es vol calcular la marca *msecpush*
  - ctrl_buffer *c: punter al *buffer* de control del paquet PES

- Valor de retorn

  - No té valor de retorn.

- Algoritme

En primer lloc, es comprova que les referències PCR actual i futura siguin diferents. Cada cop que s'extreu una referència PCR, es busca i s'extreu la següent. El proper cop que s'extregui una marca PCR en el fil d'extracció principal, aquesta correspondrà a la que, fins aquest moment, era la futura. Així doncs, si en un moment determinat no es troba la referència PCR futura en el *raw buffer*, les variables *msectime* i *next_msectime* tindran el mateix valor (la darrera marca PCR). En aquest cas, la variable *msecpush* no es calcularà amb el procediment explicat, sinó que se li assignarà la darrera referència PCR extreta (com en la versió inicial d'Mplex). No obstant, en les proves realitzades, s'ha comprovat que sempre que s'extreu una marca PCR es troba la següent dins el *raw buffer*.

```
if (m->u.m.next_msectime != m->u.m.msectime
   && m->u.m.next_PCR_byte > m->u.m.PCR_byte) {
   [...]
} else {
   c->msecpush = m->u.m.msectime;
}
```

Un cop s'ha comprovat que es disposa de les marques PCR anterior i posterior, es comprova que aquestes estiguin referides a la mateixa base de temps mitjançant la variable *discontinuity_indicator*. Si no s'ha produït cap discontinuïtat entre les dues, es calcula el *transport rate* i es desa el resultat en la variable del *mapstream* creada a tal efecte. A continuació, es calcula el temps d'arribada mitjançant la formula 5.2, i s'assigna el resultat a la variable *msecpush*. En cas que s'hagués produït una discontinuïtat en la base de temps, no es calcularia el *transport rate*, de manera que la variable mantindria el valor que s'havia calculat amb la base de temps antiga, i seria aquest l'utilitzat en la formula 5.2, tal i com s'especifica en l'estàndard.

```
if (!m->u.m.discontinuity_indicator) {
   m->u.m.transport_rate = (m->u.m.next_PCR_byte - m->u.m.PCR_byte)
                                    / (m->u.m.next_msectime - m->u.m.msectime);
}

c->msecpush = m->u.m.msectime
                + ((c->PES_byte - m->u.m.PCR_byte) / m->u.m.transport_rate);
```

6. Modificar la funció *ts_data_stream* per tal que calculi el valor de la variable *PES_byte* del *buffer* de control dels paquets PES extrets, i que cridi la funció citada en el punt anterior en el moment que comença l'extracció d'un paquet PES, per tal d'assignar-li la variable *msecpush*.

```
static boolean ts_data_stream (file_descr *f, int pid)
{
    [...]
    if (c->length == -2) {
      [...] *
      c->PES_byte = f->total;
      calc_msecpush (m, s, c);
    }
   [...]
}
```

Aquest és un fragment de codi de la funció *ts_data_stream*, en què s'avalua la capçalera del paquet PES i s'extreu la longitud del paquet. Dins aquest bloc de codi (*), es realitzen una sèrie de comprovacions per determinar si es pot començar amb l'extracció del paquet (si els tres primers *bytes* corresponen a l'inici d'un paquet PES, i hi ha prou espai al *buffer* de dades). Si no es pot extreure, es sortirà de la funció (*return*) i per tant no s'arribarà al final d'aquest bloc de codi. Si es pot començar amb l'extracció del paquet, es realitzarà l'assignació *c->PES_byte = f->total*, i es cridarà la funció *calc_msecpush*. En la versió inicial d'Mplex, l'assignació de la variable *msecpush* es realitzava al final de l'extracció del paquet PES. S'ha modificat aquest el punt pel següent motiu: per calcular el temps d'arribada del primer *byte* del paquet PES, és necessari conèixer la marca PCR anterior i posterior a aquest *byte*, i l'índex dels *bytes* en què viatgen les dues marques. Si es calcula el temps d'arribada del primer *byte*

(*msecpush*) quan s'acaba d'extreure el paquet PES, i entre l'inici de l'extracció del paquet i el final s'ha trobat alguna marca PCR, les marques emmagatzemades al *mapstream* com a *msectime* i *next_msectime* ja no correspondran a la immediatament anterior i la immediatament posterior, i el valor de *PES_byte* no estarà dins el rang [*PCR_byte, next_PCR_byte*], de manera que el càlcul seria incorrecte. Per aquest motiu, el càlcul del temps d'arribada del paquet es realitzarà quan es comenci a extreure.

5.2.2.3 Proves realitzades. Resultats.

Per tal de comprovar que la solució ha estat implementada correctament, cal centrar les proves en dos punts:

- Extracció de les marques PCR futures
- Càlcul del temps d'arribada dels paquets PES

Com en el cas anterior, les proves es realitzaran amb les captures "rete.ts" (DVB-T) i "transponder10818.ts" (DVB-S). Tal i com s'ha dit anteriorment, la captura "rete.ts" conté cinc programes, els vídeos i referències PCR dels quals viatgen en els PIDs 101, 201, 301, 401 i 501. A la taula 5.10 es pot veure un petit fragment del procés d'extracció de les marques PCR en cadascun dels fluxos que en porten. El valor de la variable *msectime* correspon a la marca PCR extreta en el fil d'extracció principal, que no s'ha modificat respecte la versió inicial d'Mplex. La nova variable *PCR_byte* emmagatzema el *byte* que conté el darrer bit d'aquesta marca. En tots els casos, aquestes dues variables tenen el mateix valor que *next_msectime* i *next_PCR_byte* en la iteració anterior. Així doncs, cada cop que s'extreu una marca PCR, aquesta correspon amb la que fins el moment era la marca futura (tant el valor, com el número de *byte* en què viatja), fet que demostra que l'extracció de la marca futura s'està realitzant correctament.

Taula 5.10 Extracció de les marques PCR de la captura "rete.ts"

PID	**Descripció**	**Valor**	**Funció**
101	msectime	2304974	ts_adaption_field
101	PCR_byte	398006	ts_adaption_field
101	next_msectime	2305010	next_PCR_ref
101	next_PCR_byte	489750	next_PCR_ref
101	msectime	2305010	ts_adaption_field
101	PCR_byte	489750	ts_adaption_field
101	next_msectime	2305048	next_PCR_ref
101	next_PCR_byte	582622	next_PCR_ref
101	msectime	2305048	ts_adaption_field
101	PCR_byte	582622	ts_adaption_field
PID	**Descripció**	**Valor**	**Funció**
201	msectime	46444553	ts_adaption_field
201	PCR_byte	162630	ts_adaption_field
201	next_msectime	46444589	next_PCR_ref
201	next_PCR_byte	253810	next_PCR_ref

201	msectime	46444589	ts_adaption_field
201	PCR_byte	253810	ts_adaption_field
201	next_msectime	46444626	next_PCR_ref
201	next_PCR_byte	346494	next_PCR_ref
201	msectime	46444626	ts_adaption_field
201	PCR_byte	346494	ts_adaption_field
PID	**Descripció**	**Valor**	**Funció**
301	msectime	37971036	ts_adaption_field
301	PCR_byte	156614	ts_adaption_field
301	next_msectime	37971072	next_PCR_ref
301	next_PCR_byte	248170	next_PCR_ref
301	msectime	37971072	ts_adaption_field
301	PCR_byte	248170	ts_adaption_field
301	next_msectime	37971110	next_PCR_ref
301	next_PCR_byte	340854	next_PCR_ref
301	msectime	37971110	ts_adaption_field
301	PCR_byte	340854	ts_adaption_field
PID	**Descripció**	**Valor**	**Funció**
401	msectime	15679246	ts_adaption_field
401	PCR_byte	156050	ts_adaption_field
401	next_msectime	15679283	next_PCR_ref
401	next_PCR_byte	248922	next_PCR_ref
401	msectime	15679283	ts_adaption_field
401	PCR_byte	248922	ts_adaption_field
401	next_msectime	15679320	next_PCR_ref
401	next_PCR_byte	340666	next_PCR_ref
401	msectime	15679320	ts_adaption_field
401	PCR_byte	340666	ts_adaption_field
PID	**Descripció**	**Valor**	**Funció**
501	msectime	7032326	ts_adaption_field
501	PCR_byte	333898	ts_adaption_field
501	next_msectime	7032363	next_PCR_ref
501	next_PCR_byte	426770	next_PCR_ref
501	msectime	7032363	ts_adaption_field
501	PCR_byte	426770	ts_adaption_field
501	next_msectime	7032400	next_PCR_ref
501	next_PCR_byte	518890	next_PCR_ref
501	msectime	7032400	ts_adaption_field
501	PCR_byte	518890	ts_adaption_field

En cas que es produís una discontinuïtat en la base de temps, el *log* mostraria una entrada indicant-ho en el moment en què s'extreu la marca PCR futura. No obstant, en un fragment de vídeo petit, com és el cas de les captures amb que s'estan realitzant les proves, és difícil que se'n produeixi. No es disposa de captures adequades per tal de comprovar el correcte funcionament en cas que es produeixi una discontinuïtat en el PCR d'entrada. Per fer-ho, es necessitarien captures en què es produís una discontinuïtat en el rellotge amb què s'està codificant el flux. D'altra banda, si en algun cas no es trobés la marca PCR futura, també apareixeria una entrada al *log* indicant-ho. Només ha aparegut aquesta entrada al final del procés de multiplexació, ja que s'havia arribat al final de la captura i no hi havia més marques PCR. En règim normal, no s'ha trobat cap cas en què no es disposi de la referència PCR futura.

S'han realitzat les mateixes proves generant un flux amb el programa 1377 (TVE1) de la captura "rete.ts" i el programa 29954 (TVV int.) de la captura "transponder10818.ts". Analitzant els *logs*, el procés d'extracció es realitza correctament, igual que en el cas anterior. Els PIDs que porten el PCR dels programes que estan sortint són el 101 (TVE1) i el 169 (TVV int.). Com es pot comprovar en les taules, el procés d'extracció de marques PCR es realitza correctament tant si el PID del qual s'extreuen està sortint a la sortida com si no. Cal recordar que Mplex extreu les marques de tots els fluxos que en porten, encara no estiguin actius, per si es vol utilitzar un altre flux el PCR del qual viatja en un flux no actiu.

Taula 5.11 Extracció de les marques PCR de les captures "rete.ts" i "transponder10818.ts".

PID	**Descripció**	**Valor**	**Funció**
101	msectime	2304974	ts_adaption_field
101	PCR_byte	398006	ts_adaption_field
101	next_msectime	2305010	next_PCR_ref
101	next_PCR_byte	489750	next_PCR_ref
101	msectime	2305010	ts_adaption_field
101	PCR_byte	489750	ts_adaption_field
101	next_msectime	2305048	next_PCR_ref
101	next_PCR_byte	582622	next_PCR_ref
101	msectime	2305048	ts_adaption_field
101	PCR_byte	582622	ts_adaption_field
PID	**Descripció**	**Valor**	**Funció**
169	msectime	14817503	ts_adaption_field
169	PCR_byte	1256414	ts_adaption_field
169	next_msectime	14817523	next_PCR_ref
169	next_PCR_byte	1330862	next_PCR_ref
169	msectime	14817523	ts_adaption_field
169	PCR_byte	1330862	ts_adaption_field
169	next_msectime	14817543	next_PCR_ref
169	next_PCR_byte	1406250	next_PCR_ref
169	msectime	14817543	ts_adaption_field
169	PCR_byte	1406250	ts_adaption_field
PID	**Descripció**	**Valor**	**Funció**
201	msectime	46444553	ts_adaption_field
201	PCR_byte	162630	ts_adaption_field
201	next_msectime	46444589	next_PCR_ref
201	next_PCR_byte	253810	next_PCR_ref
201	msectime	46444589	ts_adaption_field
201	PCR_byte	253810	ts_adaption_field
201	next_msectime	46444626	next_PCR_ref
201	next_PCR_byte	346494	next_PCR_ref
201	msectime	46444626	ts_adaption_field
201	PCR_byte	346494	ts_adaption_field
PID	**Descripció**	**Valor**	**Funció**
166	msectime	14554525	ts_adaption_field
166	PCR_byte	1249458	ts_adaption_field
166	next_msectime	14554545	next_PCR_ref
166	next_PCR_byte	1323906	next_PCR_ref

166	msectime	14554545	ts_adaption_field
166	PCR_byte	1323906	ts_adaption_field
166	next_msectime	14554565	next_PCR_ref
166	next_PCR_byte	1397414	next_PCR_ref
166	msectime	14554565	ts_adaption_field
166	PCR_byte	1397414	ts_adaption_field

Un cop s'ha comprovat que el procés d'extracció de les marques PCR anterior i posterior és correcte, s'analitzarà el procés de càlcul del temps d'arribada dels paquets PES. Per fer-ho, s'ha generat un *log* en què es mostra el valor de les variables que intervenen en el càlcul, cada cop que es calcula *msecpush*.

Taula 5.12 Càlcul de la variable *msecpush*.

PID	**Descripció**	**Valor**	**Funció**
101	msectime	2305010	calc_msecpush
101	PCR_byte	489750	calc_msecpush
101	next_msectime	2305048	calc_msecpush
101	next_PCR_byte	582622	calc_msecpush
101	transport_rate	2444	calc_msecpush
101	PES_byte	516060	calc_msecpush
101	msecpush	2305020	calc_msecpush
101	msectime	2305010	calc_msecpush
101	PCR_byte	489750	calc_msecpush
101	next_msectime	2305048	calc_msecpush
101	next_PCR_byte	582622	calc_msecpush
101	transport_rate	2444	calc_msecpush
101	PES_byte	562308	calc_msecpush
101	msecpush	2305039	calc_msecpush
PID	**Descripció**	**Valor**	**Funció**
301	msectime	37971036	calc_msecpush
301	PCR_byte	156614	calc_msecpush
301	next_msectime	37971072	calc_msecpush
301	next_PCR_byte	248170	calc_msecpush
301	transport_rate	2543	calc_msecpush
301	PES_byte	196272	calc_msecpush
301	msecpush	37971051	calc_msecpush
PID	**Descripció**	**Valor**	**Funció**
303	msectime	37971036	calc_msecpush
303	PCR_byte	156614	calc_msecpush
303	next_msectime	37971072	calc_msecpush
303	next_PCR_byte	248170	calc_msecpush
303	transport_rate	2543	calc_msecpush
303	PES_byte	199656	calc_msecpush
303	msecpush	37971052	calc_msecpush
303	msectime	37971184	calc_msecpush
303	PCR_byte	525846	calc_msecpush
303	next_msectime	37971220	calc_msecpush
303	next_PCR_byte	616650	calc_msecpush
303	transport_rate	2522	calc_msecpush
303	PES_byte	546328	calc_msecpush
303	msecpush	37971192	calc_msecpush

En la taula 5.12 es pot veure el procés de càlcul de les variables *msecpush*, per als PIDs 101, 301 i 303. Els dos primers porten referència PCR; la referència del PID 303, que conté l'àudio del programa, viatja en el 301. Per a cada cop que es determina el valor de *msecpush*, es mostra el valor de les variables que hi intervenen: *msecpush*, *PCR_next, next_msecpush, next_PCR_byte, transport_rate* i *PES_byte*. En tots els casos, el valor de la variable *PES_byte* es troba entre *PCR_byte* i *next_PCR_byte*, de manera que queda dins l'interval en què es calcularà el *transport_rate*. La variable *msecpush* pren un valor que queda enmig de *msectime* i *next_msectime*. Així doncs, sembla que el càlcul és correcte.

Quan es genera un flux de sortida amb els programes TVE1 ("rete.ts") i TVV int. ("transponder10818.ts"), els resultats també són els esperats. A continuació es mostren els *logs* del VLC per a la captura "rete.ts" remultiplexada amb aquesta nova versió del *software*. Com es pot observar, és similar al que s'havia obtingut remultiplexant aquesta mateixa captura amb la primera millora implementada, però amb menys errors relacionats amb els *timestamps*. A nivell de reproducció no s'ha detectat una millora significativa respecte les proves realitzades en el cas anterior.

```
main warning: output date isn't PTS date, requesting resampling (44348)
main warning: buffer is 44348 late, triggering upsampling
main warning: output date isn't PTS date, requesting resampling (103200)
main warning: audio drift is too big (147381), dropping buffer
main warning: audio drift is too big (123381), dropping buffer
main warning: dts != current_pts (-85398)
main warning: decoder synchro warning: pts != current_date (-40000)
main warning: timing screwed, stopping resampling
main warning: buffer is 99548 late, triggering upsampling
main warning: output date isn't PTS date, requesting resampling (99385)
main warning: audio drift is too big (172641), dropping buffer
main warning: audio drift is too big (148641), dropping buffer
main warning: audio drift is too big (124641), dropping buffer
main warning: output date isn't PTS date, requesting resampling (74318)
main warning: audio drift is too big (164896), dropping buffer
main warning: audio drift is too big (140896), dropping buffer
main warning: audio drift is too big (-231812), clearing out
main warning: output date isn't PTS date, requesting resampling (64833)
main warning: buffer is 41000 late, triggering upsampling
main warning: resampling stopped after 9664196 usec (drift: 125)
```

Com a prova definitiva, s'ha generat un múltiplex amb 10 programes procedents de tres fonts diferents:

- Rete2.ts (DVB-T), capturada del múltiplex "RGN". Novembre 2003
  - Programa 1057 (Canal +)
  - Programa 1121 (Antena 3)
  - Programa 1185 (Telecinco)
  - Programa 1377 (TVE 1)
  - Programa 1441 (TVE 2)
- Veo.ts (DVB-T). Novembre de 2003.
  - Programa 1536 (Veo TV)
  - Programa 1825 (Net TV)
  - Programa 420 (Video Promocional 2)
  - Programa 425 (Video Promocional 1)

- Transponder10818.ts (DVB-S), capturada del satèl·lit Astra. Gener de 2006.
  - Programa 29954 (TVV int.)

El flux de transport resultant també es reprodueix correctament amb l'VLC, però en aquest cas s'ha detectat un petit desfasament entre el vídeo i l'àudio dels diferents programes. El *log* extret del reproductor VLC es mostra a continuació i és similar als que s'han presentat per al a captura *rete.ts* remultiplexada amb la versió final d'Mplex, amb alguns errors de *audio drift is too big*, que en aquest cas si que s'han observat a nivell de reproducció, i d'altres relacionats amb els *timestamps*, poc significatius per les poques vegades que es produeixen.

```
main warning: output date isn't PTS date, requesting resampling (76694)
main warning: buffer is 76694 late, triggering upsampling
main warning: output date isn't PTS date, requesting resampling (73824)
main warning: dts != current_pts (-25842)
main warning: decoder synchro warning: pts != current_date (-40000)
main warning: audio drift is too big (150330), dropping buffer
main warning: audio drift is too big (126330), dropping buffer
main warning: output date isn't PTS date, requesting resampling (97223)
main warning: audio drift is too big (197095), dropping buffer
main warning: audio drift is too big (173095), dropping buffer
main warning: audio drift is too big (149095), dropping buffer
main warning: audio drift is too big (125095), dropping buffer
main warning: output date isn't PTS date, requesting resampling (68009)
main warning: audio drift is too big (157291), dropping buffer
main warning: audio drift is too big (133291), dropping buffer
main warning: output date isn't PTS date, requesting resampling (70825)
main warning: dts != current_pts (-156347)
main warning: decoder synchro warning: pts != current_date (-40000)
main warning: buffer is 70825 late, triggering upsampling
main warning: dts != current_pts (-105942)
main warning: decoder synchro warning: pts != current_date (-40000)
main warning: output date isn't PTS date, requesting resampling (71217)
main warning: buffer is 71217 late, triggering upsampling
```

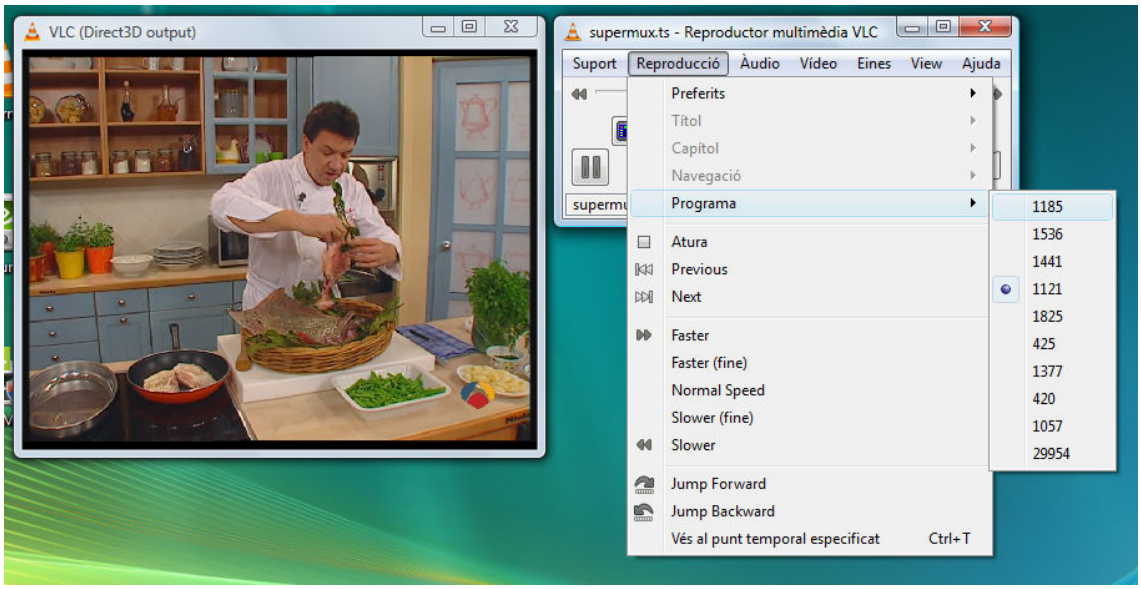


Figura 5.1 Captura de pantalla durant la reproducció del flux amb VLC.

CAPÍTOL 6. CONCLUSIONS I LÍNIES FUTURES D'INVESTIGACIÓ

6.1 Conclusions finals

Davant la perspectiva de l'actual creixement dels sistemes de difusió de continguts audiovisuals en format digital, aquest TFC s'ha centrat en obtenir una aplicació informàtica de codi obert i ús lliure capaç de generar fluxos de transport MPEG-2, orientats a la difusió de programes audiovisuals digitals mitjançant els estàndards de DVB. Per aconseguir-ho, s'ha partit d'una aplicació ja existent, l'Mplex 13818, que no funcionava correctament.

Un cop assimilada tota la informació relacionada amb els estàndards que defineixen la codificació dels fluxos de transport, una part molt important del treball ha estat l'anàlisi del codi del multiplexor, línia a línia, que ha suposat molts mesos de feina a causa del gran nombre de fitxers de codi (més de 20) i de la poca informació que es dona sobre les variables i els algoritmes, en forma de comentaris. En aquest sentit, s'ha pres consciència de la importància de realitzar una documentació correcta del codi, en forma de comentaris, durant el desenvolupament de qualsevol aplicació per tal que altres programadors siguin capaços d'entendre'l i modificar-lo en un futur. Com ja s'ha dit anteriorment, a llarg termini es pretén muntar un escenari de proves complet utilitzant multiplexació estadística i algoritmes de multiplexació més eficients del que presenta Mplex; s'ha documentat el codi generat per tal de que aquesta tasca pugui ser desenvolupada per altres persones amb més facilitat.

L'exhaustiu anàlisi que s'ha realitzat, a més de servir per descriure detalladament l'algoritme de multiplexació i per trobar problemes en la implementació, ha permès definir de forma precisa els punts del codi concrets que s'havien de modificar, amb la qual cosa la posterior implementació de les millores ha estat relativament senzilla. Les modificacions en el codi s'han realitzat respectant el màxim possible l'estructura original, i utilitzant noms per a les funcions i variables coherents amb la resta.

Per comprovar els resultats de les millores s'ha instrumentat el codi d'Mplex per tal d'extreure'n la informació necessària per valorar els resultats. També s'ha utilitzat com a referència els *logs* proporcionats pel reproductor VLC. Els resultats han estat satisfactoris, havent aconseguit que els fluxos de transport generats amb Mplex 13818, que inicialment no es podien reproduir amb cap reproductor MPEG per problemes de sincronisme, es reprodueixin correctament. Així doncs, aquesta aplicació permetrà que qualsevol persona pugui difondre continguts audiovisuals propis sense que això li suposi cap cost econòmic, per exemple a través d'Internet i seguint les especificacions de DVB-IP.

6.2 Estudi d'ambientalització

Els esforços en la obtenció d'algoritmes de multiplexació de fluxos de transport MPEG-2 ha permès que la radiodifusió massiva de continguts audiovisuals sigui possible. Aquest tipus de senyals poden transportar diversos programes audiovisuals en un únic flux de transport, que un cop modulat i emès a l'aire seguint els estàndards de DVB ocupen menys ample de banda del que ocupava un únic canal analògic, permetent un ús més eficient de l'espectre electromagnètic. D'altra banda, la potència necessària per descodificar correctament un senyal DVB, en què hi viatja el flux de transport, és molt menor que la necessària per veure amb una qualitat raonable un programa difós amb senyal analògic, la qual cosa suposa una reducció considerable de la contaminació radioelèctrica.

Concretant en el treball fet durant aquest TFC, qualsevol pas cap a la millora dels algoritmes de multiplexació redunda en un ús més eficient de l'espectre, i a la subsegüent disminució del número de múltiplexs necessaris per emetre una certa oferta de programes de TV, el que porta a una disminució del consum de potència radioelèctrica.

6.3 Línies futures d'investigació

A partir de la feina realitzada en aquest TFC, queden oberts una sèrie d'aspectes que es poden tractar en un futur per seguir millorant l'Mplex, i que s'exposen a continuació:

- Anàlisi, proves i millores, en cas necessari, de l'aplicació Mplex 13818 quan els fluxos d'entrada són fluxos PS o PES.

- Estudi de la possibilitat d'implementar un multiplexor estadístic, partint d'Mplex, que realimenti els codificadors de la capa de compressió MPEG-2 amb la finalitat de modificar-ne el *bit rate* de sortida en funció de l'ample de banda disponible en el flux de transport, reservant més espai per als programes que ho requereixen, amb imatges que canvien ràpidament, en detriment dels que, per les característiques del contingut que s'està emetent, es poden comprimir més sense perdre qualitat.

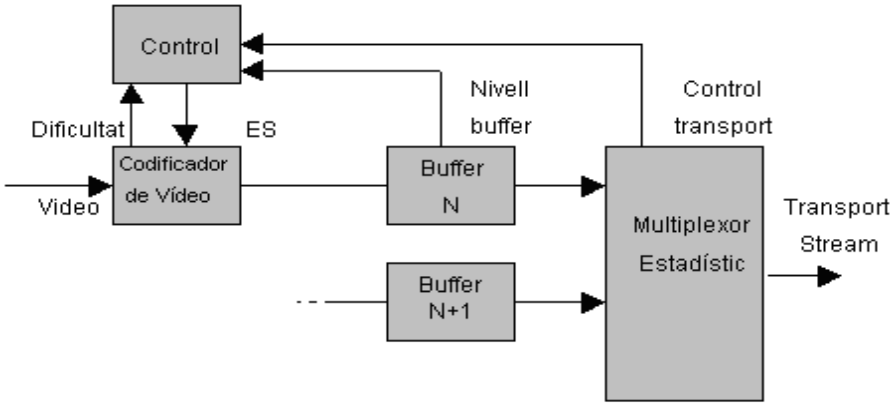


Figura 6.1 Esquema d'un sistema de multiplexació estadística. Extret de [M5]

- Implementació de la capa física de DVB-IP, permetent que els fluxos de sortida de Mplex es puguin transportar directament a través d'una xarxa IP. Estudi de l'efecte del *jitter* d'arribada de paquets de transport en la sincronització del flux.

- Estudi del comportament d'Mplex multiplexant fluxos de vídeo d'alta definició, comprimits en H.264.

- Alguns proveïdors de serveis aprofiten l'ample de banda sobrant dels fluxos de transport, habitualment omplert amb paquets de transport nuls, per encapsular dades IP i oferir servei d'accés a la xarxa Internet (quan el medi de difusió no és la pròpia xarxa). Aquest ample de banda sobrant és molt variable i depèn del nivell de compressió dels diferents fluxos elementals en cada moment. Es pot estudiar la possibilitat d'afegir aquesta funcionalitat al *software*, buscant algun mecanisme que proporcioni un ample de banda sobrant el més constant possible, per tal d'aconseguir una connexió IP de bona qualitat.

BIBLIOGRAFIA

Llibres

[L1] Chen, X., *Transporting Compressed Digital Video*, Kluwer Academic Publishers, Dordrecht (2002)
[L2] Benoit, H., *Digital television : MPEG-1, MPEG-2 and principles of the DVB system*, Focal Press, Oxford (2002)
[L3] Reimers, U., *DVB : the family of international standards for digital video broadcasting*, Springer, Berlin (2005)
[L4] Mitchell, J. L., *MPEG video compression standard*, Kluwer Academic Publishers, New York (2002)

Estàndards

[E1] ISO/IEC 13818-1, MPEG-2, *Generic coding of moving pictures and associated audio*, *information: Systems*, Segona edició (2000)
[E2] ISO/IEC 13818-2, MPEG-2, *Generic coding of moving pictures and associated audio*, *information: Video*, Segona edició (2000)
[E3] ETSI EN 300 468, *Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems*, 2009

Material Docent i acadèmic

[M1] Tarrés, F., *Sistemes de vídeo: Principis de compresió dels senyals audiovisuals, codificació entròpica i transformada cosinus, tècniques de compensació de moviement*, Universitat Politècnica de Catalunya (2008)
[M2] Delgado Gutiérrez, A., *Flujos de programa y de transporte MPEG-2. Aplicación a DVB*, Universidad Politécnica de Madrid (2001)
[M3] Gurrero, D, *Informe definitivo sobre el software de multiplexación Mplex 13818*, Activitat de soport a la docència, EPSC (2005)
[M4] Guerrero, D., *Sistema de temporització a DVB*, EPSC (2005)
[M5] Departament de vídeo i televisió, *TV Digital, Transport Stream, DVB-SI*, (http://www.salle.url.edu/Eng/elsDTA/elsVideo/webts), La Salle

PFC/TFCs

[P1] Piqueras Sáez, O., *Codificación de video según MPEG,* Directors: Olga Casals, John Cosmas, UPC (Escola Tècnica Superior d'Enginyers de Telecomunicació de Barcelona, 1994)

Pàgines Web

[W1] Transport Stream - Wikipedia (http://es.wikipedia.org/wiki/Transport_Stream)
[W2] Digital Video Broadcasting - Wikipedia (http://es.wikipedia.org/wiki/DVB)
[W3] Televisión Digital Terrestre - Wikipedia (http://es.wikipedia.org/wiki/TDT)

[W4] MovingPicturesExperts Group - Wikipedia (http://es.wikipedia.org/wiki/MPEG)
[W5] MPEG-2 – Wikipedia (http://es.wikipedia.org/wiki/MPEG-2)
[W6] DVB – Digital Video Broadcasting (http://www.dvb.org)
[W7] ETSI (http://www.etsi.org)
[W8] TS Reader (http://www.tsreader.com/tsreader)
[W9] MPEG-2 TS Packet Analyzer (http://www.pjdaniel.org.uk/mpeg)
[W10] VLC Media Player (http://www.videolan.org/vlc)
[W11] Anjuta (http://projects.gnome.org/anjuta)
[W12] Mplex 13818 (http://scara.com/~schirmer/o/mplex13818)
[W13] ISDB (http://www.dibeg.org)
[W14] ASTC (http://www.atsc.org)

ANNEX A: GLOSARI TERMINOLÒGIC

Codis de *Huffman*:
Codis de longitud variable generats mitjançant un algoritme específic que tenen la propietat que la cadena de bits que representa un determinat símbol no serà mai prefix de la cadena de bits d'un altre símbol.

Compressió:
Reducció del nombre de bits necessaris per representar una determinada informació.

Constant Bit Rate (CBR):
Taxa de bit constant.

Decoding Time Stamp (DTS):
En la capçalera d'un paquet PES, marca temporal que determina en quin moment s'ha de descodificar la unitat d'accés associada.

Elementary Stream (ES):
Terme que s'utilitza per referir-se a un flux de vídeo o àudio comprimit amb MPEG-2.

European Telecommunication Standards Institute (ETSI):
Organització reconeguda per la Unió Europea que promou estàndards relacionats amb el món les Tecnologies de la Informació i la Comunicació (TIC).

Unitat d'accés:
Resultat de la compressió MPEG-2 d'una unitat de presentació.

Variable Bit Rate (VBR)
Taxa de bit variable.

Packet Identifier (PID):
Nombre enter de 13 bits que identifica un flux elemental dins un *Transport Stream*.

Packetized Elementary Stream (PES):
Flux de dades resultat de l'empaquetatge de les dades d'un ES.

Presentation Time Stamp (PTS):
En la capçalera d'un paquet PES, marca temporal que determina en quin moment s'ha de presentar la unitat d'accés associada.

Programa:
Conjunt de fluxos elementals que es presenten de forma simultània. Un programa audiovisual està format per un únic vídeo i un o més àudios, entre altres. Els diferents fluxos elementals d'un programa tenen una base de temps comuna per aconseguir una presentació sincronitzada.

Program Clock Reference (PCR):
En un *Transport Stream*, mecanisme que proporciona MPEG-2 per sincronitzar el rellotge del descodificador amb el rellotge que s'ha utilitzat per codificar un determinat programa.

Program Specific Information (PSI):
Informació organitzada en taules i que viatja en els *Transport Streams* per tal que el descodificador sigui capaç de descodificar correctament un determinat programa.

System Clock Reference (SCR):
En un *Program Stream*, mecanisme que proporciona MPEG-2 per sincronitzar el rellotge del descodificador amb el rellotge que s'ha utilitzat per codificar el programa.

System Target Decoder (STD):
Model hipotètic de descodificador utilitzat per definir les restriccions que han de complir els fluxos de programa i de transport definits en l'ISO 13818-1.

ANNEX B: SYSTEM TARGET DECODER (STD)

El *System Target Decoder* és un model de referència hipotètic de descodificador utilitzat el l'ISO 13818 per descriure les restriccions que ha de complir un flux de transport o de programa (T-STD o P-STD). Totes les restriccions estan parametritzades en funció del tipus de flux elemental que s'ha de descodificar (per exemple, la mida dels *buffers*). Els codificadors hauran de generar fluxos de transport que compleixin aquestes restriccions, mentre que els descodificadors reals hauran de compensar d'alguna manera les diferències entre el seu disseny i l'STD.

El sistema de temporització dels fluxos, explicat anteriorment, es defineix a partir del STD (sincronització amb el rellotge de programa, temps d'arribada de cada *byte*, i *stamping* de les UA en els paquets PES). A continuació es descriu el sistema de *buffers* que formen el descodificador hipotètic.

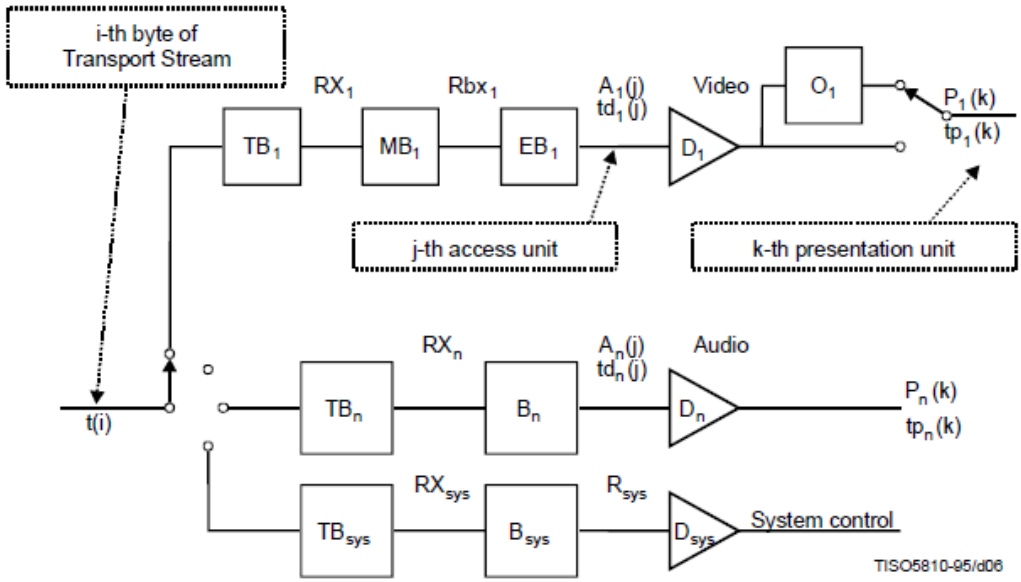


Figura B.1 Esquema de *buffers* del descodificador hipotètic [E1]

Com es pot observar a la figura B.1, els *buffers* del STD es defineixen en funció del tipus de flux elemental d'entrada (vídeo, àudio o control). L'explicació se centrarà en els *buffers* de vídeo i àudio, que són els més delicats a nivell de temporització i *buffering*.

T-STD de vídeo

- *Buffer* TB (*Transport Buffer*)

Al *buffer* TB hi entren els paquets de transport que porten el vídeo d'un programa, amb un determinat PID. La cadència d'arribada de paquets

dependrà del *bit rate* del flux de transport, que serà constant, però també de la disposició dels paquets en el múltiplex. Tenint en compte aquest segon punt, el *bit rate* d'entrada al *buffer* TB serà variable i dependrà únicament de les característiques del *Transport Stream* d'entrada. El *payload* dels paquets de transport (fragments de paquets PES) passa al següent *buffer* (MB) a velocitat R_X, sempre que el *buffer* TB no estigui buit. Aquesta velocitat està tabulada al estàndard 13818-2 i depèn del tipus de flux MPEG. Concretament,

Si el buffer TB no està buit, $R_X = 1.2\ R_{MAX}$ [profile, level]
Altrament, $R_X = 0$

on R_{MAX} és un paràmetre que s'especifica d'acord amb la taula 8-13 del ISO 13818-2 i que depèn dels paràmetres *profile* i *level* de la codificació MPEG. Segons aquests requeriments, en el *buffer* TB mai es produirà *underflow*, ja que si està buit la velocitat de transmissió cap al següent *buffer* serà zero; únicament s'ha de vetllar per tal que no es desbordi, és a dir, per a que el *bit-rate* d'entrada sigui, en mitjana, menor o igual que R_X. A més, s'ha de complir que el *buffer* estigui buit almenys un cop per segon; en l'estàndard no s'explica el motiu d'aquesta restricció. Té una mida fixa de 512 bytes.

Taula B.1 Taula a partir de la qual s'especifica el valor de R_{MAX}. Extret de [W2]

Table 8-13. Upper bounds for bit rates (Mbit/s)

Level	Profile				
	Simple	Main	SNR	Spatial	High
High		80			100 all layers 80 middle + base layer 25 base layer
High-1440		60		60 all layers 40 middle + base layers 15 base layer	80 all layers 60 middle + base layers 20 base layer
Main	15	15	- 15 both layers 10 base layer		20 all layers 15 middle + base layer 4 base layer
Low		4	- 4 both layers 3 base layer		

- *Buffer* MB (*Multiplexing Buffer*)

El *buffer* MB emmagatzema els paquets PES procedents dels paquets de transport del *buffer* TB. El *bit rate* d'entrada a MB és R_X, i dependrà de la cadència d'arribada de paquets de transport al primer *buffer*. Si el TS compleix les condicions imposades per TB, Rx serà zero almenys un cop per segon. La càrrega útil dels paquets PES (unitats d'accés) passarà al següent *buffer* (EB). Hi ha dos mecanismes per controlar la velocitat a la que es realitza aquest

traspàs de dades: *leak* i *vbv delay*. És el codificador MPEG qui determina quin dels dos mètodes s'utilitzen, i ho especifica en camps de control del flux ES.

- Mètode "Leak"

El mètode *leak* consisteix en que la informació passa al *buffer* EB a velocitat R_{BX} mentre aquest tingui espai disponible, on R_{BX} depèn de la naturalesa del flux MPEG (paràmetres *level* i *profile*).

Per als nivells *Low* i *Main*,

$$R_{BX} = R_{MAX}\,[profile, level]$$

Per als nivells High-1440 i High

$$R_{BX} = \min\{1.05\,R_{ES},\ R_{MAX}\,[profile,\ level]\}$$

Si s'utilitza aquest mecanisme, MB no podrà patir *underflow*, ja que ningú li demana dades, sinó que és ell qui passa la informació al següent *buffer* si aquest últim no està ple. S'ha de controlar que no es desbordi i que estigui buit almenys un cop per segon.

- Mètode "VBV delay"

Si s'utilitza aquest mètode, és el propi codificador MPEG qui indica al descodificador l'instant en que ha de passar la unitat d'accés al *buffer* EB. Això ho fa mitjançant el paràmetre vbv_delay, que viatja en l'ES de vídeo. Aquest camp indica amb quanta antelació respecte l'instant de descodificació, determinat pel *timestamp*, ha de passar al *buffer* ES la unitat d'accés. És a dir, la unitat d'accés passarà de MB a EB a l'instant:

$$td(j) - vbv\_delay(j),$$

on td és l'instant de descodificació determinat pels *timestamps*, i *j* és l'índex apunta a una UA concreta. En aquest cas sí que s'ha de controlar que no es produeixi *underflow* en el *buffer* MB. Evidentment, tampoc s'ha de desbordar.

Per altra banda, la mida de MB (MBS) està definida per les següents fórmules:

$$MBS = BSmux + BSoh$$

$$BSmux = 0.004\ seg \times R_{MAX}\ (multiplex\ buffering)$$

$$BSoh = (1/750)\ seg \times R_{MAX}\ (overhead\ buffering)$$

on vbv_buffer_size és un paràmetre definit pel codificador i que viatja en el flux MPEG, i VBV_max és un paràmetre definit a l'estàndard ISO11172-2.

- *Buffer EB* (*ES Buffer*)

Aquest *buffer* és el que conté les unitats d'accés que passen al descodificador a l'instant indicat pels *timestamps*. La seva mida ve determinada pel camp *vbv_buffer_size* de la capçalera *sequence_header* del flux de vídeo, especificat pel codificador. S'ha d'evitar que aquest *buffer* es desbordi, ja que suposaria una pèrdua d'informació, i en general també s'ha d'evitar una situació de falta d'informació, que provocarà una interrupció en la reproducció del flux excepte si es treballa en mode "*low_delay*" o "*trick_mode*".

T-STD d'àudio

El T-STD d'àudio és més senzill que el de vídeo i està format únicament per dos *buffers.* El *buffer* TB es defineix exactament de la mateixa manera que en el T-STD de vídeo, i la informació passa al següent (B) utilitzant el mateix mecanisme. El *buffer* B emmagatzema el *payload* dels paquets PES, és a dir, les unitats d'accés d'àudio. Resumint, els requeriments del descodificador d'àudio són els següents:

- S'ha d'evitar que el *buffer* TB es desbordi. No hi ha possibilitat de que pateixi *underflow*. També ha d'estar buit almenys un cop per segon.
- El *buffer* B no s'ha de desbordar i s'han d'evitar situacions d'*underflow*.

Tant per fluxos de vídeo com d'àudio, totes les dades que travessen el STD han de ser descodificades amb un retard màxim entre l'entrada al primer *buffer* i la sortida del darrer d'1 segon, amb excepció de les imatges fixes i els *streams* ISO/IEC 14496, amb un retard màxim de 60 i 10 segons, respectivament. En l'ISO 13818-1 no s'explica el motiu d'aquestes restriccions.

ANNEX C: ANÀLISI DEL CODI D'MPLEX 13818

Per tal de descriure l'algoritme d'Mplex 13818 i treure les conclusions exposades en el capítol 4, s'ha realitzat un anàlisi exhaustiu de les parts més importants del codi. Primerament, s'ha estudiat el diagrama de flux del bucle principal, escrit en el mòdul *Dispatch*, per veure en quin ordre es duen a terme els diferents passos descrits en el punt 4.4 de la memòria. A continuació s'han analitzat, línia a línia, les funcions més importants del programa. La majoria contenen una breu descripció feta per de l'autor en el propi codi, on s'explica quin paper juguen en el procés de multiplexat, i s'especifiquen els paràmetres formals i el valor de retorn. El que no s'indica enlloc és l'algoritme que utilitzen ni com es veuen modificades les variables que es passen com a paràmetres de la funció, aspecte clau per entendre la relació entre els diferents blocs de codi i poder descriure el funcionament global de l'algoritme de Mplex.

Com a primera aproximació per fer-se una idea de com s'organitza la feina entre els diferents mòduls, s'ha realitzat el següent esquema on es pot veure la relació funcional entre ells.

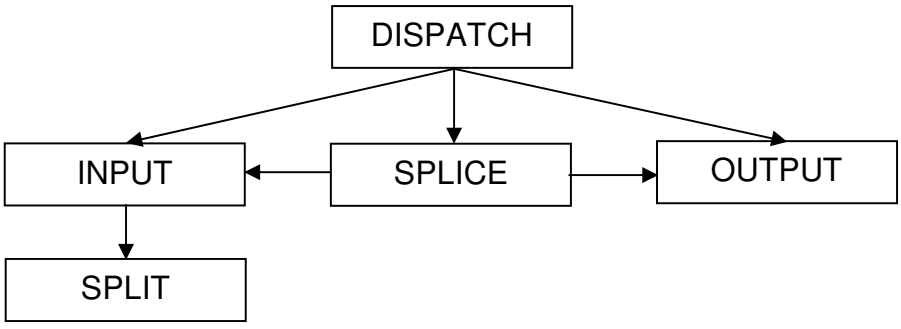


Figura C.1. Relació funcional entre els mòduls

La figura C.1 mostra com el mòdul *Dispatch* crida funcions de tots els altres mòduls, excepte les del mòdul *Split*. És el propi mòdul *Input* el que, a petició de *Dispatch*, s'ajuda de les funcions de *Split* per fer passar les dades dels diferents *raw buffers* cap als *input buffers* corresponents. D'altra banda, s'observa que el mòdul *Splice* també es relaciona amb *Input*, per demanar que obri o tanqui un determinat ES, en funció de si aquest es lliga o deslliga d'un programa de sortida, i amb *Output*, en el moment en que *Splice* demana que es comprovi la disponibilitat d'espai en el *buffer* de sortida, i que es reservi l'espai necessari per disposar-hi un paquet. En general, és el mòdul *Dispatch* qui s'encarrega de distribuir la feina i d'establir les relacions entre la resta de mòduls.

C.1. Bucle principal

A continuació s'estudia el diagrama de flux del bucle principal. Primerament, es descriuen les variables locals que s'utilitzen en el mòdul *Dispatch*, i com són inicialitzades. Tot seguit, es determinen les condicions de sortida del bucle

principal, es presenta un diagrama de flux on es veu la seqüència de passos que es realitzen, i s'explica l'algoritme pas a pas.

C.1.1. Descripció de les variables locals

- boolean bi, bo, bs;

bi: Se li assigna el valor de retorn de la funció *input_acceptable()*
TRUE > Almenys un dels *raw buffers* té espai per acceptar dades.
FALSE > Altrament.

bo: Se li assigna el valor de retorn de la funció *output_available()*
TRUE > Hi ha informació disponible al *buffer* de sortida.
FALSE > Altrament.

bs: Se li assigna el valor de retorn de la funció *split_something()*
TRUE > S'ha extret informació d'algun TS/PS
FALSE > No hi ha informació o espai disponible (al *raw buffer* o *input buffer*, respectivament).

- stream_descr *st;

Punter a l'estructura *stream_descr*, que defineix un ES. S'utilitza per manejar fluxos elementals al bucle principal. No conté els fluxos elementals oberts, aquests estan declarats al mòdul *Input*. Se li assigna el valor de retorn de la funció *input_available()*, que correspon al *stream_descr* que conté el proper paquet PES que s'ha d'encadenar.

- t_msec tmo;

Variable del tipus int32, utilitzada com a *timeout* en la funció *poll()* de la llibreria *poll.h*, que s'explica més endavant.

- unsigned int nfds, onfds, infds;

nfds: Nombre d'estructures *pollfd* que conté el vector *ufds*.
onfds: Índex del vector *ufds* corresponent al descriptor de sortida.
infds: Nombre de descriptors d'entrada que conté el vector *ufds*.

- struct pollfd ufds [MAX_POLLFD];

Vector que apunta a estructures del tipus *pollfd*, definides en la llibreria *poll.h*. Aquesta estructura conté un *file descriptor* i una sèrie de màscares, usades en la funció *poll()* de la mateixa llibreria per especificar condicions de lectura/escriptura.

- int pollresult;

Se li assigna el valor de retorn de la funció *poll()*.

C.1.2. Inicialització de variables

Abans d'entrar en el bucle principal, s'inicialitzen les variables locals de la següent manera:

bs = false;	No s'ha extret informació.
st = input_available;	Es determina el proper ES a ser encadenat (NULL).
nfds = 0;	No hi ha cap descriptor de fitxer obert.
onfds = nfds;	S'inicialitza l'índex del descriptor de sortida a zero.
command_expected (&nfds, &ufds[0]);	Es llegeix la primera comanda que s'ha introduït per teclat.
bo = output_available (&nfds, &ufds[onfds], &tmo);	Es comprova si hi ha dades disponibles al *buffer* de sortida (*false*).

C.1.3. Condicions de sortida

El bucle dura mentre es compleix almenys alguna d'aquestes condicions:

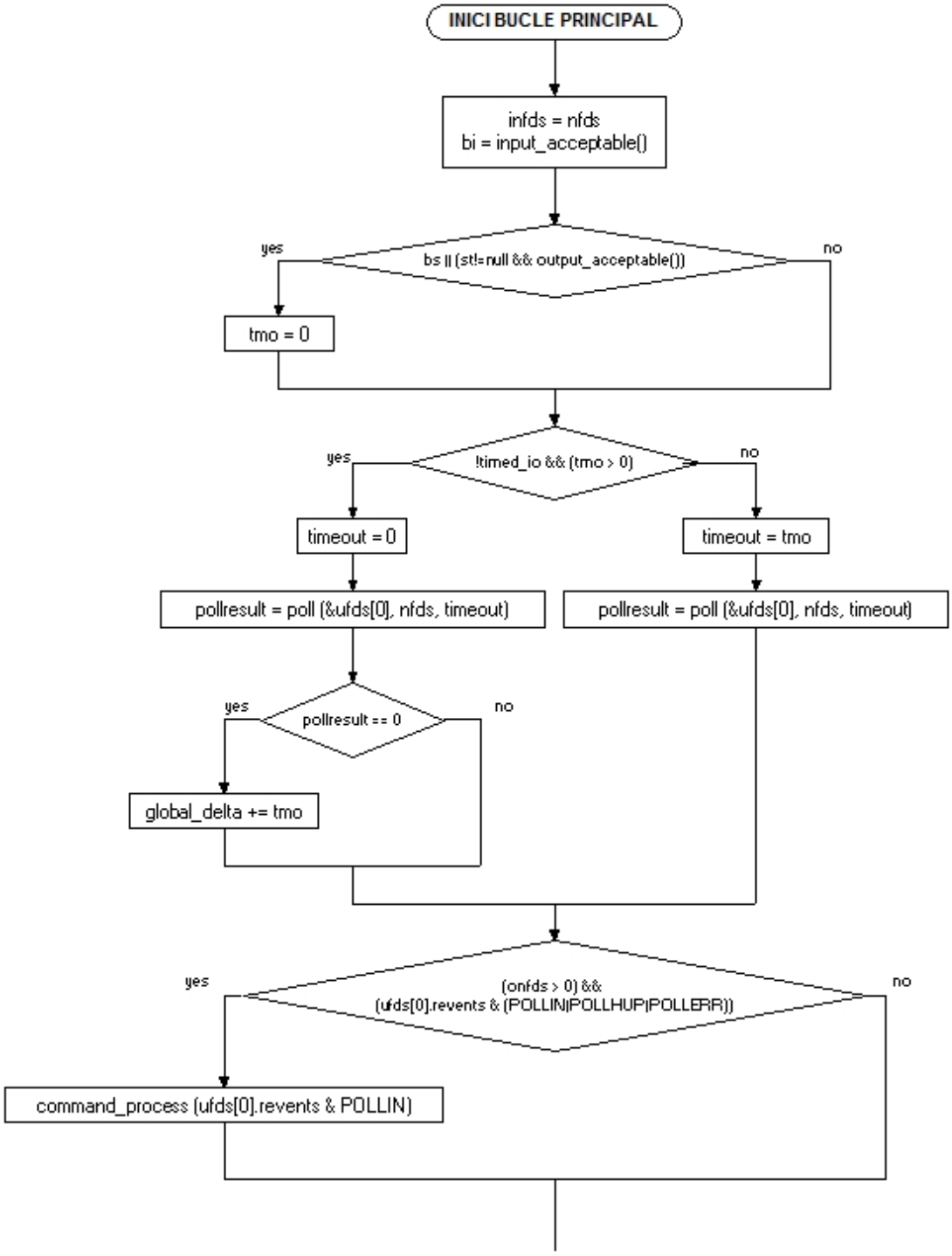
bo = true	Hi ha informació disponible al *buffer* de sortida.
bs = true	S'ha extret/s'ha d'extreure informació d'algun flux.
st != null	Hi ha algun ES preparat per ser "empalmat" en aquesta passada del bucle.
input_expected() = true	S'espera informació d'alguna de les fonts d'entrada.
(tmo >= 0) && (tmo <= MAX_MSEC_OUTDELAY)	El *timeout* és positiu i menor que el retard de sortida màxim.
busy_work = true	

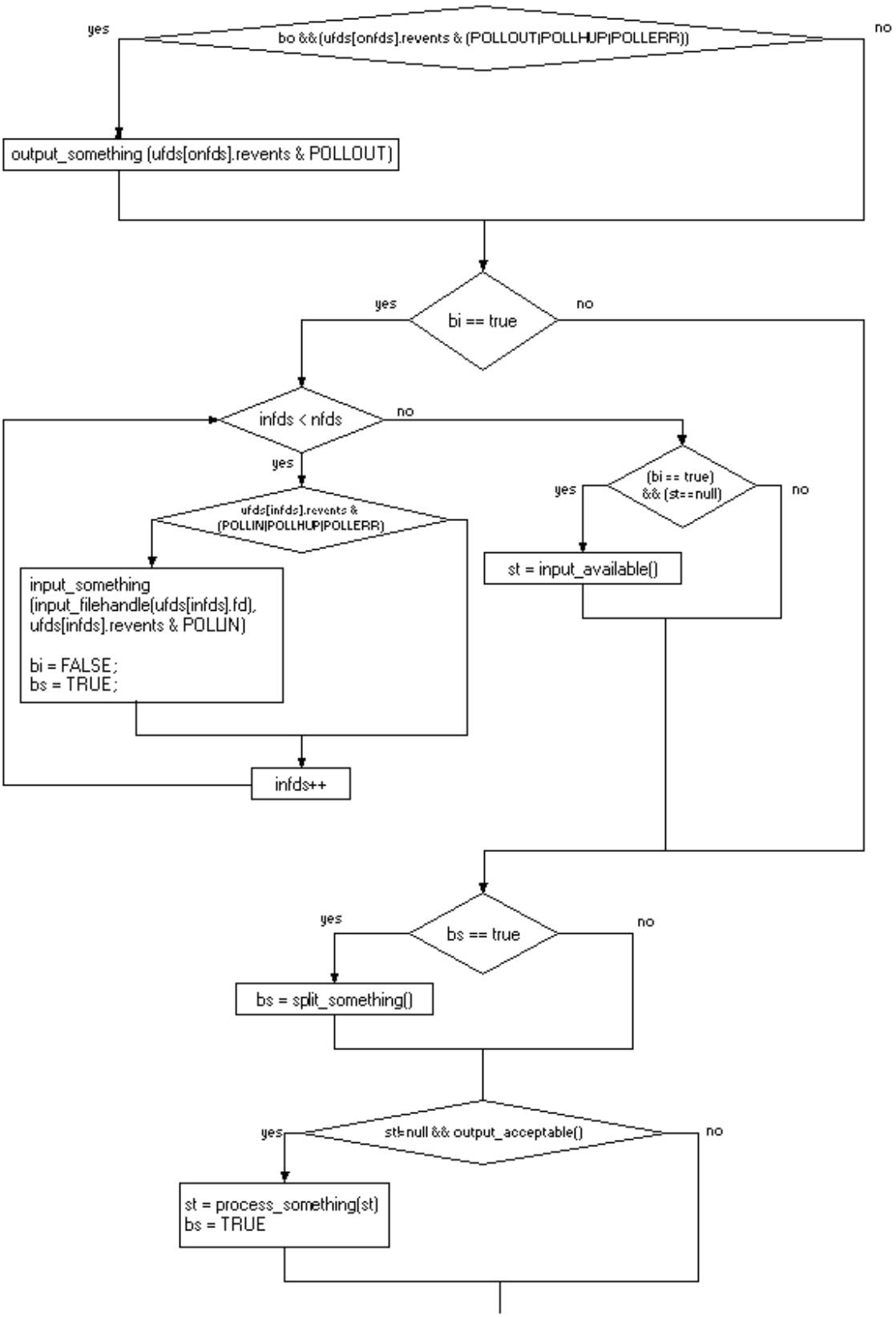
Alhora, s'han de complir aquestes dues condicions:

- fatal_error = false
- force_quit = false
    - Variables de control del procés.

C.1.4. Diagrama de flux

El bucle principal és on es defineix la seqüència de passos que realitza el multiplexor per tal d'anar processant la informació d'entrada i generant el flux de sortida. Abans d'explicar les accions que es duen a terme en el bucle, es presenta un diagrama de flux on es representa l'algoritme de forma visual.




yes
bo &&(ufds[onfds].revents & (POLLOUT|POLLHUP|POLLERR))
no
output_something (ufds[onfds].revents & POLLOUT)
yes
bi == true
no
infds < nfds
no
yes
(bi == true)
&& (st==null)
yes
no
ufds[infds].revents &
(POLLIN|POLLHUP|POLLERR)
st = input_available()
input_something
(input_filehandle(ufds[infds].fd),
ufds[infds].revents & POLLIN)
bi = FALSE;
bs = TRUE;
infds++
yes
bs == true
no
bs = split_something()
yes
st=null && output_acceptable()
no
st = process_something(st)
bs = TRUE

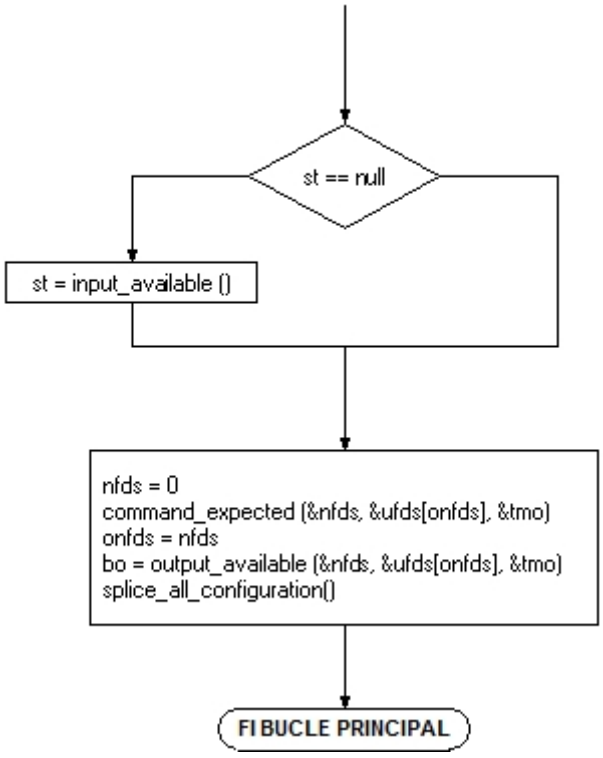


Figura C.2. Diagrama de flux del bucle principal

C.1.5. Llibreria poll.h

Com es pot observar en el diagrama, el codi utilitza la funció *poll()*, definida en la llibreria *poll.h*. Abans d'analitzar pas a pas l'algoritme del codi principal, s'explica el funcionament d'aquesta.

La llibreria *poll.h*, especificada a UNIX, proporciona una sèrie de mecanismes que permeten la multiplexació d'entrada i sortida de dades sobre un conjunt de *file descriptors*. Està formada pels següents membres:

```
struct {
  int fd                //the following descriptor being polled
  short int events      //the input event flags
  short int revents     //the output event flags
}pollfd;

int poll(struct pollfd fds[], nfds_t nfds, int timeout);
```

L'estructura *pollfd* està formada per tres camps: *fd*, que identifica el descriptor de fitxer; *events*, que especifica una sèrie de condicions de lectura/escriptura a l'entrada de la funció *poll()* mitjançant màscares construïdes a partir de la combinació d'una sèrie de *flags* (definits en la mateixa llibreria); i *revents*, que conté la màscara de sortida de la funció.

A la funció *poll()* se li passen diferents paràmetres. La variable *nfds*, del tipus *nfds_t* (definit com un enter sense signe), indica la longitud del vector *fds[]*. Cada membre del vector apuntat per *fds[]* és una estructura del tipus *pollfd*, que especifica un *file descriptor (fd)*, i una màscara *events* a l'entrada. A la sortida de la funció, *poll()* utilitzarà la variable *revents* per indicar si les condicions d'entrada especificades per *events* es compleixen o no. Com s'ha dit, les màscares són creades a partir de combinacions de diferents *flags* definits en la mateixa llibreria. De tots els possibles, Mplex utilitza els següents:

POLLIN
: Es poden llegir dades 'normals' i 'prioritàries' sense bloqueig.

POLLOUT
: Es poden escriure dades "normals" sense bloqueig.

POLLERR
: S'ha produït un error al dispositiu o *stream*. Aquest *flag* només és vàlid en la màscara *revents*. Si apareix a la màscara *events* serà ignorat.

POLLHUP
: El dispositiu ha estat desconnectat. Només és vàlid en la màscara *events*.

Els conceptes de dades *normals* i *prioritàries* són específics per a un determinat tipus de fitxer/dispositiu.

Per a cada estructura *pollfd*, la funció *poll()* posarà a la màscara *revents* els *flags* especificats a *events* que es compleixin, a més dels *flags* POLLERR o POLLHUP en cas que es produeixin errors. S'entén que un descriptor està "seleccionat" a la sortida de la funció si el camp *revents* és diferent de zero.

Si en el moment de cridar la funció *poll()* no es compleix cap de les condicions definides per *events* en cap dels *file descriptors*, s'espera dins la funció per si es produeix algun dels *events* en algun dels descriptors, durant *timeout* ms, com a màxim. Si el valor de *timeout* és 0, la funció tornarà instantàniament (sense esperar), i si és –1, no sortirà d'aquesta fins que es produeixi un dels *events* en algun dels descriptors, o fins que la crida sigui interrompuda. La funció *poll()* suporta fitxers, terminal, dispositius d'emmagatzematge, *FIFOs*, *pipes*, *sockets* i *STREAM-based files*.

Si es produeix algun error, la funció retorna –1. En cas contrari, retorna un valor positiu que indica el nombre total de *file descriptors* seleccionats, és a dir, els nombre de *file descriptors* el valor de *revents* dels quals és diferent de zero. Si el valor de retorn és 0, indica que ha passat el *timeout* i per tant no hi ha cap descriptor seleccionat.

C.1.6. Descripció de l'algoritme

En aquest punt es descriu l'algoritme utilitzat en el bucle principal, a partir del diagrama de flux presentat anteriorment. S'expliquen els passos en l'ordre en que es realitzen en cada passada de bucle. No s'explicaran els punts del codi

relacionats amb el mòdul *Command*, és a dir, amb la interfície entre el *software* i l'usuari.

1. *bi = input_acceptable (&nfds, &ufds[infds], &tmo, output_acceptable ());*

   Es comprova si pot es acceptar informació a l'entrada del multiplexor. En cas que hi hagi un o més *raw buffers* amb espai suficient, es preparen les estructures *pollfd* adequadament, especificant les condicions de lectura (POLLIN) per a cada un dels *file_descr*. En aquest cas, la funció torna *true*. Si no hi ha espai en caps dels *raw buffers*, el valor de retorn és *false*. A més, en cas que el *buffer* de sortida no estigui ple, es calcula el valor de *timeout*, i s'assigna a la variable *tmo*. Si el *buffer* de sortida està ple, s'utilitzarà el valor de *timeout* calculat en el punt 12, en la funció *output_available()* (en la passada de bucle anterior). Aquest valor correspon al temps que queda fins que s'ha de treure el proper paquet PES a la sortida, i per tant és el temps màxim durant el qual es pot esperar per llegir alguna cosa.

2. *if (bs || ((st != null) && output_acceptable())*

   Es comprova si hi ha alguna cosa a fer pendent de la passada de bucle anterior: informació extreta o per extreure (*bs == true*), o algun ES seleccionat per ser encadenat a l'*stream* de sortida (*st != null*), havent-hi espai disponible al *buffer* de sortida. En cas afirmatiu, en aquesta passada del bucle es pot fer "alguna cosa" sense llegir nova informació, i per tant, no tindria sentit esperar-se dins la funció *poll()* fins que es complís alguna de les condicions de lectura especificades (en cas que no se'n compleixi cap inicialment); així, es fa l'assignació *timeout = 0*. D'altra banda, es comprova el valor de *timed_io*. Aquesta variable, per defecte té assignat el valor de *false,* excepte si l'usuari utilitza l'opció '*--timed*' ("*force delay timing even if solely diskfiles in use"*). Com que no s'ha utilitzat en les proves realitzades, es considera només el cas en que *timed_io == false*. Així, veient el diagrama de flux, el valor de *timeout* utilitzat en la funció *poll()* serà sempre zero, independentment dels valor de *timeout* calculats en els punts 1 i 12.

3. *pollresult = poll (&ufds[0],nfds,timeout)*

   Mitjançant la funció *poll()*, es comprova si es compleixen les condicions de lectura/escriptura dels descriptors de fitxer especificades als punts 1 i 12.

4. En cas que no es compleixi la condició de POLLIN en cap dels descriptors de fitxers d'entrada, ni tampoc la condició POLLOUT en el descriptor de fitxer de sortida, i que no hi hagi res a fer pendent de l'etapa anterior (punt 2) (és a dir, que en aquesta passada de bucle no es podrà llegir ni processar nova informació, ni treure'n a la sortida), incrementa el valor de retard global (*global_delta*) en '*tmo*' ms, essent *tmo* el valor calculat en els punts 1 i 12 (si en el punt 1 no es calcula, s'utilitza el del punt 12). El valor de *global_delta* s'utilitza per ajustar la referència interna de temps. Per obtenir el valor del rellotge del multiplexor en un instant determinat, s'utilitza

la funció *msec_now()* que proporciona el temps del sistema actual en segons relatius (començant en 0 ms a l'inici de l'execució del programa). A aquesta referència, se li suma el valor de *global_delta*. El valor de *'tmo'* calculat, és el temps durant el qual no s'ha de treure res a la sortida. Per tant, la idea seria que si en un determinat moment no es pot llegir ni extreure informació, i fins d'aquí *'tmo'* ms no es pot treure res, es fa un salt en el rellotge de *'tmo'* ms per tal de "no perdre temps" en l'execució.

5. *if (bo && ((ufds[onfds].revents & (POLLOUT|POLLHUP|POLLERR)))*

   Si hi ha informació disponible de l'etapa anterior al *buffer* de sortida (l'assignació de *bo* es realitza en el punt 12), i el fitxer de sortida compleix les condicions especificades per la màscara *events* (POLLOUT*)*, treu aquesta informació del *buffer* cap a la sortida del multiplexor. En cada passada de bucle, es treu la informació generada en l'anterior, sempre i quan es pugui escriure en el fitxer destí.

6. *if (ufds[].revents & (POLLIN|POLLHUP|POLLER))*

   Per a tots els *file_descr* que en el punt 1 s'ha determinat que poden acceptar nova informació, es comprova la màscara *revents* de les estructures *pollfd*, modificades per la funció *poll()* en el punt 3, amb la intenció de veure si les diferents fonts d'informació estan preparades per a que les dades siguin llegides.

7. *input_something (input_filehandle(ufds[].fd, ufds[].revents & POLLIN)*

   Per a tots els *file_descr* que poden acceptar nova informació i que les seves fonts estan preparades, es crida la funció *input_something()*, que s'encarrega de copiar les dades d'una font d'entrada cap al *raw buffer* corresponent. Si s'ha llegit informació d'alguna font d'entrada, es realitza l'assignació:

   bs = TRUE;
   bi = FALSE;

   Indicant que ja no hi ha espai disponible (*bi = false*), i que probablement hi ha informació preparada per ser extreta (*bs = true*).

8. *if (!bi && st == NULL)*

   Si ha entrat informació nova en algun dels *raw buffers* (*!bi*), i no hi ha cap ES preparat per ser empalmat pendent de l'etapa anterior (*st==null*), es comprova si hi ha algun paquet PES d'algun ES per ser empalmat a continuació, comparant els *timestamps* dels paquets PES més antics de tots els fluxos elementals oberts, amb l'instant de temps actual. Se n'encarrega la funció *input_available.*

9. *if (bs)*

   Si ha entrat informació nova en algun dels *raw buffers*, és probable que hi hagi nova informació disponible per ser extreta cap als *input buffers* (punt 7). D'altra banda, si s'ha tret algun paquet PES d'algun *input buffer* cap al *buffer* de sortida en l'etapa anterior (punt 10), es preveu que hi ha espai en algun dels *input buffers*. En qualsevol dels dos casos, es crida la funció *split_something()*, que intentarà extreure informació dels *raw buffers,* cap als *input buffers* corresponents. Aquesta funció torna *true* si s'ha extret alguna cosa, i *false* si no s'ha pogut extreure ja sigui perquè no hi ha informació disponible a cap *raw buffer*, o bé perquè no hi ha espai disponible als *input buffers* corresponents (prèviament, no es comproven cap de les dues condicions, únicament es preveu).

   bs = split_something();

10. *if (st != null && output_acceptable())*

    Si hi ha algun paquet PES d'algun ES preparat per ser empalmat (*st!=null*), havent-se seleccionat en aquesta etapa (punt 8) o en l'anterior (punt 11), i es preveu que hi ha espai disponible al *buffer* de sortida, es crida la funció *process_something()*, del mòdul *Splice*, que s'encarrega d'empaquetar dades del *input buffer* seleccionat en un paquet TS i disposar-les en el *buffer* de sortida.

    st = process_something (st);
    bs = TRUE;

    La funció *process_something* s'encarrega d'encapsular les dades de l'ES *st* en paquets TS, i disposar-los en el *buffer* de sortida. Cada cop que es crida la funció, generarà un únic paquet TS, amb un fragment de paquet PES. La funció tornarà el mateix punter que se li ha passat (el mateix ES), fins que acabi d'empaquetar tot un paquet PES, moment en que tornarà un punter nul. Així, l'ES seleccionat per treure a la sortida serà el mateix fins que s'hagi acabat d'encapsular tot el paquet PES en paquets TS. A més, fins que no s'acabi d'encapsular no es tornaran a comprovar els *timestamps* de la resta de paquets PES (la funció *input_available* només es crida si *st==null*, en els punts 8 i 11).

11. *if (st == null)*

    Si després de generar un nou paquet TS, el punter *st* és nul, indica que el paquet PES que s'estava processant s'ha acabat d'encapsular. En aquest punt, es tornen a comprovar els *timestamps* dels diferents paquets PES mitjançant la funció *input_available()*. En cas que se'n seleccioni algun, serà encadenat en la propera passada del bucle (punt 10).

12. *bo = output_available (&nfds, &ufds[onfds], &tmo)*

    Es determina si hi ha informació disponible al *buffer* de sortida. En cas afirmatiu, prepara l'estructura *pollfd* del fitxer de sortida adequadament

(màscara POLLOUT). També es calcula el valor de *timeout*. Aquest valor de *timeout* en cas que en el punt 1 el *buffer* de sortida estigui ple. Si en el punt 1 el *buffer* de sortida no està ple, es recalcularà el *timeout* segons paràmetres dels fluxos ES.

C.2. Anàlisi de les principals funcions

Un cop explicat el flux de programa del bucle principal, s'analitzaran algunes de les funcions més importants del codi per tal d'entendre l'algoritme de multiplexació. Es definirà l'objectiu amb el que s'ha escrit cadascuna d'elles, els paràmetres formals i el valor de retorn, i l'algoritme que utilitzen internament, per veure com es modifiquen les variables que es passen per referència. No s'analitzaran en l'ordre en què apareixen en el bucle principal, sinó que es farà en l'ordre en què viatgen les dades (*input<->split->splice->output*).

boolean input_acceptable (unsigned int *nfds, struct pollfd *ufds, t_msec *timeout, boolean outnotfull)

- Mòdul: *Input*

- Descripció

S'encarrega de comprovar si hi ha espai disponible per acceptar nova informació en els *raw buffers*. En cas afirmatiu, prepara el vector *ufds[]* adequadament, especificant les condicions de lectura mitjançant la màscara *events*. A més, si es pot acceptar informació al *buffer* de sortida, calcula el valor de *timeout*.

- Paràmetres de la funció

  - *nfds: Nombre de descriptors de fitxer que conté el vector *ufds*.
  - *ufds: Vector d'estructures *pollfd*.
  - *timeout: Se li passa per referència la variable *tmo* del bucle principal, que s'usa per determinar el *timeout* de la funció *poll()*.
  - outnotfull: Indica si hi ha espai al *buffer* de sortida. Se li passa el valor de retorn de la funció *output_acceptable*.

- Valor de retorn

  - TRUE: Almenys algun dels *raw buffers* pot acceptar informació nova.
  - FALSE: Altrament.

- Comentari de l'autor

```
/* Determine whether input data is acceptable, i.e. there is space in buffers.
 * If so, set the poll struct accordingly for each file in question.
 * Check the streams for time stamps and set the timeout^ accordingly,
 * if the streams might be responsible alone for blocking pipes.
 * Return: TRUE, if at least one buffer has enough space to accept new data.
 *         FALSE otherwise.
 */
```

- Algoritme

1. La funció recorre tots els *raw buffers* i comprova si hi ha espai disponible per acceptar noves dades en cadascun d'ells, a partir dels punters *in* i *out*, i de la variable *mask*. En cas que hi hagi espai en algun d'ells, tornarà *true*. A més, prepara el vector *ufds* adequadament per tal que posteriorment, mitjançant la funció *poll()*, es pugui comprovar si les fonts d'informació estan preparades per que les dades siguin llegides. El vector *ufds* es defineix des de zero cada cop que es crida la funció *input_acceptable*, i només s'hi afegeixen els descriptors de fitxer que corresponen als *raw buffers* que poden acceptar informació.

2. Si pot es acceptar informació al *buffer* de sortida, es calcula el valor de *timeout* que s'utilitzarà per incrementar el valor de *global_delta*. Per a cada ES de dades obert i que no estigui buit, es determina el valor de *timeout* de la següent manera:

   Si té *trigger* assignat:

   tmo = s->ctrl.ptr[s->ctrl.out].msecpush + s->u.d.delta – now

   que equival a calcular:

   tmo = time_stamp - now

   Si no el té:

   tmo = s->ctrl.ptr[s->ctrl.out].msecread – now + trigger_msec_input

En el primer cas, es calcula el temps que falta fins que s'ha de treure a la sortida el paquet PES més vell (el primer que s'ha de treure) de l'ES. Si no té valor de *trigger* assignat, es calcula el temps que fa que s'ha llegit, i se li suma *trigger_msec_input*, definit a *global.h* com a 250 (ms). Aquest càlcul es realitza per a cada ES, i el valor de *timeout* final serà el més petit. S'ignoren els valors negatius.

Suposant que en règim de treball normal, passat el transitori inicial, sempre hi haurà algun ES amb *trigger*, s'entén que el valor de *timeout* correspon al temps que falta fins que s'ha de treure el proper paquet PES a la sortida.

boolean split_something (void)

- Mòdul: *Input*

- Descripció

Aquesta funció s'encarrega de recórrer tots els *raw buffers* oberts, determinar el tipus de dades que contenen (TS, PS o PES), i cridar la funció *split_ts*, *split_ps* o *split_pes*, segons correspongui. Es podria dir que la seva funció és repartir la feina entre les funcions dels fitxers *SplitTS, SplitPS* o *SplitPES*, en el moment en que *Dispatch* demana a *Input* que intenti extreure informació. Es cridarà la funció *split_xx* per a totes les fons d'informació obertes.

- Paràmetres de la funció

  - No té paràmetres

- Valor de retorn

  - TRUE: S'ha extret informació d'algun *raw buffer*.
  - FALSE: No hi ha prou informació (en cap *raw buffer*) o espai (en cap *input buffer*) disponible.

- Comentari de l'autor

```
/* Split data from raw input buffers to PES data stream buffers.
 * Return: TRUE, if something was processed, FALSE if no data/space available
 */
```

boolean split_ts (file_descr *f)

- Mòdul: *Split*

- Paràmetres de la funció

  - *f: Descriptor de fitxer, del qual es vol extreure informació.

- Valor de retorn

  - TRUE: S'ha extret informació
  - FALSE: No hi ha prou informació (al *raw buffer*) o espai (al *input buffer*) disponible.

- Comentari de l'autor

```
/* Split data from a TS stream.
 * Precondition: f!=NULL
 * Return: TRUE, if something was processed, FALSE if not enough data
 * available */
```

- Algoritme

1. Descarta els *bytes* anteriors al primer *byte* de sincronisme que troba en el flux, on comença un paquet TS. Modifica el punter *out*.
2. Comprova si el *raw buffer* conté almenys TS_PACKET_SIZE *bytes*, és a dir, si hi ha almenys un paquet TS sencer.
3. Extreu el PID del paquet a processar.
4. Comprova si el PID del paquet està dins el rang [TS_PID_LOWEST, TS_PID_HIGHEST], definit com [0x0010, 0x1FFE] = [16, 8190]. Els valors de PID inferiors a 16, estan reservats per a informació PSI/SI.

a. En cas afirmatiu, comprova si el punter del *file_descr* que apunta al *stream_descr* de l'ES on ha d'anar la informació extreta és vàlid (no nul). → **stream[sid]*, on *sid* correspon al PID del paquet.

Si és vàlid, comprova el tipus de contingut del paquet (dades o taules PSI), i extreu la informació conseqüentment, cridant una d'aquestes funcions:

- *ts_data_stream (f,pid).*

Extreu dades (vídeo, àudio, etc) del flux d'entrada. S'explica a continuació.

- *ts_psi_table_section (f,pid,TS_TABLEID_PMT)*

Extreu i analitza la informació PSI del flux original (taules PMT i PAT). En aquest cas, únicament s'extreuen taules PMT, ja que prèviament s'ha comprovat que el PID estigui dins un cert rang; la taula PAT sempre viatja en el PID zero. La informació PMT s'extreu i es desa en l'*stream_descr* apuntat pel vector **stream[sid]*, del *file_desc* corresponent, on [sid] correspon al PID en què viatja la taula en el flux original. La informació PSI d'entrada s'utilitza per lligar els *stream_descr* que formen part d'un programa d'entrada, amb el *mapstream* corresponent (*stream_descr* que conté la taula PMT d'entrada), a través del punter *s->u.d.mapstream*.

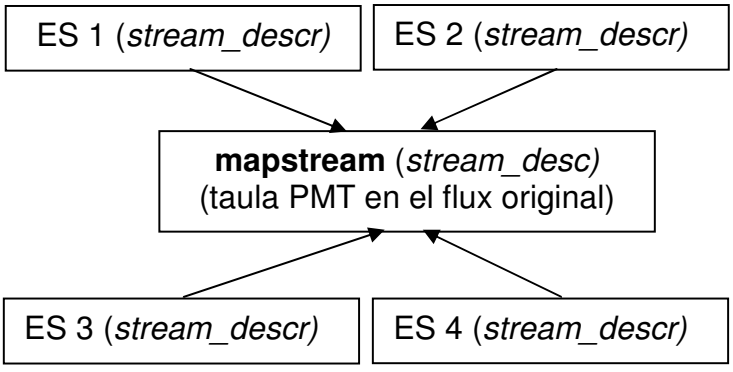


Figura C.3. Relació entre ESs i *mapstream* d'entrada

En cas que el punter sigui nul, interpreta que aquest ES no pertany a cap programa de sortida, o bé conté informació SI. Abans de descartar el paquet, busca en les taules PSI de sortida, i en cas que l'identificador de flux PES coincideixi amb algun dels fluxos de sortida, l'obre i extreu la informació del

paquet. En cas que l'ES no pertanyi a cap programa a la sortida del multiplexor, comprova si es tracta d'informació SI no analitzada; si ho és l'extreu i la guarda en l'*stream_descr* amb *sid* 'TS_UNPARSED_SI' del *file_descr*. En aquest cas, no es desencapsula el *payload* del paquet TS, sinó que es copien directament els paquets TS sencers, incloent la capçalera, en el *buffer* mencionat. A l'hora de treure'ls a la sortida, es copiaran els paquets TS sense modificar-los.

En cas que el paquet tampoc contingui informació SI, comprova si en el camp d'adaptació hi viatja informació PCR. S'extreu la informació PCR de tots els *elementary streams*, independentment de si estan sortint en el flux de sortida o no. Això es fa per si s'està utilitzant un flux ES a la sortida la referència PCR del qual viatja en un altre ES, que no s'utilitza. Tant si l'ES s'utilitza com si no, el valor de PCR es guarda en el camp *msectime* del *stream_descr* que conté la taula PMT del programa al que pertany en el flux original (*mapstream*).

b. Si el PID de l'ES no es troba dins el rang:

Comprova si el PID és el corresponent a la taula PAT (i l'extreu mitjançant la funció *ts_psi_table_section*), a un paquet nul (i incrementa els índex del *raw buffer* sense extreure'n res), o a informació SI no analitzada (i l'extreu a 'TS_UNPARSED_SI' del *file_descr*).

static boolean ts_data_stream (file_descr *f, int pid)

- Mòdul: *Split*

- Descripció

Aquesta funció s'encarrega d'extreure informació d'un paquet TS que contingui un flux ES (paquets PES amb vídeo, àudio, etc) i disposar-la en un bloc del *buffer* de dades de *l'stream_descr* que correspongui. Cada cop que es crida la funció, s'extreu el *payload* d'un únic paquet TS. Un paquet PES, però, està fragmentat en més d'un paquet TS. La funció s'encarrega de disposar les dades en cada *input buffer* de manera que cada bloc del *buffer* de dades correspongui a un paquet PES sencer. Per fer-ho, cada cop que extreu un fragment de paquet PES d'un paquet TS amb un PID determinat, l'encadena a la resta de fragments del mateix paquet PES que ha extret anteriorment, posant-lo en el mateix bloc del *buffer* de dades. Quan extreu el darrer fragment, "tanca" el bloc corresponent a aquest PES. Per controlar aquest procés, utilitza la variable *length* del *buffer* de control associat al bloc de dades del PES que està extraient, tal i com explica l'autor del codi:

```
* c->length keeps the progress of data acquisition, as follows:
*  =0: initial, waiting for payload unit start indicator
*  <-2, increasing: not yet enough data to evaluate the PES header
*  =-2: evaluate the PES packet length field
*  =-1: acquiring packet with unspecified length
*  >0, decreasing: amount of data still missing
*  =0: final, close PES packet, start new one
```

Cada cop que surti de la funció havent extret informació d'un paquet TS, la variable *length* indicarà en quin punt del procés d'extracció del PES s'ha quedat. Quan extregui el *payload* del TS que conté el final del paquet PES, tancarà el bloc i la propera vegada començarà a escriure en un nou bloc del *buffer* de dades. Aquest procés serà independent per a cada *elementary stream* obert.

- Paràmetres de la funció

  - *f: Descriptor de fitxer, del qual es vol extreure informació.
  - pid: PID del qual es vol extreure un paquet PES.

- Valor de retorn

  - TRUE: S'ha extret informació.
  - FALSE: No hi ha prou informació (al *raw buffer*) o espai (al *input buffer*) disponible.

- Comentari de l'autor

```
/* Parse one TS packet with given PID.
 * Depending on the actual state (c->length) and the contents of the packet,
 * provide the data into the stream and possibly complete a PES package.
 * c->length keeps the progress of data acquisition, as follows:
 *  =0: initial, waiting for payload unit start indicator
 *  <-2, increasing: not yet enough data to evaluate the PES header
 *  =-2: evaluate the PES packet length field
 *  =-1: acquiring packet with unspecified length
 *  >0, decreasing: amount of data still missing
 *  =0: final, close PES packet, start new one
 * Precondition: f!=NULL.
 * Return: TRUE if something was processed, FALSE if no data/space available
 */
```

- Algoritme

Es considera que l'algoritme que utilitza per copiar el *payload* d'un TS no es rellevant, únicament interessa com queden emmagatzemades les dades en cada *input buffer*, i els valors que s'assigna a les variables del *buffer* de control de cada bloc. Com ja s'ha dit, el *buffer* de dades quedarà dividit en blocs, cadascun dels quals correspon a un paquet PES. Cadascun d'aquests blocs, té una sèrie de variables associades, en el *buffer* de control. En el moment en que s'acaba d'extreure la informació d'un paquet PES, es realitzen, entre altres, les següents assignacions:

```
c->sequence = f->sequence++;
c->msecread = now
c->msecpush = s->u.d.mapstream->u.m.msectime
```

Com es pot observar, el número de seqüència associat al paquet PES, serà el número de paquet PES extret del flux original. D'altra banda, la variable *msecread* agafa el valor actual del rellotge intern en què s'ha acabat d'extreure el paquet PES, en ms, i la variable *msecpush* el valor de *msectime* del *mapstream* del programa al que pertany en el flux TS d'entrada. Com s'ha explicat anteriorment, la variable *msectime* del *mapstream* (*stream_desc* que conté la taula PMT del flux original) conté la darrera mostra de PCR extreta del programa d'entrada, en ms. Per tant, la variable *msecpush*, desarà el darrer valor de PCR que s'ha llegit en el moment d'extreure el paquet PES.

stream_descr *input_available (void)

- Mòdul: *Input*

- Descripció

S'encarrega de determinar si hi ha informació disponible en els *input buffers* dels diferents ES, de comprovar si aquests estan preparats per ser encadenats al flux de sortida, en funció de les condicions internes dels *buffers*, i de determinar quin ES serà empaquetat en cada moment, segons els *time stamps* dels diferents paquets PES.

- Paràmetres de la funció

  - No té paràmetres

- Valor de retorn

  - Punter a l'*stream_descr* que conté el proper paquet PES a ser encapsulat a la sortida.

- Comentari de l'autor

```
/* Check for every stream whether data is available to be spliced.
 * Unparsed SI from an otherwise unused TS has priority.
 * If the stream with the lowest time stamp has a corresponding map stream,
 * that provides data to be spliced first, the map stream is returned.
 * Prior, check if a stream is empty and end it, if necessary; check if a
 * stream is ready but not yet triggered, so trigger it.
 * Return: stream to be spliced next.
 */
```

- Algoritme

1. Per a tots els *raw buffers* que no tenen cap ES obert, comprova si tenen informació en l'*sid 'TS_UNPARSED_SI'.* Si troba algun *raw buffer* que compleixi aquesta condició, la funció tornarà el punter a l'*stream_descr* que conté la informació SI no analitzada. No queda clar per què prioritza la

informació SI d'un TS d'entrada del qual no s'està extraient cap *elementary stream*.

Per a tots els *elementary streams* que contenen dades (no taules), i el seu *buffer* està buit:

Comprova el valor del camp *endaction* i en funció d'aquest realitza una determinada acció, amb l'ajuda de funcions del mòdul *Input*. Aquests són els possibles valors del camp en qüestió:

- *ENDSTR_CLOSE*: Crida la funció *input_endstream.* Aquesta, comprova la taula PMT de tots els programes de sortida que contenen l'ES, i si no hi ha almenys un d'aquests programes que contingui un altre ES procedent de la mateixa font (*raw buffer*), deslliga l'ES de tots els programes i el tanca.
- *ENDSTR_KILL*: Crida la funció *input_endstreamkill.* Treu l'ES de tots els programes dels que formava part i el tanca.
- *ENDSTR_WAIT*: No fa res.

Si el *buffer* no està buit, i no té valor de *trigger* assignat, comprova si es compleix alguna de les següents condicions

- El *buffer* està ple (*list_full*).
- El *buffer* està parcialment ple (*list_partialfull*).
- El camp *endaction* té per valor ENDSTR_CLOSE o ENDSTR_KILL.

Si es compleix qualsevol de les tres condicions, es considera que l'ES està preparat per començar a ser encadenat al flux de sortida, i per tant se li assigna el *flag* de *trigger* i es calculen una sèrie de variables utilitzades en l'algoritme de temporització: *delta* i *lasttime*, mitjançant la funció ***set_trigger()***. En el moment en que s'assigna el *trigger* a un ES, també s'assigna a la resta d'ES que formen part del mateix programa de sortida. La variable *lasttime*, de *l'stream_descr*, desa el valor del *time stamp* de sortida del paquet PES més antic. L'instant de temps en què s'assigna el *trigger* es considera el *time stamp* de sortida del primer paquet PES del flux, i per tant a la variable *lasttime* se li assigna el valor de retorn de la funció *msec_now()*. La variable *delta* es calcula com la diferència de temps entre el rellotge intern, en ms, i el valor de la variable *msecpush* del paquet PES més antic. Cal recordar que la variable *msecpush* guarda el darrer valor de PCR extret del programa original, en el moment en que s'ha extret aquest paquet PES. Així doncs, la variable *delta* es pot definir la diferència temporal entre el rellotge de programa en el flux original, i el rellotge intern del multiplexor. A partir del moment en que s'ha assignat el *trigger*, s'avaluaran els *time stamps* dels paquets PES d'aquest flux.

S'avaluen els *time stamps* de sortida dels paquets PES més antics de tots els fluxos elementals que tenen *trigger* assignat, contenen dades (no taules), i el *buffer* de dades no està buit. Per a cada paquet a avaluar es calcula el *time stamp* com:

t = msecpush *(del PES)* ***+ delta*** *(del ES)*

Simplement, al *time stamp* d'entrada (*msecpush*), se li suma la diferència entre els rellotges, per tal d'obtenir un *time stamp* referit al rellotge intern. Cada cop que es calcula *t* per a un ES, s'actualitza el valor de *lasttime* d'aquest, assignant-li el nou *time stamp*. De la mateixa manera, cada cop que es calcula un nou *time stamp*, es compara amb l'anterior (*lasttime*) per assegurar que el nou és major. En cas que fos menor, es desassignaria el *trigger* a aquest ES (*clear_trigger()*); posteriorment, es tornaria a cridar la funció *set_trigger*, per recalcular la *delta.* Si el *time stamp* calculat és vàlid (major que l'anterior), es calcula el temps que falta fins que s'ha de treure el paquet.

$$t = t - now \rightarrow jiiter\ de\ sortida$$

Si el resultat és positiu, el *time stamp* del paquet PES és posterior al moment actual. En canvi, si és negatiu, implica que el paquet ja s'hauria hagut de treure. En el cas ideal tots els paquets es trauran quan el *jitter* sigui zero, és a dir, es trauran en el moment que determina el seu *time stamp*. Aquest càlcul es realitza per a tots els fluxos elementals oberts i amb *trigger* assignat. En cas que dos fluxos tinguin el mateix *time stamp*, se seleccionarà primer el que tingui el número de seqüència menor.

Un cop s'ha seleccionat un paquet PES per ser encadenat a la sortida, es comprova si hi ha informació PSI d'entrada rellevant, relacionada amb l'ES que conté el paquet. S'entén per informació rellevant, taules PAT o PMT amb un número de seqüència anterior al paquet PES. En cas afirmatiu, es processaria abans la taula PSI que el paquet, per si s'hagués produït algun canvi en el *mapstream* del programa.

stream_descr *process_something (stream_descr *s)

- Mòdul: *Splice*
- Descripció

S'encarrega de generar el *Transport Stream* de sortida, encapsulant les dades dels *input buffers* en paquets TS, dels quals en genera la capçalera i el camp d'adaptació, i disposant aquests paquets en el *buffer* de sortida. Cada cop que es crida la funció, es genera un únic paquet TS. També s'encarrega de manegar els programes de sortida, i de gestionar la informació PSI de sortida i generar-ne les taules.

- Paràmetres de la funció
  - stream_descr *s: ES del que s'ha d'encapsular informació
- Valor de retorn
  - Si la informació que conté *s són dades:
    - *s: Queda informació pendent del paquet PES que s'està processant.
    - NULL: S'ha acabat d'encapsular tot el paquet PES en el flux de sortida.

- Si la informació que conté *s són taules:
  - NULL

- Comentari de l'autor

  - No hi ha comentari.

- Algoritme

En primer lloc, es comprova el tipus de contingut del *stream_descr* *s (dades, taules PSI, o taules SI).

a. En cas que siguin dades, es determina si hi ha informació PSI de sortida rellevant relacionada amb aquest ES, mitjançant la funció *procdata_check_psi()*. Bàsicament, es comprova si hi ha hagut algun canvi en algun dels *mapstreams* de sortida, moment en que es refarien i es transmetrien les taules que ho requerissin. Aquest canvis poden ser provocats per una comanda de l'usuari, o bé per un canvi en la informació PSI d'entrada, si s'està traient un programa sencer a la sortida. Si hi ha taules PMT o PAT que han començat a ser transmeses i encara queden fragments pendents, primer es transmetran aquests fragments. Això implica que tots els fragments d'una taula determinada s'encapsulen en paquets TS consecutius. En cas que no hi hagi informació PSI rellevant, s'encapsularà el paquet PES. Tant per processar fragments de taules PSI, com fragments de paquets PES, s'utilitzen les següents funcions:

output_pushdata
Funció del mòdul *Output*. S'encarrega de reservar l'espai indicat per *Splice* al *buffer* de sortida. Torna un punter a la posició de memòria on ha de començar a escriure *Splice*.

procdata_adaptfield_flags
S'encarrega de determinar la informació que contindrà el camp d'adaptació del paquet TS, segons la informació disponible (referència PCR, OPCR, etc), de generar els *flags* corresponents, i de calcular l'espai que quedarà disponible com a *payload* del paquet TS.

procdata_adaptfield_frame
Ajusta la mida del camp d'adaptació en funció la quantitat de dades de què es disposen, si és necessari.

procdata_syn_head
S'encarrega de generar la capçalera del TS i escriure-la en el *buffer* de sortida, en l'espai assignat per *output_pushdata*.

procdata_syn_adaptfield
Genera el camp d'adaptació, en funció dels *flags* determinats anteriorment, i l'escriure en el *buffer* de sortida, a continuació de la capçalera.

procdata_syn_payload
S'encarrega de copiar la informació del *input buffer* del ES que s'està processant al *payload* del paquet TS.

En aquest cas, la funció *process_something()*, tornarà el valor de retorn de *procdata_syn_payload()*. Aquesta, torna un punter nul si el paquet PES que s'està processant s'ha acabat de treure a la sortida, o bé un punter a l'ES, si el paquet PES no s'ha acabat d'empaquetar. En el bucle principal, quan es crida la funció *process_something()* es fa la següent assignació:

st = process_something (st),

on el paràmetre *st* és un punter a una estructura del tipus *stream_descr* i s'utilitza per indicar el proper ES a ser encadenat.

Així, es pot veure que fins que un determinat paquet PES no s'ha encapsulat completament, el punter *st* del bucle principal sempre apuntarà al mateix ES. Aquest és el motiu pel qual en el flux de sortida apareixen tots els paquets TS que contenen un mateix paquet PES disposats de forma consecutiva.

La funció també tornarà l'ES que s'està processant en cas que no hi hagi espai disponible al *buffer* de sortida.

b. Si l'*stream_descr* *s conté informació PSI, aquesta es valida mitjançant la funció *validate_mapref()*. Bàsicament s'encarrega d'analitzar les taules PMT i PAT dels fluxos d'entrada, i lligar cada ES al *mapstream* d'entrada corresponent. És important que cada ES estigui lligat al *mapstream* del programa d'entrada al que pertany realment, ja que aquest porta la referència PCR.

c. Si conté informació SI no analitzada, es reserva l'espai necessari per treure un paquet i es treu a la sortida. Com s'ha explicat anteriorment, quan s'extreu la informació SI del flux d'entrada no es desencapsula, es copien directament els paquets TS un determinat *buffer*. En el moment de treure aquesta informació, es copien directament els paquets TS al *buffer* de sortida, sense realitzar-ne cap modificació.

Les conclusions extretes d'aquest anàlisi es mostren en l'apartat 4.5.

ANNEX D: PROVES REALITZADES AMB MPLEX 13818

Per tal d'entendre millor el funcionament de l'algoritme que utilitza Mplex per determinar quan s'ha d'encapsular cada paquet a la sortida, s'ha afegit al *software* el codi necessari per generar un *log* amb el valor de diferents variables i paràmetres relacionats amb l'algoritme de temporització, en diferents punts del procés de multiplexat. S'ha dissenyat un nou fitxer anomenat *timelog.c*, i la corresponent capçalera *.h*, amb les funcions necessàries per escriure en un fitxer de text la informació que es genera en diferents punts del codi, amb el següent format:

Rellotge intern	PID	Núm. Seqüència	Descripció	Valor	Funció

Figura D.1. Estructura de les entrades del *log*

Per a cada entrada del *log*, en primer lloc es mostra el valor del rellotge intern del multiplexor en el moment en què s'ha generat la informació. A continuació, s'indica el PID de sortida del flux elemental al que fa referència l'entrada, així com el número de seqüència que identifica el paquet que s'està processant. Després es mostra la descripció de la informació proporcionada, el valor, i la funció dins la qual s'ha generat.

L'objectiu d'aquest anàlisi és, per una banda, mostrar com es realitza l'assignació de les variables *msecpush* i *msecread* a cada un dels paquets PES en el moment en què s'extreuen, i com s'agafen les referències PCR del flux d'entrada i s'assignen a la variable *msectime* del *mapstream* corresponent, en ms. D'altra banda, es vol veure l'evolució dels *time stamps* de sortida dels diferents paquets, així com del valor del *jitter*, i quin és el paquet que se selecciona cada moment per ser empalmat en el flux de sortida. Per tal de facilitar l'anàlisi de la informació, els *logs* s'han exportat en un full de càlcul, i s'han separat els camps en columnes, per tal de poder utilitzar filtres i veure únicament la informació associada a un PID, un paquet, un tipus de descripció, etc. El *log* conté totes les entrades generades durant tot el procés de multiplexat, però se n'han exportat 65536, que és el nombre màxim de files amb què permet treballar l'*Excel*. S'han seleccionat les entrades a partir d'un punt en el que tots els ES han començat a ser encadenats al flux de sortida, per tal de descartar la informació associada al transitori inicial.

L'exemple s'ha realitzat amb el *log* obtingut de la remultiplexació de la captura "rete.ts", descrita en el capítol 3, mitjançant la següent comanda:

```
$ ./iso13818ts --fpsi 400 --conservativepids 1 --ts rete.ts > reteRemux.ts
```

L'opció *'conservativepids'* s'ha fet servir per tal que els ES tinguin el mateix PID a la sortida que a l'entrada.

Taula D.1. Correspondència entre programes i PIDs

Programa / PID	Vídeo i PCR	Àudio	Teletext
Canal + (1057)	501	503	-
Antena 3 (1121)	301	303	302
Telecinco (1185)	401	403	402
TVE1 (1377)	101	103 i 104	102
TVE2 (1441)	201	203 i 204	202

Totes les proves s'han realitzat amb un PC de sobretaula amb processador Intel Q9550 (quatre nuclis a 2,83 GHz, 12MB L2, bus de 1333 MHz), 8 Gigues de memòria RAM DDR2 i placa base Asus amb FSB de 1600 MHz.

D.1. Extracció de la informació

Les 65536 entrades seleccionades corresponen a l'interval de temps [3722, 3959] ms, segons el rellotge intern. Per veure com es realitza l'assignació de les variables *msecread, msecpush* i *msectime*, es presenta la informació obtinguda de cada un dels ES per separat.

Taula D.2. Evolució de les variables *msecread*, *msecpush* i *msectime* del flux amb PID 101

Rellotge Intern	PID	Seqüència	Descripció	Valor	Funció
3726	101	1586	MSECTIME	2310860	split_ts
3739	101	1591	MSECREAD	3739	split_ts
3739	101	1591	MSECPUSH	2310860	split_ts
3879	101	1595	MSECTIME	2310897	split_ts
3879	101	1597	MSECREAD	3879	split_ts
3879	101	1597	MSECPUSH	2310897	split_ts
3893	101	1603	MSECTIME	2310934	split_ts
3909	101	1604	MSECREAD	3909	split_ts
3909	101	1604	MSECPUSH	2310934	split_ts
3909	101	1606	MSECTIME	2310955	split_ts
3909	101	1613	MSECTIME	2310971	split_ts
3944	101	1623	MSECTIME	2311008	split_ts
3952	101	1629	MSECTIME	2311045	split_ts

Aquesta taula mostra els valors de les variables *msecread* i *msecpush* en el moment en què s'acaba d'extreure un paquet PES, i el valor de la variable *msectime* quan s'extreu una referència PCR del flux amb PID 101, que correspon al vídeo del programa TVE1, i hi viatja el PCR. Com es pot observar,

a la variable *msecread* se li assigna el valor del rellotge intern del multiplexor en el moment en què s'acaba d'extreure el paquet, i a la variable *msecpush*, el valor actual de la variable *msectime*, és a dir, la darrera marca PCR extreta del programa original, en ms. En el cas del fluxos amb PIDs 102, 103 i 104, que contenen el teletext, i els dos àudios del programa, respectivament, només es mostra el valor de les variables *msecread* i *msecpush*, ja que no se n'extreu cap referència PCR perquè aquesta viatja en el flux de vídeo (PID 101), tal i com es pot veure a la taula D.1.

Taula D.3. Evolució de les variables *msecread*, *msecpush* dels fluxos amb PIDs 102, 103 i 104

Rellotge Intern	PID	Seqüència	Descripció	Valor	Funció
3724	102	1579	MSECREAD	3724	split_ts
3724	102	1579	MSECPUSH	2310823	split_ts
3736	102	1589	MSECREAD	3736	split_ts
3736	102	1589	MSECPUSH	2310860	split_ts
3879	102	1595	MSECREAD	3879	split_ts
3879	102	1595	MSECPUSH	2310897	split_ts
3909	102	1607	MSECREAD	3909	split_ts
3909	102	1607	MSECPUSH	2310955	split_ts
3937	102	1615	MSECREAD	3937	split_ts
3937	102	1615	MSECPUSH	2310971	split_ts
3952	102	1626	MSECREAD	3952	split_ts
3952	102	1626	MSECPUSH	2311008	split_ts
Rellotge Intern	**PID**	**Seqüència**	**Descripció**	**Valor**	**Funció**
3725	103	1584	MSECREAD	3725	split_ts
3725	103	1584	MSECPUSH	2310823	split_ts
Rellotge Intern	**PID**	**Seqüència**	**Descripció**	**Valor**	**Funció**
3737	104	1590	MSECREAD	3737	split_ts
3737	104	1590	MSECPUSH	2310860	split_ts

Fixant-se en els valors de rellotge intern cada moment en què es realitza l'assignació de la variable *msecpush*, i buscant el valor de *msectime* actual en la taula D.2, es comprova que a *msecpush* sempre se li assigna la darrera mostra de PCR extreta del programa al que pertany en el flux d'entrada. Per exemple, el paquet amb PID 104 i número de seqüència 1590, s'extreu en l'instant 3737ms. En aquest moment, l'últim cop que s'ha actualitzat la variable *msectime* ha estat en l'instant 3726ms (paquet 1586), tal i com es pot observar en la taula D.2. La variable *msecpush* agafarà el valor actual de la variable *msectime*, en aquest cas 2310860 ms.

En cap cas es té en compte el temps que ha passat entre la darrera extracció d'una marca PCR i el moment en què s'extreu el paquet PES i s'assigna la variable *msecpush*. Aquest fet té dues conseqüències:

- Dos paquets PES poden tenir el mateix valor de *msecpush* (i per tant, el mateix *time stamp* de sortida), si entre l'extracció del primer i l'extracció del següent no apareix cap marca de PCR en el flux d'entrada associada al programa al que pertanyen.

- La marca assignada a cada paquet PES no correspon exactament amb el seu temps d'arribada, de manera que es produirà un desajust en la distancia entre els paquets PES en el flux de sortida respecte la distancia en el flux d'entrada.

En la resta de programes, s'ha comprovat que el comportament del multiplexor és exactament el mateix.

D.2. Algoritme de temporització

Com s'ha dit anteriorment, s'està realitzant l'anàlisi en el rang temporal [3722, 3959] ms, referit al rellotge intern del multiplexor. En aquest interval però, no s'ha observat que s'assigni el *trigger* a cap ES, de manera que se suposa que ha estat assignat anteriorment; això implica que en aquest interval de temps tampoc s'ha realitzat cap *clear trigger*. Tot i que s'ha seleccionat aquest rang de temps, el *log* conté totes les entrades registrades durant tot el procés de multiplexat. Així, s'ha buscat el punt en què s'assigna el *trigger* als diferents *elementary streams*:

Taula D.4. Càlcul de la variable *delta* dels diferents fluxos elementals

Rellotge Intern	PID	Seqüència	Descripció	Valor	Funció
207	401	14	DELTA	-15679039	set_trigger
207	403	49	DELTA	-15679187	set_trigger
207	402	15	DELTA	-15679039	set_trigger
209	301	16	DELTA	-37970827	set_trigger
209	302	32	DELTA	-37970901	set_trigger
209	303	48	DELTA	-37970937	set_trigger
213	501	38	DELTA	-7032113	set_trigger
213	503	84	DELTA	-7032298	set_trigger
244	201	18	DELTA	-46444309	set_trigger
244	203	88	DELTA	-46444567	set_trigger
244	204	74	DELTA	-46444531	set_trigger
244	202	19	DELTA	-46444345	set_trigger
264	101	46	DELTA	-2304746	set_trigger
264	103	99	DELTA	-2304969	set_trigger
264	102	51	DELTA	-2304746	set_trigger
264	104	73	DELTA	-2304820	set_trigger

En la taula D.4 es pot observar que en el moment en que s'assigna el *trigger* a un dels ES, també es realitza l'assignació a tots els ES que formen part del mateix programa a la sortida. Encara que les entrades de la taula es mostren de forma consecutiva, l'assignació del *trigger* es fa individualment en cada programa de sortida, en temps d'execució del programa diferents. Havent passat 264 milisegons, ja s'ha assignat el *trigger* a tots els fluxos elementals. En tots els casos el valor de *delta* és negatiu, la qual cosa indica que el rellotge intern del multiplexor va atrassat respecte els dels diferents programes d'entrada; té lògica tenint en compte que el rellotge intern comença a comptar

des de zero. Analitzant el *log*, s'ha observat que fins a l'instant 24000ms, no es torna a calcular el valor de *delta* (prèviament s'ha produït un *clear trigger*). En l'interval que s'analitza, s'utilitzaran els valors de *delta* calculats en aquest punt per determinar els *timestamps* de sortida.

Un cop calculat el valor de *delta* per a tots els ES, aquests ja estan preparats per ser seleccionats com a proper *stream* a ser empalmat a la sortida. En la taula D.5, es mostra un fragment del *log* amb les entrades que mostren l'evolució del *time stamp* de sortida, la diferència entre el *time stamp* actual i el que s'havia calculat en la passada anterior del bucle, i el *jitter* de sortida.

Taula D.5 Evolució del *time stamp* i del *jitter* del flux amb PID 101

3740			**INICI SELECCIÓ ES**		input_available
3740	101	962	ΔMSECPUSH	0	input_available
3740	101	962	TIME STAMP	3744	input_available
3740	101	962	JITTER	4	input_available
3740			**INICI SELECCIÓ ES**		input_available
3740	101	962	ΔMSECPUSH	0	input_available
3740	101	962	TIME STAMP	3744	input_available
3740	101	962	JITTER	4	input_available
3740			**INICI SELECCIÓ ES**		input_available
3740	101	962	ΔMSECPUSH	0	input_available
3740	101	962	TIME STAMP	3744	input_available
3740	101	962	JITTER	4	input_available
3744			**INICI SELECCIÓ ES**		input_available
3744	101	962	ΔMSECPUSH	0	input_available
3744	101	962	TIME STAMP	3744	input_available
3744	101	962	JITTER	0	input_available
3744	101	962	PID SELECCIONAT	101	input_available
3744			**INICI SELECCIÓ ES**		input_available
3744	101	964	ΔMSECPUSH	0	input_available
3744	101	964	TIME STAMP	3744	input_available
3744	101	964	JITTER	0	input_available
3744	101	964	PID SELECCIONAT	101	input_available
3744			**INICI SELECCIÓ ES**		input_available
3744	101	975	ΔMSECPUSH	37	input_available
3744	101	975	TIME STAMP	3781	input_available
3744	101	975	JITTER	37	input_available

Es mostra l'evolució dels paràmetres indicats anteriorment, per al flux elemental amb PID 101. Evidentment, cada cop que es crida la funció *input_available* des del bucle principal (Inici selecció ES), s'avaluen tots els ES amb *trigger* assignat, però de moment només es mostra la informació associada a un PID. Inicialment, el paquet PES que s'està avaluant (seq. 962) té un *time stamp* de sortida igual a 3744ms, i el rellotge intern està en els 3740 ms. Conseqüentment, tal i com es pot observar, el *jitter* de sortida té un valor de

4ms. Cal recordar que el *jitter* indica la diferència entre el *time stamp* i l'instant de temps actual, de manera que el fet que sigui positiu indica que el paquet encara no s'ha de treure. En les 65536 entrades del *log* analitzades, no s'han trobat valors de *jitter* negatius, de manera que sempre es treu el paquet en el moment que determina el *time stamp*. Com es pot veure, en les diferents passades de bucle el valor del *time stamp* no canviarà fins que no se seleccioni el paquet d'aquest ES, es tregui a la sortida, i es passi a avaluar el següent paquet PES. El valor de Δ*msecpush* indica la diferència entre la variable *msecpush* del paquet PES que s'ha avaluat en la passada anterior del bucle, i la variable *msecpush* del paquet que s'està avaluant actualment. Aquesta diferència equival a l'increment del *timestamp* de sortida entre els dos paquets. En cas que la diferencia entre els *timestamps* sigui negativa, es desassignarà el *trigger* de tots els ES que formen el programa de sortida al que pertany el paquet PES que s'està avaluant, i quan sigui el moment, és tornarà a calcular la *delta*.

Els paquets amb números de seqüència 962 i 964 han estat seleccionats per ser encadenats en el flux de sortida de forma consecutiva, tenint el mateix *time stamp*, i per tant un Δ*msecpush* igual a zero. El fet que tinguin el mateix valor de *msecpush* demostra que el fet de no tenir en compte el temps que ha passat des de la darrera extracció d'una marca PCR fins l'extracció del paquet PES, pot provocar en alguns casos que diferents paquets PES consecutius quedin marcats amb el mateix valor. En aquest cas, els dos paquets estaven molt propers dins el flux d'entrada, ja que tenen un número de seqüència molt proper (cal recordar que el número de seqüència és un comptador de paquets PES que compta i identifica els paquets que s'han extret d'un flux d'entrada, tenint en compte tots els PIDs). No obstant, ha de quedar clar que aquest no és el motiu pel qual tots els paquets TS en què s'encapsula un paquet PES apareixen a la sortida de forma consecutiva; això és degut al funcionament del mòduls *Splice* i *Dispatch*, com s'ha explicat en apartats anteriors. Aquest punt únicament provocarà una desviació en la distància temporal entre paquets PES en el TS de sortida respecte el flux d'entrada.

La taula D.6 l'estat dels paràmetres de tots els fluxos elementals en el moment en què s'ha seleccionat el paquet 962 del flux amb PID 101. Com es pot observar, cada un dels fluxos elementals presenta un *time stamp* (i per tant un *jitter*) diferents, de manera que és poc probable que en algun cas coincideixin dos paquets amb el mateix *time stamp*. Si es produís aquesta situació, es produiria un retard en la sortida d'un dels paquets. No obstant, analitzant el *log*, s'ha comprovat que es poden arribar a realitzar 25 passades de bucle en un milisegons, de manera que el paquet seria tret a la sortida amb un retard mínim. Evidentment, això dependrà de la potència de la màquina amb què s'executi el programa.

Les conclusions extretes a partir d'aquestes proves es mostren en l'apartat 4.5.

Taula D.6 Evolució del *time stamp* i del *jitter* del flux de sortida

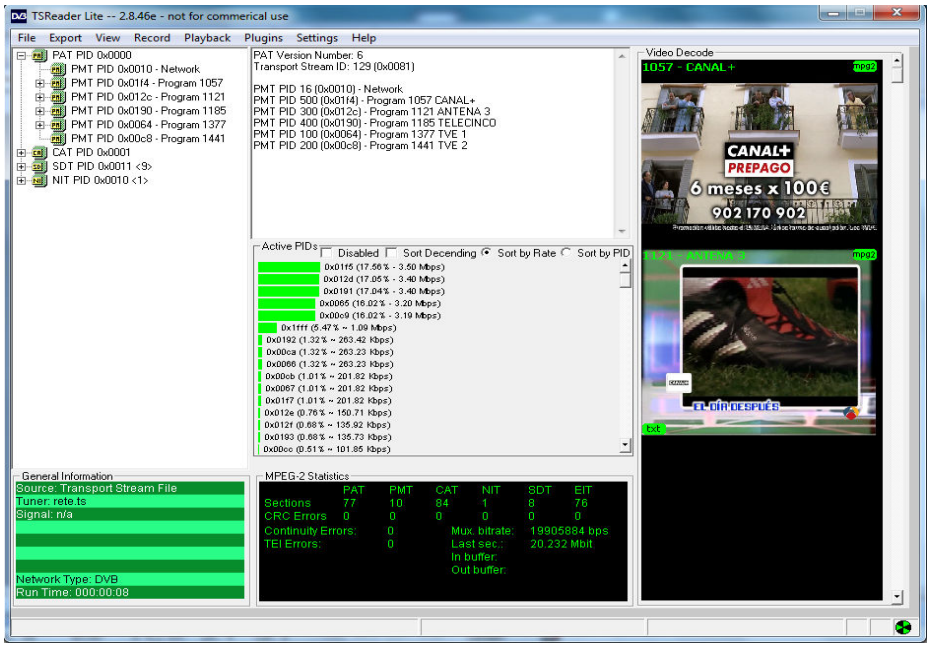
3744			**INICI SELECCIÓ ES**		input_available
3744	103	1067	ΔMSECPUSH	0	input_available
3744	103	1067	TIME STAMP	3928	input_available
3744	103	1067	JITTER	184	input_available
3744	102	967	ΔMSECPUSH	0	input_available
3744	102	967	TIME STAMP	3744	input_available
3744	102	967	JITTER	0	input_available
3744	203	1054	ΔMSECPUSH	0	input_available
3744	203	1054	TIME STAMP	3947	input_available
3744	203	1054	JITTER	203	input_available
3744	501	992	ΔMSECPUSH	0	input_available
3744	501	992	TIME STAMP	3878	input_available
3744	501	992	JITTER	134	input_available
3744	503	1049	ΔMSECPUSH	0	input_available
3744	503	1049	TIME STAMP	3878	input_available
3744	503	1049	JITTER	134	input_available
3744	104	978	ΔMSECPUSH	0	input_available
3744	104	978	TIME STAMP	3745	input_available
3744	104	978	JITTER	1	input_available
3744	204	1033	ΔMSECPUSH	0	input_available
3744	204	1033	TIME STAMP	3908	input_available
3744	204	1033	JITTER	164	input_available
3744	101	962	ΔMSECPUSH	0	input_available
3744	101	962	TIME STAMP	3744	input_available
3744	101	962	JITTER	0	input_available
3744	302	965	ΔMSECPUSH	0	input_available
3744	302	965	TIME STAMP	3763	input_available
3744	302	965	JITTER	19	input_available
3744	403	996	ΔMSECPUSH	0	input_available
3744	403	996	TIME STAMP	3798	input_available
3744	403	996	JITTER	54	input_available
3744	303	988	ΔMSECPUSH	0	input_available
3744	303	988	TIME STAMP	3838	input_available
3744	303	988	JITTER	94	input_available
3744	201	940	ΔMSECPUSH	0	input_available
3744	201	940	TIME STAMP	3760	input_available
3744	201	940	JITTER	16	input_available
3744	202	950	ΔMSECPUSH	0	input_available
3744	202	950	TIME STAMP	3762	input_available
3744	202	950	JITTER	18	input_available
3744	401	952	ΔMSECPUSH	0	input_available
3744	401	952	TIME STAMP	3761	input_available
3744	401	952	JITTER	17	input_available
3744	301	947	ΔMSECPUSH	0	input_available
3744	301	947	TIME STAMP	3763	input_available
3744	301	947	JITTER	19	input_available
3744	402	953	ΔMSECPUSH	0	input_available
3744	402	953	TIME STAMP	3773	input_available
3744	402	953	JITTER	29	input_available
3744	101	962	PID SELECCIONAT	101	input_available

ANNEX E: SOFTWARE UTILITZAT

E.1. TS Reader

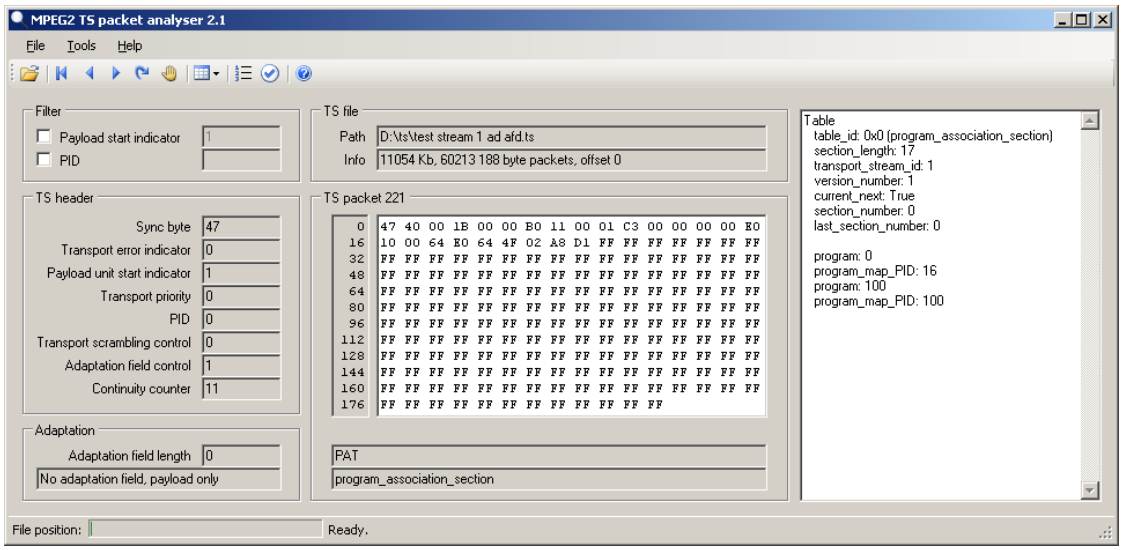
TS Reader [W8] és un analitzador i descodificador de *Transport Streams* MPEG-2. La seva finalitat és proporcionar una sèrie d'informacions sobre el flux. Entre les seves funcionalitats, han estat especialment útils les següents:

- Descodifica les taules PSI i SI i les presenta de forma resumida
- Compta el nombre de seccions de cada taula que apareixen en el flux.
- Mostra el tan per cent d'ocupació d'ample de banda dels diferents fluxos elementals, en forma de gràfic.
- Analitza les deficiències en el flux tals com errors i discontinuïtats.



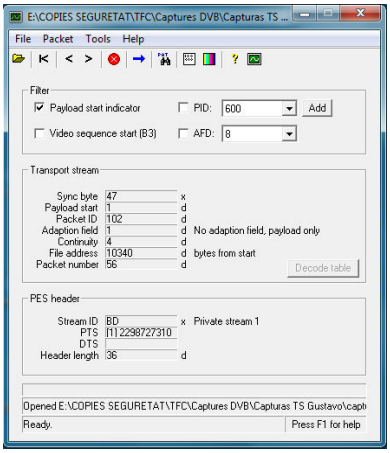
E.2. MPEG2 TS Packet Analyser 2.1

MPEG-2 *Transport Stream Packet Analyzer* [W9], desenvolupat per programador PJ Daniel, permet analitzar els fluxos de transport a baix nivell, proporcionant informació de cada paquet TS tal com el seu PID, l'estat d'alguns dels *flags* de la capçalera com *transport error indicator* o *payload unit start*, el valor del comptador de continuïtat, i la presència de camp d'adaptació, especificant-ne la mida. En cas que el paquet TS contingui una secció de taula PSI/SI, la descodifica i la mostra. D'altra banda, presenta el contingut del paquet el format hexadecimal, organitzant la informació en *bytes*, que resulta útil de cara a analitzar manualment el contingut dels paquets de transport.



E.3. MPEG-2 Analyzer 1.04

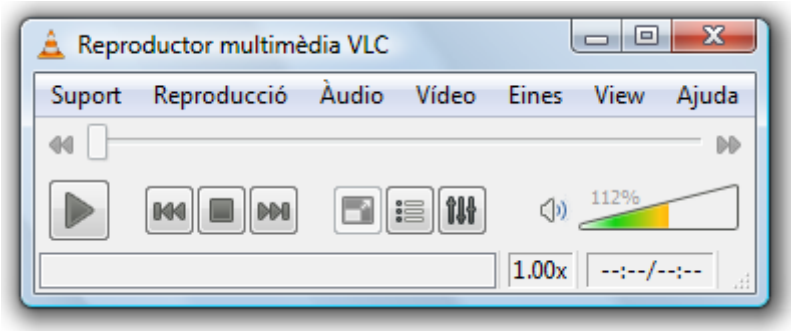
Creat també pel desenvolupador PJ. Daniel, és una versió anterior del MPEG-2 TS Packet Analyzer 2.1. Tot i que presenta menys funcionalitats que l'anterior, té un apartat que mostra es *timestamps* de descodificació i presentació dels paquets PES, quan s'analitzen paquets de transport que en contenen l'inici.



E.4. VLC Media Player

VLC Media Player [10] és un reproductor multimèdia de codi obert desenvolupat a partir del projecte VideoLAN, dedicat a la creació de *software* per aplicacions multimèdia sota llicències GPL. Permet reproduir una gran varietat de formats d'àudio i vídeo (MPEG-2, MPEG-4, H.264, DivX, mp3, ogg, acc, etc), així com DVDs, CD's d'àudio, VCD's i alguns protocols de difusió de continguts audiovisuals a través d'Internet. A més, també incorpora la funcionalitat de difusió de fluxos a través de xarxes IPv4 o IPv6, en forma *unicast* o *multicast*.

A nivell de reproducció, permet visualitzar *logs* que mostren informació sobre els problemes que es detecten en el flux d'entrada, tals com desbordament de *buffers*, desfasament entre vídeo i àudio, *timestamps* incorrectes, etc.



E.5. Entorn de programació: ANJUTA

La implementació de totes les millores que s'han realitzat en aquest TFC s'ha mitjançant l'entorn de programació Anjuta [W11].

Anjuta és un entorn integrat de desenvolupament (IDE) per a GNOME que permet programar en els llenguatges C, C++, Java i Python, en sistemes Linux/Unix. És tracta d'un *software* de codi obert, amb llicència GPL (*General Public Licence*). Va ser creat per Naba Kumber l'any 1999, i el va anomenar d'aquesta manera en honor a la seva parella a qui el va dedicar.

Inclou eines d'edició, depuració, organització de fitxers i de projectes i fins i tot assistents i plantilles que poden facilitar la feina del programador.