TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Eines de codi obert per a serveis d'streaming

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Sistemes de Telecomunicació

AUTOR: Juan Luis Rodríguez Blanco

DIRECTOR: David Rincón Rivera

DATA: 14 de Maig de 2012

Títol: Eines de codi obert per a serveis d'streaming

Autor: Juan Luis Rodríguez Blanco

Director: David Rincón Rivera

Data: 14 de Maig de 2012

Resum

Les xarxes de comunicació IP actuals i els nous hàbits de consum de tecnologies multimèdia han introduït nous tipus de serveis audiovisuals. L'alta definició ha arribat a TV i a Internet entre d'altres nous serveis per a sistemes d'àudio i vídeo. Les aplicacions d'aquests sistemes són moltes. S'utilitzen en àmbit militar, telemedicina, televigilància i emissions de continguts de TV i cinema.

En aquest marc de consum, sorgeix la necessitat de disposar d'eines que ens permetin operar i desplegar aquests tipus de serveis. Les noves tecnologies de codi obert són una solució possible davant altres tecnologies de pagament. La filosofia de codi obert afavoreix el seu ús per a docència i són una alternativa econòmica per a empreses i a usuaris.

L'objectiu d'aquest TFC és presentar aplicacions de codi obert que hem cercat per desenvolupar serveis en l'àmbit de l'streaming per a sistemes SD, HD i 3D. Consisteix en un conjunt de casos d'estudi i proves que descriuen la configuració d'aquests tipus de serveis amb codificadors de codi obert. Destaquem l'ús del framework multimèdia gstreamer.

Al primer capítol introduirem alguns conceptes relacionats amb els serveis d'streaming extrem a extrem des del procés de codificació fins a la reproducció dels clients.

Al segon capítol tractarem les aplicacions amb les que hem treballat i treure'm conclusions de la nostra experiència amb aquestes eines.

Al capítol tercer, desplegarem diversos serveis d'streaming. Farem servir diferents fonts de vídeo SD, HD i 3D, amb codificadors HW i de codi obert. Descriurem el procés complet del servidor d'streaming i configurarem streams per a dispositius mòbils.

El quart capítol tracta sobre les conclusions i les línies d'investigació futura en aquest àmbit. Farem una comparativa econòmica entre serveis de codi obert i de pagament per demostrar la seva viabilitat com a oportunitat de negoci. Els annexos són manuals d'aplicació de les eines amb els que hem treballat.

Title: Open source tools for streaming services

Author: Juan Luis Rodríguez Blanco

Director: David Rincón Rivera

Date: May 14th, 2012

Overview

The offer of multimedia services has increased thanks to IP network communications and the new behaviour of costumers. Nowadays, TVs and IP streaming based devices can play High Definition (HD) video and 3D video. Multimedia streaming is used in several environments such as telemedical assistance, IP surveillance, military security, robotics, TV and cinema.

In order to deliver multimedia services we need new tools and applications. Open source software and frameworks are one of the best options to deal with multimedia. Besides, open source philosophy takes advantage of its very economical development. This is an important reason for companies and users interested in multimedia services based in open source development, and has also several advantages for the academic and teaching point of view.

In this thesis we describe and set up several streaming media services for SD, HD and 3D video delivery. We work with different video and audio codecs and stream them to any type of devices such computers or mobile phones. Gstreamer multimedia framework outstands as the most useful tool.

Chapter one introduces the main concepts related with multimedia streaming, and describes the codecs, protocols and conventions that deal with multimedia.

Chapter two is a description of the applications and frameworks we have worked with.

Chapter three deals with multimedia services configuration and development. We work with webcams, DVB tuners, HD hardware encoders and 3D video sources and players. We stream audio and video to mobile devices, like Iphone and Android.

Chapter four concludes the thesis and describes possible future developments in this field. It includes an economical study to compare open source services and professional solutions. The annexes are a set of laboratory tests describing how to deal with gstreamer and streaming services.

ÍNDEX

INTRODUCCIÓ

La transmissió de fluxos audiovisuals és un dels serveis més utilitzats a Internet. Les capacitats de les xarxes IP en termes d'ample de banda i interactivitat fan possible un nou escenari on es poden assolir els requeriments del transport dels senyals audiovisuals sobre xarxes IP.

Els fluxos de senyals HD i 3D així com l'aparició de nous dispositius com són les TV 3D, smartphones, videojocs i l'ús de les xarxes socials han fet augmentar l'ús de tecnologies multimèdia. Davant d'aquest nou escenari de consum de serveis audiovisuals, sorgeix la necessitat per part d'usuaris i empreses de disposar d'eines que ens permetin oferir aquests serveis i configurar aquests tipus de sistemes.

Al mercat trobem dispositius codificadors, servidors d'streaming i aplicacions de pagament (amb llicència) que ens permeten fer aquesta tasca. A nivell professional les solucions basades en hardware dedicat són l'opció més directa però –tal com volem fer veure amb el desenvolupament d'aquest treball- hi ha alternatives de codi obert que les poden substituir en determinats àmbits, suposant un estalvi (de diners i temps) en la implantació i desplegament d'aquest tipus de serveis. Els usuaris finals i desenvolupadors necessiten un conjunt d'aplicacions i tècniques que els permetin treballar amb aquests tipus de sistemes de forma eficient i amb el menor cost possible.

D'aquest conjunt de necessitats i amb l'interès que es coneguin i es facin servir alternatives de lliure distribució i gratuïtes sorgeix la motivació de realitzar aquest estudi. Fer servir les eines descrites a aquest estudi pot suposar una millora als usuaris interessats en aquest tipus de sistemes, des del punt de vista econòmic per a empreses i des del punt de vista d'aplicació i estudi per a la docència.

El constant desenvolupament d'eines opensource per a Linux i altres sistemes operatius configuren un conjunt de noves eines que permeten treballar amb fluxos de dades i fitxers audiovisuals de forma eficient. Les distribucions de Linux i la utilització cada cop més estesa d'eines i sistemes de codi obert per part del usuaris han afavorit el desenvolupament d'aquestes eines que suposen un avanç tecnològic.

A partir de l'experiència laboral de l'autor d'aquest TFC en empreses de serveis audiovisuals, com a tècnic de sistemes i com a gestor de producte a un majorista i integrador de serveis audiovisuals, s'ha comprovat durant aquests anys la demanda d'aplicacions de lliure distribució.

A aquesta empresa, com a tècnic de sistemes, treballavem amb un conjunt d'equips codificadors on els periodistes seleccionaven els vídeos que volien codificar per que després es publiquéssin als diferents webs de les empreses de la corporació. Per les difusions dels senyals de directe, l'emissió del que veiem per TDT a Internet en temps real, contractaven el servei Windows Media Server (WMS) per enviar els fluxos audiovisuals a una Content Delivery

Network (CDN) que feia la tasca de repetidor del senyal a diferent nodes de la xarxa de forma que l'emissió de continguts pogués visualitzar-se a la mateixa qualitat a qualsevol ubicació IP. L'única part del procés que es feia amb eines de codi obert era el procés de codificació del còdec MPEG-2 (extret de les captures de TV del grup de canals d'aquesta corporació). La resta de gestió de directes es feia amb tecnologies de pagament. Actualment com a gestor de producte en aquest majorista audiovisual ens trobem amb clients que ens sol·liciten codificadors de vídeo per IP en temps real.

La majoria d'aquests equips són cars ja que són solucions professionals enfocades a sistemes on la latència és crítica , per exemple en aplicacions de control d'aviació militar com drones [1]o per aplicacions de telemedicina. A banda d'aquests serveis on la latència i l'encriptació de les dades és una prioritat , hi ha serveis que demanden equips més econòmics per retransmissions broadcast on la latència no és crítica. Volem demostrar que per a aquests tipus de serveis les alternatives de codi obert es poden configurar com una solució real i econòmica.

Un altre dels serveis més demandats pels clients són els serveis de vídeo sota demanda. El recent tancament de plataformes de distribució audiovisual (com ara MegaUpload[2]) per termes de legalitat deixa clara la necessitat d'implantar nous serveis més competitius que permetin adaptar-se a un marc legal. En aquest cas les tecnologies de lliure distribució permeten fer-ho ajustant els costos totals del sistema i faciliten la implantació d'aquest tipus de serveis. La possibilitat de configurar sistemes de transmissió multimèdia en temps real obre la porta a molts àmbits d'aplicació.

A l'àmbit de la telemedicina es transmeten operacions en temps real i fins i tot es comuniquen metges i hospitals que comencen a implantar aquest sistemes per a formació i en la diagnosi. També es comencen a implantar codificadors de vídeo en temps real per a avions de passatgers emetent fluxos de vídeo entre el costat aire i el control a terra. Amb l'estudi d'eines de codi obert es poden desenvolupar aplicacions en qualsevol àmbit d'aplicació i un estudi satisfactori d'aquestes eines obre la porta a una oportunitat de negoci.

L'objectiu d'aquest TFC és trobar, provar i avaluar les eines de codi obert que ens permetin treballar amb aquest tipus de serveis. Per fer-ho desenvoluparem un conjunt de proves del diferents softwares. Es descriuran diferents escenaris d'aplicació on s'argumentaran les proves fetes i es justificarà l'ús més adient de cadascuna.

En una vertent acadèmica el conjunt de pràctiques d'aquest TFC pretén ser una guia de suport per a les pràctiques de l'assignatura Serveis Audiovisuals sobre Internet d'aquesta escola, amb l'objectiu d'introduir al lector les eines i metodologies necessàries per treballar amb sistemes d'streaming. Identificarem i descriurem el funcionament dels protocols, codificadors i diferents escenaris de sistemes d'streaming. Escollirem el codificador i els protocols més adients

[1] Drones: aeronaus per observació i control no tripulats equipats amb càmeres d'alta definició i codificadors de baixa latència.
[2] Coneguda plataforma d'streaming tancada per conflices legals. Trobem la notícia del tancament de MegaUpload a [37]

per a cada servei, en configurarem els paràmetres més importants i aprendrem a operar i desplegar serveis d'streaming d'àudio i vídeo. En quant a la distribució d'aquest TFC la resta de capítols s'organitzen de la següent manera.

Al primer capítol es descriuen les bases teòriques, els conceptes que cal tenir clars, dels senyals audiovisuals i els protocols de comunicació entre els diferents blocs del sistema. Veurem els tipus de còdecs més utilitzats i, concretament, els còdecs estàndards de la família MPEG i les estructures de dades de la capa de transport. Introduirem exemples de sistemes d'streaming i estudiarem els diferents protocols de xarxa que comuniquen els diferents elements d'streaming des del codificador fins als clients. A la part d'aplicació on estudiarem les diferents eines que tenim disponibles.

Al segon capítol s'introduirà el programari i sistemes de codi obert. Es detallen les diferents aplicacions amb les que hem treballat. Les diferenciarem segons el tipus de servei que ofereixen i en descriurem la utilització de cadascuna. El conjunt d'aplicacions i frameworks multimèdia que estudiarem són:

Linux dvb-apps
Gstreamer
ffmpeg
VLC media player
MuMuDVB
Wowza
Bino 3D Player
Project-X
Avidemux

Al tercer capítol es descriuran un conjunt de casos d'estudi on farem servir les aplicacions escollides i s'estudiaran diferents escenaris d'aplicació. Principalment treballarem amb diferents tipus de fonts d'entrada d'AV, ja siguin codificadors per V4L2 (Vídeo per a Linux), solucions audiovisuals professionals com codificadors hardware i càmeres d'alta definició (HD), sintonitzadors de TV digital terrestre DVB-T i fonts i reproductors de vídeo 3D.

D'aquestes descripcions en traurem un conjunt d'eines, casos pràctics i proves fetes que ens permetran deduir conclusions generals del sistema i de les línies d'investigació futures en aquest àmbit. A més a més podrem comparar les diferents solucions hardware i de lliure distribució per tal de saber quines avantatges i inconvenients ens trobem treballant amb cadascuna.

El quart capítol inclou les conclusions i línies futures de treball en aquest àmbit. Comentarem les conclusions extretes dels diferents estudis i detallarem quins aspectes d'aquest treball poden ser interessants per a futurs estudis de sistemes audiovisuals. Detallarem un conjunt de projectes hipotètics i farem un estudi econòmic de cadascun per provar la viabilitat de l'ús d'eines de codi obert. Volem demostrar la potència d'aquestes eines i l'estalvi que poden suposar per a usuaris i a empreses.

CAPÍTOL 1. CONCEPTES BÀSICS

1.1. Introducció als conceptes bàsics

Des dels seus orígens Internet afavoreix l'intercanvi d'informació de dades entre usuaris. Inicialment les dades a transmetre eren principalment documents de text o HTML que els clients sol·liciten a un servidor per tenir-ne accés. Poc a poc van començar a introduir-se elements multimèdia en aquest tipus de presentacions que permetien visualitzar fotos i música entre d'altres continguts.

En tots el àmbits de difusió d'informació, premsa escrita, ràdio i TV, Internet ha suposat un canvi en com els usuaris consumeixen aquests tipus d'informació. Els nous dispositius mòbils amb connexió a Internet i equipats amb càmeres i micròfons han generat nous hàbits de consum i generació de continguts multimèdia que també podem compartir actualment a través d'Internet. Les aplicacions de transmissió multimèdia s'estenen en molts àmbits d'ús com ara el cinema, premsa i comunicació, robòtica, seguretat, televigilància i telemedicina.

Tecnològicament Internet s'ha millorat per poder treballar amb aquest tipus de dades i serveis. En aplicacions multimèdia es requereix una comunicació constant entre client i servidor. Els equips, les infraestructures de xarxa, i l'ample de banda dels usuaris finals, han millorat les seves capacitats, en termes de capacitat de reproducció i interactivitat, i permeten la difusió de continguts amb qualitats i serveis que abans no eren possibles.

Els nous còdecs i protocols dissenyats per aquest tipus de serveis juntament amb les raons anteriors han afavorit aquest desenvolupament. En aquest marc sorgeix la necessitat d'eines que ens permetin treballar i desenvolupar aplicacions i serveis multimèdia a Internet. Les noves tecnologies de codi obert cada cop més potents són una eina molt eficaç com veurem durant el desenvolupament d'aquest TFC. A més a més des del punt de vista de la docència i fins i tot en el desenvolupament d'aplicacions professionals són una alternativa a tenir en compte envers d'altres tecnologies de pagament o amb llicència.

La transmissió d'un flux de dades multimèdia a Internet és el que coneixem per streaming. Un sistema d'aquest tipus ens permet reproduir continguts audiovisuals a Internet, tot controlant el servei de reproducció. Això vol dir que podrem fer pausa, aturar o avançar el contingut.

Un clar exemple de web d'streaming seria Youtube [38] que ens permet pujar vídeos a un servidor per que els clients els puguin veure i compartir. En aquest cas accediríem a un contingut que físicament resideix al servidor. Aquest tipus de servei és conegut com a vídeo sota demanda.

Cal diferenciar però un servei de vídeo sota demanda d'un servei de live broadcast o directe en que els clients accedeixen als fluxos que es generen en temps real des del codificador, passant pel servidor d'streaming fins als clients.

La principal diferència entre tots dos serveis és l'origen de les dades, si és un fitxer complet que existeix al servidor serà sota demanda i en el cas en que estem reproduint un flux directe des del codificador de vídeo fins al reproductor en temps real, estem emetent un directe o live broadcast. Un exemple de directe seria el servei de directes de TV3 [7] en que accedim directament a l'stream dels codificadors que reben el senyal de TV. En aquests tipus de serveis la latència -retard del sistema per ser processat i servit d'extrem a extrem- és un dels paràmetres a tenir en compte.

Continuant amb la configuració més bàsica, un sistema d'streaming ha d'incloure almenys aquests tres elements:

- Codificador.
- Servidor d'streaming.
- Reproductor.

En aquest primer capítol descriurem les característiques d'aquests elements i els senyals que els alimenten. La comunicació entre els diferents blocs es farà mitjançant un conjunt de protocols i convencions a nivell de format d'arxiu, capes de transport i còdecs que també seran descrits en aquest primer capítol.

1.2. Escenari bàsic sistema d'streaming

Com dèiem el codificador, el servidor d'streaming i el reproductor són els tres blocs bàsics que composen el sistema d'streaming. A la Fig.1.1 podem veure el diagrama de blocs d'un sistema bàsic d'streaming.

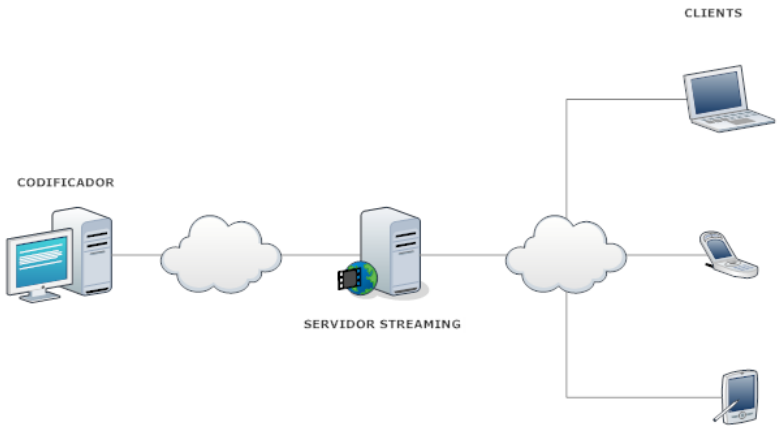


Fig.1.1 Escenari bàsic d'un servei d'streaming

El sistema està composat pel codificador que serveix els streams a un servidor ubicat a Internet. Els clients s'hi connecten per visualitzar el directe o els continguts sota demanda.

L'audiència fa servir diferents tipus de reproductors, ja sigui embegut (incrustat) a una plana web o directament com una aplicació independent.

El reproductor ataca al servidor on resideix el contingut o es rep l'stream directe del codificador. A continuació descriurem amb una mica més de detall els blocs que conformen l'escenari genèric.

1.2.1. Codificador

El codificador és l'element SW o HW que captura les senyals i les adequa al canal a transmetre mitjançant un còdec per àudio i/o vídeo . Un còdec és un conjunt d'algoritmes o aplicació capaç de codificar o descodificar fluxos i senyals audiovisuals.

L'elecció del còdec és la part més important del procés de codificació ja que d'aquest en dependran directament paràmetres del sistema com l'ample de banda de transmissió i la latència. Al nostre sistema ens trobarem diferents tipus de codificadors, solucions SW i HW que faran servir el còdec més adient per la transmissió. Els còdecs i els tipus emprats es descriuen més endavant a la secció 1.4.

El senyal que arriba al codificador, ja sigui d'àudio o vídeo, s'ha de processar abans de poder ser enviada a xarxa per tal de no ocupar més ample de banda de transmissió de l'estrictament necessari. A més a més cal protegir el senyal dels errors introduïts pel canal per garantir la comunicació. La feina del codificador és minvar el pes en bytes dels streams d'àudio i vídeo i protegir el senyal dels possibles errors que introdueixi el canal, tot garantint que les pèrdues en qualitat han de ser les mínimes possibles. Exemples de codificadors serien els elements del framework gstreamer i els codificadors de vídeo hardware que tractarem més endavant.

1.2.2. Servidor d'streaming

El servidor d'streaming reenvia els fluxos de dades als clients tot servint les peticions que li arriben. Coneixem els servidors HTTP on els clients sol·liciten documents, en el model més bàsic el servidor els hi envia el document i la transmissió entre el client i el servidor finalitza fins que arriba una altra petició.

En el cas dels servidor d'streaming, la comunicació entre client i servidor és contínua ja que en tot moment hem de tenir control del que estem reproduint. A més el flux de dades requereix un ample de banda molt més elevat que en el cas de transmissions HTTP.

A les xarxes IP s'utilitzen tècniques de reenviament de paquets per protegir-se de les pèrdues. Veurem que en el cas dels sistemes d'streaming, durant la comunicació entre client i servidor no ens interessa rebre els paquets perduts ja que reproduirem una part de la seqüència fora de temps i perdríem el sincronisme i la sensació de seqüència. Com exemple de servidor d'streaming, al nostre servei configurarem un servidor d'streaming amb l'aplicació Wowza, que tractarem més endavant a les seccions 2.8 i als annexos D.

1.2.3. Reproductor

Els reproductors són solucions software o hardware (Set Top Boxes[3] per exemple) que reben els streams del servidor i fan d'interfície humana del senyals, és a dir, després de la petició al servidor amb el reproductor comencen a rebre els fluxos audiovisuals. Aquests tipus de reproductors poden residir al mateix servidor (un reproductor java embegut a una plana web) o poden ser solucions independents que corren al costat del client. Els controls de volum, play, pause, stop i passar a pantalla completa són les característiques de control més bàsiques que ha d'oferir el reproductor.

1.2.4. Conclusions sistema d'streaming

Aquest escenari genèric ens serveix per descriure el servei més bàsic de transmissió d'un flux d'àudio i vídeo a una xarxa IP. Als propers apartats descriurem les bases teòriques que aplicarem al procés de codificació per tal de disminuir el pes del flux de les fonts d'àudio i vídeo. En primer lloc tractarem tècniques de reducció de components -submostratge de croma- per a vídeo i continuarem amb la introducció als còdecs de vídeo i àudio que farem servir en el procés de codificació. Per conèixer com s'apliquen les tècniques de reducció de croma hem d'introduir els espais de color i els diferents tipus de components dels senyals de vídeo.

1.3. Espais de color

Un espai de color és una abstracció matemàtica que descriu valors que representen els colors de l'espectre visible [39]. Per exemple a partir dels colors elementals utilitzats a impressió amb tinta ,cian, magenta i groc, podem generar combinacions d'aquests (mesclant les tintes) i parametritzar l'espectre visible del tots els colors que es poden imprimir sobre paper blanc. Alguns exemples d'espais de colors amb els que treballarem a sistemes electrònics, on la superfície sobre la qual projectem llum és negra (i s'aplica la teoria additiva del color, on la suma dels colors bàsics dóna blanc com a resultat) són:

RGB: Espai de color basat en la suma de components vermell, verd i blau.

YPbPr: Espai de color basat en RGB que separa les components de vermell, verd i blau en les components Y Iluminància, Pb (diferència entre la component iluminància i la component blava) i Pr (diferència de la component iluminància i la component vermella). Existeixen dues versions YPbPr per senyals analògiques i YCbCr o YUV per a vídeo digital. Tots els codificadors apliquen reducció de croma a les imatges.

No entrarem a definir més a fons les components de iluminància i com estan relacionades amb les components RGB. Només ens cal entendre que hi ha mecanismes per reduir el pes del component croma. Aquesta reducció es

[3] STB: Decodificador i reproductor de fluxos audiovisuals per a TV.

coneix com a Chroma Subsampling o submostratge de la component de croma. A [1] podem ampliar més informació sobre el submostratge de croma.

El motiu principal de donar prioritat a la informació de luminància i color és que l'ull humà és molt més sensible als canvis de luminància (brillantor) que als canvis de color. Això es degut a les característiques de les cèl·lules dels ulls que fan de sensors, bastons per la luminància i cons pel color.

1.3.1. Submostratge de croma

Com l'ull humà és més sensible a les variacions de luminància que a les variacions de color, s'aprofita aquesta característica de la vista per a eliminar les components a les que els nostre ulls són menys sensibles. Per a fer-ho es defineixen diferents esquemes de mostreig de la forma J:a:b (per exemple esquema 4:2:0). Els valors que indiquen les variables J:a:b són:

- J Referència de mostreig horitzontal. Normalment té un valor de 4.
- a Nombre de mostres de les components de croma Cr i Cb a la primera columna dels píxels J.
- b Nombre de mostres addicionals a la segona columna de píxels J.

A la Fig.1.2 podem veure els diferents esquemes de mostreig de croma i els valors dels paràmetres J, a i b.

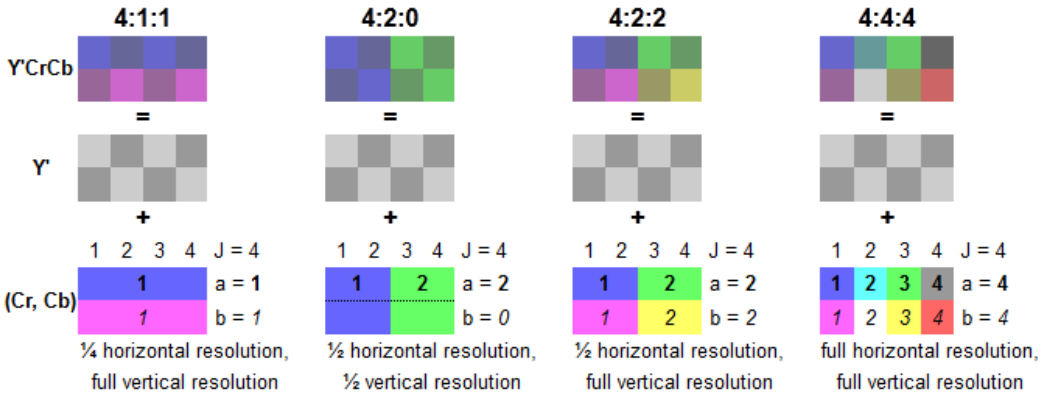


Fig. 1.2 Esquemes submostratge de croma extret de [1].

Alguns dels còdecs de la familia MPEG treballen amb un esquema de mostratge de croma 4:2:0 per a emissions broadcast (cap als consumidors finals, els espectadors), tot i que hi ha perfils de l'estàndard que permeten treballar amb altres esquemes de mostreig amb millor qualitat (4:2:2, que s'utilitza internament als estudis i als processos d'edició per no perdre qualitat). Un cop sabem com es mostregen el conjunt d'imatges tot reduint les components a les que som menys sensibles, continuem amb les tècniques de codificació amb la introducció als còdecs.

1.4. Còdecs

Els còdecs multimèdia són un conjunt d'algoritmes capaços de codificar i descodificar fluxos audiovisuals tot seguint les especificacions detallades en normes o estàndards. Per tant els còdecs són implementacions software i hardware que poden o no complir les especificacions d'un estàndard. Exemples de còdecs no estàndard serien xvid per difusió de senyals de vídeo a Internet i MxPEG per emmagatzemar vídeos en l'àmbit de la televigilància [40].

Exemples de còdec estàndard serien per exemple la família de còdecs MPEG que tractarem a la secció 1.5. Cada còdec te unes característiques concretes que el fan més o menys útil depenent del sistema complet on el volem integrar. Cal tenir alguns conceptes clars relacionats amb els còdecs com són la compressió i la latència.

1.4.1. Compressió

El terme compressió es refereix a la capacitat del còdec de reduir el pes en bytes d'un flux o fitxer. Com més comprimit i amb més qualitat vulguem els senyals AV, més temps trigarà i més latència afegirà al sistema. Cada còdec triga més o menys temps depenent de la complexitat de l'algoritme. Sempre hem de tenir en compte els requeriments del sistema concret. Al capítol 3 i als annexos A veurem exemples de com configurar correctament els diferents paràmetres del codificador aplicant diferents nivells de compressió.

1.4.2. Latència

La latència que introdueix el còdec és el temps que triga el còdec en codificar i/o descodificar el flux . Serà temps de processament que afegirà el còdec al sistema i estarà directament relacionat amb el temps que es triga en descodificar o codificar el senyal, que no ha de ser necessàriament el mateix.

En comunicacions bidireccionals interactives en temps real la latència en un sentit (transmissor – receptor viceversa) ha de ser inferior a 200 ms, [5], per garantir una comunicació extrem a extrem satisfactòria.

Els requeriments de latència varien en cada sistema i s'ha de tenir en compte durant la seva implementació. Tot element d'un sistema introdueix una latència, des de la càmera fins al reproductor. La suma total de cada una de les latències dels equips és la latència del sistema.

Una de les millors proves per mesurar la latència és fer-se amb un cronòmetre que mesuri ms i una càmera de fotos. Hem de capturar una imatge entre el cronòmetre real i el cronòmetre que veiem al reproductor. La diferència de temps serà la latència del sistema.

Continuant amb els còdecs, principalment en podem distingir dos tipus, còdecs amb pèrdues i còdecs sense pèrdues.

1.4.3. Còdecs sense perdues

Els còdecs sense pèrdues es fan servir en aplicacions que requereixen emmagatzemar dades comprimides garantint que no hi ha pèrdua de qualitat. La compressió d'aquests còdecs és menor que la obtinguda amb còdecs amb pèrdua però la qualitat ha de ser comparable. A [2], és descriuen un conjunt de còdecs sense pèrdues. Cal destacar-ne l'ús de còdecs de lliure distribució per a àudio com FLAC Free Looseles Audio Codec.

1.4.4. Còdecs amb pèrdues

La majoria de còdecs fan perdre qualitat a costa de guanyar en termes de compressió. Es pot aconseguir que les diferències de qualitat entre un flux o fitxer sense comprimir i comprimit siguin inapreciables depenent de com configurem el còdec i de la qualitat d'aquest.

L'objectiu és reduir la quantitat de bytes a transmetre entre el codificador i el servidor d'streaming. Cada còdec s'ha de fer servir depenent del sistema d'streaming i de les característiques del vídeo o àudio que estem emetent. Així doncs si volem emetre un partit de futbol, on el que prima és la sensació de moviment, hem d'ajustar el codificador per que no hi hagi pèrdues en termes de sensació de moviment. Sacrificarem per altra banda la informació de color que en aquest cas no serà crítica.

Hem introduït els tipus de còdecs amb pèrdues. A continuació descriurem la família de còdecs MPEG. D'aquest grup de còdecs en descriurem els diferents estàndards que es fan servir a comunicacions de sistemes audiovisuals de TV i ràdio actuals.

1.5. Códificació MPEG-1

MPEG-1 (estàndard ISO/IEC 11172) és un estàndard que defineix còdecs amb pèrdues per transmissió i compressió de senyals audiovisuals. És l'estàndard de transmissió i codificació multimèdia més estès a nivell mundial. MPEG-1 està composat per 5 parts en les que es descriuen les diferents capes del sistema. La part 2 tracta sobre la compressió de vídeo i la part 3 sobre la codificació d'àudio. No entrarem a definir les capes de sistema en aquesta part dedicada a l'estudi dels diferents còdecs, ho farem més endavant al punt 1.10.3.

1.5.1. Codificació vídeo – MPEG-1

El còdec de vídeo definit a MPEG-1 comprimeix el senyal de vídeo basant-se en el fet de que en les seqüències de vídeo sempre hi ha informació espacial i temporal redundant o que es repeteix. Això s'explica entenen un vídeo com un flux continu d'imatges, on si la imatge no canvia molt (com passa en un canvi de pla) la imatge següent que veurem serà la mateixa, amb la diferència dels píxels que s'han mogut.

És a dir, en una mateixa seqüència, si a la imatge N+1 li restem la imatge N només ens queden els canvis que hi ha entre N+1 i N. El còdec de vídeo d'MPEG-1 es basa en aquesta dependència entre les imatges d'una mateixa seqüència. Si transmeto la imatge N, la imatge N+1 serà igual que N amb les petites variacions de píxels (moviment i textura) entre una imatge i una altra. Eliminant la resta de la informació, que és redundant, aconseguim disminuir el pes del flux de vídeo. MPEG-1 comprimeix el vídeo com una seqüència d'imatges relacionades entre si. Per fer-ho durant el procés de codificació es defineixen diferents tipus de frames.

1.5.2. Tipus de frames

I Frame Intra-frame

És un tipus de frame que pot ser descodificat sense estar relacionat amb altres frames. S'anomena *keyframe* i és a partir d'aquests frames que es generen els frames *predicted* i *bidireccional*. Els quadres I poden ser considerades com a imatges JPEG. Cal destacar el concepte de GoP o *Group of Pictures*, que és el número d'imatges que hi ha entre dos keyframes I.

P Frame (Predicted-frame)

Conegudes com Predicted-frame. Els frames de tipus P es comprimeixen guardant només la diferència amb la referència amb la imatge anterior del tipus I. Les imatges P o B es generen dividint una imatge I en una matriu de píxels de certa mida anomenada macro bloc. El còdec calcula com s'han mogut els macro blocs a la imatge i els mou tot seguint el que s'anomenen vectors d'aproximació de moviment. El procés de reducció per predicció de moviments és una de les formes de reduir el pes dels fluxos de vídeo.

B frame Bidireccional-frame

Conegudes com Backward-predicted frames o B-pictures, un frame N de tipus B es basa en la predicció de la imatge a partir d'una referència (I o P) anterior i posterior. Així doncs per descodificar-les cal rebre dues imatges i per tant el buffer per a processat ha de ser major que en els altres casos.

Hem introduït els conceptes bàsics del còdec MPEG-1. A continuació a la secció 1.6 descriurem la versió millorada del còdec i de la mateixa família, MPEG-2.

1.6. Códificació MPEG-2

MPEG-2 és el nom d'un conjunt d'estàndards per a codificació d'AV publicats a la norma ISO/IEC 13818. Va ser definit per un grup d'empreses del sector audiovisual juntament amb l'ITU. Esta definit en diferents parts que descriuen les capes de compressió i de sistema. Més endavant a la secció 1.10 tractarem les parts relacionades amb la capa de sistema. En aquest apartat volem centrar-nos en la definició dels còdecs per àudio i vídeo.

MPEG-2 és un conjunt de mètodes de codificació amb pèrdues que permeten emmagatzemar i transportar dades audiovisuals ajustant-les a les capacitats del sistema en termes de capacitat d'emmagatzematge i ample de banda de transmissió. Inclou els còdecs que es fan servir a les normes DVB tant per SD com per HD.

1.6.1. Tipus de frames MPEG-2

MPEG-2 és força similiar a MPEG-1 implementant algunes millores com suport per a vídeo entrellaçat, com es descriu a [3]. MPEG-2 ha demostrat millors resultats de codificació que MPEG-1.

MPEG-2 es basa en la compressió de vídeo que hem vist a MPEG-1 amb algunes millores. També hi ha una estructura de GOP (imatges I, P i B) i s'aplica el submostratge de croma. Per a comunicacions de TV professionals s'utilitza l'esquema 4:2:2 i per a emissions broadcast el 4:2:0. Una seqüència d'imatges habitual és la de 15 frames de llarg amb l'estructura I-BB-P-BB-P-BB-P-BB-P-BB, referits als tipus de frames definits a la secció 1.5.2.

Els quadres I codifiquen redundància espaial i els quadres P i B redundància temporal. Els P i B es processen per compensació de moviment i es generen els vectors que mapejen els macroblocs.

La quantitat de frames de tipus I, P i B estan relacionats amb el contingut del vídeo que s'emet i la configuració de la taxa de bits del codificador que fita el flux. La configuració del codificador augmentarà la latència si es fan servir més frames del tipus B que requereixen més temps de processament. La taxa de bits de sortida d'un codificador MPEG-2 pot ser constant CBR Constant Bit Rate o variable VBR Variable Bit Rate. A la Taula 1.6.1 i Taula 1.6.2 es detallen els paràmetres de configuració de fluxos de vídeo MPEG-2 utilitzats a Europa, Estats Units i Japó.

RESOLUCIÓ	FPS
720 × 480 píxel	30
640 × 480 píxel	30
544 × 480 píxel	30
480 × 480 píxel	30
352 × 480 píxel	30
352 × 240 píxel	30
720 × 576 píxel	25
544 × 576 píxel	25
480 × 576 píxel	25
352 × 576 píxel	25
352 × 288 píxel	25
352 × 576 píxel	25
352 × 288 píxel	25

Taula 1.6.1. MPEG-2 a Europa (DVB)

RESOLUCIÓ	FPS
1920x1080 píxel 1080i	60
1280x720 píxel 720p	60
720x576 píxel 576i/576p	50/25
720x480 píxel 480i/480p	60/30
640x480 píxel	60

Taula 1.6.2. MPEG-2 a EEUU (ATSC) i Japó (ISDB).

Amb MPEG-2 ja arribem a resolucions de vídeo de fins a 1080i. Moltes de les emissions de TV actuals fan servir aquesta resolució. 1080i és la resolució de vídeo per a 1920 píxels x 1080 línies per a vídeo entrellaçat (interlaced). Això vol dir que per cada segon (a 60 fps) s'utilitzaran 30 frames per a les línies parell i 30 frames per a les línies senars. A 1080p es generen totes les línies de cada frame, en un segon tindrem 60 frames. A [41] es pot ampliar mes la informació sobre aquests modes de vídeo.

1.6.2. Codificació àudio MPEG-2

MPEG-2 introdueix nous mètodes de codificació d'àudio. Principalment permet treballar amb baixes taxes de mostratge i amb codificacions millorades MPEG-1 audio layer 1/2/3

1.6.3. Conclusions còdec MPEG-2

Les millores d'MPEG-2 són principalment permetre resolucions superiors a les que oferia el seu predecessor MPEG-1 i suporta vídeo entrellaçat. MPEG-2 s'utilitza per a retransmissions de la TDT a SD. La versió millorada del còdec, MPEG-4 concretament AVC es descriu al següent apartat. Les retransmissions de TV HD i full HD es codifiquen seguint les normes d'aquest estàndard.

1.7. Códificació MPEG-4

MPEG-4 és un còdec estàndard per a fluxos d'àudio i vídeo digital i forma part de la família de còdecs MPEG. Compleix moltes de les característiques d'altres estàndards de la família, MPEG-1 i MPEG-2. Es fa servir sobretot en difusió d'streaming multimèdia i en aplicacions de difusió de TV HD i 3D. Aquest estàndard continua en desenvolupament i es divideix com els altres estàndards en diferents parts.

La part que tracta sobre el còdec de vídeo és la definida a MPEG-4 part 10 o MPEG-4 AVC/H.264, on les sigles AVC es corresponen a Advanced Vídeo Codec. Hi ha una implementació d'aquest còdec de lliure distribució, x264enc amb la que treballarem al capítol 3 i als codificadors dels annexos A. A [33] hi ha més informació sobre x264enc. Moltes de les prestacions continuen en desenvolupament i, estant definides a l'estàndard, encara no s'han

implementat. S'ha deixat la porta oberta a desenvolupadors de software per aquesta tasca.

Inicialment el còdec va ser desenvolupat per treballar amb taxes de bits baixes per servir fluxos a dispositius mòbils però degut a la seva capacitat de compressió i versatilitat es van implementar parts per a taxes de bits de transmissió més elevades. A MPEG-4 es defineixen nivells i perfils de vídeo que permeten treballar amb aplicacions de més baix nivell fins a transmissions professionals de TV en HD.

Centrant-nos en les parts de codificació n'hem de tractar les dues més importants referides als còdecs d'àudio ,definida a la part 3 de l'estàndard, i de vídeo, definida a la part 10.

1.7.1. AAC Advanced Audio Codec

AAC és un còdec estàndard d'àudio. És un còdec amb pèrdues dissenyat per codificar fluxos d'àudio digitals i té més qualitat a bit rates similars que el seu predecessor mp3.

És el còdec d'àudio estàndard per a molts dispositius electrònics com l'Iphone amb el que treballarem als casos pràctics del capítol 3. Com en el cas d'AVC també es defineixen perfils, Low-Complexity profile (AAC-LC / LC-AAC), Main profile (AAC Main) i Scalable Sampling Rate profile (AAC-SSR). En el nostre cas treballarem amb el perfil AAC-LC i AAC Main. Una de les millores a destacar respecte mp3 és la possibilitat de mostrejar a freqüències de 8 a 96 kHz.

Com en el cas de AVC, AAC permet més flexibilitat de configuració a l'hora de desenvolupar aplicacions que treballin amb aquest còdec. Aquesta flexibilitat és la que permet aconseguir millors taxes de compressió amb més qualitat. AAC fa servir estratègies per disminuir el pes dels fluxos d'àudio que es codifiquen. Principalment elimina les components d'àudio que no són perceptibles per nosaltres (fora de la banda dels 20 Hz als 20 kHz) i n'elimina la informació redundant. El còdec AAC és força més complex que la descripció feta, només n'hem destacat els aspectes més rellevants pels casos d'aplicació que tractarem.

1.7.2. AVC Advanced Video Codec

H.264/MPEG-4 part 10 és el còdec estàndard de compressió de vídeo més estès a transmissions en alta definició. AVC va ser dissenyat per treballar amb fluxos de vídeo de baix bit rate amb alta qualitat. La millora més significativa la trobem a la capacitat de compressió de l'estàndard.

Fent una comparativa amb el seu predecessor MPEG-2, un flux a la mateixa qualitat requereix gairebé la meitat del bit rate. S'han definit diferents tipus de perfils i nivells amb diferents tipus de qualitat. Aquestes extensions permeten a l'estàndard treballar amb diferents formats de vídeo, com per exemple

l'esquema 4:2:0 per a submostratge de croma. Altres exemples de formats amb més qualitat serien el 4:2:2 i el 4:4:4 per espais de color YUV.

La versatilitat de perfils i formats afavoreixen l'ús de l'estàndard per a tot tipus d'aplicacions. Des de les que requereixen bit rates relativament baixos ,com es el cas de transmissions SD a xarxes IP i per a dispositius mòbils, fins als alts bit rates necessaris per transmissions de vídeo en alta definició.

1.7.2.1. Característiques d'AVC

Les millores de l'estàndard són moltes, i per això només en definirem algunes a destacar per entendre el funcionament del nostre sistema i les millores respecte de l'anterior estàndard.

La primera són les millores en la predicció de frames, permetent fins a un màxim de 16 imatges de referència. Suposa una millora substancial en la compressió de fluxos respecte el màxim de dues referències possibles (només una al futur, i una altra al passat) que permet MPEG-2. .

També es permet modificar la mida dels macro blocs per a compensació de moviment. La mida mínima és de 4x4 píxels i la màxima de 16x16 píxels. A més a més fa tasques de compensació de moviment pel senyal de luma i permet seleccionar la mida del macro bloc amb dimensions fixes que poden variar entre els valors 16x16, 16x8, 8x16, 8x8, 8x4, 4x8 i 4x4 píxels.

La relació de croma subsampling es manté al format que s'està fent servir. Per cada macro bloc poden haver fins a un màxim de dos vectors de compensació de moviment. També s'ha millorat la resolució fins a un màxim d'un quart de píxel (qpel) per garantir la precisió i la qualitat d'imatges en moviment.

1.7.2.2. Perfils d'AVC

Per tal de treballar amb diferents tipus de resolucions de vídeo i formats s'han definit diversos perfils amb nivells de qualitat. Alguns dels que veurem són:

- Constrained Baseline Profile (CBP): s'utilitza a aplicacions que requereixen bit rates baixos, com per exemple a videoconferències i per aplicacions mòbils.
- Baseline Profile (BP): amb característiques iguals a CBP amb la millora d'aplicar tècniques de prevenció de pèrdua de dades.
- Main Profile (MP): perfil que s'utilitza habitualment a transmissions de TV digital definits per les normes DVB.
- High Profile (HiP): Aquest és el perfil que s'utilitza a dispositius de reproducció d'alta qualitat, com el cas de Blue-Ray.
- High 4:2:2 (Hi422P): Per transmissions de video HD de qualitat profesional, seguint el format 4:2:2 de submostratge. Tots els perfils anteriors utilitzen 4:2:0. Cada mostra ocupa 10 bits millorant la qualitat de la imatge i la sensació de brillantor dels colors.

- High 4:4:4 (Hi444PP): Versió millorada de l'anterior perfil que treballa amb mostreig de croma en format 4:4:4 amb una resolució de 14 bits per mostra.

Hem introduït el còdecs d'MPEG-4. A la secció 1.8 utilitzarem un perfil específic d'AVC que permet codificar fluxos de vídeo 3D. Al punt 1.10 continuarem amb la descripció de l'estàndard des del punt de vista del transport del fluxos entre els diferents elements del sistema

1.8. Códificació de vídeo 3D

En aquest apartat volem introduir les tècniques de codificació i visualització de fluxos de vídeo 3D. En el cas de vídeo 3D, ens cal tenir dos punts de vista per tenir sensació de profunditat. La visió estereoscòpica consisteix en l'observació per part d'ambdós ulls de dues imatges amb petites variacions que representen una mateixa realitat. El nostre cervell al processar aquesta petita diferencia, genera la sensació de profunditat del que veiem.

L'aplicació d'aquest principi consisteix en donar el suport necessari per que cada ull rebi un flux concret d'imatges. D'aquesta manera el nostre cervell construirà la imatge 3D del que estem visualitzant. Això vol dir que per a sistemes 3D ens calen càmeres o dispositius amb com a mínim dos sensors de vídeo. Un cop capturades les imatges per mantenir la sensació de 3D podem fer servir diferents tècniques de reproducció. A més a més s'ha implementat una millora a l'estàndard AVC per comprimir fluxos de vídeo 3D.

1.8.1. Tècniques de reproducció 3D

Al costat de reproducció existeixen diferents tècniques per tenir sensació de profunditat amb dos o més fluxos de vídeo.

Anaglífic

A cada seqüència d'imatges se li aplica un filtre de color , vermell per a una vista i verd i blau per a una altra o altres possibles combinacions de colors que veurem a la secció 2.10 amb el reproductor 3D Bino. Al reproduir el flux les imatges es superposen i fent servir unes ulleres compatibles cada ull rep la component de cada color i ens dona una lleugera sensació de profunditat.

Ulleres polaritzades

Cada vidre està polaritzat (un verticalment i l'altre horitzontalment) de manera que cada ull rep una imatge diferent a l'estar polaritzada cadascuna de les vistes en una direcció.

Imatges alternades

Aquesta tecnologia requereix un monitor compatible i un kit d'ulleres actives. Amb aquesta tècnica les ulleres es sincronitzen amb el monitor de forma que

per a cada ull s'obre i es tanquen les lents de les ulleres de manera periòdica, llavors cada ull rep només el conjunt de seqüències de la seva vista. Aquesta és de les tècniques més actuals i la que millor sensació de profunditat ens dona. En contrapartida, el sistema de reproducció (monitor, tarja gràfica i ulleres 3D o TV 3D i ulleres actives) no és econòmic i calen unes ulleres per a cada usuari.

Autoestereoscòpic

Mètode per a reproduir imatges tridimensionals de forma que la sensació de profunditat no depèn de cap dispositiu. A [4] es descriu la tècnica d'autoestereoscòpia.

Respecte als tipus de senyals de vídeo d'entrada compatibles en destaquem els següents.

Syde by syde

Cada vista és distribueix com una composició de dos frames, amb la meitat de columnes del frame que reproduirem, en un frame esquerre i un dret. Després el reproductor mostra cada vista en un únic frame, identifica quin frame es correspon a cada vista i els va mostrant de forma sincronitzada. A la Fig 1.17 es mostra aquest tipus de composició.

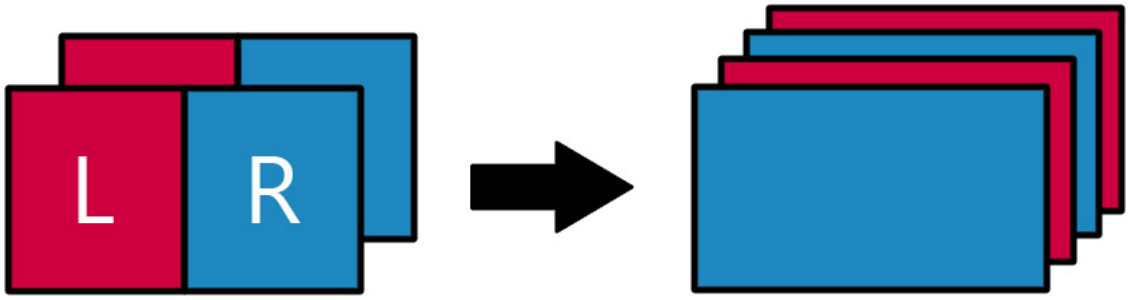


Fig. 1.17 Composició vídeo side by side. Extret de [47].

Top Bottom

En aquest cas la composició de les imatges es fa amb un frame superior i un frame inferior (meitat de files) que es correspon a cada vista. El procés de reproducció és el mateix que en el tipus side by side.

Separate frames

Els fluxos de cada vista arriben al reproductor com a fluxos independents.

Ja coneixem quines característiques tenen els fluxos 3D i com reproduir-los, a continuació explicarem la millora d'AVC per a fluxos 3D, anomenada MVC (Multi View Coding).

1.8.2. Multi View Coding – Codificació de fluxos 3D

Un dels problemes amb que ens trobem amb vídeo 3D és la necessitat de doblar l'ample de banda (si no fem servir tècniques de codificació) per transmetre el flux de vídeo estereoscòpic. A més la similitud de les imatges de totes dues vistes permet fer servir tècniques de compressió ja conegudes i aplicar-les per disminuir-ne el pes.

Per treballar amb aquests tipus de fluxos de vídeo s'ha desenvolupat una millora de la norma H.264/MPEG-4 part 10, MVC Multiview Video Coding. Aquest estàndard de codificació aplica algoritmes que es basen en la relació temporal entre fotogrames de la mateixa seqüència i en els fotogrames de les seqüències d'imatges de les altres vistes. D'aquesta doble dependència s'aconsegueixen taxes més eficients de compressió. A la figura Fig.1.8 veiem un exemple de GoP per a diferents vistes durant el temps.

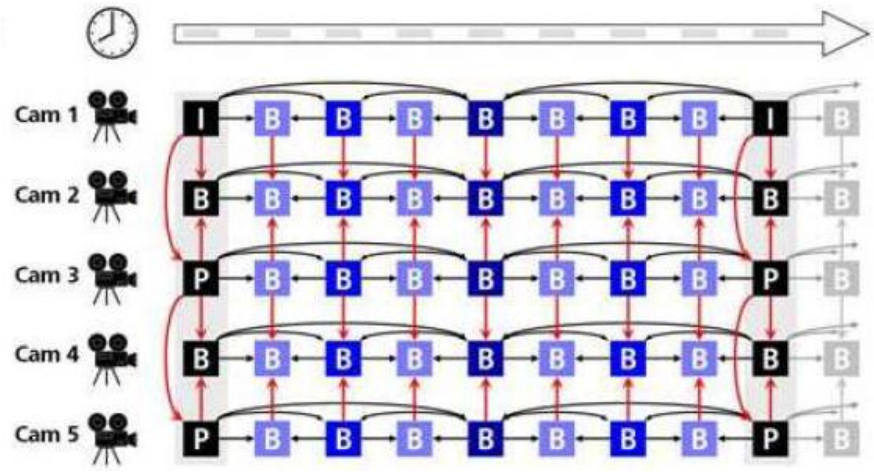


Fig. 1.8 GoP per fluxos MVC. Extret de [4]

Als actuals serveis de vídeo 3D es fa servir aquesta millora de l'estàndard. Com veurem a la secció 3.5 es poden transmetre fluxos 3D sense la necessitat de fer servir MVC, tot i així és una millora necessària per transmetre eficientment els fluxos amb diferents perspectives.

La major part de fluxos multimèdia contenen una composició de senyals audiovisuals i dades (per exemple subtítols) sincronitzades en el temps. Hem tractat el procés de codificació dels fluxos. La forma de transportar simultàniament aquestes senyals entre els diferents sistemes es fa mitjançant el que coneixem com a contenidors multimèdia.

1.9. Contenidors multimèdia

Un contenidor multimèdia és un conjunt de metadades interpretables per aplicacions que descriu com s'organitza la informació interna en un fitxer o flux. Els contenidors multimèdia són una forma de multiplexar i transportar diferents senyals audiovisuals i dades en un mateix fitxer o flux. Un contenidor

multimèdia no descriu el còdec que s'ha fet servir. Per exemple, a l'obrir un fitxer el reproductor capaç de treballar amb aquest tipus de contenidor pot llegir l'arxiu però si no hem instal·lat el còdec compatible no podrem veure ni sentir res. Així doncs no podem saber si un fitxer encapsulat en un contenidor multimèdia es reproduïble o no, ens cal saber el còdec que s'ha fet servir per a cada senyal audiovisual.

Podem anomenar els contenidors multimèdia de diferents formes, parlem de chunks (fragment d'informació multimèdia amb una capçalera que indica alguns dels valors dels senyals que transporten), atoms (moovatom de Quicktime) per a contenidors MP4 o paquets en el cas de fluxos TS transport stream que tractarem a l'apartat 1.10.3. El contingut (les dades) d'un chunk rep el nom de payload. Al capítol 3 veurem exemples de chunks en la recepció de l'stream al servidor d'streaming. A [6] podem veure una comparativa dels diferents contenidors multimèdia i de les seves característiques.

Als propers apartats descriurem els contenidors multimèdia que es fan servir en transmissions de senyals per a ràdio i TV reals, concretament els definits a les parts de sistema de la família MPEG.

1.10. Transport de fluxos multimèdia

Un cop definits alguns conceptes que cal tenir clars, hem de descriure els protocols que farem servir per comunicar els diferents blocs.

A MPEG-2 es defineix com es transporten els fluxos de dades de TV digital actuals. A continuació farem una petita introducció als orígens d'MPEG-2 i continuarem descrivint altres aspectes de l'estàndard a nivell de protocols de comunicació i capa de transport.

1.10.1. Orígens d'MPEG-2 – DVB

DVB és una organització internacional formada per diverses empreses que promouen estàndards de Televisió Digital, concretament per TV digital en SD , HD i 3D ja sigui per comunicació de dades per canals de RF, satèl·lit o per cable.

Les organitzacions que lideren aquest grup d'empreses són les següents:

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

Aquests organismes són els encarregats de crear i proposar els procediments d'estandardització per a TV digital. Els estàndards són els que s'utilitzen actualment a Europa, amb l'excepció d'altres països com és el cas de Japó, Estats Units i Canada que fan servir els seus propis estàndards propietaris. Concretament al Japó es defineix l'estàndard ISBD i l'ATSC als EEUU i Canada. A la Fig.1.9 podem veure un mapa de la distribució de tecnologies per països.

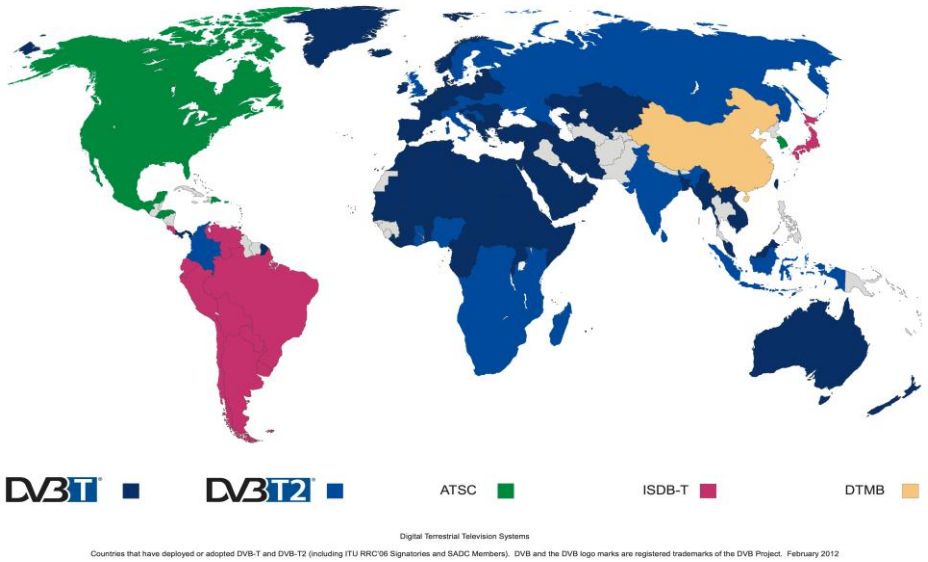


Fig. 1.9 Distribució DVB per països

Cal assenyalar que la capa de transport que fan servir els estàndards és gairebé la mateixa en els casos MPEG-2 i MPEG-4 i que les principals diferències del sistema es troben a la capa física i al procés de codificació.

Els estàndards de la família MPEG cobreixen els aspectes de codificació dels senyals d'AV i els procediments de multiplexació i sincronització d'aquests fluxos de programes tot viatjant pel canal sobre una capa de transport. Un cop definida aquesta capa de transport es selecciona el tipus de modulació utilitzada a la capa física pels diferents tipus de difusió (radiofreqüència, satèl·lit i per cable), els codis de detecció d'errors i els mecanismes d'accés condicional als serveis i programes.

DVB està compost per diferents normes en funció del canal i sistema de radiodifusió. Concretament aquestes són les normes que distribueixen les dades per DVB:

- Satèl·lit DVB-S i DVB-S2
- Cable DVB-C i DVB-C2
- Televisió terrestre DVB-T i DVB-T2
- Televisió terrestre per equips portàtils DVB-H
- Televisió per satèl·lit per equips portàtils DVB-SH
- Televisió per xarxes IP DVB-IPTV.

Les principals diferències es troben a la capa física i a la capa d'enllaç del sistema de distribució així com al tipus de modulació:

- DVB-S utilitza QPSK, 8PSK o 16QAM.
- DVB-S2 les mateixes que l'anterior i 32APSK.
- DVB-C QAM: 16QAM, 32QAM, 64QAM, 128QAM i 256QAM.

- DVB-T QPSK, 16QAM o 64 QAM combinades amb OFDM. A l'annex A.4.1 veurem com configurar un sistema d'adquisició per a DVB-T.

Els tipus de dades que es transmeten encapsulades a un TS són àudio, vídeo i dades (subtítols, teletext i informació privada). Per facilitar el suport d'aquestes dades es fan servir les normes DVB-DATA,DVB-TXT i DVB-SUB.

Els fluxos DVB es transmeten en fluxos de transport TS Transport Stream. Les característiques del TS les descriurem més endavant en aquest mateix capítol al punt 1.10.3. .

1.10.2. Capes i Sistema

La capa de sistema es defineix a la part ISO/IEC 13818-1 de MPEG-2 on es descriuen formats contenidors i de transport dels fluxos. En diferenciem dos. El primer d'ells és el que anomenem MPEG-2 Transport Stream. L'altre tipus de contenidor es conegut com a Program Stream està dissenyat per emmagatzemar fluxos a fitxers (en format .mpg) . A la Fig.1.10 podem veure un exemple de com es paquetitzen i multiplexen els fluxos en TS i PS.

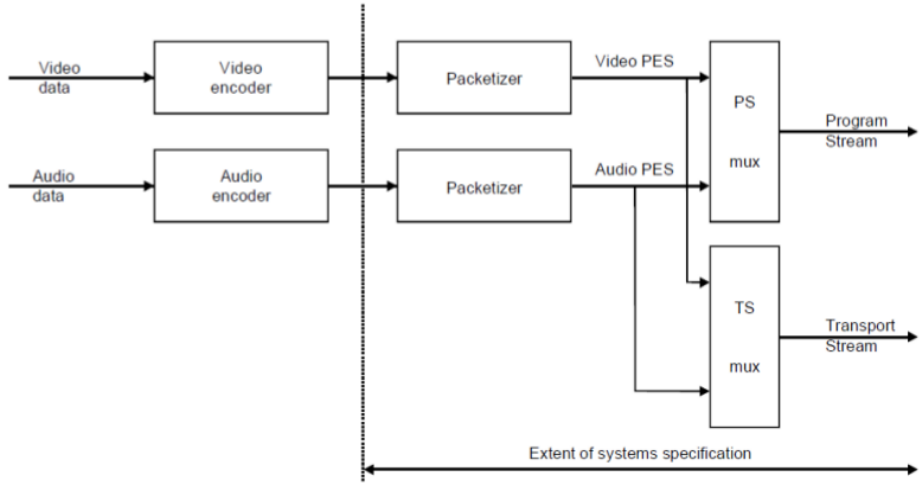


Fig. 1.10 Multiplexor de fluxos MPEG-2 TS i PS

Hem d'aclarir que un cop paquetitzats els fluxos separats d'àudio i vídeo abans del multiplexor s'anomenen Packetized Elementary Stream o PES i inclouen un únic stream amb un únic tipus de dades ja codificades. Alguns exemples de PES en casos reals d'aplicació serien fluxos d'àudio codificats en MPEG-1 layer II o mp2 i fluxos MPEG-2 MPEG-2v pel vídeo.

Transport stream és un tipus de contenidor multimèdia a la capa de transport per a fluxos MPEG-2. Els fluxos d'àudio i vídeo de cada programa que s'emet s'encapsulen independentment en el que coneixem con Elementary Stream. Cada Elementary Stream o ES conté un únic tipus de dades. Els fluxos ES es serveixen en paquets com paquetized elementary stream.

Un cop desmultiplexats, els paquets encapsulats en TS són de 188 bytes. A la Fig.1.11 es detalla informació sobre la capçalera i veiem els diferents camps que composen el paquet.

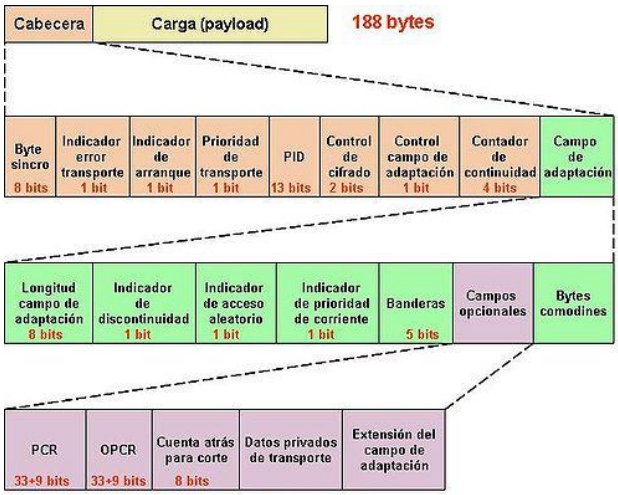


Fig 1.11 Capçalera contenidor TS

A banda d'àudio i vídeo s'emeten dades del programa (subtítols per exemple) i informació privada que es fa servir com a mecanisme condicional d'accés a continguts de pagament. Aquests fluxos passen per un multiplexor que genera a la sortida un únic flux TS que conté tots els fluxos dels N programes, com veiem a la Fig.1.12.

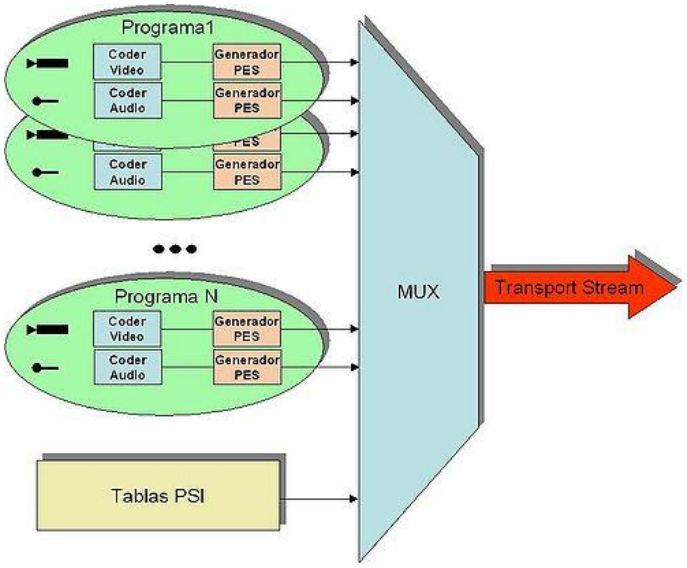


Fig 1.12 Multiplexor MPEG-2

Cada programa està sincronitzat en el temps mitjançant timestamps i els paquets tenen número de seqüencia. Cal comentar que el bit rate del flux de

sortida és constant, i encara que els fluxes elementals variïn, si cal el multiplexor afegirà paquets buits per mantenir aquest fluxe constant.

Una possible millora en aquest procès seria retroalimentar la sortida del multiplexor de forma que el bit rate de sortida s'ajusti al que s'està emetent. Això s'anomena multiplexació estadística i és com es multiplexen els fluxos a serveis profesionals, com per exemple el multiplexor de TV3 a Collserola. Idealment si s'envien molts paquets buits el multiplexor hauria de disminuir el bit rate de sortida. Aquesta millora supossaria aprofitar més eficientment l'ample de banda de transmissió necessari en funció dels continguts que s'emeten.

El TS generat conté tots els fluxes PES de cada programa i la informació de les taules Program Specific Information PSI. La senyalització del fluxe de transport es fa mitjançant les taules d'informació de servei SIT. Per poder treballar amb aquests tipus de senyals ens cal conèixer l'estructura de les taules PSI.

1.10.3. Taules PSI

Les taules PSI són una estructura de dades que contenen la informació dels programes que s'estan servint en el TS. El descodificador d'MPEG-2 consulta les taues PSI per identificar els diferents fluxes ES i descodificar-los. Cada programa té un Program Identifyer PID únic que l'identifica. Les taules que obligatoriament ha dincloure el TS són les seguents.

PAT Program Association Table

PAT ens dona la informació de tots els programes que viatgen en un TS. Amb la PAT podem saber que PID's viatgen a les taules PMT. La PID de la PAT és sempre 0.

PMT Program Map Table

Existeix una PMT per a cada programa present en el TS. Es detalla la informació dels diferents ES que viatgen associats a aquests programa:

- PID en el que viatja el fluxe elemental.
- Tipus de trama àudio, vídeo o dades.
- Descriptors.

CAT Condicional Acces Table

Ens dona informació sobre el sistema d'accés condicional. Només és obligatòria si hi ha contingut privat.

Amb aquesta introducció teorica ja coneixem com viatgen entre els elements del sistema els fluxos TS. Hem volgut donar una visió general del sistema per entendre el seu funcionament més bàsic. Amb aquesta base podem entendre els mecanismés involucrats en els processos de códificació i transport de senyals audiovisuals. Al punt 1.11 d'aques capítol tractarem els protocols que

es fan servir per comunicar els diferents blocs dels sistemes d'streaming per a transmissió de fluxos.

1.11. Xarxes IP i protocols de comunicació

1.11.1. Introducció xarxes IP

Les xarxes IP són xarxes de commutació de paquets que segueixen les normes i protocols descrites a la família Internet Protocol. Un paquet és una estructura de dades amb informació de capçalera i les dades o payload.

Són conegudes com xarxes TCP/IP ja que la majoria de comunicacions es basen en aquests dos estàndards de comunicació de xarxa, on TCP és Transmission Control Protocol i IP és Internet Protocol, tot i que es fan servir d'altres protocols. El protocol IP s'encarrega d'adreçar correctament els paquets des d'una font o host font a destí. El sistema d'adreçament basat en adreces IP facilita les funcions d'identificació del host i de localització de servei. El protocol TCP s'encarrega de la comunicació entre la capa d'aplicació i el protocol IP, i assegura la fiabilitat de la transmissió.

Les xarxes IP són un conjunt de nodes comunicats. El control del tràfic es fa pels elements de xarxa anomenats routers. En aquest tipus de xarxes, el protocol TCP fa servir tècniques de control de congestió i reenviament de paquets en el cas de que es detectin pèrdues en l'enviament.

Bàsicament els routers gestionen l'enviament dels paquets IP sobre la xarxa. Els paquets es numeren amb un número de seqüència per controlar el reenviament i l'ordre. No obstant en el cas d'streaming en temps real el reenviament de paquets no soluciona el problema de les pèrdues, ja que rebre un reenviament d'un conjunt de paquets que es corresponen a un flux que ja hauríem d'haver vist o sentit no ens interessa.

L'objectiu en aquest cas és no perdre paquets o si es perden que l'usuari detecti la menor pèrdua de qualitat possible. Dintre d'aquest model de comunicació, les xarxes IP són xarxes Best Effort , és a dir , la xarxa no asegura qualitat de servei, l'ample de banda pot variar, poden haver-hi pèrdues i el retard pot ser gran i variar.

Això fa que les comunicacions a xarxa siguin canviants. Com veurem a la secció 3.6 les noves tècniques d'streaming adaptatiu com DASH, [50], volen preveure la capacitat de la xarxa entre client i servidor i servir un flux adequat a aquesta capacitat d'ample de banda.

Els paquets IP s'encapsulen a diferents nivells per a diferenciar-ne els diferents protocols seguint el model OSI. Aquest model encapsula els paquets a diferents capes i en describim:

- Capa d'enllaç: Els protocols d'aquesta capa contenen informació de la xarxa local (LAN).
- Capa d'Internet: Interconnecta diferents xarxes locals.
- Capa de transport: Assegura la transmissió fiable d'informació extrem a extrem.
- Capa d'aplicació: comunica processos a nivell d'aplicació entre dos hosts.

A la Fig.1.13 es detalla un exemple de comunicació entre dos hosts i les diferents capes a una xarxa IP.

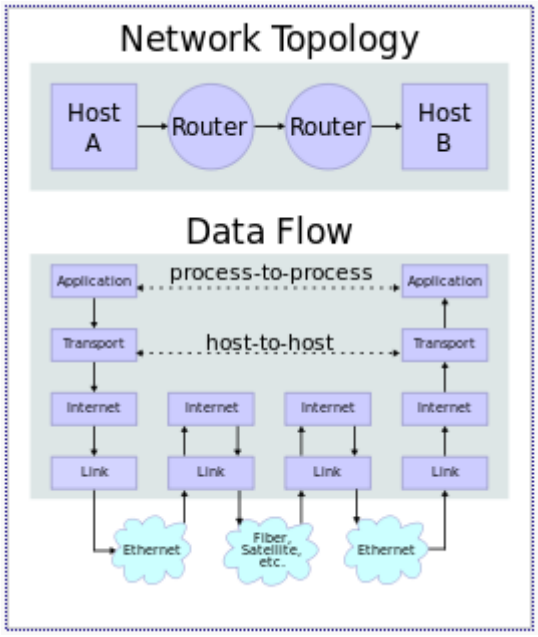


Fig 1.13 Capes i processos de comunicació. Extret de [42]

També cal distingir quatre tipus diferents d'adreçament IP tot i que nosaltres treballarem principalment amb unicast i multicast.

- Unicast: Els paquets s'adrecen d'un origen a un destí. No hi intervenen més elements que emissor i receptor.

- Multicast: L'emissor envia els paquets al router, que els reenvia al usuaris de la xarxa local interessats en rebre'l. Una adreça IP de multicast s'associa a un conjunt o grup de subscriptors interessats en rebre el flux de paquets d'aquesta adreça. Les adreces multicast es troben al rang d'adreces de 224.0.0.0 a 239.255.255.0 i reben el nom d'adreces tipus D. Per treballar amb xarxes multicast s'utilitza el protocol IGMP que descriurem al punt 1.11.6

- Broadcast: l'emissor envia a tots els destins possibles de la xarxa el flux de paquets.

- Anycast: l' emissor serveix el flux ,d'entre tots els destins possibles, al client que sigui més a prop de la xarxa. Principalment fet servir per servidors DNS.

A continuació detallarem el conjunt de protocols que intervenen en el procés de comunicació dels diferents sistemes d'streaming que volem configurar. Principalment en detallarem els protocols RTP, RTSP, HTTP i IGMP.

1.11.2. Protocol RTP

RTP és un protocol de comunicació sobre xarxes IP que defineix un format estandarditzat de paquet per a transmissions d'àudio i vídeo sobre xarxes IP.
RTP va ser desenvolupat pel grup Internet Engineering Task Force (IETF). Hi ha una versió de l'estàndard millorada que va ser publicada al 2003. RTP està dissenyat per garantir una comunicació extrem a extrem en temps real de fluxos d'àudio i vídeo. El protocol fa tasques de mesura a partir dels timestamps del paquets del jitter. El buffer del receptor RTP fa tasques de compensació del jitter. A [9] s'explica qué és i com fer-li front al jitter en comunicacions digitals.

En aplicacions de transmissió multimèdia es requereix tenir un control del temps d'entrega dels paquets, a més el protocol tolera una pèrdua relativa d'alguns dels paquets durant la transmissió.

En aplicacions multimèdia per RTP és habitual l'ús de RTP sobre UDP. En termes de compressió RTP va ser dissenyat per treballar amb diferents tipus de còdecs, com per exemple H.264, i permet la integració de nous tipus de còdecs.

1.11.2.1. Components de l'estàndard RTP i RTCP

- RTP: s'encarrega de transportar els fluxos audiovisuals. La informació que cal controlar en aquest tipus de transmissions són principalment control del temps, timestamps, nombre de seqüència (per tècniques de reenviament de paquets i reordenació, que no fa directament RTP sinó la capa d'aplicació que corre a sobre seu) i les dades o payload que detallen la compressió del senyals que transporten.

- RTCP: fa tasques de control en termes de monitorització i extracció de dades estadístiques i de qualitat de servei (QoS o Quality of Service). A més fa tasques de sincronització entre els diferents fluxos. Ocupa només el 5 % del total del bit rate de transmissió en relació a RTP.

1.11.3. Protocol RTSP

RTSP és un protocol de xarxa IP dissenyat per establir, controlar i alliberar fluxos multimèdia entre clients i servidors d'streaming. RTSP és un protocol de senyalització de fluxos i sovint es compara amb les funcionalitats d'un comandament a distància de TV. El protocol estableix una comunicació entre client i servidor que permet rebre comandes d'accions per part del client i

interpretar ordres com Play o Pause de forma que el servidor aturi temporalment l'enviament de l'stream i la torni a iniciar.

RTSP defineix el control de seqüències per garantir en tot moment el control de la reproducció. La principal diferència amb altres protocols de comunicació com HTTP és que RTSP fa servir estats. Els estats són un identificador que indica com es troba la comunicació entre client i servidor. A més a més la comunicació pot ser de client a servidor i de servidor a client. La comunicació entre client i servidor es mostra a la Fig.1.14.

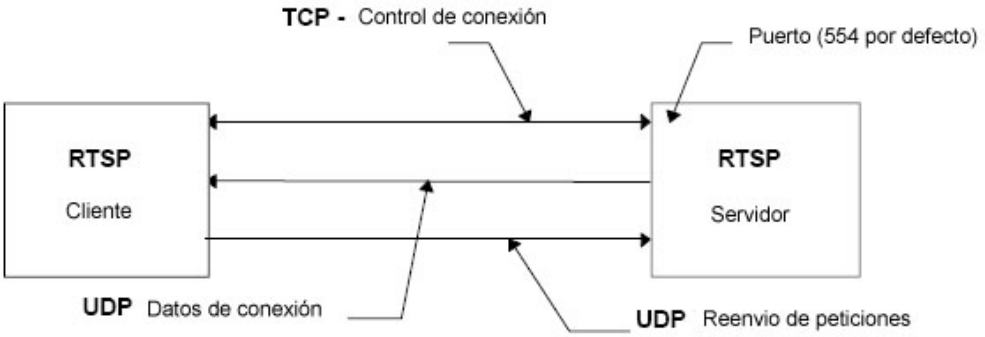


Fig. 1.14 Model client-servidor RTSP. Fluxos UDP de RTP. Extret de [43].

Els missatges que s'envien client i servidor en un procés de comunicació RTSP són:

- DESCRIBE: El servidor envia al client una descripció del contingut multimedia.
- SETUP: El client especifica al servidor per quin port rebrà els fluxos d'àudio i vídeo.
- PLAY: El client demana al servidor que li enviï els fluxos multimedia.
- TEARDOWN: El client avisa de que es vol deixar de rebre els fluxos i s'atura l'enviament de dades.

A la Fig.1.15 es descriu el procés de comunicació i els diferents missatges entre client i servidor. El client envia una petició de connexió al servidor pel port 554 (per defecte pel protocol RTSP). El client i servidor verifiquen si la connexió és possible i s'assignen ports entre 6970 i 9999 per transmetre els fluxos d'àudio i vídeo. Tot i aquest rang no està definit a cap èstandard, algunes de les aplicacions fan servir aquest rang de ports, de [45]. Les aplicacions llegiran dades dels ports assignats. Un cop s'estableix l'assignació el servidor envía els fluxos audiovisuals sobre UDP/RTP al client per aquest ports. Al capítol 3 i als annexos A veurem exemples de sesions de comunicació per RTSP.

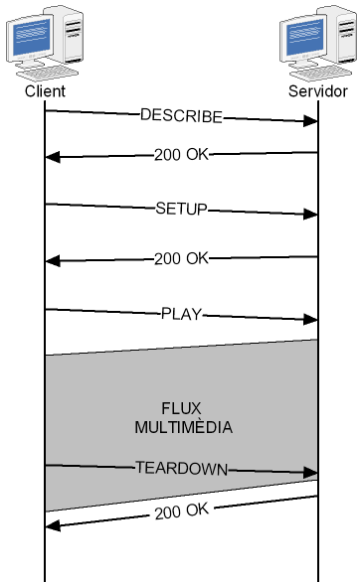


Fig. 1.15 Missatges RTSP. Extret de [47]

1.11.4. Protocol HTTP

El protocol HTTP és un protocol d'aplicació per a xarxes IP desenvolupat per IETF i el W3C World Wide Web Consortium. HTTP funciona com un model client servidor on es transmeten peticions i respostes a aquestes. El client li envia una petició al servidor i aquest li envia la resposta a aquesta petició en forma de documents HTML interpretables per un navegador d'Internet. HTTP funciona sobre capes UDP i TCP i a diferència de RTSP no hi ha estat de comunicació entre client i servidor.

Aquesta forma de gestionar les comunicacions no són la millor opció en sistemes d'streaming on es requereix una comunicació fluida entre client i servidor i servidor i client. Hi han tècniques per fer possible l'streaming de continguts amb HTML com per exemple HTML Live Streaming o HLS que permet l'enviament de fluxos a clients fent servir només transaccions HTTP.

1.11.5. Protocol HLS

HTTP Live Streaming (HLS) és un protocol basat en HTTP per a transmissions de fluxos multimèdia. Funciona dividint el flux que ha de ser servit als clients en un conjunt de seqüències en format fitxer que el reproductor interpretarà com un flux continu.
El client demana al servidor un fitxer extendend M3U que conté la llista de reproducció amb els continguts. Si l'stream d'origen s'ha codificat a diferents qualitats (diferents bit rates o diferents resolucions, per exemple) mitjançant aquest arxiu el client podrà escollir una o altra basant-se en el seu ample de banda de baixada. Així doncs HLS permet fer streaming adaptatiu. Com veurem al capítol 3, el servidor d'streaming Wowza permet als clients connectar-se amb HLS i servir fluxos adaptats als clients.

1.11.6. Protocol IGMP

Internet Group Management Protocol és un protocol de comunicació per a xarxes IP que s'encarrega de gestionar els clients que pertanyen a un determinat grup de Multicast. IGMP garanteix les comunicacions entre clients del grup i el router que està reenviant els fluxos d'àudio i vídeo. L'arquitectura de xarxa d'aquest tipus de sistemes es mostra al diagrama de blocs de la figura Fig.1.16.

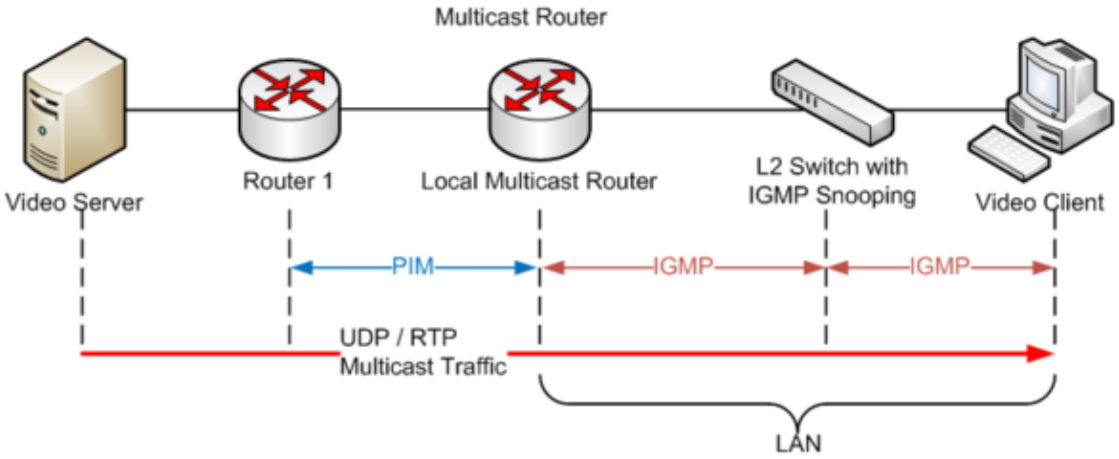


Fig 1.16 Exemple arquitectura IGMP. Extret de [44].

Aquesta arquitectura és força interessant per sistemes de difusió multimèdia. Els clients que vulguin reproduir l'stream només han d'indicar-li l'adreça de multicast per on s'accedeix al flux. El router el donarà d'alta al grup i començarà a reenviar-li a aquest client i als altres del grup els streams multimèdia.

1.11.7. URI's i URL's

Per accedir als diferents fluxos o documents dels elements de xarxa es fan servir el que anomenem, Uniform Resource Identifier o Uniform Resource Locator, URI i URL respectivament. Una URL és una cadena de caràcters que designa un recurs únic a la xarxa. Una URL es classifica pel seu esquema, el protocol que fa servir, dos punts i la seva part especifica. Un exemple seria rtsp://146.255.96.41:1935/directo/mpegts.stream/playlist.m3u8. Les URI's són l'identificador d'accés a la informació que fan servir les aplicacions com un navegador web o un reproductor multimèdia.

CAPÍTOL 2. SOFTWARE DE CODI OBERT PER A SERVEIS D'STREAMING

2. Introducció

Hem descrit els tipus de protocols i còdecs per poder treballar amb els senyals multimèdia. En aquest segon capítol descriurem les eines amb les que farem funcionar el sistema. La majoria d'aplicacions que descriurem estan disponibles per sistemes operatius Linux. L'objectiu és que els clients del nostre servei puguin fer servir qualsevol sistema operatiu i qualsevol tipus de dispositiu (PC o mòbil) per reproduir els continguts.

Per a cada aplicació descriurem el procés d'instal·lació i configuració previ. El catalogarem dins del tipus d'element dintre del nostre sistema ja sigui un codificador, servidor o reproductor i farem una descripció a mode de guia d'usuari per a la seva utilització.

2.1. Aplicacions DVB-APPS

Les aplicacions dvb-apps són un conjunt d'APIS o utilitats en Linux que ens permeten processar senyals DVB i extreure'n informació. Es poden descarregar del gestor d'aplicacions o de [10]. Les utilitats són w_scan, dvbscan, czap, tzap, szap, femon,dvbdate, dvbnet,dvbtraffic. Per l'objectiu d'aquest TFC ens cal fer servir l'aplicació w_scan. L'operació d'aquestes aplicacions es fa mitjançant línia de comandes. Per poder fer-les servir cal tenir configurada la tarja DVB correctament al nostre sistema, amb els controladors de Linux correctament instal·lats.

2.1.1. Configuració de la tarja sintonitzadora

Per capturar streams del senyal de DVB necessitem una tarja sintonitzadora i un PC amb les aplicacions DVB-apps. Ja hem descrit els tipus senyal de DVB al capítol anterior i en el nostre cas hem escollit una tarja de DVB-T per fer les captures.

Al mercat trobem molts tipus de targes sintonitzadores des de la més econòmica que permet extreure una freqüència de l'espectre a receptors enrackables que capturen tot l'espectre i en generen streams per a cada canal o programa concret. Depenent del pressupost i l'objectiu podem seleccionar-ne una o altre, en aquest cas hem escollit una tarja DVB-T econòmica de la marca Avermedia, veure annex E. Si hem instal·lat correctament la nostra tarja DVB-T executem en mode consola:

```
$ ls –lR /dev/dvb/
/dev/dvb/:
total 0
/dev/dvb/adapter0:
total 0
```

```
crw-rw----+ 1 root 212, 0 mar 27 16:33 demux0
crw-rw----+ 1 root 212, 1 mar 27 16:33 dvr0
crw-rw----+ 1 root 212, 3 mar 27 16:33 frontend0
crw-rw----+ 1 root 212, 2 mar 27 16:33 net0
```

Verifiquem que tenim configurada correctament la tarja. Han d'aparèixer les referències de l'adaptador al directori /dev/dvb/

Frontend0

La part del dispositiu receptor que adquireix el senyal de RF i n'extreu la informació subjacent que es correspon a una determinada freqüència portadora. No s'ha d'entendre el frontend només com un desmodulador del senyal ja que s'encarrega de la desmodulació i de sintonitzar a la freqüència portadora.

Demux0

La part de la tarja sintonitzadora que des multiplexa els diferents canals i programes que es reben a la tarja.

Dvr0

Sigles de Digital Video Recorder, és la part del sistema que ens permet extreure informació de la tarja i bolcar-la en forma de dades al nostre PC.

A [11] s'amplia la informació d'aquestes referències per a sistemes Linux.

Un cop configurat correctament podem fer servir les aplicacions dvb-apps, la més útil en el nostre cas w_scan per escanejar l'espectre de freqüències i trobar els canals associats a aquesta freqüència portadora.

2.1.2. w_scan

w_scan es pot instal·lar des del gestor d'aplicacions de Linux. Aquesta aplicació ens permet escanejar el senyal DVB que rebem i extreure'n informació dels diferents canals disponibles, en termes de freqüència i serveis. A consola executem:

```
:~$ w_scan –c ES

w_scan version 20091230 (compiled for DVB API 5.1)

using for SPAIN
DVB aerial
DVB-T Europe
frontend_type DVB-T, channellist 4
output format vdr-1.6
Info: using DVB   auto detection.
      /dev/dvb/adapter0/frontend0 -> DVB-T "Afatech AF9013 DVB-T": good
Using DVB-T frontend ( /dev/dvb/adapter0/frontend0)
-_-_-_-_ Getting frontend capabilities-_-_-_-_

Using DVB API 5.1
frontend Afatech AF9013 DVB-T
INVERSION_AUTO
```

```
QAM_AUTO
TRANSMISSION_MODE_AUTO
GUARD_INTERVAL_AUTO
HIERARCHY_AUTO
FEC_AUTO
-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_

Scanning 7MHz  es...
```

Amb w_scan sabem les freqüències a les que hem de sintonitzar la tarja DVB per a un grup de canals en concret. Per exemple la freqüència:

```
tune to: QAM_64   f = 794000 kHz I999B8C23D0T8G4Y0
(time: 04:43)      service = TV3 (TVC)
       service = 33 (TVC)
       service = 3/24 (TVC)
       service = Super3/3XL (TVC)
       service = CatRadio (TVC)
       service = CatInfo (TVC)
```

És la que inclou els serveis de TVC, 4 canals de vídeo, àudio i dades TV3, 33, 3/24, Super3/3XL i 2 més d'àudio i dades, CatRàdio i CatInfo. Les CAPS o capabilities són el conjunt de serveis de DVB-T compatibles amb la nostra tarja:

```
INVERSION_AUTO
QAM_AUTO
TRANSMISSION_MODE_AUTO
GUARD_INTERVAL_AUTO
HIERARCHY_AUTO
FEC_AUTO
```

La principal utilitat de w_scan és sintonitzar la freqüència portadora d'un grup de canals en concret.

2.1.3. Conclusions

Amb les aplicacions de DVB-apps podem fer tasques d'escaneig de l'espectre freqüencial. Són necessàries per poder treballar amb senyals DVB en sistemes Linux. La importància d'aquestes aplicacions és conèixer determinats valors de la freqüència portadora amb l'objectiu de sintonitzar un grup de canals en concret que contengui les dades d'AV del programa que ens interessa.

2.2. DVBsnoop

Instal·lada des del gestor d'aplicacions de Linux, també es pot descarregar de [34]. Dvbsnoop és un analitzador de fluxos TS. Ens permet extreure tot tipus d'informació relacionada amb les taules PSI. És força complert, encara que hi ha funcions que no són compatibles amb la nostra tarja DVB-T. També ens permet extreure dades d'un PID en concret. Si executem:

```
$ dvbsnoop -n 1 0x00

dvbsnoop V1.4.50 - http://dvbsnoop.sourceforge.net/
SECT-Packet: 00000001   PID: 0 (0x0000), Length: 40 (0x0028)
  0000:  00 b0 25 00 61 c5 00 00  00 00 e0 10 03 21 e0 6e   ..%.a........!.n
  0010:  03 22 e0 78 03 23 e0 82  03 24 e0 8c 03 29 e0 96   ."x.#...$...)..

  0020:  03 2a e0 a0 2d 9d ac 4f                            .*..-..O
PID:   0 (0x0000)   [= assigned for: ISO 13818-1 Program Association Table
(PAT)]
Guess table from table id...
PAT-decoding....
Table_ID: 0 (0x00)  [= Program Association Table (PAT)]
section_syntax_indicator: 1 (0x01)
(fixed): 0 (0x00)
reserved_1: 3 (0x03)
Section_length: 37 (0x0025)
Transport_Stream_ID: 97 (0x0061)
reserved_2: 3 (0x03)
Version_number: 2 (0x02)
 t_next_indicator: 1 (0x01)  [ now]
Section_number: 0 (0x00)
Last_Section_number: 0 (0x00)
    Program_number: 0 (0x0000)
    reserved: 7 (0x07)
    Network_PID: 16 (0x0010)
    Program_number: 801 (0x0321)
    reserved: 7 (0x07)
    Program_map_PID: 110 (0x006e)
    Program_number: 802 (0x0322)
    reserved: 7 (0x07)
    Program_map_PID: 120 (0x0078)
    Program_number: 803 (0x0323)
    reserved: 7 (0x07)
    Program_map_PID: 130 (0x0082)
    Program_number: 804 (0x0324)
    reserved: 7 (0x07)
    Program_map_PID: 140 (0x008c)
    Program_number: 809 (0x0329)
    reserved: 7 (0x07)
    Program_map_PID: 150 (0x0096)
    Program_number: 810 (0x032a)
    reserved: 7 (0x07)
    Program_map_PID: 160 (0x00a0)
CRC: 765307983 (0x2d9dac4f)
```

El PID 0x00 està reservat per la PAT que és una de les taules PSI que conformen un senyal DVB. Si volem extreure informació d'un altre Program Identifier, del programa 110 per exemple, executem:

```
:~$ dvbsnoop -n 1 0x006e

dvbsnoop V1.4.50 - http://dvbsnoop.sourceforge.net/

SECT-Packet: 00000001   PID: 110 (0x006e), Length: 204 (0x00cc)
```

```
Stream_type: 6 (0x06)  [= ITU-T Rec. H.222.0 | ISO/IEC 13818-1 PES packets
containing private data]
 Elementary_PID: 133 (0x0085)
 Stream_type: 6 (0x06)  [= ITU-T Rec. H.222.0 | ISO/IEC 13818-1 PES packets
containing private data] Elementary_PID: 801 (0x0321)
 Stream_type: 6 (0x06)  [= ITU-T Rec. H.222.0 | ISO/IEC 13818-1 PES packets
containing private data]Elementary_PID: 811 (0x032b)
Stream_type: 5 (0x05)   [= ITU-T Rec. H.222.0 | ISO/IEC 13818-1 private
sections] Elementary_PID: 901 (0x0385)
Stream_type: 12 (0x0c)  [= ISO/IEC 13818-6 Stream Descriptors]Elementary_PID:
601 (0x0259)
```

De la sortida anterior deduïm que el programa inclou 1 ES de vídeo (tipus 5), 3 ES d'audio (tipus 6) i 1 ES de dades. Hi ha diferents tipus d' ES de Dades, per exemple el `Elementary_PID: 901 (0x0385)` és un descriptor de senyalització i el `Elementary_PID: 811 (0x032b)` és un descriptor de subtítols.

Aquesta informació ens serà molt útil més endavant als cassos d'aplicació del capítol 3 per entendre el procés de des multiplexat del flux TS amb gstreamer. La utilitat de DVBsnoop és conèixer les estructures de transport descrites a l'apartat 1.10.4. taules PSI. Ens cal conèixer aquests valors que li haurem de passar al receptor de DVB-T per fer captures de programes concrets dins l'estructura de les taules PSI.

2.3. Ffmpeg

És pot instal·lar des del gestor d'aplicacions de Linux o descarregar de [12]. Ffmpeg és un conjunt de llibreries i API's que componen un conjunt d'eines multi plataforma (Linux, Windows i MAC) que ens permeten treballar amb senyals, còdecs i protocols d'àudio i vídeo. És una solució de codi obert, completa i compatible amb frameworks multimèdia com gstreamer mitjançant plugins que fan de pont entre les dues aplicacions. Amb ffmpeg podem codificar, descodificar, multiplexar i desmultiplexar fluxos i fitxers multimèdia. Les aplicacions que inclou són les següents:

- *ffmpeg* eina per a intèrpret de comandes. Molt útil principalment per fer transcodificacions i transformacions de formats de fluxos d'àudio i vídeo.

- *ffserver* aplicació que implementa un servidor d'streaming multimèdia en temps real.

- *ffplay* Reproductor de ffmpeg.

- *ffprobe* Analitzador de fluxos multimedia.

Es basa en les llibreries libavcodec, libavutil, libavdevice, libswscale i libswresample. A continuació en detallem les característiques.

- Libavutil llibreries que contenen funcions útils per a treballar amb estructures de dades multimèdia. Inclou funcions de generació de números aleatoris i rutines per fer operacions matemàtiques.

- Libavcodec llibreries que contenen codificadors i descodificadors d'àudio i vídeo.

- Libavformat Llibreria que conté multiplexors i desmultiplexors per a contenidors multimèdia.

- Libavdevice Permet extreure fluxos d'àudio i vídeo de dispositius capturadors com tarja sintonitzadores o capturadores de vídeo.

- Libavfilters Filtres multimèdia.

- Libswescale Permeten fer tasques de conversió d'espais de color.

- Libswresample Permet re mostrejar fluxos i fitxers audiovisuals i operacions de conversió.

Un dels avantatges de ffmpeg és poder fer servir aquest conjunt de llibreries per programar les nostres aplicacions amb el conjunt de llibreries de ffmpeg.

ffmpeg funciona per intèrpret de comandes Linux i és juntament amb gstreamer un dels frameworks multimèdia més conegut i potent. Principalment es fa servir per treballar amb fluxos d'AV en format fitxer i permet transcodificar un gran nombre de còdecs estàndard i no estàndard.

A [12] és detallen el conjunt de còdecs i llibreries compatibles amb ffmpeg. Ffmpeg, com gstreamer, permet codificar en format H.264 AVC ja que inclou el codificador x264 desenvolupat pel grup VideoLAN del que parlarem a la secció 2.9.

Per introduir-ne l'ús, als annexos B.1 i C.1 es detallen casos pràctics d'aplicació amb ffmpeg. A C.1 generarem un vídeo 3D a partir de dos fluxos de vídeo. Amb aquests exemples volem demostrar la versatilitat de ffmpeg per treballar amb senyals multimèdia i codificar fluxos i fitxers de vídeo.

2.3.1. Conclusions

Dels casos d'aplicació dels annexos B.1 i C.1 hem comprovat que ffmpeg és una aplicació força útil per realitzar codificació i escalat de vídeo. A més a més a l'annex A.5 veiem que gstreamer integra totes les definicions d'espais de color de ffmpeg amb l'element ffmpegcolorspace. Tot i que és una eina útil està limitada en la part d'adquisició del senyal i d'enviament a xarxa, encara que inclou llibreries que ens permeten fer aquests tipus de tasca. En definitiva és una eina potent i el seu ús està força estès.

2.4. Gstreamer

Gstreamer és un framework multimedia per a Linux. Com a framework multimedia entenem el conjunt de llibreries i API's que ens permeten desenvolupar les nostres pròpies aplicacions i serveis multimedia. El manual complet en anglès el podem trobar al web de gstreamer [13].

2.4.1. Introducció a gstreamer

Gstreamer és un framework, o kit de desenvolupament, per aplicacions multimedia. Ha estat creat per poder treballar amb qualsevol tipus de font de dades d'àudio o vídeo, i pel seu disseny basat en mòduls i plugins es poden afegir nous còdecs d'AV o fins i tot desenvolupar-ne de nous.

Està dissenyat com un conducte genèric per on pot passar qualsevol tipus de flux de dades. Específicament gstreamer inclou les següents característiques:

- API's per aplicacions multimèdia.
- Arquitectura basada en plugins.
- Arquitectura basada en conductes per on passen fluxos de dades.
- Mecanismes per processar i interconnectar tipus multimèdia.
- Eines de desenvolupament i un conjunt de 150 plugins pre definits.

Els plugins que s'inclouen es poden diferenciar en:

- Per manipular protocols específics (MPEG-2 per exemple).
- Fonts o sources d'AV.
- Formats: contenidors, metadades i subtítols.
- Còdecs: Codificadors i descodificadors.
- Sinks d'AV o elements consumidors de fluxos.

A la Fig.2.3 veiem les diferents capes i l'estructura del nucli de gstreamer.
Gstreamer està format per diferents conjunts de llibreries:

- Nucli o core de gstreamer.
- Gst-plugins-base: un conjunt bàsic d'elements.
- Gst-plugins-good: un conjunt de plugins de qualitat amb llicència LGPL.
- Gst-plugins-ugly: conjunt de plugins de qualitat amb restriccions de distribució per termes de llicència.
- Gst-plugins-bad: conjunt de plugins que necessiten millores.
- Gst-python: plugins per fer servir amb aplicacions desenvolupades en Python.

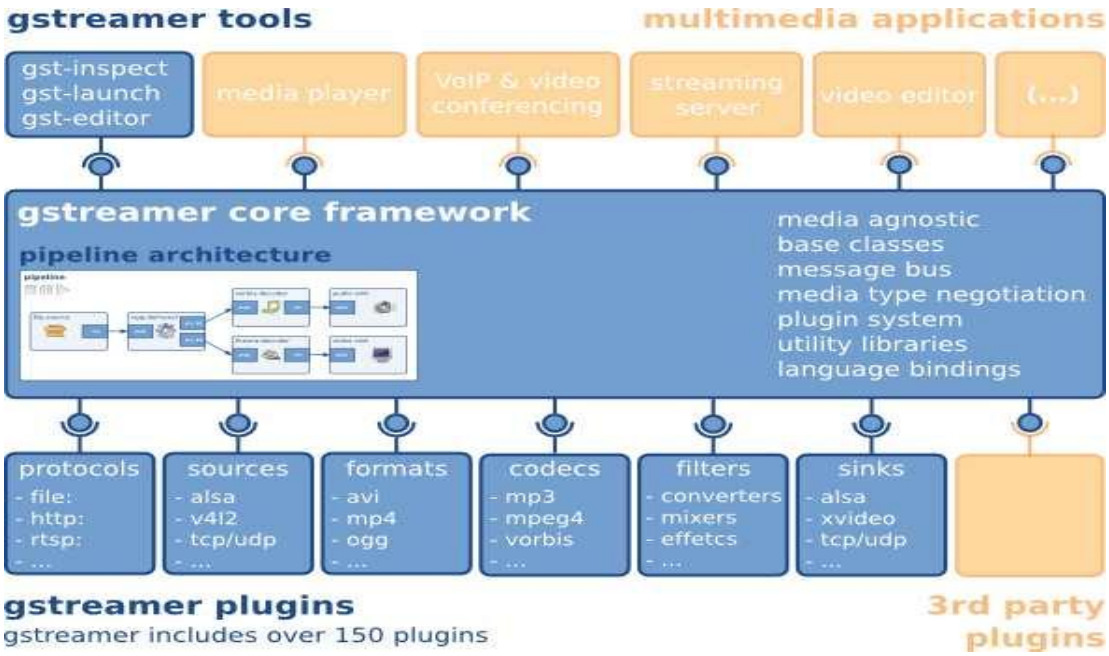


Fig 2.3 Capes framework gstreamer

2.4.2. Conceptes bàsics

Per començar a treballar amb gstreamer cal tenir clars alguns conceptes relacionats amb el disseny de la seva arquitectura i com es comuniquen entre ells els diferents elements que compondran la pipeline.

Elements

Són els tipus d'objectes més importants de gstreamer. Normalment una pipeline serà una cadena o conjunt d'elements per on passarà el flux de dades. Cada element ha de tenir una funció específica, llegir un arxiu de vídeo, codificar o reproduir per exemple. Es poden programar nous elements.

Pads

Són elements d'entrada o sortida als que es poden connectar altres elements. S'utilitzen per adaptar els enllaços del flux de dades entre diferents elements que treballen amb diferents tipus de dades. Podríem fer una analogia d'un pad com si fos un connector o port de comunicació.

Bins i pipelines

Són contenidors d'elements. La pipeline és un tipus concret de bin que permet l'execució en cadena de tots els elements que la componen. Un exemple de pipeline, per a un reproductor d'arxius OGG es mostra a la Fig. 2.4.

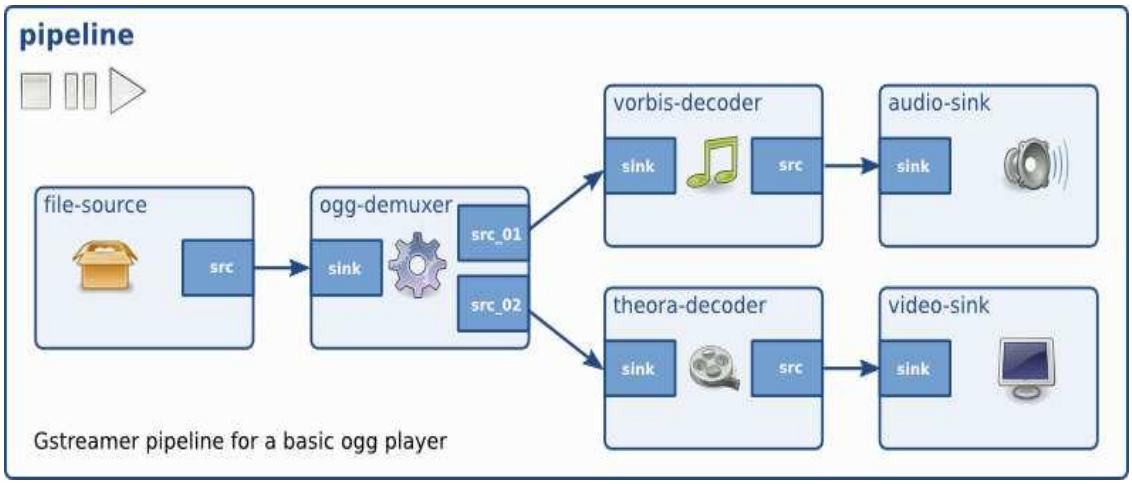


Fig 2.4 Exemple de pipeline reproductor OGG

A la Fig. 2.4 veiem el primer exemple de pipeline amb gstreamer. Com es descriu a l'annex A configurarem els diferents codificadors dissenyant pipelines o conductes. En aquesta pipeline la font d'àudio i vídeo és un fitxer multiplexat en un contenidor OGG. El desmultiplexor separa els fluxos d'AV que descodificàrem abans de reproduir amb elements consumidors (una pantalla i uns altaveus en aquest cas).

A banda de la pipeline, existeixen altres elements de l'aplicació que permeten rebre events per saber en tot moment que està passant al conducte que hem generat. Els elements que comuniquen la pipeline amb les altres capes són els següents:

- Buffers: Són objectes per passar dades entre elements de la pipeline. Els buffers tenen una única direcció pel seu flux de dades, van de la font a l'element consumidor (a aquest sentit únic de les dades s'anomena downstream). Emmagatzemen temporalment els fluxos de dades i serveixen de comunicació unidireccional entre elements.
- Events: Són objectes que s'envien entre elements o de l'aplicació als elements. Els events poden fluir upstream o downstream.
- Missatges i peticions.

A la Fig.2.5 detallem les diferents capes de l'aplicació i com es comuniquen els diferents elements. A la sortida (consola) quan executem una ordre amb gstreamer comencem a rebre missatges de com s'està generant la pipeline i si les capacitats de cada element permeten inter connectar-los amb èxit.

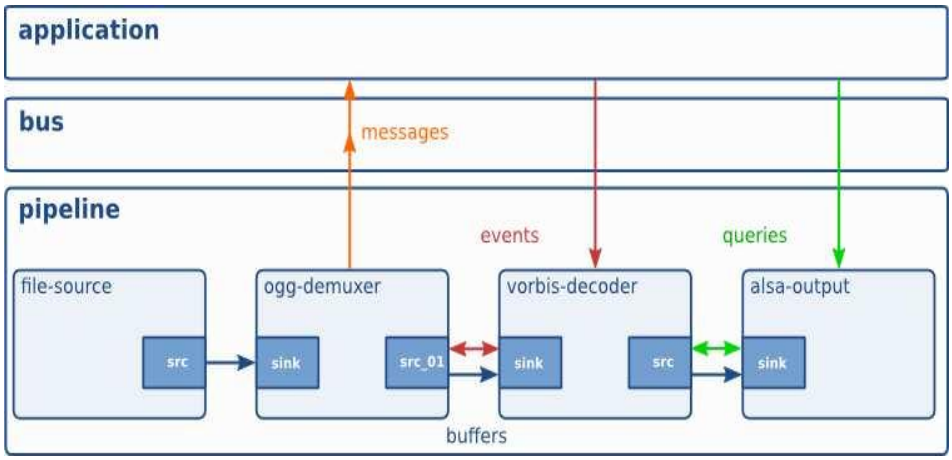


Fig 2.5 Comunicació entre elements i capes.

A l'annex A descrivim un conjunt de casos d'aplicació de gstreamer per introduir el seu ús més bàsic i entendre com configurarem el codificador als casos d'estudi del capítol 3.

2.4.3. Conclusions

Un cop introduït gstreamer i havent aprés el seu funcionament a la guia i els casos d'aplicació del l'annex A descobrim l'enorme potencial d'aquest framework multimèdia. El principal punt fort és el disseny modular i el seu plantejament de la pipeline com a paradigma de programació. Un cop entès el seu funcionament per a generar diferents tipus de pipelines que treballin amb aquests tipus d'estructures de dades, còdecs i protocols, hem de buscar les peces que ens permetin fer la tasca que vulguem.

La principal tasca és familiaritzar-se amb els elements amb els que treballarem per tal d'encadenar-los i generar els fluxos que ens interessi. La dificultat la trobem a l'hora de cercar els diferents elements i estudiar les capacitats de cadascun per saber si els podem inter connectar. A més cal entendre com s'han de fer les crides de cada element a la pipeline per encadenar-los de forma satisfactòria.

2.5. Project-x

Project-X és una aplicació alliberada sota llicència GNU General Public License. Està programada en Java, i per tant és compatible amb qualsevol sistema operatiu on hi instal·lem la màquina virtual.

És bàsicament un des multiplexor de fluxos TS. Té certes limitacions com la de no treballar amb resolucions HD, així doncs només podrem desmultiplexar captures de senyal SD. La podem descarregar des del gestor d'aplicacions de Linux o al seu web [14].

Te una interfície visual encara que també la podem invocar des de l'intèrpret de comandes. A l'annex B.1 podem veure un exemple d'ús de l'aplicació Project-x.

```
$ sudo projectx
```

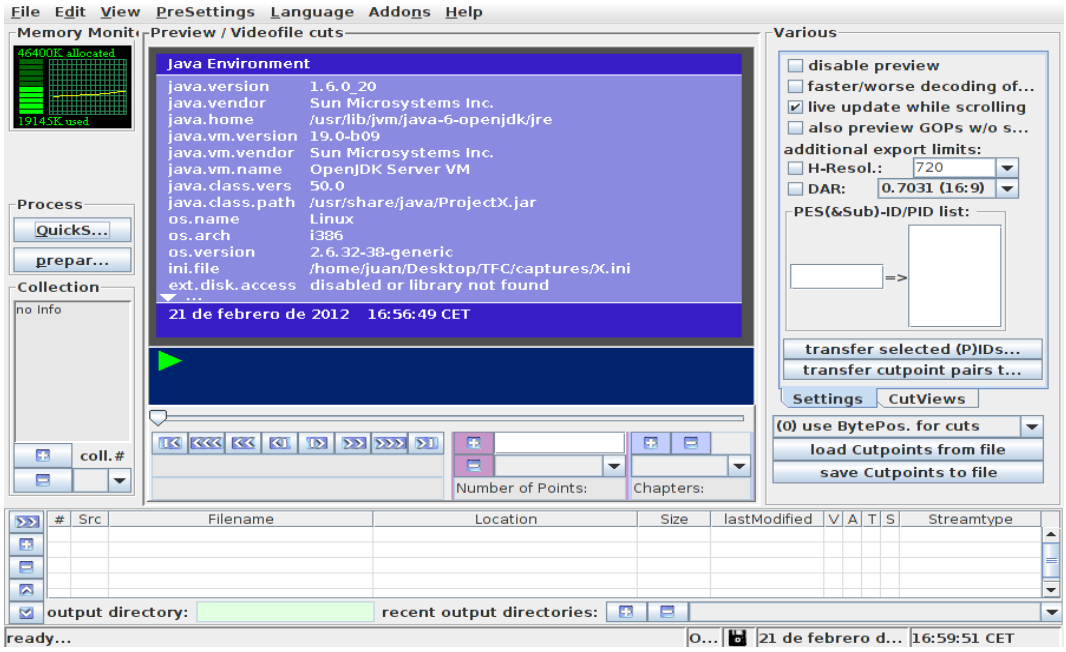


Fig. 2.18 Interface Project-X

2.5.1. Conclusions

Per complir l'objectiu d'aquest TFC ens cal una eina capaç de des multiplexar fluxos TS, gratuïta i de codi obert. El principal problema de project-x és que només ens permet treballar amb fluxos en format fitxer i és només un desmultiplexor de fluxos TS. El que volem es comparar project-x amb gstreamer per demostrar la potència d'aquest últim per treballar amb fluxos en temps real. Només amb una senzilla crida d'un dels elements de gstreamer fem tot el que és capaç de fer project-x.

2.6. Avidemux

Avidemux és un codificador de fluxos i multiplexor que ens permet treballar amb fitxers multimèdia. És un SW gratuït i de codi obert. Funciona en sistemes operatius Linux, Windows i Mac. Es pot descarregar des del gestor de descarregues de Linux o de [15].

A l'annex B es detalla un exemple d'aplicació amb Avidemux per multiplexar fluxos d'àudio i vídeo en un únic fitxer contenidor mp4.

L'objectiu d'aquest estudi de l'aplicació avidemux és adonar-se de la potència de gstreamer en termes de codificació i manipulació de fluxos i fitxers

multimèdia. Per poder extreure una captura de la tarja DVB-T i recodificar-la han calgut diverses aplicacions, com hem vist a l'annex B. Amb gstreamer podem integrar-ho tot en la mateixa pipeline en temps real com veurem als exemples d'aplicació del capítol 3.

2.6.1. Conclusions

Avidemux és una eina que ens permet trans codificar i multiplexar fluxos multimèdia en fitxers. A més permet fer tasques d'edició de vídeo i el seu entorn visual és intuïtiu i fàcil de fer servir. El principal problema d'avidemux és que permet treballar principalment amb fitxers i no seria possible fer-lo servir en un sistema en temps real. Un cop més gstreamer es configura com l'opció més vàlida entre les eines multimèdia amb les que hem treballat.

2.7. MuMuDVB

MuMuDVB (Multi Multicast DVB) és un distribuïdor de fluxos DVB a xarxes IP. Captura tots els programes i canals associats a una freqüència portadora i per a cada canal li assigna una adreça IP multicast.

Es pot descarregar del gestor d'aplicacions de Linux o de [16]. Un cop instal·lat executem:

```
$ mumudvb
mumudvb is a program who can redistribute stream from DVB on a network, in
multicast or in http unicast.
 It's main feature is to take a whole transponder and put each channel on a
different multicast IP.

Usage: mumudvb [options]
-c, --config : Config file
-s, --signal : Display signal power
-t, --traffic : Display channels traffic
-d, --debug  : Don't deamonize
-v           : More verbose
-q           : Less verbose
-h, --help   : Help

mumudvb Version 1.6
Based on dvbstream 0.6 by (C) Dave Chapman 2001-2004
Released under the GPL.
```

A l'annex B es detalla l'ús de mumudvb i tenim un exemple d'aplicació per capturar tots els fluxos que viatgen associats en TS a una mateixa freqüència portadora. És una aplicació força útil per enviar a xarxa local diferents fluxos del senyal de DVB-T de forma ràpida i senzilla.

2.7.1. Conclusions

MuMuDVB és una aplicació interessant des del punt de vista de la docència i per a usuaris que vulguin distribuir senyals de TV en una xarxa local. Analitzant els diferents fluxos que se'ns generen a xarxa podem visualitzar amb VLC les

senyals que viatgen multiplexades. A més és una opció a gstreamer per a capturar fluxos de DVB-T.

2.8. Wowza

Wowza és un servidor d'streaming desenvolupat en Java. Funciona sota qualsevol sistema operatiu i es pot descarregar de [17]. Un servidor d'streaming ens permet servir els diferents fluxos que rebem a tots els clients que s'hi connectin.

A l'annex D es detalla una guia d'instal·lació i configuració prèvia. Wowza no és una solució de codi obert, tot i que hi ha versions de desenvolupament gratuïtes. Hi han diferents tipus de llicencies depenent de l'ús que en fem. Entre les seves prestacions en destaquem les següents:

- Directes: Wowza és capaç de re transmetre fluxos d'àudio i vídeo encapsulats en TS i codificats en H.264 AVC ,en el cas del vídeo, i AAC en el cas de l'àudio. A més és compatible amb fluxos adaptatius HLS.

- Video sota demanda: Wowza és capaç de servir fluxos d'AV de fitxers que resideixin al propi servidor.

- Gravació: Wowza permet emmagatzemar els fluxos que s'emeten de forma que siguin accessibles com a vídeo sota demanda.

És una solució completament escalable, podem configurar diferents servidors simultàniament en el cas de que necessitem servir fluxos a una audiència elevada. Als annexos D desenvolupem un conjunt d'aplicacions d'exemple que podem integrar amb els diferents casos d'estudi del capítol 3.

2.8.1. Conclusions

Als annexos D hem descrit com configurar el servidor d'streaming. Hem triat wowza per ser un servidor d'streaming escalable per servir fluxos SD, HD i 3D. Aquesta aplicació correrà a la màquina Linux que hem contractat pel desenvolupament d'aquest TFC.

Wowza ens permet treballar amb diferents tipus de codificadors i podem desenvolupar aplicacions per a rebre més d'un directe. També podem configurar serveis de vídeo sota demanda i facilita el desenvolupament de webs multimèdia. A la banda del reproductor, és compatible amb tot tipus de dispositius i ens facilitarà la tasca per arribar al major nombre d'usuaris en aplicacions que així ho requereixin.

Una de les limitacions de Wowza és que només ens permet servir fluxos codificats en AVC i AAC. Es pot configurar per a servir fluxos codificats amb altres còdecs però ens cal fer servir una aplicació externa de pagament. Amb gstreamer no hi ha cap problema perquè podem configurar el codificador a la nostre elecció i fer servir aquests còdecs, estalviant-nos haver de fer servir més aplicacions de pagament.

2.9. VLC Media Player

VLC media Player és un reproductor de codi obert i gratuït descarregable de [35]. És compatible amb els SO's Linux, Windows i Mac. És un dels reproductors multimèdia més coneguts precisament per ser gratuït, multi plataforma i compatible amb la majoria de còdecs estàndards i no estàndards.

És un dels projectes del grup VideoLAN que aposta pel desenvolupament d'aplicacions multimèdia de codi obert. Entre els seus projectes en destaquem el reproductor VLC media Player i el codificador de fluxos de vídeo x264. Un cop descarregat i instal·lat la Interface de l'aplicació es que el mostra a la Fig. 2.19.

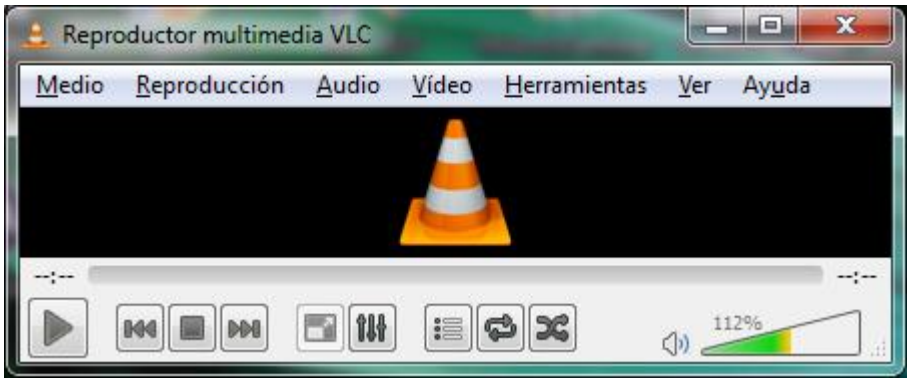


Fig. 2.19 Interface aplicació VLC media Player

Cal destacar la utilització de VLC durant bona part de la realització d'aquest TFC. Ha estat el reproductor que hem fet servir per verificar que els streams generats són reproduïbles i compleixen els diferents estàndards a nivell de codificació i de contenidors multimèdia. Principalment ens permet reproduir qualsevol tipus de fluxos multimèdia i extreure'n informació com el tipus de còdec, la resolució i frames per segon. És un reproductor potent ja que ens permet reproduir fitxers, dispositius HW com càmeres web i totes aquelles fonts de vídeo o àudio que generin fluxos audiovisuals.

2.10. Bino 3D Player

Bino 3D és un reproductor de fluxos estereoscòpics per a plataformes Linux, Mac i Windows. És de codi obert i gratuït i es tracta del primer reproductor 3D d'aquestes característiques:

- Basat en les llibreries de ffmpeg.
- Permet integració de subtítols.
- Integra openAL, un conjunt de API's per a reproducció d'àudio 3D.

El podem instal·lar des del gestor d'aplicacions de Linux o de [18]. Un cop instal·lat la Interface de l'aplicació és el que es mostra a la Fig. 2.20.

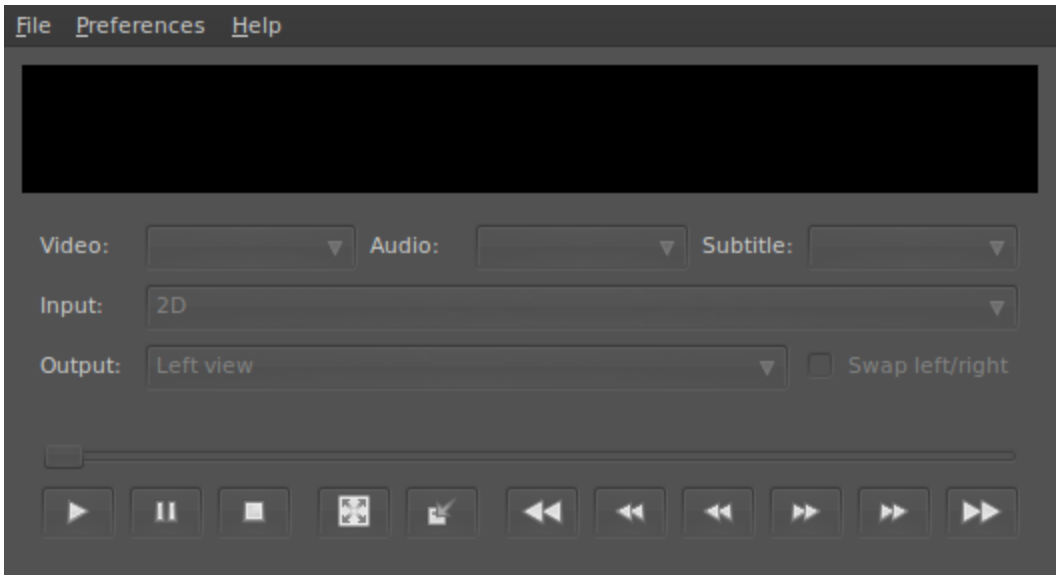


Fig 2.20 Interface Bino 3D

Dels tipus de vídeo 3D definits a la secció 1.8, Bino 3D és compatible amb:

- *Separate left-right*
- *Top-bottom*
- *Side by side*

En quant al mode de reproducció ens interessen:

- *HDMI-frame-pack*, que farem servir per a TV 3D compatible amb ulleres actives.
- *Mode anaglífic*, per a ulleres amb components per a colors.
- *OpenGL Quad Buffer Stereo,* compatible amb l'aplicació OpenGL que ens permet treballar amb tarjes gràfiques 3D Quad-Buffer sota Linux. Al punt 3.5 descriurem l'estudi dels diferents HW 3D disponibles per a la reproducció amb Linux.

2.10.1. Conclusions

Bino 3D és el primer reproductor de codi obert que ens permet visualitzar vídeos amb sensació de profunditat gratuïtament. Els equips 3D tot comencen a introduir-se al mercat. És en aquest moment quan sorgeixen oportunitats de negoci en aquests àmbit. Encara que és una aplicació completa, té algunes limitacions com la incompatibilitat amb fluxos codificats en MVC. En qualsevol cas és una eina molt útil que ens permet reproduir fluxos 3D amb èxit.

CAPÍTOL 3. CASOS D'ESTUDI - SERVEIS D'STREAMING

3. Introducció

En aquest tercer capítol aplicarem els coneixements adquirits als anteriors per descriure un conjunt de casos pràctics de sistemes d'streaming. Configurarem diferents tipus de codificadors i senyals, aplicacions del servidor d'streaming i reproductors. Concretament configurarem els següents sistemes:

- Codificador gstreamer font v4l2 → wowza → reproductors. En aquest cas configurarem el codificador de gstreamer per a fer captures de la font de vídeo i àudio de la web cam integrada d'un PC. Es codificarà àudio i vídeo a AAC i AVC respectivament i s'enviaran els fluxos al servidor d'streaming. Al costat de reproducció tindrem clients PC i per a dispositius mòbils.

- Codificador gstreamer font DVB-T → wowza → reproductors. En aquest cas configurarem el codificador de gstreamer per a fer captures del senyal de TDT. Codificarem a AAC i AVC i s'enviaran els fluxos al servidor d'streaming. Reproduirem amb PC i per a dispositius mòbils.

- Codificador HD HW → wowza → reproductors. En aquest cas configurarem un codificador HW HD i l'integrarem al nostre sistema d'streaming. L'objectiu és fer una comparativa entre solucions HW i solucions de codi obert.

- Codificador 3D → wowza → reproductor. En aquest cas llegirem una font de vídeo 3D de fitxer i l'integrarem al nostre sistema d'streaming. L'objectiu és reproduir un flux 3D amb el reproductor Bino 3D.

3.1. Equips del sistema

Per estudiar els diferents escenaris, descriurem el conjunt d'equips que farem servir. L'escenari és el mateix que es descriu a la figura 1.1 i els equips involucrats són els següents. Les fulles d'especificacions dels equips es troben a l'annex E.

- PC1: Sistema operatiu Linux Opensuse 11.4 Intel core 2 U3500 a 1.4 GHz. És l'equip on tenim instal·lat gstreamer i farà de codificador de sistema a tots els casos tret del cas del codificador HW. Integra una web cam i un micròfon que farem servir com a font d'àudio i vídeo del sistema.

- Servidor: Sistema operatiu Linux Ubuntu server 10.04.03 LTS 1 GB RAM Dual Core a 4.8 GHz. Hem configurat un servidor a Internet contractant un cloud VPS amb accés root. A D.3 es detalla l'elecció

d'aquest tipus de servidor. És on corre l'aplicació wowza i serà el servidor d'streaming del sistema.

- Codificador HW Makito de Haivision DVI. Codificador HW que configurarem a la secció 3.3 per a transmissions en HD i serveis de baixa latència.

- Càmera HD700 de szreach. Càmera DVI-I i SDI-HD amb resolucions màximes fins a 1080p a 30 fps.

- Tarja sintonitzadora DVB-T Avermedia Volar TV Black.

- Telèfon HTC Wildfire Sistema operatiu Android Froyo amb CPU a 500 MHz

- Iphone 4 d'Apple.

- Smart TV Samsung 3D i ulleres actives.

- Kit 3D Nvidia: Monitor 120 Hz, ulleres actives, tarja gràfica Nvidia i emissor de ràdio. A [48] es detallen les característiques i especificacions d'aquest sistema reproductor 3D.

3.2. Sistema codificador V4l2

En aquest cas d'estudi volem obtenir fluxos dels senyals d'AV de la webcam, font v4l2 integrada al nostre PC i el seu micro.

L'objectiu és generar un stream compatible amb el nostre servidor d'streaming wowza (flux vídeo codificat a H.264 i àudio AAC encapsulat en TS). L'escenari de proves és el descrit a la Fig.1.1. Volem configurar el codificador com es mostra al diagrama de blocs de la Fig.3.1.

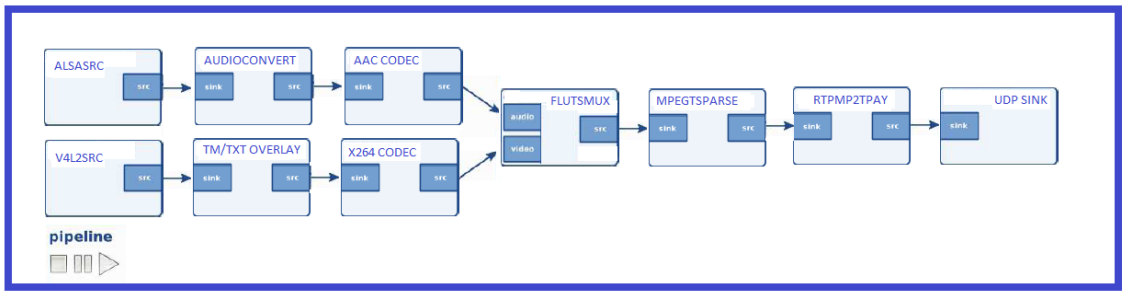


Fig. 3.1 Pipeline del codificador v4l2

Llavors queda definit el servei de la següent manera:

- Codificador: Aplicació de gstreamer que captura el vídeo (font vídeo per a Linux o v4l2) i àudio de la tarja de só del PC i el codifica complint els

estàndards de compressió H.264 per a vídeo i AAC per àudio. L'stream viatjarà encapsulat en un flux MPEG-2 TS.

- Servidor d'streaming: Wowza, configurat per rebre un flux encapsulat en TS.

- Reproductors: Reproductor Java incrustat en un document HTML / VLC Media Player RTSP.

3.2.1. Configuració del codificador

A l'annex A.4.1 expliquem con configurar el codificador de gstreamer per aquest cas d'aplicació. Com veiem a la Fig.3.1 capturem de la webcam i del micròfon. Cal configurar correctament els codificadors perquè no es perdin mostres en el procés i tant àudio com vídeo estiguin sincronitzats. A més s'ajusten els paràmetres per configurar el codificador de forma que reduïm la latència total del sistema. En paral·lel codifiquem tots dos fluxos amb els còdecs seleccionats, l'encapsulem en un contenidor TS i l'enviem a xarxa.

3.2.2. Configuració del servidor d'streaming

A l'annex D documentem la configuració del servidor d'streaming Wowza. Hem de crear una nova aplicació al servidor i configurar els reproductors per visualitzar els streams. Configurem el servidor per entregar els fluxos fent servir l'aplicació de directe de l'annex D.2. A més a l'annex D.3 detallem la selecció del servidor d'streaming que hem contractat.

3.2.3. Reproductor

En aquest apartat descriurem els reproductors HTML i VLC per RTSP del sistema. En el cas de VLC per RTSP analitzarem el que està passant a xarxa per verificar el procés del protocol RTSP.

3.2.3.1. Reproductor HTML

Al costat del reproductor (clients) hem generat un document HTML amb un reproductor embegut. Els elements que composen el reproductor web són un fitxer HTML i un arxiu javascript que descarreguem del directori d'instal·lació de Wowza, `/usr/local/WowzaMediaServer3.0.3/examples/SimpleVideoStreaming/client` obrim el document amb el navegador i configurem la recepció de l'stream. A la Fig. 3.2 veiem la reproducció dels flux amb VLC. Al logs del servidor se'ns indica si s'hi connecta un client:

```
root@xtreaming:/usr/local/WowzaMediaServer-3.0.3/bin# INFO session connect-
pending [IPCLIENT] -
INFO session connect [IPCLIENT] -
INFO stream create - -
INFO stream play mpegts.stream -
```

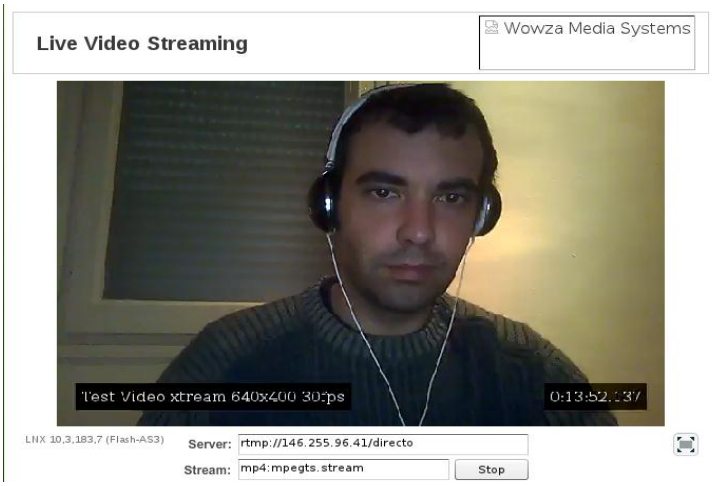


Fig 3.2 Reproducció client flash HTTP

Estem rebent l'streaming en directe amb una latència aproximada de 5 segons. Verifiquem el sincronisme entre l'àudio i vídeo del flux en reproducció.

3.2.3.2. Reproductor VLC

Un altre tipus de reproductor que volem provar és VLC per RTSP. En aquest cas ja hem configurat el servidor wowza per que treballi amb aquests tipus de clients a l'arxiu de configuració `Application.xml`. Per obrir l'stream introduïm la URI [8]. Wowza fa servir el port 1935 per RTSP. Veurem una imatge com la Fig. 3.3.

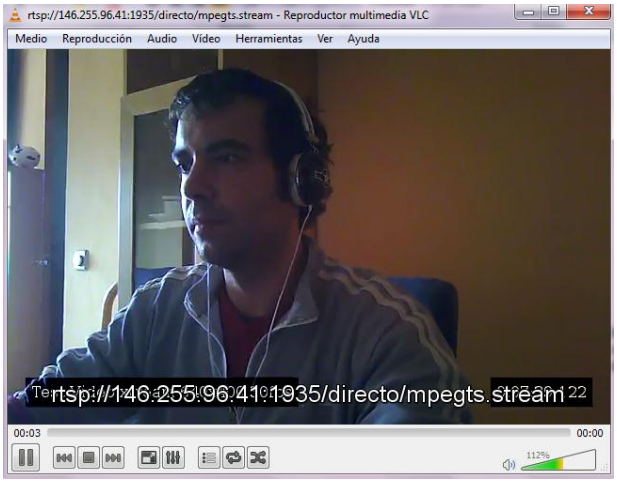


Fig 3.3 Reproducció RTSP amb VLC

Al costat del servidor, comprovem els logs. S'han negociat els ports de transmissió de la següent manera:

```
INFO stream play mpegts.stream –
```

```
INFO rtsp play 487015969 –
INFO server comment - UDPTransport.firstPacket: bind:0.0.0.0/0.0.0.0:6973
msg:/ipclient:59459
INFO server comment - UDPTransport.firstPacket: bind:0.0.0.0/0.0.0.0:6971
msg:/ipclient:59461
INFO server comment - RTPUDPTransport.unbind[directo/_definst_]:
0.0.0.0/0.0.0.0:6972 sent:179 recv:0
INFO server comment - RTPUDPTransport.unbind[directo/_definst_]:
0.0.0.0/0.0.0.0:6973 sent:4 recv:2
INFO server comment - RTPUDPTransport.unbind[directo/_definst_]:
0.0.0.0/0.0.0.0:6970 sent:763 recv:0
INFO server comment - RTPUDPTransport.unbind[directo/_definst_]:
0.0.0.0/0.0.0.0:6971 sent:4 recv:2
```

Verifiquem que els ports on s'envien els datagrames UDP estan dintre del rang 6970 – 9999 com vam veure a la secció 1.11.3.

3.3. Sistema codificador DVB-T

En aquest cas d'estudi aplicarem la teoria descrita al primers capítols i desenvoluparem una pipeline capaç de capturar un flux DVB-T i transcodificar-la a un stream H264 AAC compatible amb el nostre servidor d'streaming.

Durant el cas estudiarem la forma d'ajustar les resolucions per equips mòbils que tenen certes limitacions en quant a les característiques multimèdia. A l'annex A.2 i A.3 aprenem a configurar el codificador per a capturar fonts DVB-T i servir-la a clients PC i per a dispositius mòbils. El codificador i la pipeline de gstreamer es descriu al diagrama de blocs de la Fig.3.4.

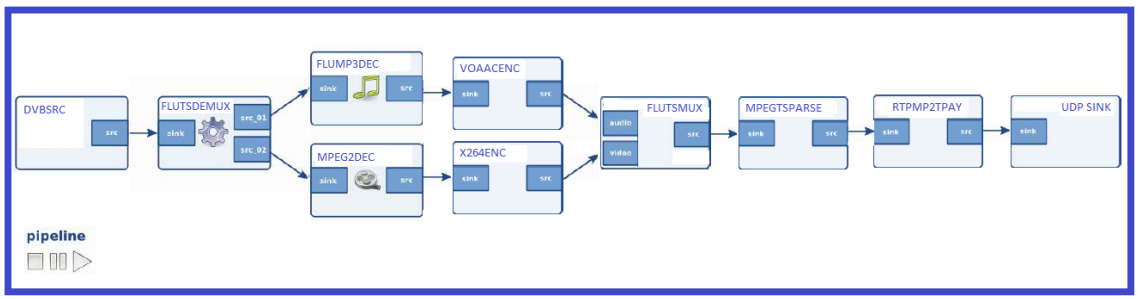


Fig. 3.4 Pipeline del codificador DVB-T

De la font de DVB-T hem de sintonitzar una freqüència portadora amb el programa d'AV que ens interessa. De tot l'espectre sintonitzem la freqüència a 685 MHz que inclou el programa 805 que és el canal d'esports de TV3 per les transmissions de DVB-T a Barcelona. Per saber la freqüència d'un canal concret podem escanejar l'espectre amb l'aplicació w_scan de dvb-apps i podrem saber quin programa en concret volem capturar.

Hem aconseguit l'objectiu de capturar i codificar el senyal de TV amb èxit i servir-lo a xarxa (com es tracta als annexos A.2 i A.4). Amb aquesta configuració podem desplegar xarxes de difusió de TV de forma gratuïta. Degut a l'elevat preu d'aquest tipus de dispositius per a vídeo professional, un altre cop gstreamer es caracteritza per la seva potència i versatilitat.

3.3.1. Configuració del codificador per a dispositius mòbils

A les seccions A.4.2.1 i A.4.2.2 es descriu com configurar el codificador de gstreamer per a dispositius mòbils. Haurem d'escalar i modificar els fps dels fluxos de vídeo per tal de fer-los compatibles amb dispositius mòbils. A la taula 3.1 es descriuen les característiques que han de complir els fluxos d'àudio i vídeo per a Android i Iphone.

VÍDEO CÒDEC	RESOLUCIÓ	FPS	BIR RATE VIDEO	ÀUDIO CÒDEC	CANALS	BIT RATE ÀUDIO
H.264BaselineProfile	174x144píxels	12	56 kbps	AAC LC	1	24 kbps
H.264BaselineProfile	480x360píxels	30	500 kbps	AAC LC	2	128 kbps
H.264BaselineProfile	1280x720píxels	30	2 Mbps	AAC LC	2	192 kbps
						ANDROID
VÍDEO CÒDEC	RESOLUCIÓ	FPS	BIR RATE VIDEO	ÀUDIO CÒDEC	CANALS	BIT RATE ÀUDIO
H.264BaselineProfile	640x480píxels	30	1.5 Mbps	AAC LC	2	160 kbps
H.264Baselin Profile	640x480píxels	30	2.5 Mbps	AAC LC	2	160 kbps
						IPHONE

Taula 3.1 Característiques audiovisuals per Iphone i Android

3.4. Sistema codificador HW

A banda de solucions SW com el framework gstreamer, hi han disponibles al mercat codificadors HW professionals. N'hi ha de molts tipus, però les principals prestacions que es busquen en aquests tipus de dispositius, són la baixa latència, la resolució màxima compatible de la font del vídeo i els fps.

Les aplicacions són moltes però en destaquem aplicacions de directes en HD, aplicacions militars i telemedicina. En aquest cas d'estudi hem aconseguit fer-nos amb un codificador de la marca Haivision, concretament el model Makito DVI HD. Les prestacions dels dispositius és poden consultar a la seva fulla d'especificacions a l'annex E. Es farà servir juntament amb una càmera HD professional, el model HD700 del fabricant xinés szreach. Aquesta càmera genera fluxos de vídeo de fins a 1080p a 30 fps. L'escenari del sistema es mostra al diagrama de blocs de la Fig 1.1.

3.4.1. Configuració del codificador HW

Als anteriors apartats hem configurat la part del servidor d'streaming i dels reproductors. Així doncs només cal configurar la solució HW per que serveixi l'stream al nostre servidor. Mitjançant el navegador web, podem accedir a la configuració de l'equip, la interfície es mostra a la Fig.3.5.

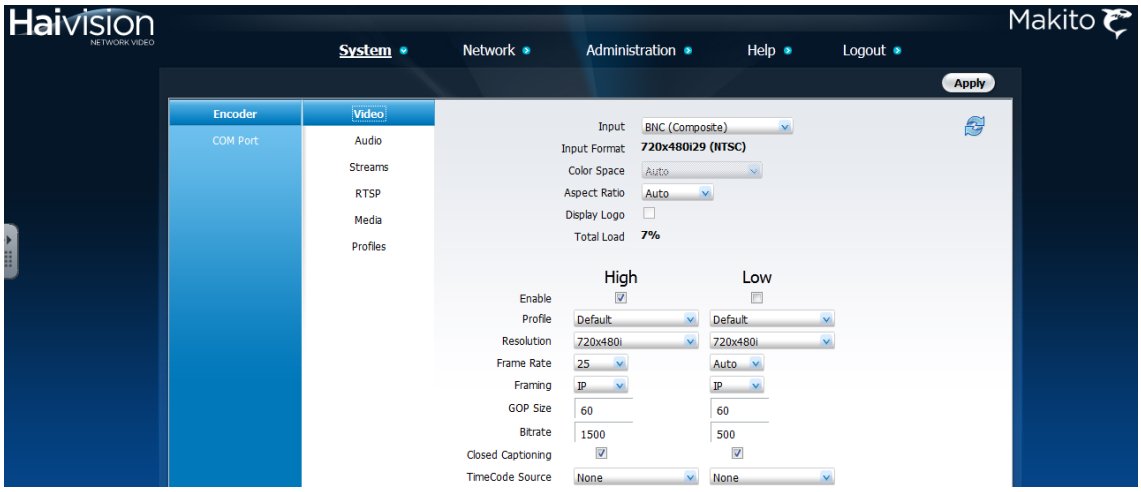


Fig 3.5 Interface codificador HW HD

Hem configurat l'equip per treballar amb resolució 720x480 píxels a 25 fps per tal de no saturar la xarxa on està el Makito servint fluxos de vídeo massa pesats. A la configuració del vídeo veiem paràmetres com el valor de GoP i el bit rate. Un cop configurat el vídeo, hem de configurar l'enviament a xarxa de l'stream. Seleccionem la pestanya stream i configurem els paràmetres com a la Fig.3.6.

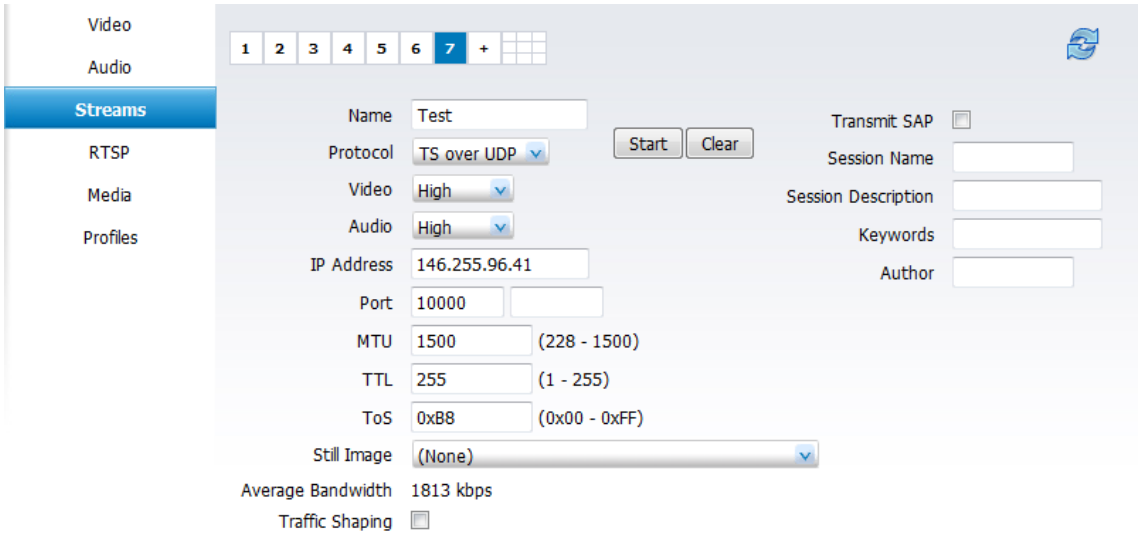


Fig 3.6 Configuració stream codificador HW

És important el valor TTL (Time To Live), fixa el límit de vida del paquets servits pel nombre de transicions entre routers a la xarxa. El fixem a valor màxim per garantir que els paquets arribin a destí. Amb aquests dos passos ja hem configurat el codificador HW per servir l'stream.

3.4.2. Configuració del servidor d'streaming

La configuració del servidor és la mateixa que es descriu a l'annex D pel servidor d'streaming. Rebrà un stream servit per RTP encapsulat en un flux TS pel port 10000. Activem la recepció de l'stream des de l'stream manager de Wowza i fem un cop d'ull als logs del servidor:

```
INFO server comment - RTPDePacketizerMPEGTS.handleRTPPacket: IMPORT: videoPID[prg:0x1,pid:0x21,filter:none]: streamType:H264:27
INFO server comment - RTPDePacketizerMPEGTS.handleRTPPacket: IMPORT: audioPID[prg:0x1,pid:0x24,filter:none]: streamType:AAC:15 audioLanguage:uknown
```

L'stream que rebem és exactament igual que el desenvolupat amb gstreamer, una captura de l'stream amb VLC a la Fig. 3.7.

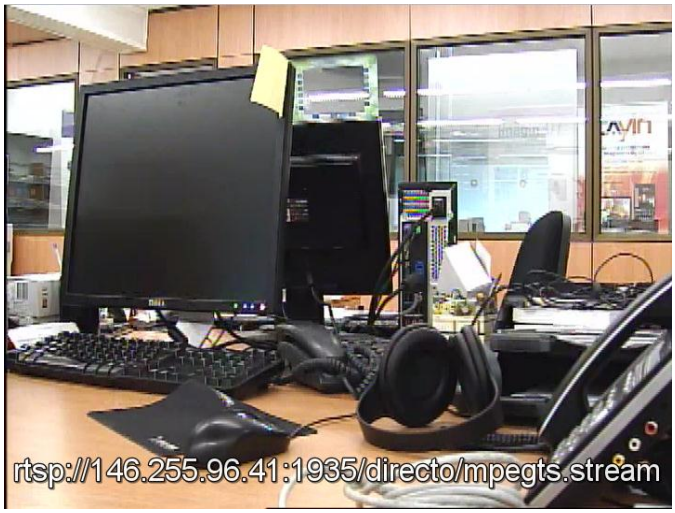


Fig 3.7 Captura amb VLC codificador HW

Amb aquest exemple volem demostrar la potència de gstreamer, ja que amb un desenvolupament simple d'una pipeline hem aconseguit un codificador molt semblant a una solució professional, sense tenir en compte aspectes força importants con la latència. Volem demostrar que gstreamer és una alternativa de lliure distribució molt potent. A les conclusions finals de la secció 4.3 detallarem més aspectes d'aquesta comparativa

3.5. Sistema 3D

En aquest cas d'estudi volem integrar un flux de vídeo 3D al sistema per tal de que el podem reproduir amb l'aplicació 3D Bino. L'escenari del sistema és el que es mostra al diagrama de blocs de la Fig.1.1.

3.5.1. Configuració 3D amb gstreamer

L'objectiu és configurar una pipeline com es mostra al diagrama de flux del codificador de la Fig. 3.8.

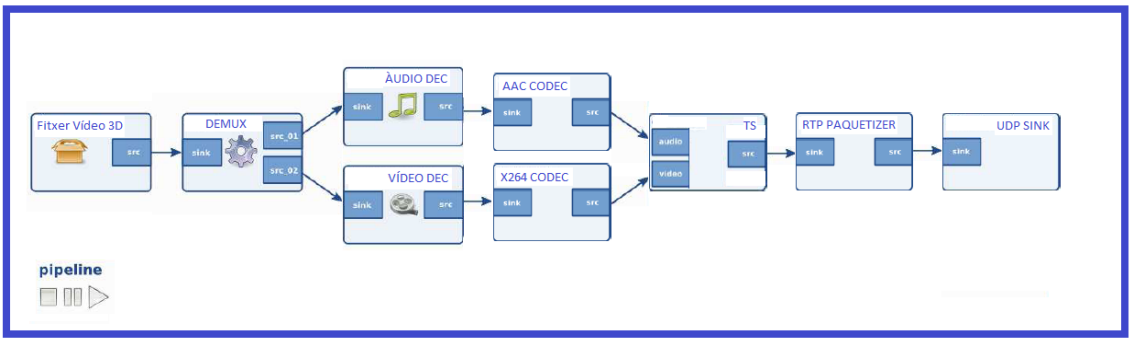


Fig 3.8 Pipeline codificador 3D

En aquest cas d'estudi tindrem com a font de vídeo un fitxer de vídeo 3D (del tipus side by side). Volem enviar-lo al servidor d'streaming per que pugui ser reproduïble per Bino 3D. Com a codificador o font de vídeo també farem servir gstreamer, en aquest cas llegim de font fitxer. A més a més la capacitat de la nostra CPU no ens permet treballar en temps real amb aquests tipus de senyals que requereixen de capacitats de processat força elevades.

Per altra banda gstreamer no integra actualment mòduls per a generació de vídeo 3D i no és capaç de treballar amb fluxos MV, descrits al punt 1.7.2. Ens fem amb una font de vídeo 3D (side by side a 854x240 píxels a 30 fps) i generem la pipeline amb gstreamer: A l'annex A.4.3 expliquem com configurar les pipelines de gstreamer per a vídeo 3D.

3.5.2. Configuració del servidor d'streaming

La configuració del servidor d'streaming es detalla a l'annex D. Un cop configurat activarem el sistema i farem una prova amb el reproductor VLC per comprovar que ho rebem correctament. Evidenment no tindrem sensació de 3D. Per tenir sensació de 3D ens cal un reproductor (SW bino 3D) i un dispositiu hardware (TV o monitor 3D i les ulleres). A l'annex E es detallen els equips 3D que podríem fer servir. A la Fig.3.8 tenim una captura del flux 3D. Verifiquem que rebem l'stream 3D correctament. Fem el mateix amb el reproductor Bino 3D per tal de comprovar si rebem correctament un stream reproduïble i amb sensació 3D. Abans hem de configurar un sistema HW reproductor 3D per a sistemes Linux.

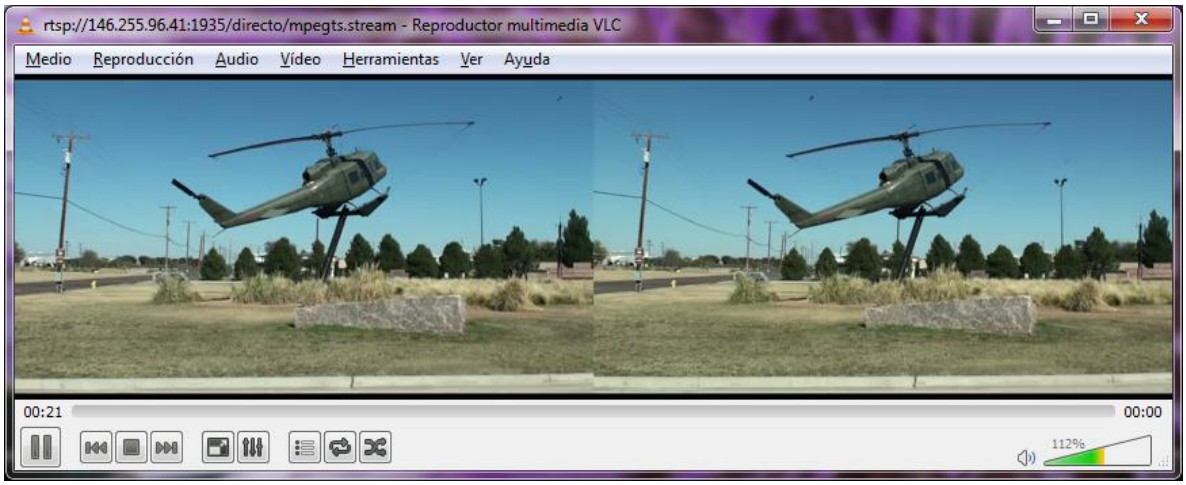


Fig 3.14 Reproducció VLC stream 3D

3.5.3. Configuració del reproductor 3D per a Linux

A banda de buscar una aplicació de codi obert per a Linux hem hagut d'estudiar les compatibilitats dels sistemes reproductors 3D amb aquest sistema operatiu. L'objectiu és aprofitar el kit de reproducció 3D (monitor 120 Hz, ulleres actives, sensor i tarja gràfica Nvidia) per a PC que tenim al laboratori. Les especificacions de l'equip es poden trobar a l'annex E.

A [19] trobem les especificacions de les targes que hi ha disponibles al mercat. Per a sistemes Linux ens cal una tarja gràfica compatible amb OpenGL. De la gama de targes disponibles són només compatibles per a Linux les Quad-Buffered. Com aquesta alternativa no és una opció econòmica i havíem de comprar una tarja nova vam decidir fer servir una de les TV del laboratori, SmartTV de Samsung que ja teníem disponible. Amb un ordinador Linux i fent servir la TV de pantalla seleccionant la sortida HDMI de Bino 3D hem verificat el seu funcionament.

3.5.4. Conclusions

Les possibles millores del sistema amb la integració de Bino 3D es descriuran al capítol 4. La difusió de fluxos 3D tot just comença a implantar-se amb nous dispositius compatibles com telèfons mòbils 3D i SmartTV. Actualment ja hi ha programari de codi obert que ens permet reproduir fluxos estereoscòpics. Tot i així, no tota la gama de targetes gràfiques són compatibles amb Linux i els sistemes de reproducció 3D actuals encara tenen preus força elevats.

CONCLUSIONS I LÍNIES FUTURES

4. Introducció

En aquest últim capítol volem fer una comparativa entre els diferents sistemes descrits al capítol anterior i solucions HW professionals. Ja hem comprovat que podem fer servir les aplicacions de codi obert en els diferents capítols d'aquest treball de forma eficient. Volem fer una comparativa econòmica entre els sistemes desplegats amb tecnologia de codi obert i solucions de mercat. Plantejarem els sistemes del capítol 3 amb solucions professionals i confeccionarem un pressupost per a cadascuna, comparant-les.

4.1. Solució mercat SD

Per confeccionar el sistema descrit a la Fig.1.1 amb solucions HW hem de buscar al mercat el següents equips:

- Càmera SD Resolució PAL/NTSC a 30 fps amb sortida de vídeo compost
- Codificador HW entrada de vídeo compost per a resolució SD
- Servidor d'streaming i llicència de wowza

A l'annex E es descriuen el conjunt d'equips HW que hem cercat. Amb aquestes dades volem comparar-ho amb una solució de codi obert:

- PC integra càmera web i codificador de vídeo amb gstreamer.
- Servidor d'streaming i llicència de wowza.

A la Taula 4.1 veiem una comparativa entre totes dues solucions. L'ús d'eines de codi obert poden suposar fins a un estalvi de més 1300 € si ja tenim PC

Cámera	Codificador SD	Servidor d'streaming
600 €	700 €	25 € /mes
Cámera	**Codificador**	**Servidor d'streaming**
0 €	0 €	25 € /mes

Taula 4.1 Comparativa solucions hardware i de codi obert

4.2. Solució mercat HD

Per confeccionar el sistema descrit a la Fig.1.1 amb solucions HW hem de buscar al mercat els següents equips:

- Càmera HD i codificador de vídeo

- Servidor d'streaming i llicència wowza

A l'annex E es descriuen el conjunt de solucions HW. Una alternativa amb solucions de codi obert s'implementaria amb els equips:

- PC integra codificador amb gstreamer
- Càmera web
- Servidor d'streaming i llicència wowza.

A la taula 4.2 de [36] , es descriu un pressupost de totes dues tecnologies i el seu preu aproximat. Fer servir solucions de codi obert pot suposar fins a un estalvi de més de 5000 €.

Cámera	Codificador HW	Servidor d'streaming
1.500 €	3000/5.000 €	25 € /mes
Cámera	**Codificador GST**	**Servidor d'streaming**
80 €	0 €	25 € /mes

Taula 4.2 Valoració económica HD

4.3. Conclusions

Un del avantatges que volem destacar amb aquest estudi és l'estalvi que suposa l'ús d'eines de codi obert davant altres solucions HW. Volem demostrar-ho presentant aquesta valoració econòmica. Durant la realització d'aquest treball hem contractat un servidor a Internet havent fet un estudi previ de l'oferta de servidors disponibles al mercat. A l'annex D.3 s'explica l'elecció del servidor.

L'objectiu és poder variar la capacitat del nostre servidor d'streaming en funció del nombre d'usuaris que s'hi connectin. D'aquesta manera podem fer una comparativa entre solucions HW que tenen limitació d'usuaris i cobren per augmentar-ne el nombre a alternatives escalables com wowza. A la taula 4.3 detallem els preus de solucions que integrarien diferents codificadors HW que coneixem i que són actualment al mercat. A l'annex E trobem les fulles d'especificacions dels equips.

Codificador	Servidor d'streaming	Nombre usuaris
500 €	25 € / mes + 5 € / dia	Escalable
		TOTAL OS
		560 €
Codificador	**Servidor d'streaming**	**Nombre usuaris**
Marca X. 2.000 €	Marca X 1.500 €	10
		TOTAL Solució X
		3.500 €
Codificador	**Servidor d'streaming**	**Nombre usuaris**
Marca Y. 3.000 €	25 € / mes + 5 € / dia	Escalable
		TOTAL Solució Y
		3.060 €
Codificador	**Servidor d'streaming**	**Nombre usuaris**
Marca Z 6.000 €	25 € / mes + 5 € / dia	Escalable
		TOTAL Solució Z
		5.060 €

Taula 4.3 Valoració económica solucions hardware

A banda de la cotització econòmica, creiem en la importància de la divulgació i l'ús d'eines de codi obert. Envers les tecnologies de pagament, la configuració d'eines de codi obert requereix tenir coneixement tècnics i experiència en operació amb sistemes Linux. Hem volgut facilitar l'aprenentatge d'aquestes eines de forma que puguin ser fetes servir per tot tipus d'usuaris. Creiem que les tecnologies de codi obert són una alternativa, degut a la seva filosofia de lliure distribució, ideal per a la docència.

Hem vist durant els anteriors capítols com configurar i desplegar sistemes d'streaming, partint d'una base teòrica i verificant en cada punt els resultats esperats. Esperem que la feina feta durant aquest TFC hagi motivat a estudiants i professionals d'aquest àmbit a animar-se a fer servir eines de codi obert, en virtut del seu propi benefici i dels altres.

4.4. Línies futures

Dels sistemes, aplicacions i protocols amb els que hem estat treballant n'hem pogut treure conclusions que ens indiquen el camí a seguir en línies d'investigació futures en aquest àmbit. Les millores que creiem que es poden implementar amb gstreamer i altres eines de codi obert per a aquests tipus de sistemes són les següents:

- Implementació d'un multiplexor estadístic de fluxos TS a la taxa de codificació dels senyals multiplexats. Com hem vist a la teoria de la secció 1.10.3 es poden implementar millores en aquest àmbit. S'hauria d'implementar millores a l'element `flutsmux` de gstreamer.

- Implementació de la millora del còdec AVC MVC al codificador de codi obert x264enc, referenciat a l'enllaç [33] per poder transmetre fluxos 3D HD sobre xarxes IP. S'haurien d'implementar millores als elements de gstreamer per poder alimentar-se amb fluxos estereoscòpics. Implementació de la millora de Bino 3D Player per a descodificar fluxos de vídeo MVC. Aquestes implementacions de MVC es podrien ajustar com a desenvolupament d'un TFC.

- Implementar un codificador compatible amb DASH per a generació de fluxos adaptatius. Una de les necessitats que sorgeixen és poder codificar la mateixa font en diversos streams a diferents resolucions i frames per segon per tal d'arribar a qualsevol tipus de dispositiu. A més una xarxa IP, pateix de variacions en l'ample de banda efectiu entre el servidor d'streaming i cada client. En aquest cas només anomenarem i introduirem algunes de les tècniques que permeten servir múltiples streams al clients adaptant-se als diferents tipus de dispositius i les variacions d'ample de banda disponible en cada cas. El grup MPEG ha desenvolupat un estàndard que solucioni els problemes que hem descrit. Aquest estàndard es diu DASH, [50], i ha estat publicat recentment. No entrarem a definir la tecnologia DASH en aquest TFC però degut a la seva importància en pròximes línies d'investigació volíem anomenar-la per a futurs casos d'estudi. La integració amb el protocol DASH s'està implementat per gstreamer en la seva última versió 0.11, [23].

GLOSSARI DE TERMES

AVC Advanced Video Codec
APSK Amplitude Phase Shift Keying
AAC Advanced Audio Codec
AVI Audio Video Interleave
ATSC Advanced Television System Committee
CDN Content Data Network
CAT Condicional Acces Table
CBR Constant Bit Rate
DASH Dynamic Adaptative Streaming over HTTP
DNS Domain Name Server
DVB Digital Video Broadcasting
FLAC Free Looseless Audio Codec
HTML Hiper Text Meta Language
HLS HTTP Live Streaming
HW Hardware
HD High Definition
IGMP Internet Group Management Protocol
ISDB Integrated Services Digital Broadcasting
IP Internet Protocol
LAN Local Area Network
MVC Multi View Coding
MPEG Motion Pictures Expert Group
OFDM Orthogonal Frequency Division Multiplexing
PS Program Stream
PSI Program Specific Information
PID Program Identifier
PAT Program Association Table
PMT Program Map Table
QPSK Quadrature Phase Shift Keying
QAM Quadrature Amplitude Modulation
RGB Red Green Blue
RTP Real Time Protocol
RTSP Real Time Streaming Protocol
SD Standard Definition
SW Software
STB Set Top Boxes
TS Transport Stream
TCP Transport Control Protocol
URI Uniform Resource Identifier
URL Uniform Resource Locator
UDP User Datagram Protocol
VBR Variable Bit Rate
VPS Virtual Private Server
WMS Windows Media Server

BIBLIOGRAFIA

[1] Submostreig de croma
http://en.wikipedia.org/wiki/Chroma_subsampling

[2] Còdecs sense pèrdues
http://wiki.hydrogenaudio.org/index.php?title=Lossless_comparisón

[3] Vídeo entrellaçat
http://es.wikipedia.org/wiki/Exploraci%C3%B3n_entrelazada

[4] Reproducció 3D autoestereoscòpia
http://es.wikipedia.org/wiki/Autoestereoscopia

[5] Haivision
http://www.haivision.com

[6] Comparació formats contenidors
http://en.wikipedia.org/wiki/Comparisón_of_container_formats

[7] Televisió de Catalunya
http://tv3.cat

[9] Jitter
http://es.wikipedia.org/wiki/Jitter

[10] Introducció a v4l2
http://linuxtv.org/wiki/index.php/LinuxTV

[11] Linux TV
http://www.linuxtv.org

[12] ffmpeg
http://ffmpeg.org/

[13] Documentació gstreamer
http://gstreamer.freedesktop.org/documentation/

[14] Aplicació project-x
http://project-x.sourceforge.net/

[15] Aplicació avidemux
http://fixounet.free.fr/avidemux/

[16] MuMuDVB
http://mumudvb.braice.net/mumudrupal/

[17] Servidor d'streaming wowza
http://www.wowza.com

[18] Bino 3D Player
http://www.nongnu.org/bino/download.html

[19] Nvidia
www.nvidia.com

[20] http://gstreamer.freedesktop.org/documentation/

[21] Llibreries gstreamer RTSP
http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-rtsp-server/html/ch01.html

[22] rtsp://146.255.96.41:1935/directo/mpegts.stream/playlist.m3u8

[23] Plugins de gstreamer
http://gstreamer.freedesktop.org/documentation/plugins.html

[24] Mostres de vídeo 3D
http://blog.planar3d.com/2009/07/3dstereoscopic-video-samples.html

[25] Java Virtual Machine
http://www.java.com/es/download/

[26] Codi DASH per a gstreamer
https://gitorious.org/ylatuya-gstreamer/gst-plugins-bad/commits/hlswip

[27] http://146.255.96.41:8086/streammanager

[28] Aplicacions de Wowza
http://www.wowzamedia.com/downloads/tutorials/vod/Application.xml

[29] Tipus d'stream Wowza
http://www.wowzamedia.com/forums/content.php?217#streamTypes

[30] Avermedia
http://www.avermedia.eu/avertv/SP/Upload/ProductImages/AVerTV%20Volar%20Black%20HD.pdf

[31] SmartTV Samsung
http://www.samsung.com/us/video/tvs/UN55D8000YFXZA-specs#

[32] Reach
http://www.szreach.com

[33] VideoLan x264
http://www.videolan.org/developers/x264.html

[34] dvbsnoop
http://dvbsnoop.sourceforge.net/

[35] VLC Media Player
http://www.videolan.org/vlc/

[36] Webcams USB HD
http://www.logitech.com/en-us/webcam-communications/webcams/devices/hd-pro-webcam-c920

[37] Notícia tancament MegaUpload
http://www.ara.cat/xarxes/claus-del-tancament-Megaupload_0_631136983.html

[38] Youtube
http://www.youtube.com

[39] Espais de color
http://en.wikipedia.org/wiki/Color_space

[40] Mobotix i MxPEG
http://developer.mobotix.com/docs/why.html

[41] Vídeo entrellaçat i resolucions
http://es.wikipedia.org/wiki/1080i

[42] Protocol IP
http://en.wikipedia.org/wiki/Internet_Protocol

[43] Protocol RTSP
http://es.wikipedia.org/wiki/Real_Time_Streaming_Protocol

[44] Protocol IGMP
http://en.wikipedia.org/wiki/Internet_Group_Management_Protocol

[45] Ports RTSP/UDP
http://soundscreen.com/streaming/firewall.html

[46] Exemples syde by side
http://www.best-3dtvs.com/what-is-side-by-side-3d/

[47] Solé, O., “Serveis de control i consum de recursos multimèdia”. TFC 2007.
http://upcommons.upc.edu/pfc/handle/2099.1/4215

[48] Nvidia 3D Vision
http://www.nvidia.es/object/3d-vision-technology-es.html

[49] Mack, S, *Streaming Media Bible*, Ed. Hungry Minds, Nova York 2002.

[50] Tecnologia DASH
http://en.wikipedia.org/wiki/Dynamic_Adaptive_Streaming_over_HTTP

ANNEX A. OPERACIÓ AMB GSTREAMER

En aquest annex volem incloure tots els desenvolupaments i operacions fetes amb el framework multimèdia gstreamer. En primer lloc introduirem l'ús més elemental de gstreamer, explicarem el procés d'instal·lació i desenvoluparem una primera aplicació a l'annex A.1. A l'annex A.2 introduirem gstreamer per a tècniques més avançades d'edició i manipulació de vídeo i operació de fonts de vídeo DVB-T. A l'annex A.3 desenvoluparem un servidor RTSP per a gstreamer i estudiarem el tràfic de xarxa amb wireshark. A l'annex A.4 desenvoluparem les pipelines pels codificadors de vídeo dels casos d'estudi. L'annex A.5 detalla tots els elements de gstreamer que hem fet servir durant la realització d'aquest treball.

A.1. Primeres passes amb gstreamer

L'objectiu d'aquest annex es introduir al lector a fer les seves primeres passes amb el framework gstreamer. Explicarem el procés d'instal·lació i escriurem el codi de la nostra primera aplicació en llenguatge C.

A.1.1. Instal·lació de gstreamer:

Des del gestor d'aplicacions de Linux necessitem instal·lar els diferents paquets i els plugins que hem d'utilitzar, que inclouran els elements de gstreamer corresponents. Depenent del tipus de dades multimedia amb el que hem de treballar necessitarem instal·lar uns plugins en concret.

Per treballar amb DVB hem d'instal·lar el mòdul GStreamer Bad Plugins, instal·larem també els mòduls Gstreamer Core, Base, i Good plugins que també ens caldrà tenir instal·lats. A l'annex A.5 detallarem el conjunt d'elements que hem fet servir i descriurem a quin conjunt de paquets (base, good, bad o ugly pertanyen). A banda del nucli i els plugins ens calen les llibreries libgstreamer0.10-dev, per poder desenvolupar aplicacions.

Un cop instal·lades, hem de verificar al directori:

/usr/lib/pkgconfig

```
/usr/lib/pkgconfig$ ls -l

[...]
-rw-r--r-- 1 root root 332 feb 19  2011 gst-python-0.10.pc
-rw-r--r-- 1 root root 474 feb 22 20:02 gstreamer-0.10.pc
-rw-r--r-- 1 root root 430 feb 22 20:32 gstreamer-app-0.10.pc
-rw-r--r-- 1 root root 430 feb 22 20:32 gstreamer-audio-0.10.pc
-rw-r--r-- 1 root root 377 feb 22 20:02 gstreamer-base-0.10.pc
-rw-r--r-- 1 root root 305 mar  2 11:03 gstreamer-basevideo-0.10.pc
-rw-r--r-- 1 root root 399 feb 22 20:32 gstreamer-cdda-0.10.pc
-rw-r--r-- 1 root root 402 feb 22 20:02 gstreamer-check-0.10.pc
-rw-r--r-- 1 root root 297 mar  2 11:03 gstreamer-codecparsers-0.10.pc
```

```
-rw-r--r-- 1 root root 394 feb 22 20:02 gstreamer-controller-0.10.pc
-rw-r--r-- 1 root root 385 feb 22 20:02 gstreamer-dataprotocol-0.10.pc
-rw-r--r-- 1 root root 363 feb 22 20:32 gstreamer-fft-0.10.pc
-rw-r--r-- 1 root root 361 feb 22 20:32 gstreamer-floatcast-0.10.pc
-rw-r--r-- 1 root root 343 feb 19  2011 gstreamer-gl-0.10.pc
-rw-r--r-- 1 root root 388 feb 22 20:32 gstreamer-interfaces-0.10.pc
-rw-r--r-- 1 root root 394 feb 22 20:02 gstreamer-net-0.10.pc
-rw-r--r-- 1 root root 405 feb 22 20:32 gstreamer-netbuffer-0.10.pc
-rw-r--r-- 1 root root 377 feb 22 20:32 gstreamer-pbutils-0.10.pc
-rw-r--r-- 1 root root 314 mar  2 11:03 gstreamer-plugins-bad-0.10.pc
-rw-r--r-- 1 root root 391 feb 22 20:32 gstreamer-plugins-base-0.10.pc
-rw-r--r-- 1 root root 364 feb 22 20:32 gstreamer-riff-0.10.pc
-rw-r--r-- 1 root root 398 feb 22 20:32 gstreamer-rtp-0.10.pc
-rw-r--r-- 1 root root 400 feb 22 20:32 gstreamer-rtsp-0.10.pc
-rw-r--r-- 1 root root 355 feb 22 20:32 gstreamer-sdp-0.10.pc
-rw-r--r-- 1 root root 378 feb 22 20:32 gstreamer-tag-0.10.pc
-rw-r--r-- 1 root root 404 feb 22 20:32 gstreamer-video-0.10.pc
-rw-r--r-- 1 root root 272 feb 19  2011 gst-rtsp-server-0.10.pc
[...]
```

Que hi són les referències a les llibreries de gstreamer. Sense aquests llibreries no podrem compilar i fer servir les aplicacions de gstreamer. A més a més al directori:

```
/usr/lib/gstreamer-0.10$ ls -l
total 21480
-rwxr-xr-x 1 root root    5680 feb 22 20:02 gst-plugin-scanner
-rwxr-xr-x 1 root root   14216 feb 19  2011 libfsfunnel.so
-rwxr-xr-x 1 root root   55544 feb 19  2011 libfsmsnconference.so
-rwxr-xr-x 1 root root    9996 feb 19  2011 libfsrtcpfilter.so
-rwxr-xr-x 1 root root  191404 feb 19  2011 libfsrtpconference.so
-rw-r--r-- 1 root root     968 feb 23 20:08 libfsselector.la
-rwxr-xr-x 1 root root   30856 feb 19  2011 libfsselector.so
-rwxr-xr-x 1 root root    9996 feb 19  2011 libfsvideoanyrate.so
-rwxr-xr-x 1 root root  150084 feb 19  2011 libgnl.so
[...]
```

A aquest directori trobem les referències de cada element de les llibreries de gstreamer. A [20] es detallen el conjunt d'elements que trobem a cada conjunt de llibreries i podem saber on es troba un element en concret. Un cop instal·lades podem fer servir l'aplicació gst-inspect-0.10 per fer cerques entre les llibreries instal·lades. Si volem cercar tots els elements que contenen les sigles dvb, executem:

```
$ gst-inspect-0.10 | grep dvb
dvb:  dvbsrc: DVB Source
dvb:  dvbbasebin: DVB bin
dvbsuboverlay:  dvbsuboverlay: DVB Subtitles Overlay
ffmpeg:  ffdec_dvbsub: FFmpeg DVB subtitles decoder
ffmpeg:  ffenc_dvbsub: FFmpeg DVB subtitles encoder
```

I si necessitem saber les capacitats de l'element dvbsrc:

```
$ gst-inspect-0.10 dvbsrc
Factory Details:
 Long name:    DVB Source
 Class:    Source/Video
```

```
 Description:    Digital Video Broadcast Source
 Author(s):    P2P-VCR, C-Lab, University of Paderborn,Zaheer Abbas Merali
<zaheerabbas at merali dot org>
 Rank:        none (0)

Plugin Details:
 Name:            dvb
 Description:        DVB elements
 Filename:        /usr/lib/gstreamer-0.10/libgstdvb.so
 Version:        0.10.23
 License:        LGPL
 Source module:    gst-plugins-bad
 Source release date:    2012-02-20
 Binary package:    GStreamer Bad Plug-ins source release
 Origin URL:        Unknown package origin

GObject
+----GstObject
      +----GstElement
            +----GstBaseSrc
                  +----GstPushSrc
                        +----GstDvbSrc
```

Se'ns descriu la versió de l'element, qui i quan es va realitzar. També es detallen les capacitats de l'element `dvbsrc` que detallarem a l'annex A.2.

A.1.2. Primera aplicació en C amb gstreamer:

Per verificar que la instal·lació s'ha fet correctament farem la nostra primera aplicació amb gstreamer. Volem desenvolupar un executable que ens digui la versió de gstreamer que estem fent servir. El codi és el següent:

```
#include <stdio.h>
#include <gst/gst.h>
int main (int argc, char *argv[])
{
      const gchar *nano_str;
      guint major, minor, micro, nano;
      gst_init (&argc, &argv);
      gst_version (&major, &minor, &micro, &nano);
      if (nano == 1)
            nano_str = "(CVS)";
      else if (nano == 2)
            nano_str = "(Prerelease)";
      else
            nano_str = "";
      printf ("Aquest programa esta linkat amb gstreamer %d.%d.%d
      %s", major, minor, micro, nano_str);
      return 0;
}
```

Per poder fer servir les llibreries de Gstreamer, sempre les haurem d'inicialitzar amb la funció `gst_init`, i fer una crida a aquesta funció des del `main`. També és aquesta funció la que s'encarrega de gestionar les crides a les funcions necessàries quan fem servir crides directes a gstreamer des de la consola.

Fixem-nos que hem d'incloure la llibreria `<gst/gst.h>` a l'aplicació, que inclou les definicions dels objectes i funcions de gstreamer.

Compilem i executem la nostra primera aplicació:

```
$ gcc -Wall $(pkg-config --cflags --libs gstreamer-0.10) ginicializa.c -o
ginicializa
$ ls
ginicializa  ginicializa.c
$ ginicializa
Aquest programa esta linkat amb gstreamer 0.10.31
```

Hem d'incloure les flags `$(pkg-config --cflags --libs gstreamer-0.10)` per dir-li al compilador que ha d'anar a buscar les llibreries de gstreamer.

A.1.3. Conclusions

En aquest annex hem aprés a compilar aplicacions per a gstreamer des de l'intèrpret de comandes. Hem aprés a cercar elements a les llibreries i a verificar si estan ben instal·lades al nostre sistema.

A.2. Introducció a l'edició de vídeo amb gstreamer

En aquest annex començarem a treballar amb el model de programació de gstreamer. Volem generar les nostres primeres pipelines i introduir la metodologia a seguir per desenvolupar-les. L'objectiu serà desenvolupar un conjunt de proves que ens permetin operar amb fluxos d'àudio i vídeo i començarem a treballar amb la font DVB.

A.2.1 Primeres passes amb gstreamer

Primer hem d'instal·lar el paquet d'aplicacions `gstreamer-tools` (des del gestor d'aplicacions de Linux o en mode consola). També hem de verificar que la nostra tarja de vídeo estigui ben configurada. Per fer-ho, executem:

```
$ xvinfo
X-Video Extension version 2.2
screen #0
 no adaptors present
```

Si no tenim els adaptadors de la tarja correctament instal·lats no funcionarà. Els instal·lem i executem un altre cop:

```
$ xvinfo
X-Video Extension version 2.2
screen #0
  Adaptor #0: "ATI Radeon AVIVO Video"
    number of ports: 4
    port base: 131
    operations supported: PutImage
    supported visuals:
      depth 24, visualID 0x23
```

```
      depth 24, visualID 0x24
      depth 24, visualID 0x25
 [...]
    number of attributes: 10
      "XV_SET_DEFAULTS" (range 0 to 1)
              client settable attribute
 [...]
    maximum XvImage size: 4096 x 4096
    Number of image formats: 4
      id: 0x32315659 (YV12)
        guid: 59563132-0000-0010-8000-00aa00389b71
        bits per pixel: 12
        number of planes: 3
        type: YUV (planar)
      id: 0x30323449 (I420)
        guid: 49343230-0000-0010-8000-00aa00389b71
        bits per pixel: 12
        number of planes: 3
        type: YUV (planar)
      id: 0x32595559 (YUY2)
        guid: 59555932-0000-0010-8000-00aa00389b71
        bits per pixel: 16
        number of planes: 1
        type: YUV (packed)
      id: 0x59565955 (UYVY)
        guid: 55595659-0000-0010-8000-00aa00389b71
        bits per pixel: 16
        number of planes: 1
        type: YUV (packed)
```

Comprovem els diferents espais de color compatibles (YUV) amb la nostra targeta gràfica. Ja estem llestos per poder operar a consola i fer crides al generador de pipelines i els elements que ens generaran les fonts multimèdia. Executem l'exemple més bàsic de pipeline amb gstreamer des de consola:

```
$ gst-launch-0.10 videotestsrc ! xvimagesink
Estableciendo el conducto a PAUSA …
El conducto está PREPARÁNDOSE …
El conducto está PREPARADO …
Estableciendo el conducto a REPRODUCIENDO …
New clock: GstSystemClock
```

Amb aquest exemple, generem una nova pipeline amb la crida a `gst-launch-0.10` . Generem una font de vídeo de prova amb `videotestsrc` i redirigim el flux amb el símbol `!` per reproduir-lo amb l'element `xvimagesink`. És molt útil fer un diagrama del flux multimèdia per treballar amb gstreamer. En aquest primer cas és força clar ja que només són dos elements com es mostra a la figura A.2.1:

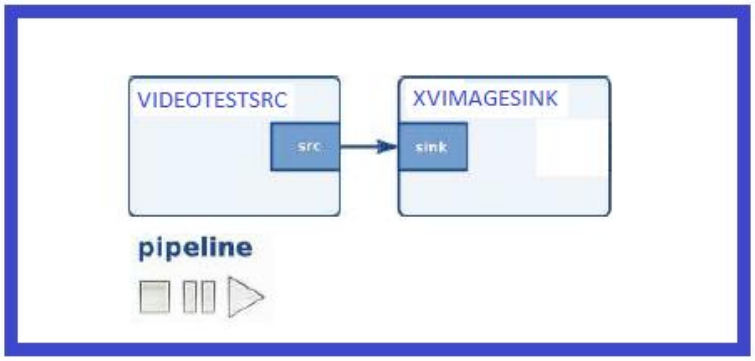


Fig. A.2.1 Exemple pipeline elemental gstreamer.

I veurem una finestra com a la figura A.2.2.

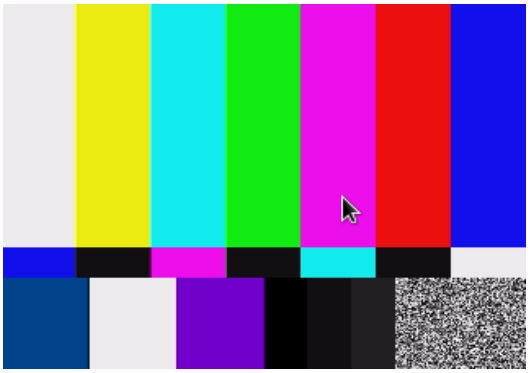


Fig A.2.2 Sortida prova gstreamer

Un cop la tanquem a la sortida de la shell:

```
ERROR: del elemento
/GstPipeline:pipeline0/GstXvImageSink:xvimagesink0:
Output window was closed
Información adicional de depuración:
xvimagesink.c(1299):            gst_xvimagesink_handle_xevents            ():
/GstPipeline:pipeline0/GstXvImageSink:xvimagesink0
Execution ended after 34625024144 ns.
Estableciendo el conducto a PAUSA …
Estableciando el conducto a PREPARADO …
Estableciendo el conducto a NULL …
Liberando la tubería…
```

Hem generat com a font un vídeo de prova del tipus `videotestsrc` i per reproduir-lo un objecte del tipus `xvimagesink`. Amb gstreamer es fan servir analogies amb conductes, tot objecte que genera un tipus de dades és una font i un altre objecte, el que consumeix, aquest tipus de dades en concret.

En els propers exemples de pipelines, veurem el funcionament dels móduls de gstreamer que ens permetran fer composicions de video amb diferents fonts simultaneament.

A.2.2. PiP (Picture in Picture) amb gstreamer

En aquest cas, volem fer proves amb diferents fonts de vídeo per visualitzar una font de video amb una resolució inferior a sobre d'una altra amb resolució superior. Veiem un diagrama de flux de la pipeline a la Fig. A.2.3.

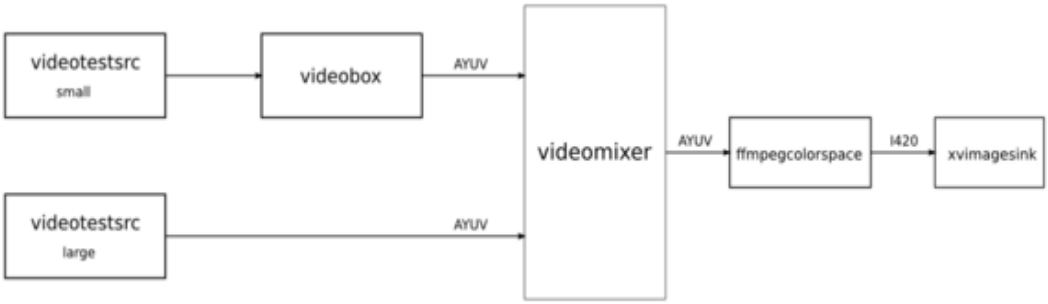


Fig A.2.3 Edició de vídeo amb gstreamer exemple PiP

Del diagrama, tenim dos fonts de vídeo de prova, escollim les resolucions CIF 352 x 288 píxels per la petita i PAL 720 x 576 píxels per la més gran. Definirem un freqüència de 25 fps.

A la pipeline haurem de definir dos fonts de vídeo del tipus `videotestsrc`, li haurem d'indicar la resolució i els fps de cada font. Ens caldrà un element mesclador que s'encarregarà de la composició del vídeo, abans de reproduir-lo amb el sink `xvimagesink`:

Fonts de vídeo:

```
videotestsrc pattern=”snow” ! vídeo/xraw-yuv, framerate=25/1, width=352,
height=288 !
videotestsrc ! video/xraw-yuv, framerate=25, width=720, height=576 !
```

Abans de fer la composició total, hem de redirigir la sortida de les dues entrades a un element `videomixer` que definirem com `mesclador`.

A la sortida del `mesclador` amb l'element `ffmpegcolorspace` ens assegurem que les capacitats entre l'element `videomixer` i `xvimagesink` siguin iguals. Si no ho són `ffmpegcolorspace` farà la transformació d'un espai de color a un altre. Reproduïm la sortida amb `xvimagesink`. Executem la pipeline amb `gst-launch-0.10`:

```
$ gst-launch-0.10 videotestsrc pattern="snow" ! video/x-raw-yuv,
framerate=25/1, width=352, height=288 ! videomixer name=mesclador !
ffmpegcolorspace ! xvimagesink videotestsrc ! video/x-raw-yuv,
framerate=25/1, width=720, height=576 ! mesclador.
```

Fixem-nos que hem definit una senyal d'entrada que simula soroll amb el `pattern=snow` per la resolució CIF. Hem indicat l'element `videomixer` amb el nom `mesclador`. És molt important que ens fixem com es redirigeix més d'una entrada a l'element videomixer, per fer-ho hem d'indicar el nom de l'element a qui volem

redirigir el flux amb un punt. A l'exemple anterior amb `mesclador` indiquem que la font de vídeo PAL és una entrada més de l'element `mesclador`.

Un cop executada hem de veure una imatge com la de la Fig A.2.4.

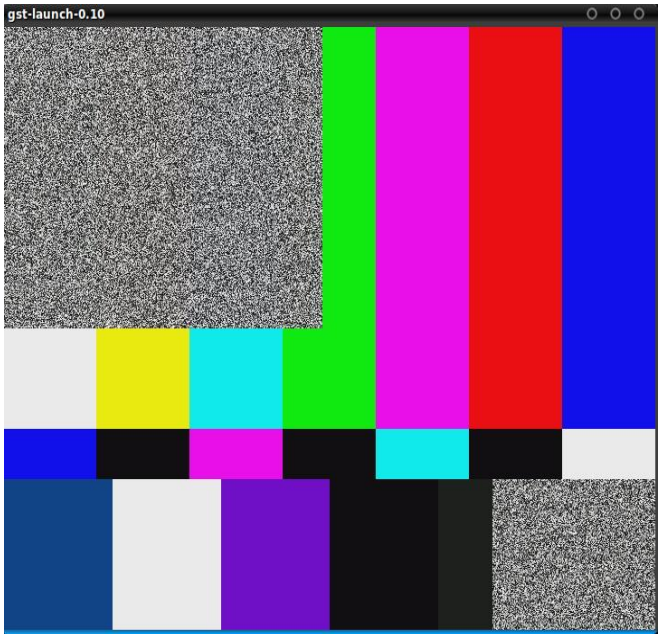


Fig A.2.4 Sortida vídeo gstreamer PiP

A la vora superior esquerra es mostra el flux a resolució CIF a sobre del de resolució PAL. Si ens cal modificar la posició a la composició podem fer servir un dels elements de gstreamer que ens ho permet, en aquest cas l'element `videobox`.

L'element `videobox` és el perímetre i la posició de la finestra de visualització, sempre podem obtenir més informació dels elements de gstreamer fent servir l'aplicació `gst-inspect`, per exemple :

```
$ gst-inspect-0.10 videobox
Plugin Details:
  Name:                   videobox
  Description:            resizes a video by adding borders or cropping
  Filename:         /usr/lib/gstreamer-0.10/libgstvideobox.so
  Version:          0.10.21
  License:          LGPL
  Source module:    gst-plugins-good
  Binary package:   GStreamer Good Plugins (Ubuntu)
  Origin URL:             https://launchpad.net/distros/ubuntu/+source/gst-
plugins-good0.10
GObject
 +----GstObject
       +----GstElement
             +----GstBaseTransform
                   +----GstVideoBox
```

```
Pad Templates:
  SINK template: 'sink'
    Availability: Always
    Capabilities:
      video/x-raw-yuv
                 format: AYUV
                  width: [ 1, 2147483647 ]
                 height: [ 1, 2147483647 ]
              framerate: [ 0/1, 2147483647/1 ]
      video/x-raw-yuv
                 format: I420
                  width: [ 1, 2147483647 ]
                 height: [ 1, 2147483647 ]
              framerate: [ 0/1, 2147483647/1 ]
  SRC template: 'src'
    Availability: Always
    Capabilities:
      video/x-raw-yuv
                 format: AYUV
                  width: [ 1, 2147483647 ]
                 height: [ 1, 2147483647 ]
              framerate: [ 0/1, 2147483647/1 ]
      video/x-raw-yuv
                 format: I420
                  width: [ 1, 2147483647 ]
                 height: [ 1, 2147483647 ]
              framerate: [ 0/1, 2147483647/1 ]
```

ens detalla la capacitat de l'aplicació en quant als formats de vídeo compatibles, resolucions i el tipus d'element entre d'altres. Com modifiquem la pipeline si volem situar la font CIF 50 píxels més a baix i 25 píxels a la dreta i sense vores? La sortida es veu a la Fig. A.2.5.

```
$ gst-launch-0.10 videotestsrc pattern="snow" ! video/x-raw-yuv, framerate=25/1, width=352, height=288 ! videobox border-alpha=0 top=-50 left=-25 ! videomixer name=mesclador ! ffmpegcolorspace ! xvimagesink videotestsrc ! video/x-raw-yuv, framerate=25/1, width=720, height=576 ! mesclador.
```

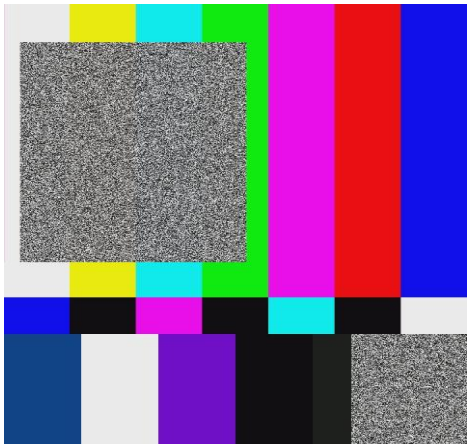


Fig A.2.5 Sortida PiP amb `xvimagesink`

Hem buscat les variables que fan aquestes modificacions de la sortida de `gst-inspect`:

```
videobox border-alpha=0 top=-50 left=-25
```

I si volem afegir transparència a la font amb resolució CIF:

```
$ gst-launch-0.10 videotestsrc pattern="snow" ! video/x-raw-yuv, framerate=25/1, width=352, height=288 ! videobox border-alpha=0 alpha=0.6 top=-50 left=-25 ! videomixer name=mesclador ! ffmpegcolorspace ! xvimagesink videotestsrc ! video/x-raw-yuv, framerate=25/1, width=720, height=576 ! mesclador.
```

A.2.3 Videowall amb gstreamer

Un videowall és una composició de diferents fluxos de vídeo mostrats a simultàniament. Continuem amb un altre exemple, volem configurar un videowall amb una imatge de fons, per exemple una retransmissió d'un esdeveniment a Mart amb 3 streams de vídeo a resolució 350x250 píxels. Per fer-ho farem servir l'element que obre un fitxer d'imatge con un flux de vídeo, `multifilesrc`. A cada stream s'ha indicar el número de càmera amb un missatge de text CAMERA_X. També s'ha de mostrar el text "Directe des de Mart" al videowall. Ens caldrà l'element `textoverlay`.

El diagrama de flux de la pipeline seria:

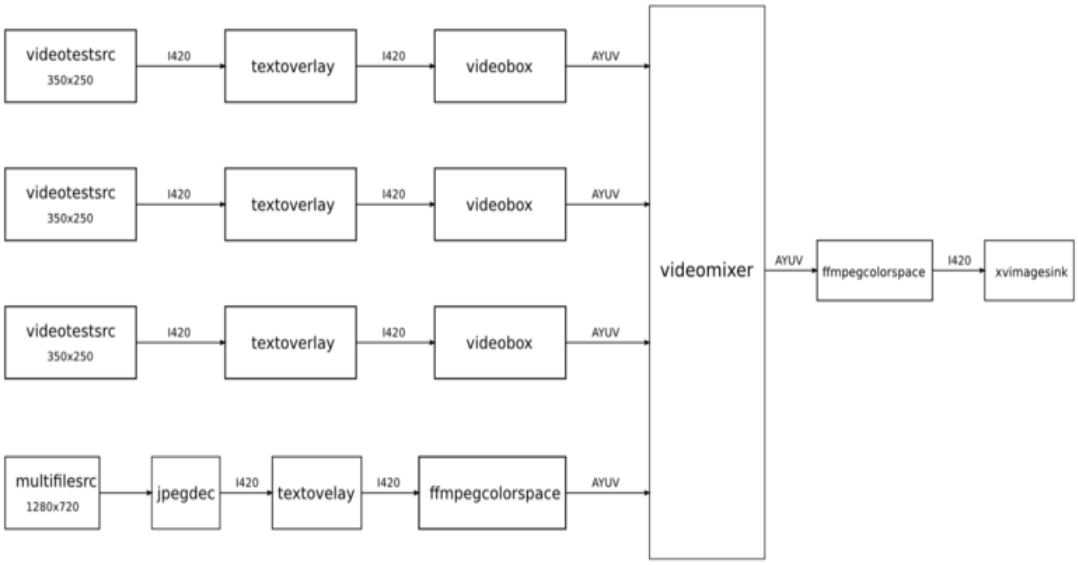


Fig A.2.6 Diagrama de flux amb gstreamer videoworld

Definiríem la pipeline, hem separat cada font de vídeo per línies per que resulti mes entenedor.

```
$ gst-launch-0.10 -e videomixer name=mesclador ! ffmpegcolorspace ! xvimagesink \
   videotestsrc pattern=0 ! video/x-raw-yuv, framerate=1/1, width=350, height=250 ! \
     textoverlay font-desc="Sans 24" text="CAMERA_1" valign=top halign=left shaded-background=true ! \
     videobox border-alpha=0 top=-200 left=-50 ! mesclador. \
```

```
   videotestsrc pattern="snow" ! video/x-raw-yuv, framerate=1/1, width=350,
height=250 ! \
     textoverlay font-desc="Sans 24" text="CAMERA_2" valign=top halign=left
shaded-background=true ! \
     videobox border-alpha=0 top=-200 left=-450 ! mescladors. \
   videotestsrc pattern=13 ! video/x-raw-yuv, framerate=1/1, width=350,
height=250 ! \
     textoverlay font-desc="Sans 24" text="CAMERA_3" valign=top halign=left
shaded-background=true ! \
     videobox border-alpha=0 top=-200 left=-850 ! mescladors. \
   multifilesrc location="mart.jpg" caps="image/jpeg,framerate=1/1" ! jpegdec
! \
     textoverlay font-desc="Sans 26" text="Directe desde Mart" halign=left
shaded-background=true auto-resize=false ! \
     ffmpegcolorspace ! video/x-raw-yuv,format=\(fourcc\)AYUV ! mescladors.
```

A la sortida veiem:

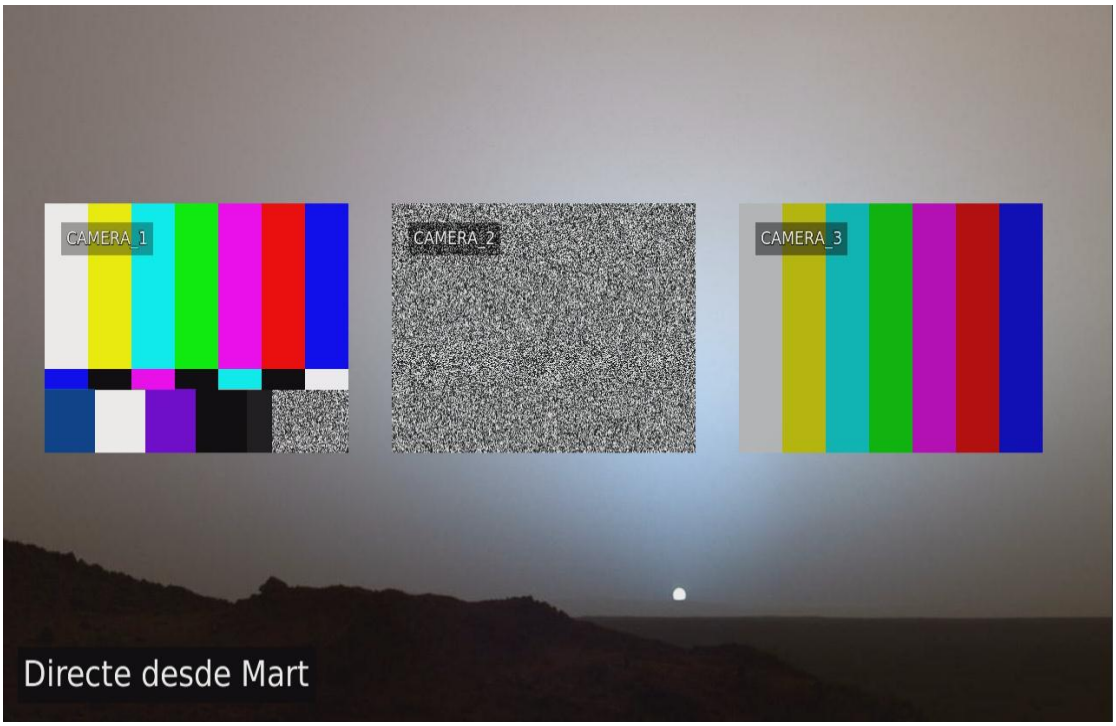


Fig. A.2.7 Sortida videoworld gstreamer

Un altre exemple de videoworld més simple. Si volem generar una matriu 2x2 composició de 4 streams de vídeo amb resolució 320x180 píxels i 5 fps:

```
gst-launch-0.10 -e videomixer name=mescladors ! ffmpegcolorspace ! xvimagesink
\

   videotestsrc pattern=1 ! video/x-raw-
yuv,framerate=5/1,width=320,height=180 ! videobox border-alpha=0 top=0 left=0
! mescladors. \

   videotestsrc pattern=11 ! video/x-raw-
yuv,framerate=5/1,width=320,height=180 ! videobox border-alpha=0 top=0 left=-
320 ! mescladors. \
```

```
   videotestsrc pattern=9 ! video/x-raw-
yuv,framerate=5/1,width=320,height=180 ! videobox border-alpha=0 top=-180
left=0 ! mesclador. \

   videotestsrc pattern=13 ! video/x-raw-
yuv,framerate=5/1,width=320,height=180 ! videobox border-alpha=0 top=-180
left=-320 ! mesclador. \

   videotestsrc pattern=3 ! video/x-raw-
yuv,framerate=5/1,width=640,height=360 ! mesclador.
```

A la sortida veurem una composició amb els 4 streams com aquesta:

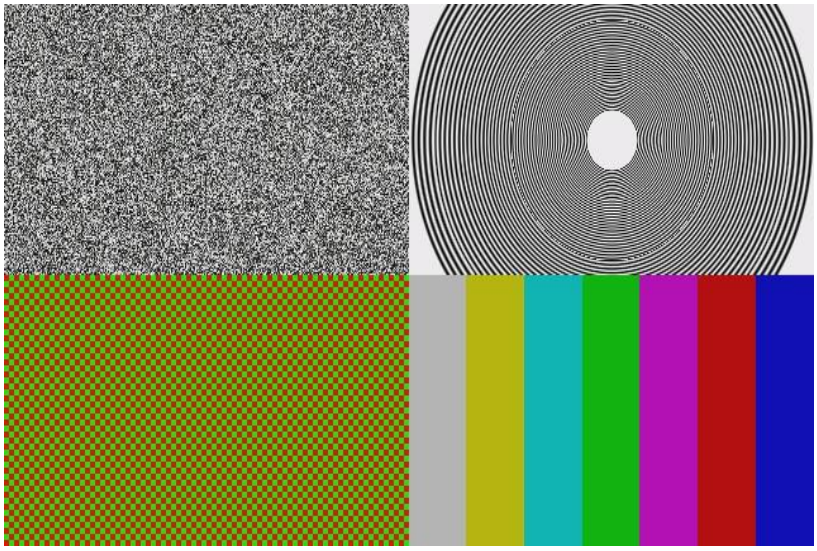


Fig A.2.8 Sortida gstreamer videoworld 2.

A.2.4 Fonts DVB-T i gstreamer

Un altre exemple de pipeline. El que volem és llegir d'una font DVB, i reproduir-lo per pantalla. Ens caldrà fer servir com a font un element del tipus `dvbsrc`, a aquesta font li haurem de passar uns paràmetres de configuració que hem de conèixer prèviament. Després com en el cas anterior li haurem de dir com volem consumir els elements, en aquest cas tindrem àudio i vídeo i li haurem d'indicar amb quines sinks els voldrem reproduir. El diagrama de flux és el que es mostra a la figura A.2.9.

Hem de llegir de la font `dvbsrc`, des multiplexem els fluxos amb `mpegtsdemux`, descodifiquem àudio i vídeo amb `mad` i `MPEG-2dec` respectivament. Finalment reproduïm. Per trobar els elements hem hagut de fer cerques amb `gst-inspect` i comprovat que les capacitats entre cada element de la cadena són les mateixes o compatibles.

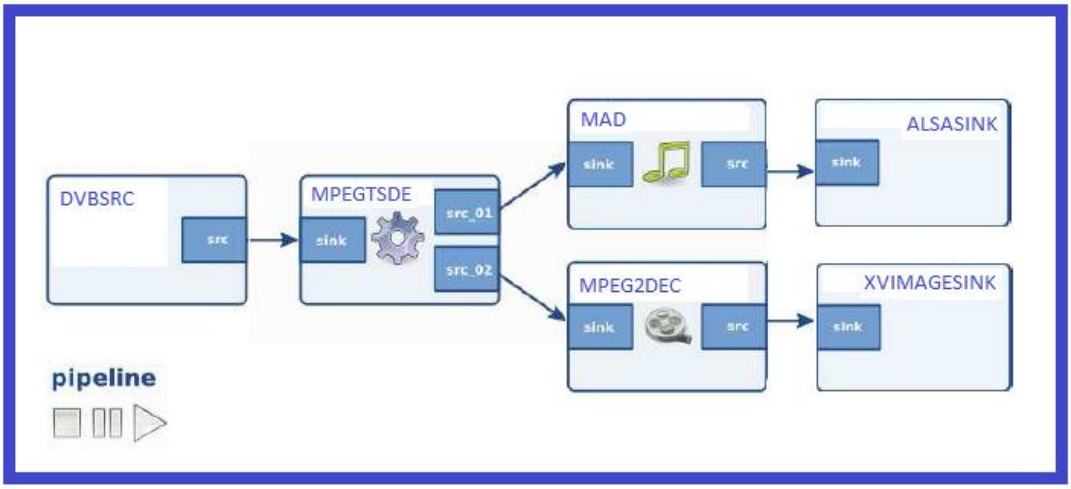


Fig. A.2.9 Descodificador / reproductor DVB-T gstreamer

Per fer-ho des de la consola executem:

```
$ gst-launch-0.10 dvbsrc modulation="QAM 64" trans-mode=8k bandwidth=8 frequency=794000000 code-rate-lp=AUTO code-rate-hp=2/3 guard=4 hierarchy=0 ! mpegtsdemux name=demux ! queue max-size-buffers=0 max-size-time=0 ! MPEG-2dec ! xvimagesink demux. ! queue max-size-buffers=0 max-size-time=0 ! mad ! alsasink
```

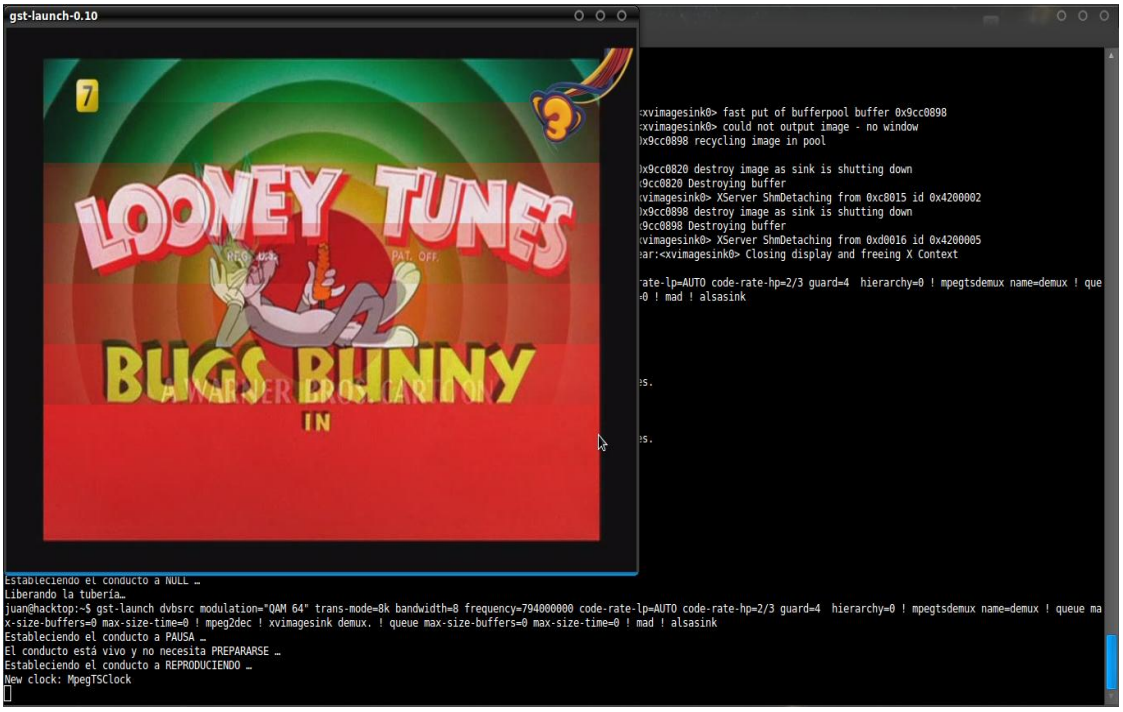


Fig. A.2.10 Reproductor font DVB-T amb gstreamer

Amb aquesta comanda reproduïm el primer PID del TS que troba per la freqüència portadora indicada. A la figura A.2.10 visualitzem i sentim la font

DVB-T. També amb `gst-inspect` sabem els paràmetres de configuració que li podem passar als diferents elements. Si volem reproduir uns PID's en concret, hem d'introduir:

```
$ gst-launch dvbsrc modulation="QAM 64" trans-mode=8k bandwidth=8 
frequency=794000000 code-rate-lp=AUTO code-rate-hp=2/3 guard=4 hierarchy=0 
pids=130:131:132 ! mpegtsdemux name=demux ! queue max-size-buffers=0 max-
size-time=0 ! MPEG-2dec ! xvimagesink demux. ! queue max-size-buffers=0 max-
size-time=0 ! mad ! alsasink
```

aquests PIDS 130:131:132 es corresponen al canal 3cat24, visualitzem la sortida a la figura A.2.11:

Fig A.2.11 Captura DVB per PID's

Per saber els PID's en concret que volem obtenir i reproduir, podem fer servir l'aplicació dvbsnoop:

```
$ dvbsnoop -pd 9 -f 24 -s pidscan
dvbsnoop V1.4.50 -- http://dvbsnoop.sourceforge.net/
   DEMUX : /dev/dvb/adapter0/demux0
   DVR   : /dev/dvb/adapter0/dvr0
   FRONTEND: /dev/dvb/adapter0/frontend0

-----------------------------------------------------------
Transponder PID-Scan...
-----------------------------------------------------------
scanning pid   0x0000 to 0x00fe  (got 255 dmx filters)
...
PID found:    0 (0x0000)  [SECTION: Program Association Table (PAT)]
PID found:    1 (0x0001)  [SECTION: Conditional Access Table (CAT)]
PID found:   16 (0x0010)  [SECTION: Network Information Table (NIT) - actual
network]
PID found:   17 (0x0011)  [SECTION: Service Description Table (SDT) - actual
transport stream]
PID found:   18 (0x0012)  [SECTION: Event Information Table (EIT) - actual
transport stream, present/following]
PID found:   20 (0x0014)  [SECTION: Time Date Table (TDT)]
```

```
PID found:   21 (0x0015)  [SECTION: ITU-T Rec. H.222.0|ISO/IEC13818 reserved]
PID found:  110 (0x006e)  [SECTION: Program Map Table (PMT)]
PID found:  111 (0x006f)  [PS/PES: ITU-T Rec. H.262 | ISO/IEC 13818-2 or
ISO/IEC 11172-2 video stream]
PID found:  112 (0x0070)  [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio
stream]
PID found:  114 (0x0072)  [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio
stream]
PID found:  115 (0x0073)  [PS/PES: private_stream_1]
PID found:  116 (0x0074)  [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio
stream]
PID found:  120 (0x0078)  [SECTION: Program Map Table (PMT)]
PID found:  121 (0x0079)  [PS/PES: ITU-T Rec. H.262 | ISO/IEC 13818-2 or
ISO/IEC 11172-2 video stream]
PID found:  122 (0x007a)  [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio
stream]
PID found:  124 (0x007c)  [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio
stream]
PID found:  126 (0x007e)  [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio
stream]
PID found:  130 (0x0082)  [SECTION: Program Map Table (PMT)]
PID found:  131 (0x0083)  [PS/PES: ITU-T Rec. H.262 | ISO/IEC 13818-2 or
ISO/IEC 11172-2 video stream]
PID found:  132 (0x0084)  [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio
stream]
PID found:  133 (0x0085)  [PS/PES: private_stream_1]
PID found:  134 (0x0086)  [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio
stream]
PID found:  140 (0x008c)  [SECTION: Program Map Table (PMT)]
PID found:  141 (0x008d)  [PS/PES: ITU-T Rec. H.262 | ISO/IEC 13818-2 or
ISO/IEC 11172-2 video stream]
PID found:  142 (0x008e)  [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio
stream]
PID found:  144 (0x0090)  [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio
stream]
PID found:  146 (0x0092)  [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio
stream]
PID found:  150 (0x0096)  [SECTION: Program Map Table (PMT)]
PID found:  151 (0x0097)  [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio
stream]
PID found:  160 (0x00a0)  [SECTION: Program Map Table (PMT)]
PID found:  161 (0x00a1)  [PS/PES: ISO/IEC 13818-3 or ISO/IEC 11172-3 audio
stream]
...
PID found:  802 (0x0322)  [PS/PES: private_stream_1]
PID found:  901 (0x0385)  [SECTION: MHP- Application Information Table (AIT)]
PID found:  902 (0x0386)  [SECTION: MHP- Application Information Table (AIT)]
PID found:  903 (0x0387)  [SECTION: MHP- Application Information Table (AIT)]
PID found:  904 (0x0388)  [SECTION: MHP- Application Information Table (AIT)]
...
PID found: 8191 (0x1fff)  [stuffing]
```

Comprovem l'estructura de les taules PSI quant llegim d'una font DVB-T que vam descriure a l'apartat 1.10.4. El PID 130 conté la PMT amb la informació dels programes. El PID 131 es correspon a un Elementary Stream ES de vídeo, el PID 132 és l'àudio estèreo d'aquest programa sincronitzat amb el vídeo. A l'annex A.4.2 ampliem el desenvolupament d'altres pipelines per des

multiplexar fonts DVB-T amb altres elements que ens donaran més informació dels diferents programes que viatgen al TS complet.

A.2.5 Conclusions

En aquest annex hem aprés a desenvolupar pipelines d'edició de vídeo i a operar i inter connectar elements en una mateixa cadena. A més a més hem introduït les fonts DVB i hem explicat com reproduir el fluxos amb gstreamer. Amb aquests exemples podem començar a adonar-nos de la potència de gstreamer. Encadenant els diferents elements podem manipular i configurar els fluxos d'acord amb les nostres necessitats. Hem de buscar els elements que ens calen i configurar correctament cada un dels paràmetres per aconseguir pipelines que generin els fluxos que volem.

A.3. Servidor RTSP amb gstreamer

En aquest annex volem desenvolupar un servei d'streaming que treballi amb el protocol RTSP. En primer lloc programarem una aplicació en C que faci servir les llibreries de gstreamer. Per fer-ho hem d'afegir algunes llibreries extres de gstreamer que inclouen les API's que ens calen. A [21] trobem les definicions de les API's que composen el servidor RTSP per a gstreamer. Des del gestor de descarrega de Linux, fem una cerca d'aplicacions. Ens cal instal·lar les següents:

- gst-rtsp-debugsource
- libgstrtspserver

És possible que el gestor ens sol·liciti complir algunes dependències amb altres aplicacions. Un cop instal·lades correctament al directori `/usr/lib/pkgconfig` han d'aparèixer les referencies `gstreamer-rtsp-0.10pc`. Al directori `/usr/lib/gstreamer-0.10` hem de verificar que es troben les referències a les llibreries de `gst-rtsp-server`. Un cop hem verificat que tot està correctament instal·lat, obrim l'arxiu de configuració del servidor que trobem al directori d'instal·lació, per exemple `rtsp.c`, i el modifiquem afegint les capçaleres necessàries:

```
#include <gst/gst.h>
#include <gst/rtsp-server/rtsp-server.h>
#include <gst/rtsp-server/rtsp-session-pool.h>
#include <gst/rtsp-server/rtsp-media-mapping.h>
#include <gst/rtsp-server/rtsp-media-factory.h>
#include <gst/rtsp-server/rtsp-media.h>
#include <gst/rtsp-server/rtsp-session.h>
#include <gst/rtsp-server/rtsp-params.h>
#include <gst/rtsp-server/rtsp-sdp.h>
#include <gst/rtsp-server/rtsp-client.h>

/* define this if you want the resource to only be available when using
 * user/admin as the password */
#undef WITH_AUTH
```

```
/* this timeout is periodically run to clean up the expired sessions from the
 * pool. This needs to be run explicitly currently but might be done
 * automatically as part of the mainloop. */
static gboolean
timeout (GstRTSPServer *server, gboolean ignored)
{
  GstRTSPSessionPool *pool;

  pool = gst_rtsp_server_get_session_pool (server);
  gst_rtsp_session_pool_cleanup (pool);
  g_object_unref (pool);

  return TRUE;
}

int
main (int argc, char *argv[])
{
  GMainLoop *loop;
  GstRTSPServer *server;
  GstRTSPMediaMapping *mapping;
  GstRTSPMediaFactory *factory;
#ifdef WITH_AUTH
  GstRTSPAuth *auth;
  gchar *basic;
#endif

  gst_init (&argc, &argv);

  loop = g_main_loop_new (NULL, FALSE);

  /* create a server instance */
  server = gst_rtsp_server_new ();

  /* get the mapping for this server, every server has a default mapper
object
   * that be used to map uri mount points to media factories */
  mapping = gst_rtsp_server_get_media_mapping (server);

#ifdef WITH_AUTH
  /* make a new authentication manager. it can be added to control access to
all
   * the factories on the server or on individual factories. */
  auth = gst_rtsp_auth_new ();
  basic = gst_rtsp_auth_make_basic ("user", "admin");
  gst_rtsp_auth_set_basic (auth, basic);
  g_free (basic);
  /* configure in the server */
  gst_rtsp_server_set_auth (server, auth);
#endif

  /* make a media factory for a test stream. The default media factory can
use
   * gst-launch syntax to create pipelines.
   * any launch line works as long as it contains elements named pay%d. Each
   * element with pay%d names will be a stream */
  factory = gst_rtsp_media_factory_new ();

  gst_rtsp_media_factory_set_launch (factory, "( "
```

```
          "videotestsrc ! video/x-raw-yuv,width=320,height=240,framerate=10/1
! "
          "x264enc ! queue ! rtph264pay name=pay0 pt=96 ! audiotestsrc !
audio/x-raw-int,rate=8000 ! alawenc ! rtppcmapay name=pay1 pt=97 "")");

  /* attach the test factory to the /test url */
  gst_rtsp_media_mapping_add_factory (mapping, "/test", factory);

  /* don't need the ref to the mapper anymore */
  g_object_unref (mapping);

  /* attach the server to the default maincontext */
  if (gst_rtsp_server_attach (server, NULL) == 0)
    goto failed;

  /* add a timeout for the session cleanup */
  g_timeout_add_seconds (2, (GSourceFunc) timeout, server);

  /* start serving, this never stops */
  g_main_loop_run (loop);

  return 0;

  /* ERRORS */
failed:
  {
    g_print ("failed to attach the server\n");
    return -1;
  }
}
```

En aquest exemple emetrem un flux de prova que podrem visualitzar amb el reproductor VLC. Un cop tenim l'arxiu `rtsp.c` l'hem de compilar indicant que farem servir les llibreries de gstreamer que hem afegit. Per fer-ho al directori on tenim `rtsp.c` executem:

```
$ gcc `pkg-config gstreamer-0.10 gst-rtsp-server-0.10 --libs` `pkg-config
gstreamer-0.10 gst-rtsp-server-0.10 --cflags` -o server rtscp.c
```

Molt important afegir les flags `` `pkg-config gstreamer-0.10 gst-rtsp-server-0.10 --libs` `pkg-config gstreamer-0.10 gst-rtsp-server-0.10 --cflags` `` per poder compilar l'aplicació. Li estem indicant al compilador que ha d'anar a buscar les llibreries `gst-rtsp-server-0.10`. Un cop compilat executem:

```
$ sudo ./server
```

I amb VLC obrim l'stream introduint l'URI: rtsp://ipservidor:8554/test fixem-nos que gstreamer fa servir el port 8554 per RTSP. Veiem la sortida amb VLC a la Fig. A.3.1.

Fig A.3.1 Sortida servidor RTSP amb gstreamer

Amb aquest exemple podem generar qualsevol pipeline de gstreamer que invoquem des de consola i servir-la per RTSP als clients que s'hi connectin. Hem de fer les modificacions que calguin al codi, concretament a la crida `gst_rtsp_media_factory_set_launch` i podrem visualitzar els streams que generem amb altres pipelines.

A.4. Configuració codificador mòbils amb gstreamer

En aquest annex volem implementar diferents codificadors amb gstreamer que integrarem als sistemes descrits al capítol 3. L'objectiu és configurar els codificadors com un mòdul integrable dintre d'altres sistemes d'streaming. Així obtindrem una alternativa intercanviable amb altres codificadors, com solucions HW. El codificador sempre s'ha d'integrar al sistema tenint en compte tots els blocs que el composen d'extrem a extrem. La necessitat sorgeix d'haver de transcodificar els fluxos a diferents resolucions i fps depenent del tipus de dispositiu que fem servir al costat de reproducció. Al punt A.4.1 configurarem el reproductor per a fonts v4l2 per a dispositius PC, mòbils Android i Iphone. Al punt A.4.2 farem el mateix per a fonts DVB-T. Al punt A.4.3 treballarem amb fonts de vídeo 3D en format fitxer.

A.4.1 Configuració codificador v4l2

L'objectiu és capturar vídeo de la web cam configurada com a dispositiu v4l2 i àudio del micro del PC. El primer que hem de fer es inspeccionar les CAPS o CAPABILITIES d'aquest element de gstreamer que es diu `v4l2src`. Executem en gst-inspect i obtenim aquesta sortida:

```
$ gst-inspect-0.10 v4l2src

  video/x-raw-yuv
```

```
            format: YVU9
             width: [ 1, 32768 ]
            height: [ 1, 32768 ]
         framerate: [ 0/1, 100/1 ]
video/x-raw-yuv
            format: YV12
             width: [ 1, 32768 ]
            height: [ 1, 32768 ]
         framerate: [ 0/1, 100/1 ]
video/x-raw-yuv
            format: YUY2
             width: [ 1, 32768 ]
            height: [ 1, 32768 ]
         framerate: [ 0/1, 100/1 ]
video/x-raw-yuv
            format: UYVY
             width: [ 1, 32768 ]
            height: [ 1, 32768 ]
         framerate: [ 0/1, 100/1 ]
video/x-raw-yuv
            format: Y42B
             width: [ 1, 32768 ]
            height: [ 1, 32768 ]
         framerate: [ 0/1, 100/1 ]
video/x-raw-yuv
            format: Y41B
             width: [ 1, 32768 ]
            height: [ 1, 32768 ]
         framerate: [ 0/1, 100/1 ]
video/x-raw-yuv
            format: Y41P
             width: [ 1, 32768 ]
            height: [ 1, 32768 ]
         framerate: [ 0/1, 100/1 ]
video/x-raw-yuv
            format: NV12
             width: [ 1, 32768 ]
            height: [ 1, 32768 ]
         framerate: [ 0/1, 100/1 ]
video/x-raw-yuv
            format: NV21
             width: [ 1, 32768 ]
            height: [ 1, 32768 ]
         framerate: [ 0/1, 100/1 ]
video/x-raw-yuv
            format: YUV9
             width: [ 1, 32768 ]
            height: [ 1, 32768 ]
         framerate: [ 0/1, 100/1 ]
video/x-raw-yuv
            format: I420
             width: [ 1, 32768 ]
            height: [ 1, 32768 ]
         framerate: [ 0/1, 100/1 ]
```

Amb `gst-inspect` podem comprovar les capacitats que té aquesta font en concret. En el nostre cas la font de v4l2 ens està generant un stream de vídeo en format x-raw-yuv/I420. Fixem-nos que aquest tipus de font és capaç de

treballar amb tots els formats de vídeo (CAPS de vídeo) descrites per `gst-inspect`.

La propietat `device` és el dispositiu ja configurat al sistema Linux i està definit al directori `/dev/video0`. Al nostre servei tenim només una webcam però si en tinguéssim més d'una podríem seleccionar-la com a `/dev/video1`.

La propietat flags determina les possibilitats hardware del dispositiu:

```
flags               : Device type flags
flags: legible
Flags "GstV4l2DeviceTypeFlags" Default: 0x00000000, "(none)" Current:
0x00000000, "(none)"
(0x00000001): capture          - Device supports video capture
(0x00000002): output           - Device supports video playback
(0x00000004): overlay          - Device supports video overlay
(0x00000010): vbi-capture      - Device supports VBI capture
(0x00000020): vbi-output       - Device supports the VBI output
(0x00010000): tuner            - Device has a tuner or modulator
(0x00020000): audio           - Device has audio in / out
```

Si volem ajustar les característiques del vídeo que estem capturant en termes de lluminositat, contrast i saturació també podem ajustar-les com desitgem o seleccionar un tipus de vídeo normalitzat amb aquest paràmetres:

```
brightness        : Picture brightness, or more precisely, the black level
                        flags: legible, escribible, controlable
                        Integer. Range: -2147483648 - 2147483647 Default: 0

contrast            : Picture contrast or luma gain
                        flags: legible, escribible, controlable
                        Integer. Range: -2147483648 - 2147483647 Default: 0

saturation         : Picture color saturation or chroma gain
                        flags: legible, escribible, controlable
                        Integer. Range: -2147483648 - 2147483647 Default: 0

hue                 : Hue or color balance
                        flags: legible, escribible, controlable
                        Integer. Range: -2147483648 - 2147483647 Default: 0

norm                : video standard

                        flags: legible, escribible

                        Enum "V4L2_TV_norms" Default: 0, "none" Current: 0,
                        "none"
```

Hem d'indicar-li al dispositiu com serà el vídeo que capturem en termes de format , frames per segon o fps i resolució. La nostra webcam és compatible amb una resolució de 640x420 píxels i 320x240 píxels, genera frames en format `raw-yuv/I420` codificats en MPEG-1 a una freqüència màxima de 30 frames per segon.

L'objectiu és dissenyar un codificador amb gstreamer com es mostra al diagrama de flux de la figura A.4.1:

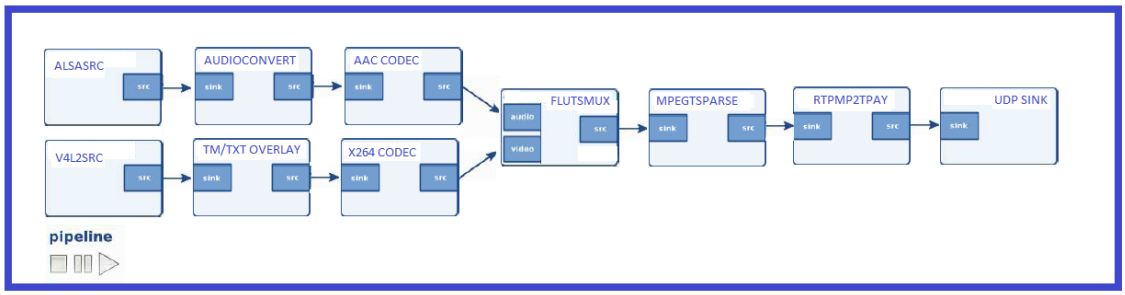


Fig. A.4.1 Diagrama de flux codificador v4l2

Llavors la primera part de la pipeline quedaria configurada de la següent manera:

```
v4l2src device="/dev/video0" ! video/x-raw-yuv, framerate=30/1, width=640, height=400 !
```

Volem afegir al vídeo un text amb el temps que fa que retransmetem i les característiques del vídeo que estem capturant. Amb gstreamer podem fer-ho fàcilment amb els elements `timeoverlay` i `textoverlay` de la següent manera:

```
v4l2src device="/dev/video0" ! video/x-raw-yuv, framerate=30/1, width=640, height=400 ! timeoverlay halign=right valign=bottom shaded-background=true ! textoverlay text="Test Video xtream 640x400 30fps" halign=left valign=bottom shaded-background=true
```

Ara mateix a la pipeline estem capturant imatges de la web cam i hi sobreescriurem el text. El següent element ha de codificar el vídeo a H.264 de forma que no ens generi un stream gaire pesat i amb una qualitat acceptable. Executem:

```
$ gst-inspect-0.10 x264enc
```

En primer lloc hem d'analitzar les CAPS d'aquest element integrant-se dins de la cadena com un element sink, fixem-nos que molts dels elements de gstreamer poden funcionar com a font o source o com a element consumidor o sink. Les CAPS (capabilities o capacitats) entre elements han de ser les mateixes o les esperades perquè podem integrar els diferents elements de la cadena un rere l'altre.

En aquest cas les CAPS com a element sink del `x264enc` són:

```
SINK template: 'sink'
   Availability: Always
   Capabilities:
     video/x-raw-yuv
                format: { I420, YV12 }
             framerate: [ 0/1, 2147483647/1 ]
                 width: [ 16, 2147483647 ]
```

```
                height: [ 16, 2147483647 ]
```

Com les CAPS com a element font de v4l2 són compatibles podem enllaçar els elements `v4l2src` i `x264enc` sempre i quan l'element v4l2 estigui capturant frames amb el format I420 o YV12.

En aquesta pipeline en concret ens interessa fixar el bit rate per no generar un stream gaire pesat i ens cal ajustar la velocitat del processat de vídeo de forma que es pugui processar i sincronitzar amb l'àudio al mateix temps. També volem ajustar el màxim possible la latència del sistema. No tota la latència del sistema és la que afegeix el codificador però si que l'hem ajustar el màxim possible per garantir el mínim de retard. La qualitat final de l'stream, la compressió i la latència dependrà de la configuració del les propietats `quantizer` i `speed-preset`. També podem seleccionar el tipus de perfil H.264 amb el que volem codificar.

Molt important: si estem treballant amb un stream de vídeo li hem d'indicar al codificador (en cas contrari gstreamer donarà error -4 i la pipeline no funcionarà). Per fer-ho hem de modificar la propietat `byte-stream=true`.

Hem de modificar i configurar algunes de les propietats. Fixem-nos en el conjunt de propietats que inclou `x264enc`

```
quantizer           : Constant quantizer or quality to apply
                      flags: legible, escribible.
                             Unsigned Integer. Range: 1 - 50 Default: 21
                      Current: 21

bitrate             : Bitrate in kbit/sec
                           flags:  legible,  escribible,  intercambiable
                      entre los estados NULL,            READY, PAUSED o
                      PLAYING Unsigned Integer. Range: 1 -
                      102400 Default: 2048 Current: 2048

profile              : Apply restrictions to meet H.264 Profile constraints.
This will                   override other properties if necessary. This
will only be used if downstream elements do not specify a profile in their
caps (DEPRECATED)
flags: legible, escribible
Enum "GstX264EncProfile" Default: 2, "main" Current: 2, "main"
                         (0): None             - No profile
                         (1): baseline         - baseline
                         (2): main             - main
                         (3): high             - high
                         (4): high10           - high10
                         (5): high422          - high422
                         (6): high444          - high444



speed-preset          : Preset name for speed/quality tradeoff options (can
affect                        decode compatibility - impose restrictions
separately for                      your target decoder)
                      flags: legible, escribible
                     Enum "GstX264EncPreset" Default: 6, "medium"
                        (0): None             - No preset
                        (1): ultrafast        - ultrafast
```

```
                           (2): superfast        - superfast
                           (3): veryfast         - veryfast
                           (4): faster           - faster
                           (5): fast             - fast
                           (6): medium           - medium
                           (7): slow             - slow
                           (8): slower           - slower
                           (9): veryslow         - veryslow

psy-tune            : Preset name for psychovisual tuning options
                             flags:      legible,      escribible      Enum
"GstX264EncPsyTune"                            Default: 0, "none"
                           (0): none             - No tuning
                           (1): film             - Film
                           (2): animation        - Animation
                           (3): grain            - Grain
                           (4): psnr             - PSNR
                           (5): ssim             - SSIM

tune                : Preset name for non-psychovisual tuning options
                        flags: legible, escribible
                        Flags "GstX264EncTune" Default: 0x00000000, "(none)"
                          (0x00000001): stillimage       - Still image
                          (0x00000002): fastdecode       - Fast decode
                          (0x00000004):  zerolatency          -  Zero  latency
(requires constant                        framerate)

byte-stream         : Generate byte stream format of NALU

                        flags: legible, escribible
                        Boolean. Default: false Current: false
```

Configurem l'encoder de la següent manera:

```
x264enc bitrate=498 speed-preset=fast tune=zerolatency
```

La pipeline quedaria:

```
v4l2src device="/dev/video0" ! video/x-raw-yuv, framerate=30/1, width=640,
height=400 !    timeoverlay halign=right valign=bottom shaded-background=true
!        textoverlay text="Test Video xtream 640x400 30fps" halign=left
valign=bottom shaded-background=true ! x264enc bitrate=498 speed-preset=fast
tune=zerolatency !
```

Volem fer la prova de que tot està funcionant, li direm a gstreamer que volem generar un flux de vídeo encapsulat en un TS transport stream. Haurem de parsejar amb `mpegtsparse` l'stream de vídeo que tenim a TS i guardarem el fitxer en local per comprovar que tot funciona correctament:

```
$ gst-launch-0.10 -e -v v4l2src device="/dev/video0" ! video/x-raw-yuv,
framerate=30/1, width=640, height=400 !        timeoverlay halign=right
valign=bottom shaded-background=true !    textoverlay text="Test Video xtream
640x400 30fps" halign=left valign=bottom shaded-background=true ! x264enc
bitrate=498 speed-preset=fast tune=zerolatency byte-stream=true ! queue !
mpegtsparse ! filesink location=videoH264_00.ts
```

Executem i comprovem que no funciona:

```
ADVERTENCIA: conducto erróneo: no se pudo enlazar queue0 a mpegtsparse0
```

El problema és que estem enllaçant una font source amb les CAPS de `x264enc`

```
SRC template: 'src'
    Availability: Always
    Capabilities:
      video/x-h264
              framerate: [ 0/1, 2147483647/1 ]
                  width: [ 1, 2147483647 ]
                 height: [ 1, 2147483647 ]
          stream-format: { byte-stream, avc }
              alignment: { au }
                profile: { high-10, high, main, constrained-baseline, high-
10-intra }
```

a l'element mpegtsparse, amb les CAPS com a sink:

```
Pad Templates:
  SINK template: 'sink'
    Availability: Always
    Capabilities:
      video/mpegts
           systemstream: true
```

Ens cal un element inter mig que consumeixi un stream de vídeo (sink) amb `CAPS=video/x-h264` i en generi un stream amb `CAPS=video/mpegts`, per això ens cal un des multiplexor d'streams que rebi un o més streams d'AV i en generi un únic TS amb els streams des multiplexats. Aquest element es diu `flutsmux`, amb les CAPS:

```
Pad Templates:
  SRC template: 'src'
    Availability: Always
    Capabilities:
      video/mpegts
           systemstream: true
             packetsize: { 188, 192 }
  SINK template: 'sink_%d'
    Availability: On request
      Has request_new_pad() function: 0xb70b8e20
    Capabilities:
      video/mpeg
            mpegversion: { 1, 2 }
           systemstream: false
      video/x-dirac
      video/x-h264
      audio/mpeg, mpegversion: { 1, 2, 4 }
```

Invoquem gstreamer des de l'intèrpret de comandes amb les opcions -e (d'aquesta manera al fer control+c des de consola s'envia l'ordre *EOS End Of Signal* d'aturar l'execució) i -v per que se'ns doni informació dels errors i el log funcioni en mode verbose:

```
$ gst-launch-0.10 -e -v v4l2src device="/dev/video0" ! video/x-raw-yuv, framerate=30/1, width=640, height=400 ! timeoverlay halign=right valign=bottom shaded-background=true ! textoverlay text="Test Video xtream 640x400 30fps" halign=left valign=bottom shaded-background=true ! x264enc bitrate=498 speed-preset=fast tune=zerolatency byte-stream=true ! queue ! flutsmux ! queue ! mpegtsparse ! filesink location=videoH264_00.ts

Estableciendo el conducto a PAUSA …

/GstPipeline:pipeline0/GstV4l2Src:v4l2src0.GstPad:src: caps = video/x-raw-yuv, format=(fourcc)I420, width=(int)640, height=(int)400, interlaced=(boolean)false, pixel-aspect-ratio=(fraction)1/1, framerate=(fraction)30/1, mpegversion=(int)1, systemstream=(boolean)false

El conducto está vivo y no necesita PREPARARSE …

Estableciendo el conducto a REPRODUCIENDO …

New clock: GstSystemClock

/GstPipeline:pipeline0/GstCapsFilter:capsfilter0.GstPad:src: caps = video/x-raw-yuv, format=(fourcc)I420, width=(int)640, height=(int)400, interlaced=(boolean)false, pixel-aspect-ratio=(fraction)1/1, framerate=(fraction)30/1, mpegversion=(int)1, systemstream=(boolean)false
/GstPipeline:pipeline0/GstCapsFilter:capsfilter0.GstPad:sink: caps = video/x-raw-yuv, format=(fourcc)I420, width=(int)640, height=(int)400, interlaced=(boolean)false, pixel-aspect-ratio=(fraction)1/1, framerate=(fraction)30/1, mpegversion=(int)1, systemstream=(boolean)false
/GstPipeline:pipeline0/GstTimeOverlay:timeoverlay0.GstPad:src: caps = video/x-raw-yuv, format=(fourcc)I420, width=(int)640, height=(int)400, interlaced=(boolean)false, pixel-aspect-ratio=(fraction)1/1, framerate=(fraction)30/1, mpegversion=(int)1, systemstream=(boolean)false
/GstPipeline:pipeline0/GstTimeOverlay:timeoverlay0.GstPad:video_sink: caps = video/x-raw-yuv, format=(fourcc)I420, width=(int)640, height=(int)400, interlaced=(boolean)false, pixel-aspect-ratio=(fraction)1/1, framerate=(fraction)30/1, mpegversion=(int)1, systemstream=(boolean)false
/GstPipeline:pipeline0/GstTextOverlay:textoverlay0.GstPad:src: caps = video/x-raw-yuv, format=(fourcc)I420, width=(int)640, height=(int)400, interlaced=(boolean)false, pixel-aspect-ratio=(fraction)1/1, framerate=(fraction)30/1, mpegversion=(int)1, systemstream=(boolean)false
/GstPipeline:pipeline0/GstTextOverlay:textoverlay0.GstPad:video_sink: caps = video/x-raw-yuv, format=(fourcc)I420, width=(int)640, height=(int)400, interlaced=(boolean)false, pixel-aspect-ratio=(fraction)1/1, framerate=(fraction)30/1, mpegversion=(int)1, systemstream=(boolean)false
/GstPipeline:pipeline0/GstX264Enc:x264enc0.GstPad:src: caps = video/x-h264, width=(int)640, height=(int)400, framerate=(fraction)30/1, pixel-aspect-ratio=(fraction)1/1, stream-format=(string)byte-stream, alignment=(string)au, level=(string)3, profile=(string)main
/GstPipeline:pipeline0/GstX264Enc:x264enc0.GstPad:sink: caps = video/x-raw-yuv, format=(fourcc)I420, width=(int)640, height=(int)400, interlaced=(boolean)false, pixel-aspect-ratio=(fraction)1/1, framerate=(fraction)30/1, mpegversion=(int)1, systemstream=(boolean)false
/GstPipeline:pipeline0/GstQueue:queue0.GstPad:sink: caps = video/x-h264, width=(int)640, height=(int)400, framerate=(fraction)30/1, pixel-aspect-ratio=(fraction)1/1, stream-format=(string)byte-stream, alignment=(string)au, level=(string)3, profile=(string)main
/GstPipeline:pipeline0/GstQueue:queue0.GstPad:src: caps = video/x-h264, width=(int)640, height=(int)400, framerate=(fraction)30/1, pixel-aspect-
```

```
ratio=(fraction)1/1, stream-format=(string)byte-stream, alignment=(string)au,
level=(string)3, profile=(string)main
/GstPipeline:pipeline0/FluTsMux:flutsmux0.GstPad:sink_64: caps = video/x-
h264, width=(int)640, height=(int)400, framerate=(fraction)30/1, pixel-
aspect-ratio=(fraction)1/1, stream-format=(string)byte-stream,
alignment=(string)au, level=(string)3, profile=(string)main
/GstPipeline:pipeline0/FluTsMux:flutsmux0.GstPad:src: caps = video/mpegts,
systemstream=(boolean)true, packetsize=(int)188
/GstPipeline:pipeline0/MpegTSParse:mpegtsparse0.GstPad:src0: caps =
video/mpegts, systemstream=(boolean)true, packetsize=(int)188
/GstPipeline:pipeline0/GstFileSink:filesink0.GstPad:sink: caps =
video/mpegts, systemstream=(boolean)true, packetsize=(int)188

^CCaught interrupt -- handling interrupt.

Interrumpir: parando el conducto …
EOS al apagar activado; Forzando EOS en la tubería
Esperando a EOS…
Se recibió un EOS del elemento «pipeline0».
EOS recibido: parando el conducto …
Execution ended after 22549876417 ns.
Estableciendo el conducto a PAUSA …
Estableciando el conducto a PREPARADO …
/GstPipeline:pipeline0/GstFileSink:filesink0.GstPad:sink: caps = NULL
/GstPipeline:pipeline0/MpegTSParse:mpegtsparse0.GstPad:src0: caps = NULL
/GstPipeline:pipeline0/FluTsMux:flutsmux0.GstPad:sink_64: caps = NULL
/GstPipeline:pipeline0/FluTsMux:flutsmux0.GstPad:src: caps = NULL
/GstPipeline:pipeline0/GstQueue:queue0.GstPad:src: caps = NULL
/GstPipeline:pipeline0/GstQueue:queue0.GstPad:sink: caps = NULL
/GstPipeline:pipeline0/GstX264Enc:x264enc0.GstPad:src: caps = NULL
/GstPipeline:pipeline0/GstX264Enc:x264enc0.GstPad:sink: caps = NULL
/GstPipeline:pipeline0/GstTextOverlay:textoverlay0.GstPad:src: caps = NULL
/GstPipeline:pipeline0/GstTextOverlay:textoverlay0.GstPad:video_sink: caps =
NULL
/GstPipeline:pipeline0/GstTimeOverlay:timeoverlay0.GstPad:src: caps = NULL
/GstPipeline:pipeline0/GstTimeOverlay:timeoverlay0.GstPad:video_sink: caps =
NULL
/GstPipeline:pipeline0/GstCapsFilter:capsfilter0.GstPad:src: caps = NULL
/GstPipeline:pipeline0/GstCapsFilter:capsfilter0.GstPad:sink: caps = NULL
/GstPipeline:pipeline0/GstV4l2Src:v4l2src0.GstPad:src: caps = NULL
Estableciendo el conducto a NULL …
Liberando la tubería…
```

Fixem-nos en la informació del log, que ens dona els valors de les diferents propietats dels elements. La figura A.4.1 mostra el flux de la pipeline del codificador.

Como tot ha anat bé, gstreamer genera tota la informació del sistema indicant-nos els valors de les CAPS i de les propietats de cada element quan funciona com a element consumidor i com a font. Reproduïm l'arxiu generat amb VLC, com veiem a la Fig. A.4.2.

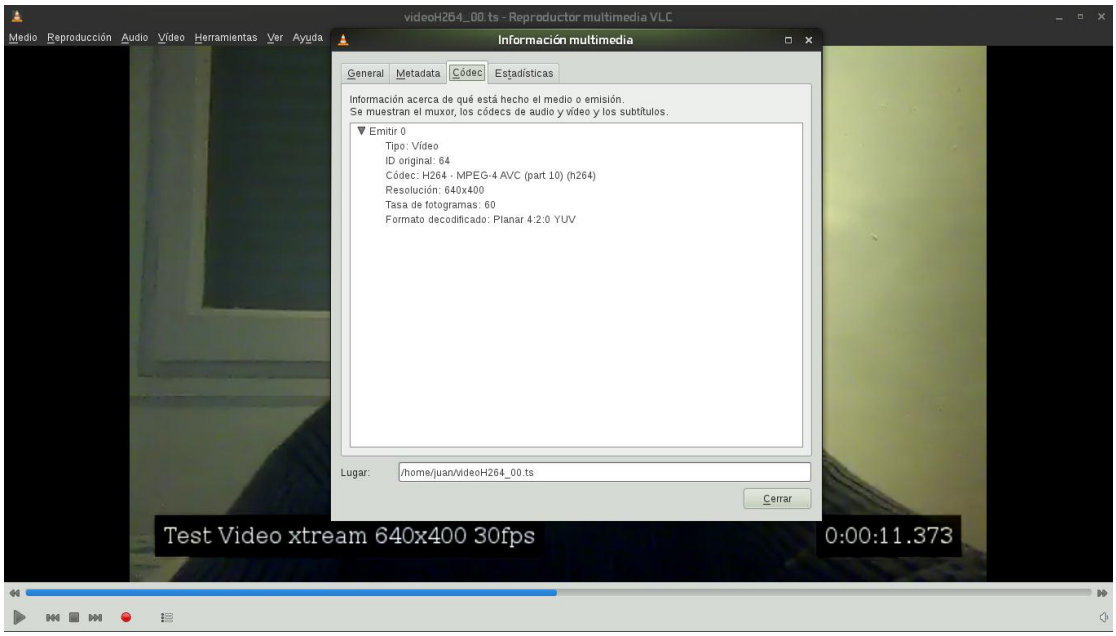


Fig A.4.2 Sortida captura flux TS compressió AVC amb VLC

S'ha generat un fitxer encapsulat com a TS amb només un flux de vídeo a resolució 640x420 píxels. Volem posar a prova el codificador configurant-lo amb més qualitat:

```
gst-launch-0.10 -e -v v4l2src device="/dev/video0" ! video/x-raw-yuv, framerate=30/1, width=640, height=400 ! timeoverlay halign=right valign=bottom shaded-background=true ! textoverlay text="Test Video xtream 640x400 30fps" halign=left valign=bottom shaded-background=true ! x264enc bitrate=1000 speed-preset=fast quantizer=50 profile=high10 tune=zerolatency byte-stream=true ! queue ! flutsmux ! queue ! mpegtsparse ! filesink location=videoH264_01.ts
```

El vídeo que es genera té més qualitat però s'estan perdent frames i sensació de moviment, a la nostra CPU no li dona temps de codificar amb aquesta qualitat i aquest perfil, baixem la qualitat del quantificador a la meitat:

```
gst-launch-0.10 -e -v v4l2src device="/dev/video0" ! video/x-raw-yuv, framerate=30/1, width=640, height=400 ! timeoverlay halign=right valign=bottom shaded-background=true ! textoverlay text="Test Video xtream 640x400 30fps" halign=left valign=bottom shaded-background=true ! x264enc bitrate=1000 speed-preset=fast quantizer=25 profile=high10 tune=zerolatency byte-stream=true ! queue ! flutsmux ! queue ! mpegtsparse ! filesink location=videoH264_02.ts
```

Continuem amb problemes i perdem frames i sensació de moviment. Deixem el perfil com està i el quantificador al valor per defecte:

```
gst-launch-0.10 -e -v v4l2src device="/dev/video0" ! video/x-raw-yuv, framerate=30/1, width=640, height=400 ! timeoverlay halign=right valign=bottom shaded-background=true ! textoverlay text="Test Video xtream 640x400 30fps" halign=left valign=bottom shaded-background=true ! x264enc bitrate=1000 speed-preset=fast tune=zerolatency byte-stream=true ! queue ! flutsmux ! queue ! mpegtsparse ! filesink location=videoH264_03.ts
```

Aquesta configuració és la més eficient, no perdem frames ni sensació de moviment i el vídeo és de la millor qualitat possible d'acord amb la nostra CPU. En qualsevol cas encara hem de carregar més la CPU amb el processat de l'àudio i la transmissió de l'stream cap al servidor. Ara que ja tenim el vídeo codificat hem de fer el mateix amb l'àudio i des multiplexar-lo juntament amb el vídeo. Abans d'això generarem només un stream d'àudio codificat en format AAC. Prendrem mostres del micròfon integrat del portàtil, aquest tipus de dispositiu ha d'estar prèviament configurat al sistema Linux com un dispositiu `alsasrc`. Aquest serà l'element font `alsasrc` de la pipeline millorada:

```
alsasrc device=hw:0,0 provide-clock=false typefind=true num-buffers=-1 ! audio/x-raw-int, signed=true, rate=48000, channels=2, width=16, depth=16
```

Fixem-nos a les propietats `provide-clock=false`, més endavant en aquest cas pràctic volem que l'stream estigui sincronitzat amb el vídeo que ja te el seu propi rellotge de sistema. Si volem que l'àudio es sincronitzi li hem d'indicar que no generi el seu propi rellotge i indicar a l'últim element sink que volem que tots dos fluxos estiguin sincronitzats .

Fem el mateix procés, obtenim les mostres de la font `alsasrc` d'acord amb les CAPS i codifiquem l'stream amb un element codificador. Necessitem l'element `voaacenc` que és l'element codificador d'àudio. Verifiquem que la font d'alsasrc és compatible amb la sink voaacenc executant `gst-inspect` per `alsasrc` i `voaacenc` i verificant les caps de font i sink:

Font `alsasrc`:

```
      audio/x-raw-int
             endianness: { 1234, 4321 }
                 signed: { true, false }
                  width: 16
                  depth: 16
                   rate: [ 1, 2147483647 ]
               channels: [ 1, 2147483647 ]

Sink voaacenc:
 SINK template: 'sink'
    Availability: Always
    Capabilities:
      audio/x-raw-int
                  width: 16
                  depth: 16
                 signed: true
             endianness: 1234
                   rate: { 8000, 11025, 12000, 16000, 22050, 24000, 32000, 44100, 48000, 64000, 88200, 96000 }
                channels: [ 1, 2 ]
```

Executem la pipeline:

```
gst-launch-0.10 -e -v alsasrc device=hw:0,0 provide-clock=false typefind=true num-buffers=-1 ! audio/x-raw-int,signed=true,rate=48000,channels=2,width=16,depth=16 ! queue ! voaacenc ! flutsmux ! mpegtsparse ! filesink location=audioAAC_00.ts
```

Reproduïm el flux amb VLC i sentim que funciona i el tipus multimedia és l'esperat. Un cop tenim codificat AV hem de multiplexar tots dos fluxos en un únic TS. A gstreamer per enllaçar més d'un stream en una mateixa pipeline hem de fer servir un multiplexor, en aquest cas de TS. Afegim la part de vídeo:

```
v4l2src device="/dev/video0" ! video/x-raw-yuv, framerate=30/1, width=640, height=400 !    timeoverlay halign=right valign=bottom shaded-background=true !        textoverlay text="Test Video xtream 640x400 30fps" halign=left valign=bottom shaded-background=true ! x264enc bitrate=498 speed-preset=fast tune=zerolatency byte-stream=true ! queue !
```

El següent element ha de ser el multiplexor, però com també necessitem enllaçar amb l'àudio hem d'indicar-li a gstreamer que el flux de vídeo es multiplexa amb el d'àudio, definirem un multiplexor `flutsmux` amb el nom `mux`. Per indicar que enllacem el flux hem d'afegir aquest element al començament del flux d'àudio i enllaçar-lo amb el vídeo abans de la crida a l'element multiplexor, bolquem a fitxer:

```
$ gst-launch-0.10 -e -v v4l2src device="/dev/video0" ! video/x-raw-yuv, framerate=30/1, width=640, height=400 !         timeoverlay halign=right valign=bottom shaded-background=true !    textoverlay text="Test Video xtream 640x400 30fps" halign=left valign=bottom shaded-background=true ! x264enc bitrate=498 speed-preset=fast tune=zerolatency ! queue ! mux. alsasrc device=hw:0,0 provide-clock=false typefind=true num-buffers=-1 ! audio/x-raw-int,signed=true,rate=48000,channels=2,width=16,depth=16 ! queue ! voaacenc ! mux.    flutsmux       name=mux    !    mpegtsparse    !    queue    !    filesink location=av_mpegts_00.ts
```

Per completar el codificador hem d'enviar a xarxa els fluxos multiplexats cap al servidor d'streaming. Els protocols de comunicació poden ser principalment dos RTP (sobre UDP) i RTSP, en aquest cas la comunicació entre codificador i servidor serà per RTP (sobre UDP). Inspeccionem les CAPS de l'element `rtpmp2tpay` que ens permet paquetitzar complint el protocol RTP encapsulat en el flux TS (d'aqui el nom de `rtpmp2t`). A més gstreamer diferencia a quin costat de la cadena de transmissió som, si transmetem fem servir `rtpmp2tpay`, en el cas de que estem rebent el flux al costat del servidor (recepció) hauríem de fer servir `rtpmp2tdepay`.

```
Pad Templates:
  SRC template: 'src'
    Availability: Always
    Capabilities:
      application/x-rtp
                  media: video
                payload: [ 96, 127 ]
             clock-rate: 90000
          encoding-name: MP2T-ES

SINK template: 'sink'
    Availability: Always
    Capabilities:
      video/mpegts
             packetsize: 188
```

```
systemstream: true
```

Comprovem que les CAPS de sink `video/mpegts` de rtpmp2tpay són compatibles amb les caps de flutsmux com a font, llavors sent la IP del nostre servidor 146.255.96.41 i el port per on escolta el servidor d'streaming el 10000, fixem la condició d'àudio i vídeo sincronitzats amb la propietat `sync=true`:

```
gst-launch-0.10 -e -v v4l2src device="/dev/video0" ! video/x-raw-yuv, framerate=30/1, width=640, height=400 ! timeoverlay halign=right valign=bottom shaded-background=true ! textoverlay text="Test Video xtream 640x400 30fps" halign=left valign=bottom shaded-background=true ! x264enc bitrate=498 speed-preset=fast tune=zerolatency ! queue ! mux. alsasrc device=hw:0,0 provide-clock=false typefind=true num-buffers=-1 ! audio/x-raw-int,signed=true,rate=48000,channels=2,width=16,depth=16 ! queue ! audioconvert qos=true ! queue ! voaacenc ! mux. flutsmux name=mux ! mpegtsparse ! rtpmp2tpay ! udpsink host=146.255.96.41 port=10000 sync=true
```

Analitzem la sortida del log de gstreamer, en concret les CAPS de `rtpmp2tpay` i `udpsink`:

```
/GstPipeline:pipeline0/GstRTPMP2TPay:rtpmp2tpay0.GstPad:src: caps = application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-name=(string)MP2T-ES, payload=(int)33, ssrc=(uint)1966298605, clock-base=(uint)4185301433, seqnum-base=(uint)32939
/GstPipeline:pipeline0/GstRTPMP2TPay:rtpmp2tpay0.GstPad:sink: caps = video/mpegts, systemstream=(boolean)true, packetsize=(int)188
/GstPipeline:pipeline0/GstRTPMP2TPay:rtpmp2tpay0: timestamp = 4185301433
/GstPipeline:pipeline0/GstRTPMP2TPay:rtpmp2tpay0: seqnum = 32939
/GstPipeline:pipeline0/GstUDPSink:udpsink0.GstPad:sink: caps = application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-name=(string)MP2T-ES, payload=(int)33, ssrc=(uint)1966298605, clock-base=(uint)4185301433, seqnum-base=(uint)32939
```

Comprovem que es generen números de seqüència i timestamps per controlar el sincronisme del flux i l'ordre de recepció dels paquets. Tot ha anat com esperàvem, analitzem la sortida amb wireshark per comprovar que s'estan enviant paquets al destí indicat:

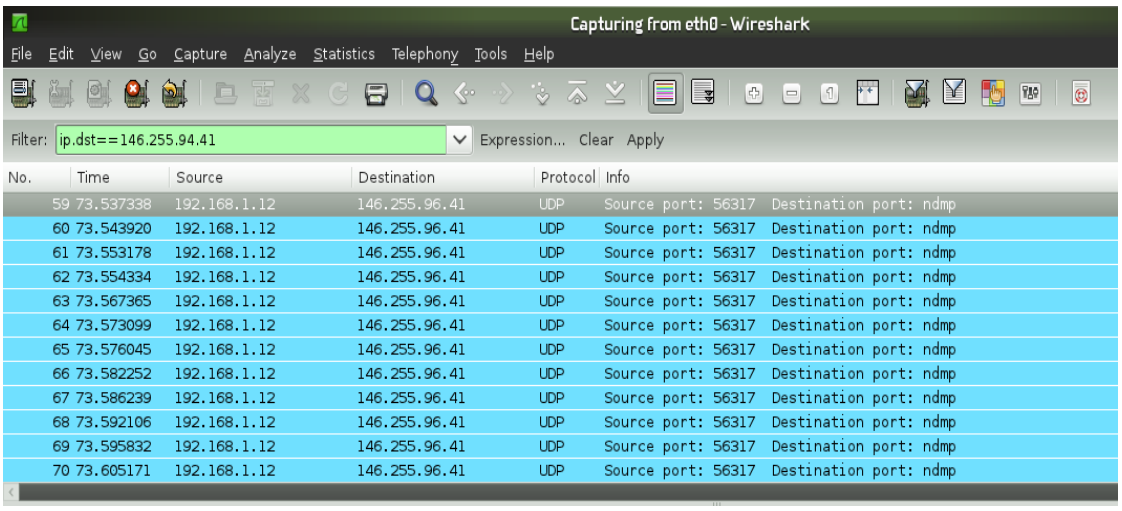


Fig A.4.3 Captura de wireshark codificador per RTP

A.4.1.1 Configuració del codificador gstreamer per a Android

El proper pas és modificar la cadena per tal de servir l'stream per a dispositius mòbils amb el SO Android. A la taula.A.4.1 es descriuen les característiques del flux de vídeo compatibles amb aquest SO.

VÍDEO CÒDEC	RESOLUCIÓ	FPS	BIR RATE VIDEO	ÀUDIO CÒDEC	CANALS	BIT RATE ÀUDIO
H.264 Baseline Profile	174x144 píxels	12	56 kbps	AAC LC	1	24 kbps
H.264 Baseline Profile	480x360 píxels	30	500 kbps	AAC LC	2	128 kbps
H.264 Baseline Profile	1280x720píxels	30	2 Mbps	AAC LC	2	192 kbps
						ANDROID

Taula A.4.1 Característiques senyals AV per Android.

Les modificacions que hem de fer ens han de permetre modificar els paràmetres de fps i resolucions en el cas del vídeo i freqüència de mostreig i número de canals ,principalment, en el cas de l'àudio. A la figura A.4.4 es mostra el flux de la cadena que volem dissenyar:

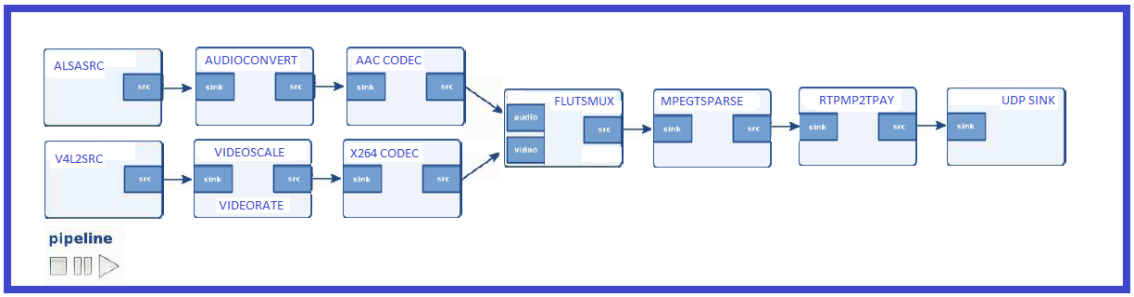


Fig A.4.4 Codificador v4l2 per Android

La principal modificació és fer servir els elements `videorate` i `videoscale` per ajustar els paràmetres que ens cal modificar segons la taula A.4.1. Eliminem els elements per afegir text a sobre del vídeo:

```
$ gst-launch-0.10 -e -v v4l2src device="/dev/video0" ! video/x-raw-yuv, framerate=30/1, width=640, height=400 ! ffmpegcolorspace ! queue ! videoscale ! video/x-raw-yuv, width=174, height=144 ! videorate ! video/x-raw-yuv, framerate=12/1 ! queue ! x264enc byte-stream=true speed-preset=superfast bitrate=150 tune=fastdecode byte-stream=true ! queue ! flumux. demux. ! queue ! flump3dec ! queue ! voaacenc ! queue ! flumux. flutsmux name=flumux ! queue ! mpegtsparse ! rtpmp2tpay ! udpsink host=146.255.96.41 port=10000 sync=true
```

A.4.1.2 Configuració del codificador gstreamer per a Iphone.

En aquest apartat volem fer modificacions a la pipeline del punt A.4.1.1 per servir fluxos d'AV a millor qualitat per aquest dispositiu. A la Taula A.4.3 es detallen les característiques que han de complir els fluxos per a que siguin reproduïbles per l'Iphone.

VÍDEO CÒDEC	RESOLUCIÓ	FPS	BIR RATE VIDEO	ÀUDIO CÒDEC	CANALS	BIT RATE ÀUDIO
H.264 Baseline Profile	640x480 píxels	30	1.5 Mbps	AAC LC	2	160 kbps
H.264 Baseline Profile	640x480 píxels	30	2.5 Mbps	AAC LC	2	160 kbps
						IPHONE

Taula A.4.3 Característiques senyals audiovisuals per Iphone

Les modificacions les farem als paràmetres `videorate` i `videoscale`, la cadena de elements del codificador quedaria:

```
$ gst-launch-0.10 -e -v v4l2src device="/dev/video0" ! video/x-raw-yuv, framerate=30/1, width=640, height=400 ! ffmpegcolorspace ! queue ! videoscale ! video/x-raw-yuv, width=320, height=240 ! videorate ! video/x-raw-yuv, framerate=25/1 ! queue ! x264enc byte-stream=true speed-preset=superfast bitrate=1500 tune=fastdecode byte-stream=true ! queue ! flumux. demux. ! queue ! flump3dec ! queue ! voaacenc ! queue ! flumux. flutsmux name=flumux ! queue ! mpegtsparse ! rtpmp2tpay ! udpsink host=146.255.96.41 port=10000 sync=true
```

A.4.2 Configuració codificador DVB-T

En aquest annex volem configurar el codificador de gstreamer per a capturar els fluxos d'àudio i vídeo d'una font DVB-T. L'objectiu és capturar i codificar a AVC i AAC per vídeo i àudio respectivament. Després volem servir un flux TS compatible amb els sistemes descrits al capítol 3. El diagrama de flux del codificador de gstreamer es mostra a la Figura A.4.7.

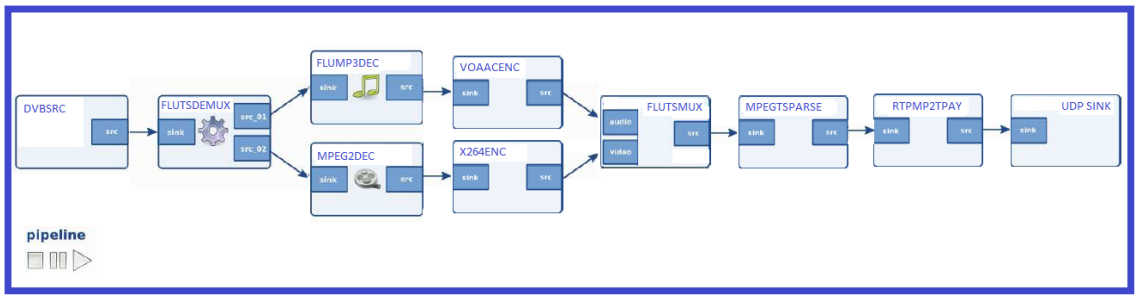


Figura A.4.7 Codificador DVB-T a AVC i AAC en TS gstreamer

```
dvbsrc modulation="QAM64", trans-mode=8k, bandwidth=8, frequency=658000000, code-rate-lp=AUTO, code-rate-hp=2/3 guard=4 hierarchy=0 ! flutsdemux program-number=807 name=demux demux. !
```

En aquest cas estem fent servir un des multiplexor de fluxos TS per a DVB-T, `flutsdemux`. Li passem com a paràmetres els que caracteritzen la font DVB-T i seleccionem el `program-number=807` del canal Esports3. A la sortida de `flutsdemux` tindrem diferents fonts de vídeo, àudio i dades corresponents als programa que hem des multiplexat. N'haurem de codificar l'àudio i el vídeo. Pel vídeo en primer lloc descodifiquem de la compressió MPEG-2 i codifiquem a H.264:

```
MPEG-2dec ! queue max-size-buffers=0 max-size-time=0 ! ffmpegcolorspace ! queue ! x264enc byte-stream=true speed-preset=superfast bitrate=500 tune=zerolatency byte-stream=true !
```

Aquest tros de pipeline està optimitzada per transmetre l'stream a resolució PAL i ser enviat al servidor d'streaming. Al punt A.4.2.1 i A.4.2.2 veurem com modificar el vídeo per ajustar-lo a un dispositiu mòbil. Per l'àudio fem el mateix descodifiquem de MPEG-1 Audio Layer II i codifiquem a AAC:

```
flump3dec ! queue ! voaacenc hard-resync=true ! queue !
```

Ho volem multiplexar tot en un mateix flux TS per ser enviat al servidor, llavors la pipeline quedaria ,afegint a la cadena la part d'enviament a xarxa:

```
$ gst-launch-0.10 -e -vvv dvbsrc modulation="QAM 64" trans-mode=8k bandwidth=8 frequency=658000000 code-rate-lp=AUTO code-rate-hp=2/3 guard=4 hierarchy=0 ! flutsdemux program-number=807 name=demux demux. ! queue ! MPEG-2dec ! queue max-size-buffers=0 max-size-time=0 ! ffmpegcolorspace ! queue ! x264enc byte-stream=true speed-preset=superfast bitrate=500 tune=zerolatency byte-stream=true ! queue ! flumux. demux. ! queue ! flump3dec ! queue ! voaacenc hard-resync=true ! queue ! flumux. flutsmux name=flumux ! queue ! mpegtsparse ! rtpmp2tpay ! udpsink host=146.255.96.41 port=10000 sync=true
```

Executem i reproduïm el directe amb VLC dels fluxos del servidor:

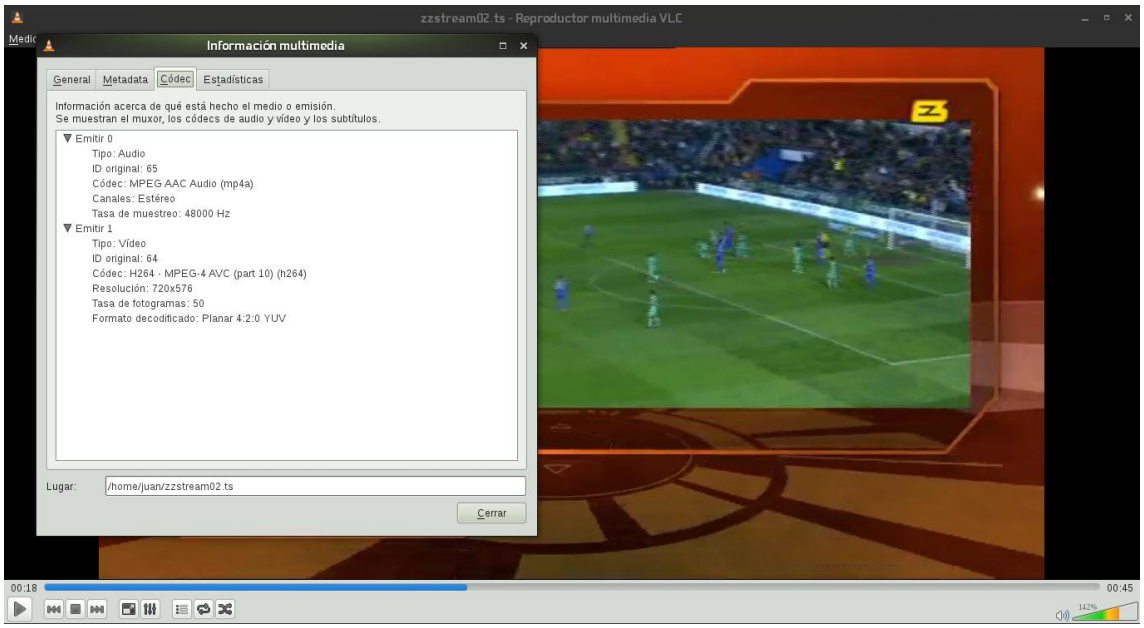


Fig A.4.8 Reproducció codificador DVB-T gstreamer

Verifiquem l'AV sincronitzat en reproducció i codificat correctament a resolució 720x576 píxels. Podem extreure molta més informació de la sortida de consola de gstreamer on es detalla tot el procés que fa el codificador:

```
Estableciendo el conducto a PAUSA …
/GstPipeline:pipeline0/GstDvbSrc:dvbsrc0.GstPad:src: caps = video/mpegts,
mpegversion=(int)2, systemstream=(boolean)true
El conducto está vivo y no necesita PREPARARSE …
Estableciendo el conducto a REPRODUCIENDO …
New clock: FluTSClock
/GstPipeline:pipeline0/GstFluTSDemux:demux.GstPad:sink: caps = video/mpegts,
mpegversion=(int)2, systemstream=(boolean)true
/GstPipeline:pipeline0/GstFluTSDemux:demux: pat-info = ((GValueArray*)
0x81ced30)
/GstPipeline:pipeline0/GstFluTSDemux:demux: pmt-info = ((FluTsPmtInfo*)
0x8269560)
/GstPipeline:pipeline0/GstQueue:queue4.GstPad:sink: caps = audio/mpeg,
mpegversion=(int)1
/GstPipeline:pipeline0/GstQueue:queue4.GstPad:src: caps = audio/mpeg,
mpegversion=(int)1
/GstPipeline:pipeline0/FluMp3Dec:flump3dec0.GstPad:sink: caps = audio/mpeg,
mpegversion=(int)1
/GstPipeline:pipeline0/FluMp3Dec:flump3dec0.GstPad:src: caps = audio/x-raw-
int, endianness=(int)1234, signed=(boolean)true, width=(int)16,
depth=(int)16, rate=(int)48000, channels=(int)1
/GstPipeline:pipeline0/GstQueue:queue5.GstPad:sink: caps = audio/x-raw-int,
endianness=(int)1234, signed=(boolean)true, width=(int)16, depth=(int)16,
rate=(int)48000, channels=(int)1
/GstPipeline:pipeline0/GstQueue:queue5.GstPad:src: caps = audio/x-raw-int,
endianness=(int)1234, signed=(boolean)true, width=(int)16, depth=(int)16,
rate=(int)48000, channels=(int)1
```

```
/GstPipeline:pipeline0/GstVoAacEnc:voaacenc0.GstPad:src: caps = audio/mpeg,
mpegversion=(int)4, channels=(int)1, rate=(int)48000, stream-
format=(string)raw, level=(string)2, base-profile=(string)lc,
profile=(string)lc, codec_data=(buffer)1188
/GstPipeline:pipeline0/GstVoAacEnc:voaacenc0.GstPad:sink: caps = audio/x-raw-
int, endianness=(int)1234, signed=(boolean)true, width=(int)16,
depth=(int)16, rate=(int)48000, channels=(int)1
/GstPipeline:pipeline0/GstQueue:queue6.GstPad:sink: caps = audio/mpeg,
mpegversion=(int)4, channels=(int)1, rate=(int)48000, stream-
format=(string)raw, level=(string)2, base-profile=(string)lc,
profile=(string)lc, codec_data=(buffer)1188
/GstPipeline:pipeline0/GstQueue:queue6.GstPad:src: caps = audio/mpeg,
mpegversion=(int)4, channels=(int)1, rate=(int)48000, stream-
format=(string)raw, level=(string)2, base-profile=(string)lc,
profile=(string)lc, codec_data=(buffer)1188
/GstPipeline:pipeline0/FluTsMux:flumux.GstPad:sink_65: caps = audio/mpeg,
mpegversion=(int)4, channels=(int)1, rate=(int)48000, stream-
format=(string)raw, level=(string)2, base-profile=(string)lc,
profile=(string)lc, codec_data=(buffer)1188
/GstPipeline:pipeline0/GstQueue:queue0.GstPad:sink: caps = video/mpeg,
mpegversion=(int)2, systemstream=(boolean)false
/GstPipeline:pipeline0/GstQueue:queue0.GstPad:src: caps = video/mpeg,
mpegversion=(int)2, systemstream=(boolean)false
/GstPipeline:pipeline0/GstMPEG-2dec:MPEG-2dec0.GstPad:sink: caps =
video/mpeg, mpegversion=(int)2, systemstream=(boolean)false
/GstPipeline:pipeline0/GstMPEG-2dec:MPEG-2dec0.GstPad:src: caps = video/x-
raw-yuv, format=(fourcc)I420, width=(int)720, height=(int)576, pixel-aspect-
ratio=(fraction)64/45, framerate=(fraction)25/1, interlaced=(boolean)true
/GstPipeline:pipeline0/GstQueue:queue1.GstPad:sink: caps = video/x-raw-yuv,
format=(fourcc)I420, width=(int)720, height=(int)576, pixel-aspect-
ratio=(fraction)64/45, framerate=(fraction)25/1, interlaced=(boolean)true
/GstPipeline:pipeline0/GstQueue:queue1.GstPad:src: caps = video/x-raw-yuv,
format=(fourcc)I420, width=(int)720, height=(int)576, pixel-aspect-
ratio=(fraction)64/45, framerate=(fraction)25/1, interlaced=(boolean)true
/GstPipeline:pipeline0/GstFFMpegCsp:ffmpegcsp0.GstPad:src: caps = video/x-
raw-yuv, format=(fourcc)I420, width=(int)720, height=(int)576, pixel-aspect-
ratio=(fraction)64/45, framerate=(fraction)25/1, interlaced=(boolean)true
/GstPipeline:pipeline0/GstFFMpegCsp:ffmpegcsp0.GstPad:sink: caps = video/x-
raw-yuv, format=(fourcc)I420, width=(int)720, height=(int)576, pixel-aspect-
ratio=(fraction)64/45, framerate=(fraction)25/1, interlaced=(boolean)true
/GstPipeline:pipeline0/GstQueue:queue2.GstPad:sink: caps = video/x-raw-yuv,
format=(fourcc)I420, width=(int)720, height=(int)576, pixel-aspect-
ratio=(fraction)64/45, framerate=(fraction)25/1, interlaced=(boolean)true
/GstPipeline:pipeline0/GstQueue:queue2.GstPad:src: caps = video/x-raw-yuv,
format=(fourcc)I420, width=(int)720, height=(int)576, pixel-aspect-
ratio=(fraction)64/45, framerate=(fraction)25/1, interlaced=(boolean)true
/GstPipeline:pipeline0/GstX264Enc:x264enc0.GstPad:src: caps = video/x-h264,
width=(int)720, height=(int)576, framerate=(fraction)25/1, pixel-aspect-
ratio=(fraction)64/45, stream-format=(string)byte-stream,
alignment=(string)au, level=(string)3, profile=(string)main
/GstPipeline:pipeline0/GstX264Enc:x264enc0.GstPad:sink: caps = video/x-raw-
yuv, format=(fourcc)I420, width=(int)720, height=(int)576, pixel-aspect-
ratio=(fraction)64/45, framerate=(fraction)25/1, interlaced=(boolean)true
/GstPipeline:pipeline0/GstQueue:queue3.GstPad:sink: caps = video/x-h264,
width=(int)720, height=(int)576, framerate=(fraction)25/1, pixel-aspect-
ratio=(fraction)64/45, stream-format=(string)byte-stream,
alignment=(string)au, level=(string)3, profile=(string)main
/GstPipeline:pipeline0/GstQueue:queue3.GstPad:src: caps = video/x-h264,
width=(int)720, height=(int)576, framerate=(fraction)25/1, pixel-aspect-
```

```
ratio=(fraction)64/45,                    stream-format=(string)byte-stream,
alignment=(string)au, level=(string)3, profile=(string)main
/GstPipeline:pipeline0/FluTsMux:flumux.GstPad:sink_64:  caps  =  video/x-h264,
width=(int)720,   height=(int)576,   framerate=(fraction)25/1,   pixel-aspect-
ratio=(fraction)64/45,                    stream-format=(string)byte-stream,
alignment=(string)au, level=(string)3, profile=(string)main
/GstPipeline:pipeline0/FluTsMux:flumux.GstPad:src:   caps   =   video/mpegts,
systemstream=(boolean)true, packetsize=(int)188
/GstPipeline:pipeline0/MpegTSParse:mpegtsparse0.GstPad:src0:      caps      =
video/mpegts, systemstream=(boolean)true, packetsize=(int)188
/GstPipeline:pipeline0/GstQueue:queue8.GstPad:sink:   caps   =   video/mpegts,
systemstream=(boolean)true, packetsize=(int)188
/GstPipeline:pipeline0/GstQueue:queue8.GstPad:src:   caps   =   video/mpegts,
systemstream=(boolean)true, packetsize=(int)188
/GstPipeline:pipeline0/GstFileSink:filesink0.GstPad:sink:        caps        =
video/mpegts, systemstream=(boolean)true, packetsize=(int)188
```

I al aturar la pipeline veurem més informació interessant:

```
EOS recibido: parando el conducto …
Execution ended after 215196501527 ns.
Estableciendo el conducto a PAUSA …
Estableciendo el conducto a PREPARADO …
/GstPipeline:pipeline0/GstFileSink:filesink0.GstPad:sink: caps = NULL
/GstPipeline:pipeline0/GstQueue:queue8.GstPad:src: caps = NULL
/GstPipeline:pipeline0/GstQueue:queue8.GstPad:sink: caps = NULL
/GstPipeline:pipeline0/MpegTSParse:mpegtsparse0.GstPad:src0: caps = NULL
/GstPipeline:pipeline0/FluTsMux:flumux.GstPad:sink_65: caps = NULL
/GstPipeline:pipeline0/FluTsMux:flumux.GstPad:sink_64: caps = NULL
/GstPipeline:pipeline0/FluTsMux:flumux.GstPad:src: caps = NULL
/GstPipeline:pipeline0/GstQueue:queue6.GstPad:src: caps = NULL
/GstPipeline:pipeline0/GstQueue:queue6.GstPad:sink: caps = NULL
/GstPipeline:pipeline0/GstQueue:queue3.GstPad:src: caps = NULL
/GstPipeline:pipeline0/GstQueue:queue3.GstPad:sink: caps = NULL
/GstPipeline:pipeline0/GstVoAacEnc:voaacenc0.GstPad:src: caps = NULL
/GstPipeline:pipeline0/GstVoAacEnc:voaacenc0.GstPad:sink: caps = NULL
/GstPipeline:pipeline0/GstX264Enc:x264enc0.GstPad:src: caps = NULL
/GstPipeline:pipeline0/GstX264Enc:x264enc0.GstPad:sink: caps = NULL
/GstPipeline:pipeline0/GstQueue:queue5.GstPad:src: caps = NULL
/GstPipeline:pipeline0/GstQueue:queue5.GstPad:sink: caps = NULL
/GstPipeline:pipeline0/GstQueue:queue2.GstPad:src: caps = NULL
/GstPipeline:pipeline0/GstQueue:queue2.GstPad:sink: caps = NULL
/GstPipeline:pipeline0/FluMp3Dec:flump3dec0.GstPad:src: caps = NULL
/GstPipeline:pipeline0/FluMp3Dec:flump3dec0.GstPad:sink: caps = NULL
/GstPipeline:pipeline0/GstFFMpegCsp:ffmpegcsp0.GstPad:src: caps = NULL
/GstPipeline:pipeline0/GstFFMpegCsp:ffmpegcsp0.GstPad:sink: caps = NULL
/GstPipeline:pipeline0/GstQueue:queue4.GstPad:src: caps = NULL
/GstPipeline:pipeline0/GstQueue:queue4.GstPad:sink: caps = NULL
/GstPipeline:pipeline0/GstQueue:queue1.GstPad:src: caps = NULL
/GstPipeline:pipeline0/GstQueue:queue1.GstPad:sink: caps = NULL
/GstPipeline:pipeline0/GstMPEG-2dec:MPEG-2dec0.GstPad:src: caps = NULL
/GstPipeline:pipeline0/GstMPEG-2dec:MPEG-2dec0.GstPad:sink: caps = NULL
/GstPipeline:pipeline0/GstQueue:queue0.GstPad:src: caps = NULL
/GstPipeline:pipeline0/GstQueue:queue0.GstPad:sink: caps = NULL
/GstPipeline:pipeline0/GstFluTSDemux:demux.GstPad:audio_00e8: caps = NULL
/GstPipeline:pipeline0/GstFluTSDemux:demux.GstPad:audio_00ec: caps = NULL
/GstPipeline:pipeline0/GstFluTSDemux:demux.GstPad:video_00e7: caps = NULL
/GstPipeline:pipeline0/GstFluTSDemux:demux.GstPad:audio_00ea: caps = NULL
/GstPipeline:pipeline0/GstFluTSDemux:demux.GstPad:private_00e9: caps = NULL
```

```
/GstPipeline:pipeline0/GstFluTSDemux:demux.GstPad:sink: caps = NULL
/GstPipeline:pipeline0/GstDvbSrc:dvbsrc0.GstPad:src: caps = NULL
Estableciendo el conducto a NULL …
Liberando la tubería…
```

Si analitzem les línies veiem la descripció de tot el procés de cada element durant l'execució de la pipeline. Verifiquem els diferents canals d'informació que contenia el programa 807 de l'element `flutsdemux`. El programa 807 porta associats tres canals d'àudio (estereo, mono i versió original), un canal de vídeo i un de dades privades. És el que vam veure a la introducció feta al capítol 1. Hem configurat correctament el codificador per a servir fluxos d'AV codificats en AVC i AAC. L'integrarem als sistemes descrits al capítol 3 juntament amb el servidor d'streaming i el reproductor.

A.4.2.1 Configuració del codificador gstreamer per a Android

El proper pas és modificar la cadena per tal de servir l'stream per a dispositius mòbils amb el sistema opratiu Android. A la taula A.4.3 es descriuen les característiques del flux de vídeo compatibles amb aquest sistema operatiu.

VÍDEO CÒDEC	RESOLUCIÓ	FPS	BIR RATE VIDEO	ÀUDIO CÒDEC	CANALS	BIT RATE ÀUDIO
H.264 Baseline Profile	174x144píxels	12	56 kbps	AAC LC	1	24 kbps
H.264 Baseline Profile	480x360píxels	30	500 kbps	AAC LC	2	128 kbps
H.264 Baseline Profile	1280x720píxels	30	2 Mbps	AAC LC	2	192 kbps
						ANDROID

Taula A.4.3 Característiques audiovisuals per Android.

Les modificacions que hem de fer ens han de permetre modificar els paràmetres de fps i resolucions en el cas del vídeo i freqüència de mostreig i número de canals ,principalment, en el cas de l'àudio. A la Fig. A.4.9 es mostra el flux de la cadena que volem dissenyar:

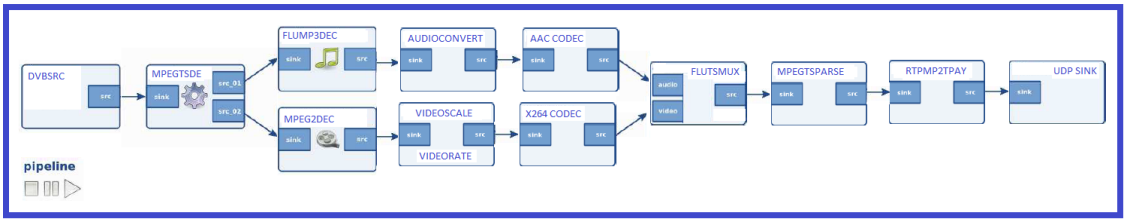


Fig A.4.9 Codificador DBV per a Android.

Hem de fer modificacions a la pipeline per adaptar els valors de resolució, fps i canals d'àudio i bit rate en tots dos casos. Ens cal introduir els elements que ho facin per A i V. En el cas del vídeo:

```
Elements: videoscale ! video/x-raw-yuv, width=176, height=144, pixel-aspect-
ratio=64/45 ! videorate ! video/x-raw-yuv, framerate=12/1 !
```

Ens permetran convertir la resolució del vídeo , molt important mantenir `pixel-aspect-ratio` de la captura, en cas contrari no veurem res més que un bloc verd constant sense àudio reproduïble. Com el que volem és servir el flux directament al servidor d'streaming i hem optimitzat les característiques pel dispositiu Android d'acord amb la taula A.4.3 i la Fig. A.4.9 la pipeline ens quedaria:

```
$ sudo gst-launch-0.10 -e -vvv dvbsrc modulation="QAM 64" trans-mode=8k
bandwidth=8 frequency=658000000 code-rate-lp=AUTO code-rate-hp=2/3 guard=4
hierarchy=0 ! flutsdemux program-number=807 name=demux. ! queue ! MPEG-2dec !
queue max-size-buffers=0 max-size-time=0 ! ffmpegcolorspace ! queue !
videoscale ! video/x-raw-yuv, width=174, height=144, pixel-aspect-ratio=64/45
! videorate ! video/x-raw-yuv, framerate=15/1 ! queue ! x264enc byte-
stream=true speed-preset=superfast bitrate=150 tune=fastdecode byte-
stream=true ! queue ! flumux. demux. ! queue  ! flump3dec ! queue ! voaacenc
! queue ! flumux. flutsmux name=flumux ! queue ! mpegtsparse ! rtpmp2tpay !
udpsink host=146.255.96.41 port=10000 sync=true
```

I activem el servidor d'streaming per rebre el flux del codificador. Per reproduir-lo amb el telèfon ens cal descarregar una aplicació. Al market d'Android en trobem diverses com Meridian Media Player o Vplayer. Un cop instal·lada introduïm la URI [8] i començarem a rebre l'stream sincronitzat i a la màxima qualitat possible.

A.4.2.2 Configuració codificador per a Iphone

Ja hem vist al cas del codificador per a v4l2 del punt A.4.1.2. A més a més, volem afegir el logo de l'EETAC a l'stream de vídeo que servirem. Ho podem fer amb l'element de gstreamer `coglogoinsert`. Modifiquem la pipeline, per no carregar massa la CPU farem servir una resolució de 320x240 píxels a 25 fps i afegim aquest element per visualitzar el logo:

```
$ sudo gst-launch-0.10 -e -vvv dvbsrc modulation="QAM 64" trans-mode=8k
bandwidth=8 frequency=658000000 code-rate-lp=AUTO code-rate-hp=2/3 guard=4
 hierarchy=0 ! flutsdemux program-number=807 name=demux demux. ! queue !
MPEG-2dec ! queue max-size-buffers=0 max-size-time=0 ! coglogoinsert
location=logo_eetac.png ! ffmpegcolorspace ! queue ! videoscale ! video/x-
raw-yuv, width=320, height=240, pixel-aspect-ratio=64/45 ! videorate !
video/x-raw-yuv, framerate=25/1 ! queue ! x264enc byte-stream=true speed-
preset=superfast bitrate=1500 tune=fastdecode byte-stream=true ! queue !
flumux. demux. ! queue  ! flump3dec ! queue ! voaacenc ! queue ! flumux.
flutsmux name=flumux ! queue ! mpegtsparse ! rtpmp2tpay ! udpsink
host=146.255.96.41 port=10000 sync=true
```

I una captura de l'stream amb VLC a la Fig. A.4.10:

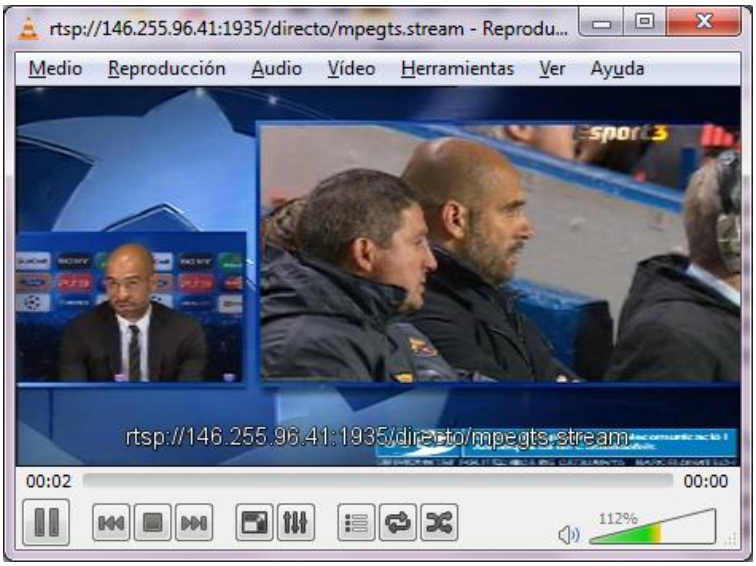


Fig A.4.10 Reproducció codificador gstreamer amb VLC/Iphone

Podem modificar les característiques del senyal de vídeo amb els valors descrits a la Taula A.4.4.

VÍDEO CÒDEC	RESOLUCIÓ	FPS	BIR RATE VIDEO	ÀUDIO CÒDEC	CANALS	BIT RATE ÀUDIO
H.264 Baseline Profile	640x480 píxels	30	1.5 Mbps	AAC LC	2	160 kbps
H.264 Baseline Profile	640x480 píxels	30	2.5 Mbps	AAC LC	2	160 kbps
						IPHONE

Taula A.4.4 característiques senyals audiovisuals Iphone

Amb l'Iphone per poder reproduir el directe obrim el navegador web Safari i introduïm la URI [22] i visualitzem l'stream. Ens sorprèn la sensació de millora de qualitat, sobretot en l'àudio es nota la diferència amb altres dispositius mòbils amb els que hem treballat.

A.4.3 Configuració codificador 3D

En aquest annex volem configurar el codificador de gstreamer per servir fluxos 3D sobre xarxes IP. L'objectiu és configurar el codificador com es mostra al diagrama de flux de la Fig. A.4.10.

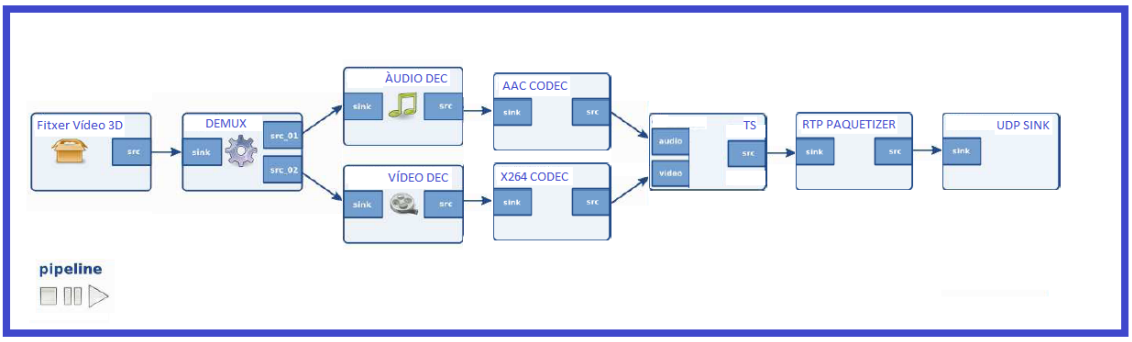


Fig. A.4.10 Pipeline codificador 3D.

Ens fem amb una font de vídeo 3D (side by side a 854x240 píxels a 30 fps) i generem la pipeline amb gstreamer. Depenent del tipus de fitxer font fem servir uns elements o altres.
Introduïm els elements `flvdemux` i `ffdemux_mov_mp4_m4a_3gp_3g2_mj2.`

```
sudo gst-launch-0.10 -e -vvv filesrc location=3D.flv ! flvdemux
name=flashdemux ! queue ! flumux. flashdemux. ! queue ! flumux. flutsmux
name=flumux ! queue ! mpegtsparse ! rtpmp2tpay ! udpsink host=146.255.96.41
port=10000 sync=true
```

Si el fitxer que tenim fos del tipus mp4 , hauríem de fer servir

```
gst-launch-0.10 -e -vvv filesrc location=lumi3d.mp4 !
ffdemux_mov_mp4_m4a_3gp_3g2_mj2 name=flashdemux ! queue ! flumux. flashdemux.
! queue ! flumux. flutsmux name=flumux ! queue ! mpegtsparse ! rtpmp2tpay !
udpsink host=146.255.96.41 port=10000 sync=true
```

Amb aquestes pipelines obtenim els fluxos estereoscòpics que reproduirem amb Bino 3D.

A.5. Elements de gstreamer

A [23] es detallen la majoria d'elements amb els que podem treballar amb gstreamer. Executant un `gst-inspect` i el nom de l'element podem saber les seves capacitats.

Videotestsrc
Xvimagesink
Videobox
Videomixer
Multifilesrc
Ffmpegcolorspace
Mad
Mpegtsdemux
Alsasink
Dvbsrc
V4l2src
Udpsrc

Mpeg2dec
Filesink
Flutsmux
Flutsdemux
Coglogoinsert
Ffdemux_mov_mp4
Rtpmp2tpay
Udpsink
Mpegtsparse
X264enc
Voaacenc

.

ANNEX B. ALTERNATIVES A GSTREAMER

B.1. Cercant alternatives a gstreamer

En aquest annex B.1 volem descriure un conjunt proves fetes amb diferents aplicacions descrites al capítol 2. L'objectiu d'aquest annex és comparar gstreamer amb altres aplicacions i treure`n conclusions. En primer lloc configurarem l'aplicació MuMuDVB per tenir una font de vídeo.

Segons la descripció feta al punt 2.9, aquesta aplicació rep un TS amb diferents canals (d'un receptor DVB-T o DVB-S) i els multiplexa en diferents sortides a xarxa, per cada portadora (vídeo, àudio i dades) assigna una adreça IP multicast, diferent per a cada programa. Ja tenim el receptor DVB-T funcionant. Abans de fer-ho servir hem de generar un arxiu de configuració ,al directori de l'aplicació, de configuració prèvia. Un exemple d'arxiu de configuració:

```
autoconfiguration=2
freq= 794000
```

El valor de freqüència en kHz. Volem fer difusió del senyal de TVC, coneixem f de les dades extretes de w_scan. Executem:

```
$ sudo mumudvb -d -c ./config
MuMuDVB Version 1.6
Latest version available from http://mumudvb.braice.net/
Full autoconfiguration, we activate SAP announces. if you want to desactivate
them see the README. The autoconfiguration auto update is enabled. If you
want to disable it put "autoconf_pid_update=0" in your config file.
/var/run/mumudvb/mumudvb_generated_conf_card0: No such file or directory
WARNING : Can't create /var/run/mumudvb/chaines_diffusees_carte0: No such
file or directory
WARNING : Can't create /var/run/mumudvb/chaines_non_diffusees_carte0: No such
file or directory
Streaming. Freq 794000000
Using DVB card "Afatech AF9013 DVB-T"
tuning DVB-T to 794000000 Hz, Bandwidth: 8
FE_STATUS:
FE_STATUS:
     FE_HAS_SIGNAL : found something above the noise level
     FE_HAS_CARRIER : found a DVB signal
     FE_HAS_VITERBI : FEC is stable
     FE_HAS_SYNC : found sync bytes
     FE_HAS_LOCK : everything's working...
Event:  Frequency: 794000000
Bit error rate: 0
Signal strength: 50933
SNR: 120
FE_STATUS:
     FE_HAS_SIGNAL : found something above the noise level
     FE_HAS_CARRIER : found a DVB signal
     FE_HAS_VITERBI : FEC is stable
     FE_HAS_SYNC : found sync bytes
     FE_HAS_LOCK : everything's working...
Card 0 tuned
Autoconfiguration Start
```

```
Autoconfiguration done
Diffusion 4 channels
Channel number :   0, ip : 239.100.0.0:1234, name : "TV3"
Channel number :   1, ip : 239.100.0.1:1234, name : "33"
Channel number :   2, ip : 239.100.0.2:1234, name : "3/24"
Channel number :   3, ip : 239.100.0.3:1234, name : "Super3/3XL"
/var/run/mumudvb/mumudvb_generated_conf_card0: No such file or directory
```

Fem un escaneig amb wireshark a veure que esta passant a la xarxa:

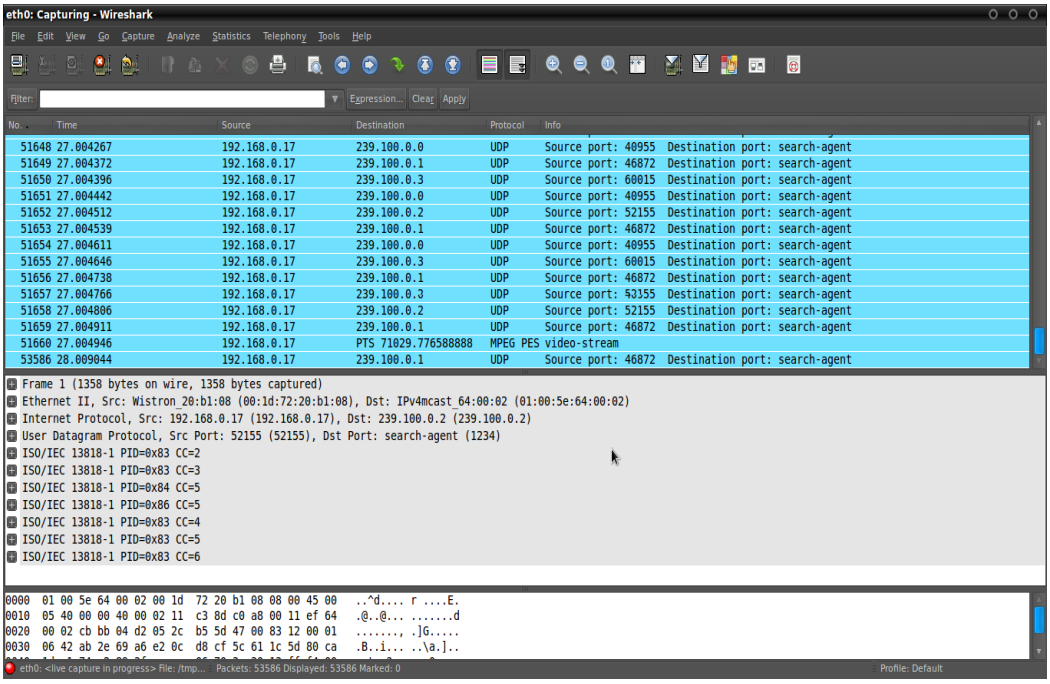


Fig B.1.1 Captura wireshark mumuDVB

Estem emetent com esperàvem, els 4 programes de la f seleccionada. Provem de reproduir a l'altre PC dins de la mateixa LAN amb VLC a l'URI udp://@239.100.0.0:1234 a la figura B.1.2 en veiem un exemple. Verifiquem que rebem perfectament el senyal d'àudio i vídeo. El que volem és trans codificar el flux d'àudio i vídeo per aconseguir un flux o fitxer codificat a AVC i AAC. Aprofitant el flux de dades que ens genera a la xarxa l'aplicació MuMuDVB volem aconseguir dos objectius.

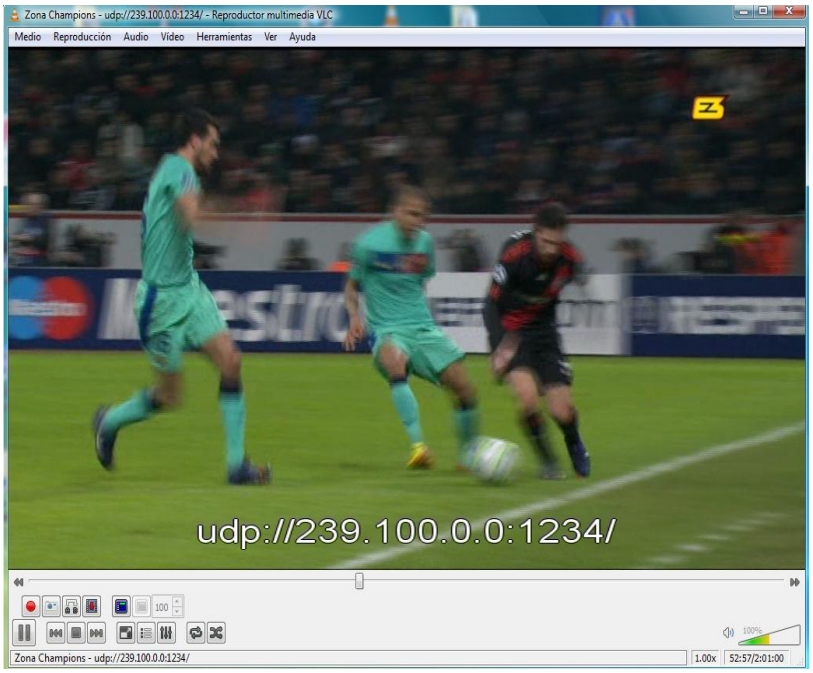


Fig B.1.2 Reproducció multicast mumudvb amb VLC

Gravar a fitxer el programa que vulguem de la graella de canals TDT i escalar, comprimir i servir aquest stream a un servidor a Internet amb l'objectiu de que sigui reproduïble per un dispositiu mòbil (un telèfon android amb un reproductor RTSP estàndard). Fem una prova. Executem l'aplicació mumudvb per sintonitzar la freqüència d'alguns canals de TV3, concretament a 658 MHz. Modifiquem l'arxiu de configuració i executem:

```
$ sudo mumudvb -d -c ./config
[...]

Event:  Frequency: 658000000

Bit error rate: 0
Signal strength: 57823
SNR: 120
Autoconfiguration Start
Autoconfiguration done

Diffusion 4 channels
Channel number :   0, ip : 239.100.0.0:1234, name : "Esport3"
Channel number :   1, ip : 239.100.0.1:1234, name : "TV3HD"
Channel number :   2, ip : 239.100.0.2:1234, name : "Canal 9"
Channel number :   3, ip : 239.100.0.3:1234, name : "IB3"
```

Hem comprovat que tenim àudio i vídeo a les diferents URI's multicast que tenim a la LAN. Ara amb una senzilla comanda de gstreamer, guardem a fitxer les dades, generarem un arxiu .TS :

```
$ gst-launch-0.10 udpsrc uri=udp://239.100.0.0:1234 ! filesink
location=captura00.ts

Estableciendo el conducto a PAUSA …
El conducto está vivo y no necesita PREPARARSE …
Estableciendo el conducto a REPRODUCIENDO …
New clock: GstSystemClock
^CCaught interrupt -- handling interrupt.
Interrumpir: parando el conducto …
Execution ended after 16223424707 ns.
Estableciendo el conducto a PAUSA …
Estableciando el conducto a PREPARADO …
Estableciendo el conducto a NULL …
Liberando la tubería…
```

En aquest exemple tenim només dos elements a la pipeline o conducte, una font d'àudio, vídeo i dades, en aquest cas la URI generada per mumudvb i l'element que consumeix el flux és la gravació a fitxer, fem un cop d'ull a l'arxiu que s'ha generat a la Fig. B.1.3:

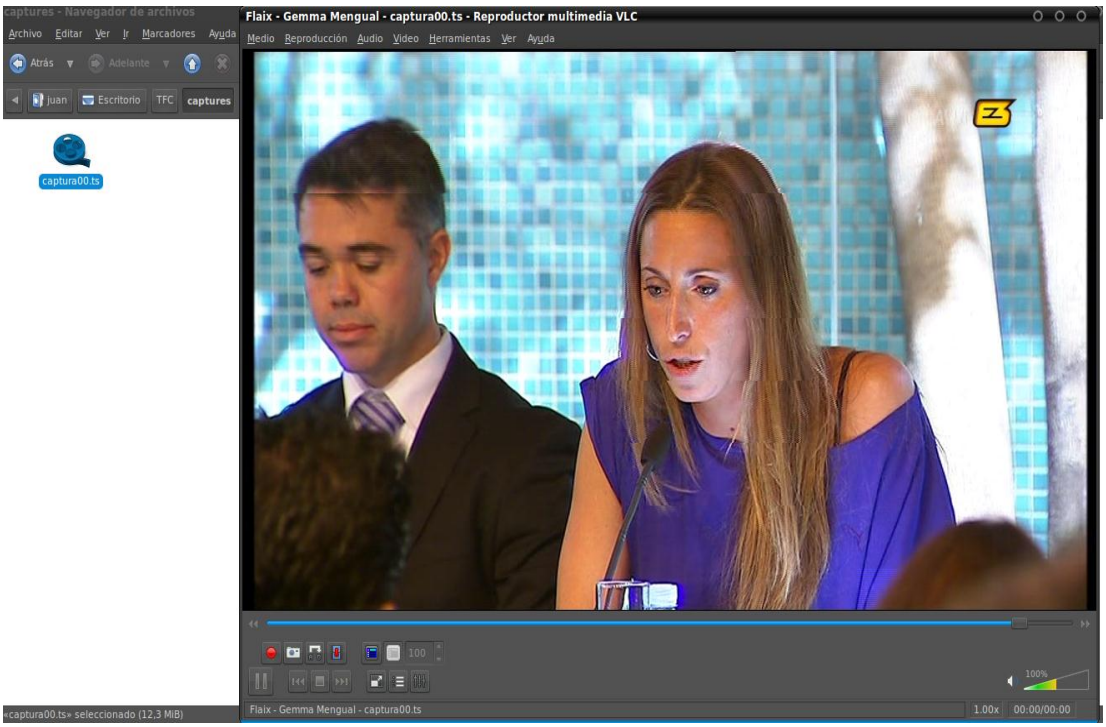


Fig B.1.3 Captura d'un TS sense des multiplexar

Fixem-nos en la informació que ens dona VLC del TS a la Fig. B.1.4. De la captura que hem pres, hem bolcat tota la informació que contenia el TS i la info dels programes de la PMT. Era d'esperar la informació obtinguda amb VLC, ja que abans de guardar a fitxer no hem des multiplexat el vídeo i àudio que ens interessava del TS complet. Per això hem de modificar la pipeline i afegir els elements inter mitjos que des multiplexin el senyal d'àudio i vídeo. Fixem-nos també en el tipus de codificació del vídeo, és mpgv (MPEG-2 en aquest cas) i de l'àudio, és mpga.

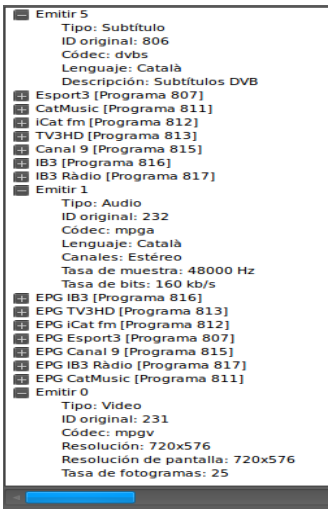


Fig B.1.4 Info multimèdia TS VLC

El que volem fer és desmultiplexar el TS, codificar àudio i vídeo amb valors compatibles amb una pantalla de mòbil, multiplexar el vídeo amb un dels fluxos d'àudio i guardar a fitxer. Per continuar amb el procés hem de des multiplexar les captures. Farem servir l'aplicació project-x que ja hem descrit. Provem de carregar una de les captures de TS que hem fet amb gstreamer. Tenim l'arxiu final00.ts de captures anteriors. És una captura de 30 segons que ocupa 28,7 Mbytes, fixem-nos en les propietats multimèdia de l'arxiu resolució 720x256 píxels, 25 fps, MPEG-2 vídeo, MPEG-1 àudio layer 2. Obrim la captura amb project-x, com es mostra a la Fig. B.1.5. Passem a des multiplexar el senyal, extraurem tots els canals de vídeo i àudio del TS. Fem un cop d'ull als logs que ens donaran informació, alguna ja coneguda de la informació de les taules PSI i de dvbsnoop:

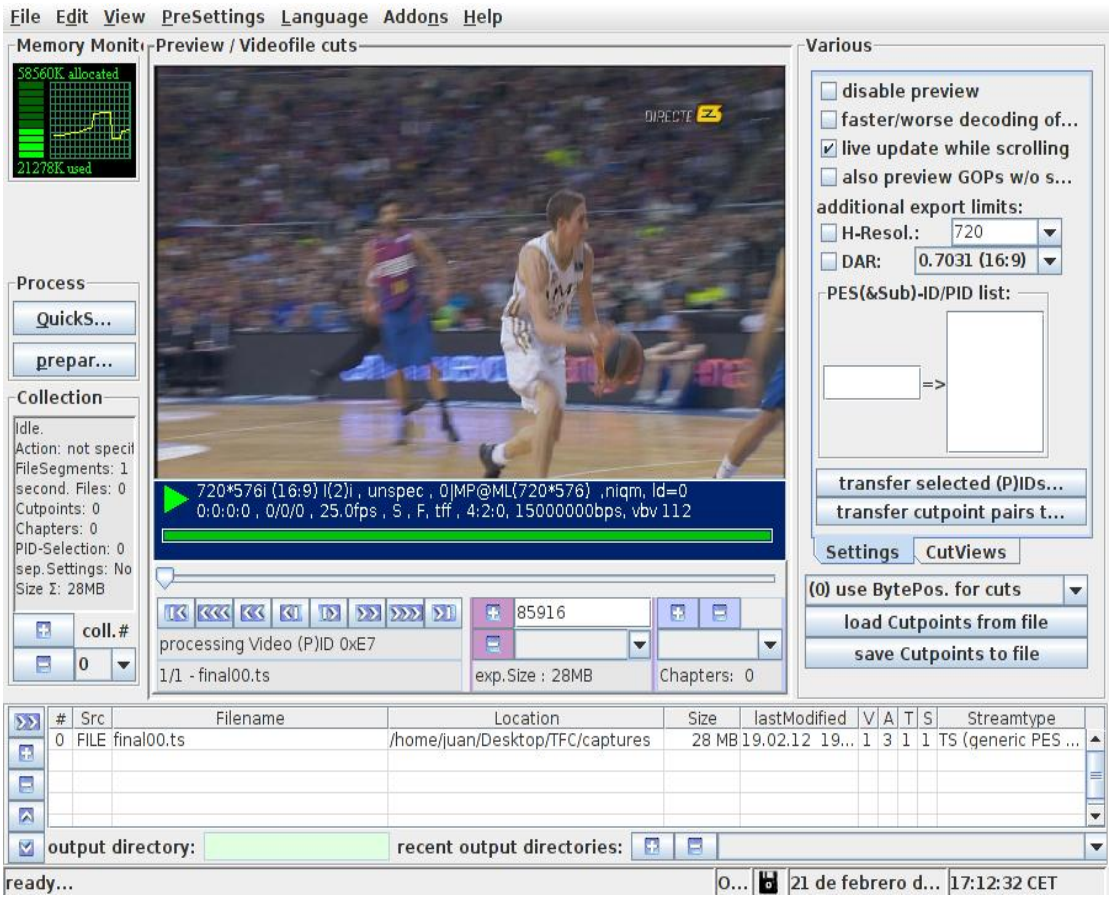


Fig B.1.5 Captura TS amb projectx multiplexat

```
<<< session infos >>>
ProjectX 0.90.4.00 (30.03.2006)

-> write output files to: '/home/juan/Desktop/TFC/captures'
-> Input File 0: '/home/juan/Desktop/TFC/captures/final00.ts' (30.078.496
bytes)
-> Filetype is TS (generic PES Container)
-> demux
-> Service ID 0x327
-> PMT 0xE6 refers to these usable streams:
Video:
PID: 0xE7
Audio:
PID: 0xE8(cat)
PID: 0xEA(v.o)
PID: 0xEC(a.d)
Teletext:
PID: 0xE9(cat_i100 )
Subpict.:
PID: 0x326(cat_0x10_p1_a1 )
ok> PID 0xE8 has PES-ID 0xC0 (MPEG Audio) (10904 #59)
!> PID 0x1 (CAT) (22184 #119) -> ignored
!> PID 0x12 (EIT) (24252 #130) -> ignored
ok> PID 0xE9 has PES-ID 0xBD (private stream 1) (TTX)  (30644 #164)
!> PID 0x0 (PAT) (33652 #180) -> ignored
ok> PID 0xE7 has PES-ID 0xE0 (MPEG Video) (37976 #203)
!> PID 0xE6 (PMT) (45872 #245) -> ignored
```

```
ok> PID 0xEA has PES-ID 0xC2 (MPEG Audio) (60912 #325)
!> PID 0x14 (TDS) (85540 #456) -> ignored
ok> PID 0xEC has PES-ID 0xC3 (MPEG Audio) (87044 #464)
!> PID 0x10 (NIT) (90240 #481) -> ignored
!> PID 0x11 (SDT) (207176 #1103) -> ignored
-> video basics: 720*576 @ 25fps @ 0.7031 (16:9) @ 15000000bps, vbvBuffer 112
-> Video: fr/ ct/ 1p/ cg/ og/ dg -> 821/ 2/ 0/ 40/ 0/ 1
-> Video length: 821 frames @ 00:00:32.840
-> GOP summary: min. 18, max. 54 fields; contains interlaced frames
-> avg. nom. bitrate 6327195bps (min/max: 4198400/8060800)
-> set first sequenceheader bitrate to 8060800bps
---> new File: /home/juan/Desktop/TFC/captures/final00.m2v
--> MPEG Audio (0xC0) on PID 0xE8
-> check CRC of AC-3 / MPEG-Audio L1,2
-> delete CRC in MPEG-Audio Layer1,2
-> add frames
Audio PTS: first packet 10:08:06.963, last packet 10:08:40.875
Video PTS: start 1.GOP 10:08:07.331, end last GOP 10:08:41.091
-> adjusting audio at video-timeline
-> src_audio: MPEG-1, Layer2, 48000Hz, jstereo, 160kbps, CRC @ 00:00:00.000
!> missing syncword @  609121, @ 00:00:30.048
!> found syncword @ 609720
audio frames: wri/pre/skip/ins/add 1368/0/0/0/0 @ 00:00:32.832 done...
---> new File: '/home/juan/Desktop/TFC/captures/final00.mp2'
--> Teletext on PID 0xE9 (SubID 0x10)
--> MPEG Audio (0xC2) on PID 0xEA
-> check CRC of AC-3 / MPEG-Audio L1,2
-> delete CRC in MPEG-Audio Layer1,2
-> add frames

summary of created media files:
.Video (m2v):      821 Frames   00:00:32.840
      '/home/juan/Desktop/TFC/captures/final00.m2v'
Audio 0 (mp2):     1368 Frames  00:00:32.832 0/0/0/0
      '/home/juan/Desktop/TFC/captures/final00.mp2'
Audio 1 (mp2):     1368 Frames  00:00:32.832 0/0/0/0
      '/home/juan/Desktop/TFC/captures/final00[1].mp2'
Audio 2 (mp2):     1368 Frames  00:00:32.832 0/0/0/0
      '/home/juan/Desktop/TFC/captures/final00[2].mp2'
```

Comprovem que ens ha generat un total de 4 arxius de sortida, 1 de vídeo en format m2v i 3 d'audio en format mp2 (MPEG-1 Layer 2).

Si els reproduïm amb VLC veurem les característiques multimedia de cada fitxer:

- final00.m2v Tipus de dades Vídeo 725x675 píxels fps 25 Còdec mpgv
- final00.mp2 Tipus de dades Audio Còdec mpga F de mostreig: 48 kHz 160 kbps ESTEREO.
- final00[1].mp2 Tipus de dades Audio Còdec mpga F de mostreig: 48 kHz 96 kbps MONO.
- final00[2].mp2 Tipus de dades Audio Còdec mpga F de mostreig: 48 kHz 96 kbps MONO.

L'arxiu final00.mp2 és el que conté els dos canals d'audio en estèreo. Ara que ja tenim les peces multimèdia cal codificar el senyal de vídeo a un format compatible i generar un arxiu contenidor d'mp4 amb les senyals multiplexades. En aquest cas, volem codificar el vídeo per a dispositius mòbils, a la taula A.4.3 es mostra que el còdec de vídeo a de ser H.264, amb una resolució de 176x144 píxels. Hem de disminuir els fps a un valor compatible.

Codifiquem el vídeo amb una comanda de ffmpeg:

```
$ ffmpeg -i final00.m2v -s 176x144 -vcodec MPEG-4 -r 13 -ab 32000 -aspect 3:2
vid-final00.mp4

  libavutil     49.15. 0 / 49.15. 0
  libavcodec    52.20. 1 / 52.20. 1
  libavformat   52.31. 0 / 52.31. 0
  libavdevice   52. 1. 0 / 52. 1. 0
  libavfilter    0. 4. 0 /  0. 4. 0
  libswscale     0. 7. 1 /  0. 7. 1
  libpostproc   51. 2. 0 / 51. 2. 0

Seems stream 0 codec frame rate differs from container frame rate: 50.00
(50/1) -> 25.00 (25/1)
Input #0, mpegvideo, from 'final00.m2v':
  Duration: 00:00:30.77, bitrate: 6752 kb/s
    Stream #0.0: Video: MPEG-2video, yuv420p, 720x576 [PAR 64:45 DAR 16:9],
6752 kb/s, 25 tbr, 1200k tbn, 50 tbc
Output #0, mp4, to 'vid-final00.mp4':
    Stream #0.0: Video: MPEG-4, yuv420p, 176x144 [PAR 27:22 DAR 3:2], q=2-31,
200 kb/s, 90k tbn, 13 tbc
Stream mapping:
  Stream #0.0 -> #0.0
Press [q] to stop encoding
frame=  428 fps= 58 q=7.9 Lsize=     903kB time=32.92 bitrate= 224.7kbits/s
video:899kB audio:0kB global headers:0kB muxing overhead 0.470409%
```

Ara que ja hem trans codificat el vídeo l'hem de multiplexar en un contenidor mp4. Fem servir l'aplicació avidemux, com es mostra a la figura B.1.6

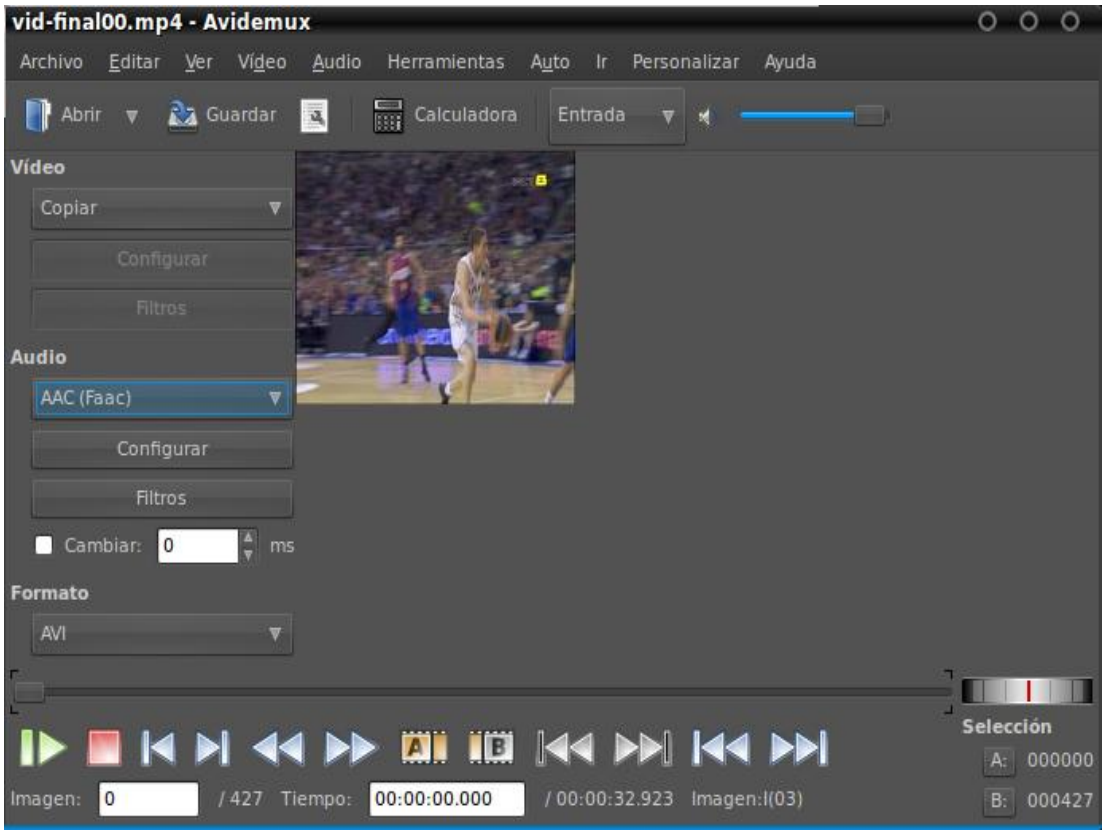


Fig B.1.6 Multiplexació amb avidemux

Les característiques del senyal d'àudio han de ser: codificació mp4a, Còdec AAC, estèreo a 160 kbps i freqüència de mostreig 48 kHz, per multiplexar el senyal juntament amb el vídeo. Si ho hem fet bé provem de reproduir l'arxiu i veure la informació multimèdia amb VLC a la Fig.B.1.7.

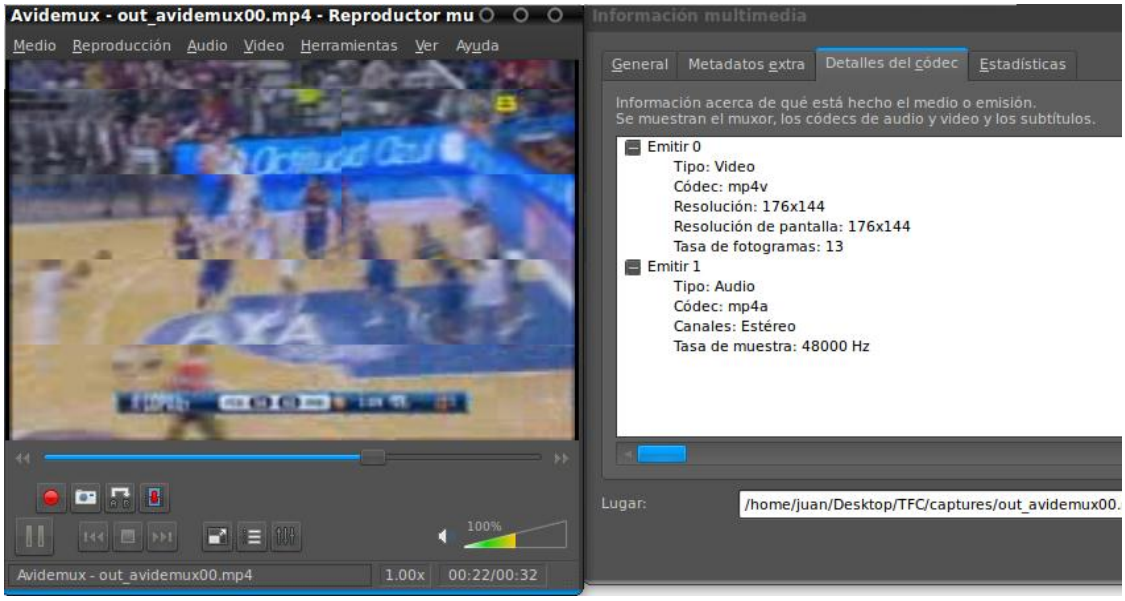


Fig B.1.7 Sortida fitxer mp4 per Android amb avidemux

Provem de reproduir-lo amb un telèfon mòbil android, amb resultat satisfactori. Finalment hem pogut fer la tasca que volíem però no ho hem pogut fer en temps real. A més a més hem hagut de fer servir més de 4 aplicacions diferents per tractar els diferents processos. Hem demostrat que gstreamer és una aplicació molt potent ja que integra en un únic framework totes les eines necessàries per poder treballar amb estructures de dades i fluxos multimèdia en temps real i en facilita la transmissió a Internet.

ANNEX C. GENERACIÓ DE VÍDEO 3D

C.1. Generació fluxos vídeo 3D amb ffmpeg

Per generar un vídeo 3D necessitem dos fluxos de vídeo. Cada vídeo serà la percepció de la visió de cada ull. L'objectiu en aquest annex és generar un vídeo amb el doble de l'ample de la resolució horitzontal del vídeo original. Cada nou frame serà una composició de vídeo 3D del tipus side by side esquerra-dreta. Al reproduir-lo s'expandeix cada imatge de forma alterna de forma que cada ull rebi el flux de cada vista.

Hem de generar un vídeo amb les dues perspectives 2D per obtenir sensació de profunditat. Descarreguem uns vídeos de mostra amb 2 fluxos de vídeos amb perspectives esquerra-dreta. A [24] podem descarregar-ne alguns.

Descarreguem les mostres i les anomenen test01-left.avi i test01-right.avi. Fixem-nos en els fps a la Fig.C.1.1.

Fig. C.1.1 Generació vídeo 3D side-by-side

Creem un nou directori i hi copiem els vídeos, els anomenem frames-left i frames-done. A consola executem:

```
$ ffmpeg -r 20 -i test01-left.avi -r 20 -s 1024x1024 frames-left/%08d.png
```

Amb aquesta ordre extrèiem tots els frames del vídeo de la vista esquerra. Fem el mateix amb l'altre vista:

```
$ ffmpeg -r 20 -i test01-right.avi -r 20 -s 1024x1024 frames-right/%08d.png
```

Com hem vist al directori frames-left s'han extret tots els frames de test01-left.avi. El mateix amb l'altre directori. Programem un petit script en bash que generi un únic frame composició de totes dues vistes.

```
$ sudo gedit mezclai.sh

#!/bin/bash
a=1
 while [ $a -lt 2416 ]
 do
    b=`printf "%08d" $a`
    montage frames-left/$b.png   frames-right/$b.png -geometry 1024x1024+0+0
frames-done/$b.jpg
    let a++
 done
```

El fem executable i executem:

```
$ sudo chmod 755 mezclai.sh
$ sudo ./mezclai.sh
```

A la figura C.1.2 veiem els frames del directori frames-done on tenim la unió de les dues vistes.

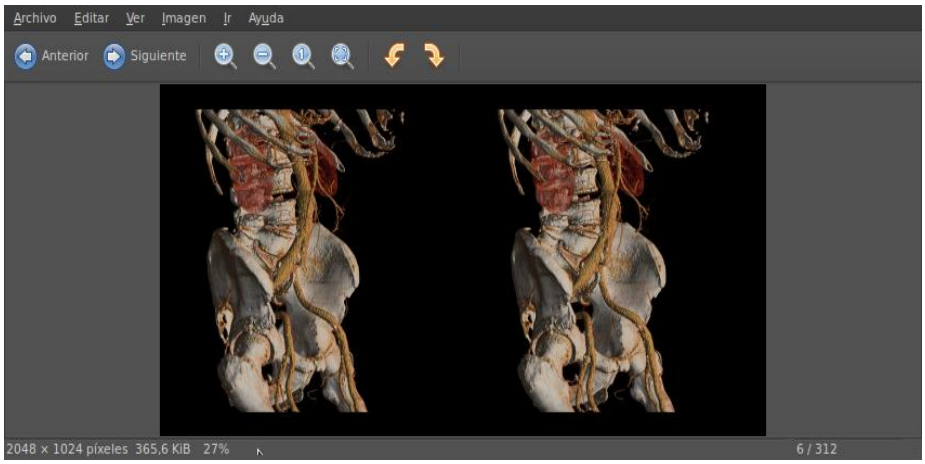


Fig. C.1.2 Captura 2 vídeo 3D generat amb ffmpeg

Només falta ordenar els frames per generar una seqüència de vídeo. Programem un altre script:

```
$ sudo gedit conviertei.sh

#!/bin/bash
for file in *.jpg
  do
     ffmpeg -i $file -acodec copy -vcodec copy -f avi ${file%.*}.avi
  done
```

el fem executable i unim tots els contenidors .avi que s'han generat:

```
$ sudo cat *.avi > output.avi
```

L'última comanda per generar el vídeo 3D:

```
$  mencoder  -forceidx  -ovc  copy  -oac  copy  output.avi  -o  final.avi
```

ANNEX D. CONFIGURACIÓ SERVIDOR D'STREAMING WOWZA

L'objectiu d'aquest annex és instal·lar i configurar el servidor d'streaming per poder servir fluxos en directe i per a vídeo sota demanda. Al punt D.1 tractarem la instal·lació de l'aplicació en el PC per fer proves en local i desenvoluparem les nostres primeres aplicacions. Al punt D.2 continuarem amb la instal·lació i configuració en remot d'un servidor Linux al núvol. Desenvoluparem les aplicacions per a gestió de directes i configurarem el servidor per rebre els fluxos del codificador.

D.1. Configuració Windows servidor streaming Wowza

En aquest annex descriurem una guia bàsica d'instal·lació i configuració del servidor d'streaming wowza. Tota la informació està extreta dels manuals d'instal·lació i configuració de Wowza. Es poden trobar a [17].

D.1.1. Instal·lació i posada en marxa en Windows

Com a prerequisit ens cal tenir instal·lada l'última versió de Java. La podem descarregar de [25]. Wowza és una aplicació disponible per a diferents sistemes operatius. Destaquem que Wowza no és una eina de codi obert, existeixen diferents tipus de llicències. En el nostre cas hem contractat una llicència d'ús diari. Un cop instal·lat l'executable podem activar el servidor amb Wowza Startup i Shutdown per aturar-lo.

A l'executar l'startup s'inicia una consola de Java i el servidor comença a mostrar missatges. El servidor deixa traces, ens mostra dades dels diferents processos que s'estan executant i ens informa si es connecta un client al servidor. Per assegurar-nos de que el servidor ja està funcionant podem accedir a l'host local pel port 1935.

Se'ns mostrarà la informació de la versió de Wowza que està corrent al nostre servidor. En aquesta exemple d'instal·lació esta corrent en local. En cas de que volguéssim accedir-hi remotament ens caldria introduir l'adreça IP del servidor i assegurar-nos de que és accessible des de la nostre ubicació.

D.1.2. Configuració de les aplicacions

A wowza corren diferents tipus d'aplicacions que donen servei a les necessitats de reproducció de cada client i als diferents continguts. El seu funcionament està basat en un conjunt de fitxers XML que contenen els paràmetres de configuració del servidor. Els fitxers de configuració els podem trobar a:

```
\Wowza Media Server\conf
```

Cada nova aplicació necessita un fitxer de configuració que s'ha d'incloure a dins d'aquesta carpeta. Per crear la nostra primera aplicació ens movem al directori:

```
\Wowza Media Server \applications
```

Creem un nou directori amb el mateix nom que l'aplicació:

```
\Wowza Media Server\applications\HelloWorld
```

Un cop creada l'aplicació l'hem de configurar, al directori:

```
\Wowza Media Server\conf
```

Creem un directori amb el nom de la nova aplicació:

```
\Wowza Media Server 2.2.4\conf\HelloWorld
```

En aquest directori afegim el fitxer xml de configuració Application.xml que trobem al directori:

```
\Wowza Media Server 2.2.4\conf\
```

I l'afegim al directori:

```
\Wowza Media Server 2.2.4\conf\HelloWorld
```

Amb aquesta configuració hem creat la nostra primera aplicació per a Wowza. Fem un cop d'ull al fitxer Application.xml i veurem els tipus d'streams amb els que pot treballar wowza i els tipus de clients que s'hi podran connectar.

D.1.3 Instal·lació de les aplicacions

Wowza media server inclou un conjunt d'aplicacions de demostració ja configurades. Per a instal·lar-les en Windows ens movem al directori:

```
\Wowza Media Server\examples
```

Executem l'arxiu installall.bat, ara al directori :

```
\Wowza Media Server\applications
```

Han aparegut noves aplicacions i nous arxius de configuració :

```
Wowza Media Server\conf\vod
```

Al directori anterior veiem una aplicació força útil. L'aplicació VoD ens permet reproduir un flux d'un fitxer que resideixi al nostre servidor. L'arxiu de configuració necessari per fer servir amb aquesta aplicació el podem descarregar de [28]

Si volem fer stream d'una peça de vídeo que tenim al directori:

```
\Wowza Media Server\content\sample.mp4
```

Iniciem el servidor amb wowza startup. Ja tenim el servidor funcionant, ara ens cal un reproductor que es connecti i ens permeti visualitzar el flux. Farem servir VLC media Player, fixem-nos en la URI que li haurem de passar al reproductor [28]. En primer lloc fem servir el protocol RTSP, la IP del servidor, l'aplicació que fem servir, el nom de l'stream i el contingut que volem reproduir. En aquest cas l'stream que volem reproduir es mp4:myvideos/sample.mp4. El nom del flux varia en funció del format de l'arxiu d'origen, a [29] se'ns detalla els diferents noms que poden tenir els fluxos. Amb VLC introduïm la URI i veurem una imatge com la de la Fig. D.1.1.

Fig. D.1.1 Sortida RTSP wowza amb VLC

Si volguéssim afegir més vídeos només hauríem de copiar el fitxer al directori del servidor `\content` i fer el mateix procés.

D.2. Configuració directes Wowza per a Linux

En aquest punt volem descriure com configurar el servidor d'streaming per rebre els fluxos del diferents codificadors. L'hem de configurar per rebre un flux TS enviat per RTP sobre UDP. Configurarem clients per a dispositius mòbils i per a PC.

D.2.1. Desenvolupament aplicació de directes

Per fer una nova aplicació al directori de instal·lació, primer creem el directori `/applications/directo`. Ens connectem al servidor per SSH o SFTP i creem el directori. Fem el mateix al directori de configuració /conf/directe hem d'afegir l'arxiu de configuració Application.xml que podem descarregar de

http://www.wowzamedia.com/downloads/tutorials/live/Application.xml

Hem de modificar uns quants paràmetres d'aquest arxiu XML de configuració.

El tipus d'stream és un directe:

```
<StreamType>live</StreamType>
```

Configurem la propietat HTTPStreamers per que l'accés sigui compatible per clients que accedeixen amb iOS, Silverlight i per HTTP Flash. Definim:

```
<LiveStreamPacketizers>cupertinostreamingpacketizer,smoothstreamingpacketizer
, sanjosestreamingpacketizer</LiveStreamPacketizers>
```

- cupertinostreamingpacketizer és el paquetitzador per iOS (Iphone).
- smoothstreamingpacketizer és per a clients Silverlight.
- sanjosestreamingpacketizer és per a clients HTTP Flash amb reproductor embegut.

Fem el mateix per la propietat `Streams/LivePacketizers`

```
<LiveStreamPacketizers>cupertinostreamingpacketizer,
smoothstreamingpacketizer, sanjosestreamingpacketizer</LiveStreamPacketizers>
```

I fixem que els clients no s'hagin d'identificar fixant el valor a none de la propietat `RTP/Authentication/PlayMethod:`

```
<PlayMethod>none</PlayMethod>
```

Guardem les dades i verifiquem que totes les propietats de l'stream estan configurades correctament. Wowza escolta per defecte pel port 10000 haurem de configurar el codificador per que serveixi l'stream a la IP del servidor per aquest port. Hem de crear un arxiu de configuració en mode text anomenat tipus mpegts.stream al directori `/content/` :

```
udp://0.0.0.0:10000
```

Podríem modificar aquesta adreça per una de multicast si fos necessari. Al directori `/conf/admin.password` modifiquem els valor de `myuser` i `mypassword` per identificar-nos i poder accedir al document streammanager de Wowza.

La configuració del servidor d'streaming està enllestida, només ens cal activar la recepció de l'stream des de la URL [27]. El procés wowza ha d'estar actiu, ens connectem al server per SSH i l'activem. Per no tenir la sessió ocupada l'executem com a dimoni afegint &:

```
root@xtreaming:/usr/local/WowzaMediaServer-3.0.3/bin# sudo ./startup.sh &
[1] 5362
root@xtreaming:/usr/local/WowzaMediaServer-3.0.3/bin#    Configure    logging:
file:///usr/local/WowzaMediaServer/conf/log4j.properties
INFO server server-start Wowza Media Server 3 Daily Edition 3.0.3 build882 -
```

```
INFO server comment - Server License Key: SVRD3-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
INFO server comment - Maximum Connections: Unlimited
INFO server comment - Transcoder Streams Available: Unlimited
INFO server comment - Transcoder Watermark: No
INFO server comment - nDVR Available: Yes
INFO server comment - DRM Available: Yes
INFO server comment - Hardware Available Processors: 2
INFO server comment - Hardware Physical Memory: 597MB/997MB
INFO server comment - Hardware Swap Space: 1023MB/1023MB
INFO server comment - Max File Descriptor Count: 20000
INFO server comment - Open File Descriptor Count: 50
INFO server comment - OS Name: Linux
INFO server comment - OS Version: 2.6.32-24-server
INFO server comment - OS Architecture: amd64
INFO server comment - Java Name: OpenJDK 64-Bit Server VM
INFO server comment - Java Vendor: Sun Microsystems Inc.
INFO server comment - Java Version: 1.6.0_20
INFO server comment - Java VM Version: 19.0-b09
INFO server comment - Java Spec Version: 1.6
INFO server comment - Java Home: /usr/lib/jvm/java-6-openjdk/jre
INFO server comment - Java Max Heap Size: 1160MB
INFO server comment - Java Architecture: 64
INFO server comment - Java Encoding[file.encoding]: UTF-8
INFO server comment - Java Args[0]: -Xmx1200M
INFO server comment - Java Args[1]: -Djava.net.preferIPv4Stack=true
INFO server comment - Java Args[2]: -Dcom.sun.management.jmxremote=true
INFO server comment - Java Args[3]: -Dcom.wowza.wms.runmode=standalone
INFO server comment - Java Args[4]: -Dcom.wowza.wms.native.base=linux
INFO server comment - Java Args[5]: -
Dcom.wowza.wms.AppHome=/usr/local/WowzaMediaServer
INFO server comment - Java Args[6]: -Dcom.wowza.wms.ConfigURL=
INFO server comment - Java Args[7]: -
Dcom.wowza.wms.ConfigHome=/usr/local/WowzaMediaServer
INFO server comment - Java GC[0]: Copy
INFO server comment - Java GC[1]: MarkSweepCompact
INFO server comment - Server runmode: standalone
INFO server comment - Server native.platform: linux
INFO server comment - Server threads[h/t]: 10/10
INFO server comment - CMDInterface now listening: [any]:8083
INFO vhost vhost-start _defaultVHost_ -
INFO server comment - _defaultVHost_ threads[h/t]:120/80
home:/usr/local/WowzaMediaServer
INFO vhost comment _defaultVHost_ Bind attempt ([any]:1935:4)
INFO vhost comment _defaultVHost_ Bind successful ([any]:1935)
INFO vhost comment _defaultVHost_ Bind attempt ([any]:8086:1)
INFO vhost comment _defaultVHost_ Bind successful ([any]:8086)
INFO server comment - Wowza Media Server is started!
```

Ens connectem a l'streammanager, tenim la nova aplicació directe, activem l'stream i veurem una finestra com la de la Fig. D.1.2. Només volem reenviar l'stream sense emmagatzemar-lo llavors seleccionem RTP i indiquem el nom de l'stream que volem reproduir mpegts.stream. Abans hem de tenir activat el codificador per que la negociació del socket es faci correctament i el servidor comenci a rebre l'stream de l'encoder.

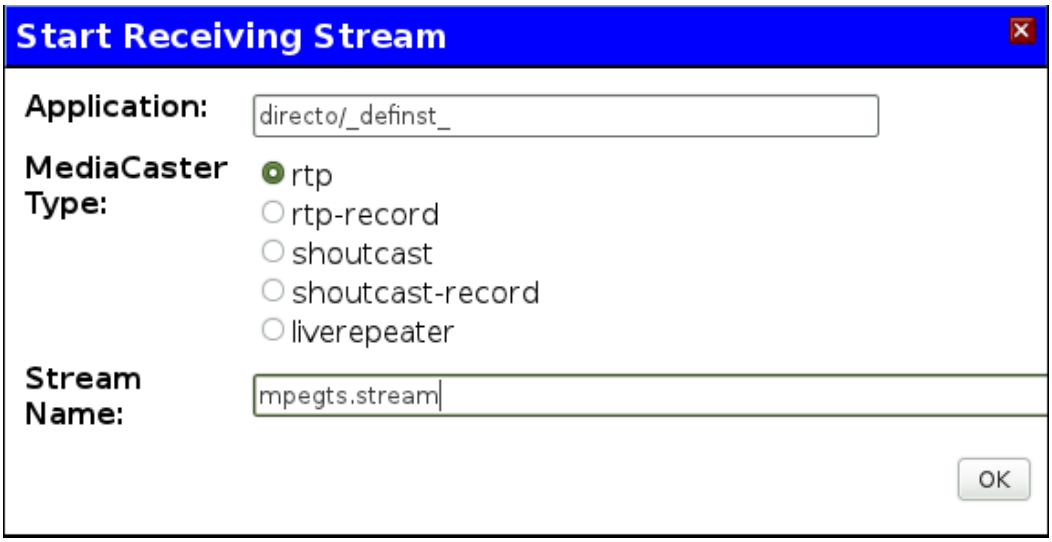


Fig. D.1.2 Stream manager de wowza, activació stream

Si tot ha anat bé el servidor ha de mostrar la informació del inici de sessió:

```
INFO server comment - RTPMediaCaster.create[1928786358]
INFO server comment - RTPMediaCaster.init[1928786358]
INFO server comment -
RTPMediaCaster.Reconnector[1928786358:directo/_definst_:mpegts.stream]:
start: 1
INFO server comment - HTTPStreamManager.onHTTPRequest: Publish stream
successfully started [directo/_definst_]: flv:mpegts.stream
INFO server comment -
RTPSessionDescriptionDataProviderBasic.getStreamInfo[directo/_definst_]: URI:
udp://0.0.0.0:10000
INFO stream create - -
INFO server comment - RTPUDPTransport.bind[directo/_definst_]: /0.0.0.0:10000
INFO stream publish mpegts.stream -
INFO server comment -
RTPMediaCaster.Reconnector[1928786358:directo/_definst_:mpegts.stream]: done:
1
INFO server comment - RTPDePacketizerMPEGTS.handleRTPPacket: MPEG-TS over RTP
INFO server comment - RTPDePacketizerMPEGTS.handleRTPPacket: IMPORT:
programID[prg:0x1,filter:none]
INFO server comment - RTPDePacketizerMPEGTS.handleRTPPacket: IMPORT:
audioPID[prg:0x1,pid:0x41,filter:none]: streamType:AAC:15
audioLanguage:uknown
INFO server comment - RTPDePacketizerMPEGTS.handleRTPPacket: IMPORT:
videoPID[prg:0x1,pid:0x40,filter:none]: streamType:H264:27
INFO server comment - UDPTransport.firstPacket: bind:/0.0.0.0:10000
msg:/[clientIP]:57010
```

Molt important verificar que l'stream que rebem és reproduïble als logs del server. El valor de les variables:

```
RTPDePacketizerMPEGTS.handleRTPPacket: MPEG-TS over RTP
H264 isCompatible:true
AAC isCompatible:true
```

```
INFO server comment -
LiveStreamPacketizerCupertino.handlePacket[directo/_definst_/mpegts.stream]:
Video codec:H264 isCompatible:true
INFO server comment -
LiveStreamPacketizerCupertino.handlePacket[directo/_definst_/mpegts.stream]:
Audio codec:AAC isCompatible:true
INFO server comment -
LiveStreamPacketizerCupertino.handlePacket[directo/_definst_/mpegts.stream][m
p4a.40.2]: AAC Audio info: {AACFrame: size: 0, rate: 48000, channels: 2,
samples: 1024, errorBitsAbsent: true, profileObjectType: "LC"}
```

Dels logs anteriors comprovem que hem configurat correctament el servidor per rebre fluxos TS codificats en AVC i AAC fent servir el protocol RTP/UDP.

D.3. Contractació servidor d'streaming

En aquest annex volem descriure el procés de selecció del servidor d'streaming. Voliem escollir un servidor escalable en termes de HW i SW. Aquest servidor s'ha de poder dimensionar depenent del nombre d'usuaris del sistema i de la qualitat a la que hem de transmetre. És per aquest motiu que vam estudiar la possibilitat de contractar un d'aquests dos serveis.

Cloud Virtual Private Server VPS

Els servidors privats virtuals fan servir tecnologies de virtualització i ens permeten accés root com si fossin un servidor dedicat. Els VPS ens permeten modificar la CPU i RAM contractada per adaptar-se al tràfic de xarxa que transportin. Són l'alternativa més econòmica a la contractació d'un servidor dedicat.

Cloud Data Center

Servidor dedicat virtual i escalable fins a 96 GB de RAM i 100 GHz de CPU. Ens permetria desenvolupar aplicacions per a un nombre de clients elevats.

D'entre els diferents hostings, vam trobar una valoració econòmica que es mostra a la Fig. D.3.1. Finalment vam decidir contractar el VPS a Gigas. És l'opció més econòmica d'entre les que vam trobar i ens permet oferir una solució completament escalable, en quant a nombre de servidors i en quant a configuració HW. En el cas de voler desenvolupar una xarxa de servidors d'streaming podríem fer-ho contractant diferents nodes a la xarxa per gestionar el tràfic entre un parc de servidors. Volem demostrar que hem tingut en compte paràmetres d'escalabilitat a l'hora de dissenyar el nostre sistema.

	Ram Garantizada	Procesador	Cloud	Panel Plesk	Permanencia	Primer pago	Precio final mensualizado
gigas	1 GB	2 cores 4,8 GHz	✔	¡Gratis!	Sin permanencia	8€ -IVA	19€ -IVA
Arsys	0,25 GB	no indica	✘	Gratis	mínimo 3 meses	104€ -IVA	34,8€ -IVA
Piensa Solutions	0,13 GB	no indica	✘	Gratis	mínimo 3 meses	74,85€ -IVA	24,95€ -IVA
Hostalia	1 GB	no indica	✘	Gratis	mínimo 3 meses	45€ -IVA	15€ -IVA
Dinahosting	1 GB	no indica	✘	72€ año	No	40€ -IVA	40€ -IVA
1and1	0,5 GB	no indica	✘	Gratis	No	19,99€ -IVA	19,99€ -IVA
RedCoruna	0,13 GB	no indica	✘	Gratis	No	19,95€ -IVA	19,95€ -IVA

Fig. D.3.1. Comparativa entre hostings servei VPS

ANNEX E. EQUIPS I FULLES D'ESPECIFICACIONS

En aquest annex s'adjunten les fulles d'especificacions dels equips amb els que hem treballat durant la realització d'aquests TFC. Concretament en detallem les especificacions dels següents:

A la Fig E.1.1. es detallen les especificacions del receptor de DVB-T Volar TV Black. També es poden descarregar de [30].

Fig. E.1.1 Especificacions sintonitzador DVB-T

A la Fig. E.1.2 es detallen les especificacions del codificador HW de Haivision amb el que vam treballar a la secció 3.5.

Haivision
Makito
HD H.264 Encoder

SPECIFICATIONS

Makito (x-290E-DVI)
Video (Inputs):
Y, Pb, Pr / RGBHV component analog
Y, Cb, Cr / DVI component digital
Video Resolutions:
1920x1080p 60/59.94/50/30/24/23.98/29.97/25 Hz
1920x1080i 60/59.94/50 Hz
1280x720p 60/59.94/50/30/29.97/25 Hz
720x480i/576i 60/59.94/50 Hz
720x480/576p 60/59.54/50 Hz
(Interlaced shown in fields per second)
Computer Resolutions:
1920x1080 60 Hz
1280x1024 75/60 Hz
1280x768 85/75/60 Hz
1024x768 85/75/60 Hz
Audio (Input):
Available through terminal block connector:
Balanced Stereo Analog Audio
Unbalanced Stereo Analog Audio

Makito-SDI (x-290E-HDSDI)
Video (Inputs):
S-Video NTSC/PAL
Composite NTSC/PAL
SD-SDI SMPTE 259M-C
HD-SDI SMPTE 292M
SMPTE 274M
SMPTE 296M
3G-SDI SMPTE 424M
SMPTE 425M
Video Resolutions:
1920x1080p 60/59.94/50/30/24/23.98/29.97/25 Hz
1920x1080i 60/59.94/50 Hz
1280x720p 60/59.94/50/30/29.97/25 Hz
1080p24/23.98
720x480i/576i 60/59.94/50 Hz
(Interlaced shown in fields per second)
Audio (Input):
Available through terminal block connector:
Balanced Stereo Analog Audio
Unbalanced Stereo Analog Audio
Embedded Audio
SD-SDI SMPTE 272M
HD-SDI SMPTE 299M

ADVANCED FEATURES
HiLo-Streaming
SD De-interlacing
Built-in Downscaling
Deblocking Filter
EIA-608-B Closed Captioning (NTSC Line 21)
Forward Error Correction
AES Encryption 128-bit or 256-bit
Logo overlay
Still image transmission
SD aspect ratio configuration
SD AFD and WSS (x-290E-HDSDI)
Color space configuration (x-290E-DVI)

VIDEO ENCODING
Compression Standard:
H.264 (MPEG-4 AVC part 10)
ISO/IEC 14496-10
Main Profile
Level 4.2 and lower Intermediate Levels
I, IP framing
Configurable Group of Picture (GOP) size
Configurable frame rate
Bit Rates:
SD/HD from 150 kbps to 15 Mbps
Rate Control:
CBR/VBR
Latency (encode only):
Less than 70ms

AUDIO ENCODING
Compression Standard:
MPEG-2 AAC-LC ISO/IEC 13818-7
MPEG-4 AAC-LC ISO/IEC 14496-3
Audio Channels:
2 per video channel
Bit Rates:
From 32 to 448 kbps per audio pair
Frequency Response:
From 20 Hz to 22 kHz

IP NETWORK INTERFACES
Standard:
Ethernet 10/100/1000 Base-T, auto-detect, Half/Full-duplex
Connector:
RJ45
Networking Protocols:
Unicast Streaming
Multicast Streaming (IGMP v3)
Multiple Unicast Streaming
MPEG Transport Stream over UDP / RTP
Direct RTP - H.264 over RTP (RFC 3984)
RTP / RTCP (RFC 3550)
QuickTime RTSP* (RFC 3640)
SAP (RFC 2974)
* Progressive resolutions only

MANAGEMENT INTERFACES
Standard:
RS-232 (optional for x-290E-DVI)
RJ45 to RS-232 (DB-9 Management Cable Req'd.)
Management:
HTTP (web browser)
Command line over SSH/Telnet/RS-232
FTP/TFTP
SNMP v3
Furnace Portal Server (VF Pilot)

Single Blade Appliance (S-290E-HDSDI/DVI)
Dimensions:
24mm H x 149mm W x 202mm D (0.92"H x 5.85"W x 8.0"D)
Weight:
2.5 lbs.
Power:
5VDC, 13W (each blade),
100-240VAC 15W external locking power supply
Temperature:
Operating: -20º to 50ºC
Non-operating: -40º to 50ºC

6 Blade Chassis (F-MB6-XX)
Dimensions:
19" rack mountable, 1RU
44.45mm H x 438.15mm W x 425.45mm D
(1.75"H x 17.25 W x 16.75"D)
Weight:
6 slot empty chassis – 10lbs
Single encoder blade – 0.5lbs
Power – Internal Power Supplies:
AC type 90-264VAC 47Hz-63Hz
200 Watt max.
Medical Grade 90-264VAC 47Hz-63Hz
300 Watt max.
DC type 20-36 VDC
200 Watt max.
Operating Temperature:
0º to 50ºC (32º to 122ºF).
Non-Operating Temperature:
-40º to 70ºC (-40º to 158ºF).
Humidity:
up to 95% Non-Condensing.

21 Blade Chassis (F-280-21DP8)
Dimensions:
19" rack mountable, 4 RU
178mm H x 445mm W x 400mm D
7.00"H x 17.50"W x 15.75"D
Weight:
21 slot empty chassis – 20lbs
Single encoder blade – 0.5lbs
Power – Internal Power Supply:
90-132V and 180-240VAC 47Hz-63Hz
400 Watt max.
Operating Temperature:
0º to 40ºC (32º to 104ºF).
Non-Operating Temperature:
-40º to 70ºC (-40º to 158ºF).
Humidity:
up to 95% Non-Condensing

METADATA (Optional)
CoT to KLV Conversion
KLV over serial RS-232/422
KLV over UDP and SDI
SMPTE 336M compliant
MISB EG0601.1 compliant
Metadata Multiplexing per MISB RP 0604
Synchronous Mode

Fig. E.1.2 Especificacions codificador HW.

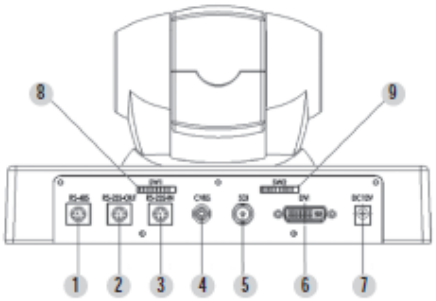
Es poden descarregar les especificacions de [5].

A la Fig. F.1.3. Es detallen les especificacions de la càmera HD700 del fabricant xinés Reach. A [32] hi ha més informació d'aquest fabricant xinés.

Specifications:

Item	Specifications
Image sensor	1/2.7"HD CMOS sensor
Effective pixels	Two million
Signal system	HD:1080p30,1080p25,1080i/60,1080i/50, 720p/60,720p/50,720p/30,720p/25 SD:NTSC,PAL
Shutter speed	1/2 - 1/10000s
Minimum illumination	12Lx(50IRE, F1.6)
Horizontal viewing angle	8°(tele) to 55.2°(wide) at HD signal output
Lens	18x Optical + 4x Digital
S/N Ratio	≥50dB
Focus system	Auto/Manual
White balance	Auto/Indoor/Outdoor/One push auto/ Manual
Backlight Compensation	On/Off
Pan/Tilt	Pan:±150°(0.1°-200°/s) Tilt:-30-90°(0.1°-150°/s)
Position preset	128 positions
Video Output	HD:HD-SDI,DVI-I(is selectable from Y/Pb/Pr, HDMI, DVI-D, VGA, RGBHV) SD:VBS
Control Protocol	PELCO-D,PELCO-P,VISCA
Control Interface	RS-485/422,RS-232
Address	0-255
Temperature	Operating temperature:0°C~+50°C, Storage temperature: -20°C~+70°C
Power requirements	12VDC
Power consumption	<10W
Dimensions	260(D)×160(W)×140(H) mm

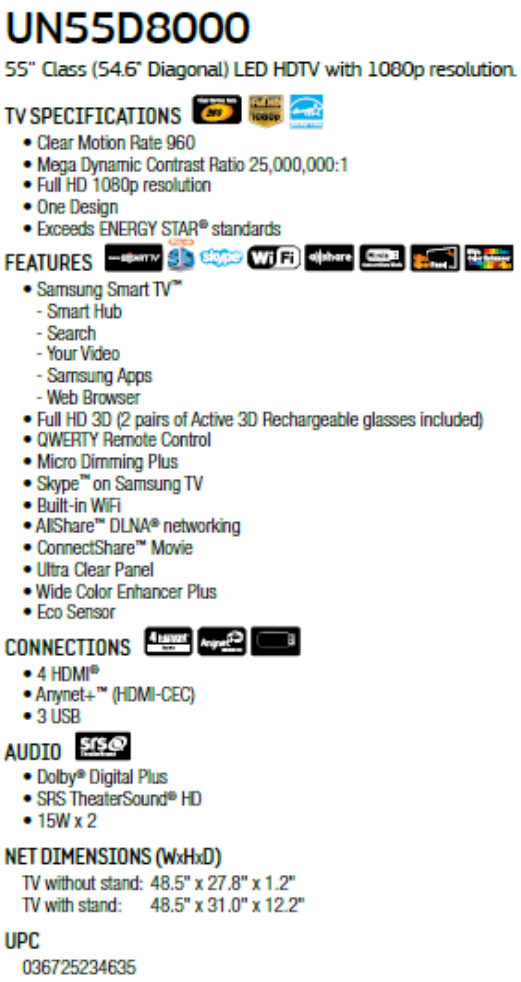
INTERFACE



1 RS-485/RS-422
2 RS-232 out
3 RS-232 in
4 Composite video out
5 HD SDI
6 DVI-I
7 Power in
8 Switch 1
9 Switch 2

Fig. E.1.3. Especificacions càmera HD700 szReach.

A la Fig. E.1.4. es detallen les especificacions de la TV Samsung Smart TV que hem fet servir per les reproduccions 3D. A més es poden descarregar de [31].



UN55D8000

55" Class (54.6" Diagonal) LED HDTV with 1080p resolution.

TV SPECIFICATIONS

- Clear Motion Rate 960
- Mega Dynamic Contrast Ratio 25,000,000:1
- Full HD 1080p resolution
- One Design
- Exceeds ENERGY STAR® standards

FEATURES

- Samsung Smart TV™
  - Smart Hub
  - Search
  - Your Video
  - Samsung Apps
  - Web Browser
- Full HD 3D (2 pairs of Active 3D Rechargeable glasses included)
- QWERTY Remote Control
- Micro Dimming Plus
- Skype™ on Samsung TV
- Built-in WiFi
- AllShare™ DLNA® networking
- ConnectShare™ Movie
- Ultra Clear Panel
- Wide Color Enhancer Plus
- Eco Sensor

CONNECTIONS

- 4 HDMI®
- Anynet+™ (HDMI-CEC)
- 3 USB

AUDIO

- Dolby® Digital Plus
- SRS TheaterSound® HD
- 15W x 2

NET DIMENSIONS (WxHxD)

TV without stand: 48.5" x 27.8" x 1.2"
TV with stand: 48.5" x 31.0" x 12.2"

UPC

036725234635

Fig. E.1.4. Especificacions Smart TV

A la Fig. E.1.5. Es detallen les especificacions del codificador de Reach a SD

Specifications:

Management
Web interface
Central control system (AMX, CRESTRON ect.)
Front panel (One Button Recording)
REACH MP

Encoding
Video encoding: WMV3/H.264
Audio encoding: WMA/AAC

Multi-record Mode
1×SD Video + 1×Stereo Audio
1×VGA + 1×Stereo Audio
1×SD Video + 1×VGA + 1×Stereo Audio

Resolution
Video: CIF/D1 (NTSC and PAL)
VGA: Up to 1400×1050

Fame rate
Video: 25/30 f/s
VGA: 10 f/s

Streaming
Unicast
Multicast

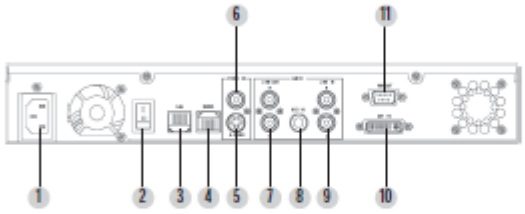
Storage and File Management
Storage capability of 500G
Network storage support (FTP)

Physical Dimensions
Length: 350mm
Width: 270mm
Height: 52mm

Environment Data
Operating temperature: 0° C to 50° C
Relative humidity: 5% to 95%
Storage and transport temperature: -20° C to 80° C

Power
100–240VAC, 50–60Hz, 100W

INTERFACE



1 Power Input
2 Power Switch
3 LAN
4 RS232
5 S-Video In
6 Composite Video In
7 Line Out
8 MIC In
9 Line In
10 DVI-I In
11 VGA Out

Fig. E.1.5 Codificador SD Reach

A la Fig. E.1.6 es detallen les especificacions del codificador HD de Reach.

Specifications:

Management
Web interface
Central control system (AMX, CRESTRON ect.)
Front panel (One Button Recording)

Video Definition
CIF, 4CIF, 480I, 576I, 720P
1080i, 1080P, NTSC and PAL

VGA Resolution
640×480, 800×600, 1024×768
1280×720, 1280×768, 1280×800
1280×960, 1280×1024, 1366×768
1400×1050, 1440×900, 1600×1200

Storage and File Management
Hard disc inside, capability up to 2T
FTP function
File authorization to different user

Encoding
Video encoding: H.264
Audio encoding: AAC

Frame Rate
1 to 60 fps

Audio Sampling Rate
16K, 32K, 44.1K, 48K

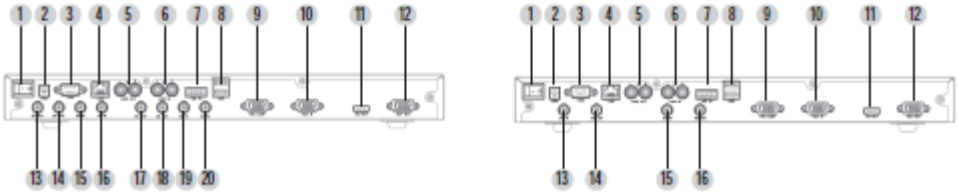
Bandwidth Control
VBR
CBR

Physical Dimensions
Length: 430mm
Width: 373mm
Height: 45mm
19" rack-mountable, 1U height

Environment Data
Operating temperature: 0° C to 50° C
Relative humidity: 5% to 95%
Storage and transport temperature: -20° C to 80° C

Power
100–240VAC, 50–60Hz, 100W

INTERFACE



(4 SD + 1 VGA + Decoding)

1 Power Switch	8 USB × 2	15 Composite Video In 1
2 Power In	9 VGA Loop Out	16 Composite Video In 2
3 COM (RS232)	10 VGA In	17 Composite Video Out 3
4 LAN	11 HDMI Out	18 Composite Video Out 4
5 Line Out	12 VGA Out	19 Composite Video In 3
6 Line In	13 Composite Video Out 1	20 Composite Video In 4
7 MIC In	14 Composite Video Out 2	

(2 HD SDI + 1 VGA + Decoding)

1 Power Switch	7 MIC In	13 SDI In 1
2 Power In	8 USB × 2	14 SDI Out 1
3 COM (RS232)	9 VGA Loop Out	15 SDI In 2
4 LAN	10 VGA In	16 SDI Out 2
5 Line Out	11 HDMI Out	
6 Line In	12 VGA Out	

Fig. E.1.6. Configuracions codificador HD de Reach