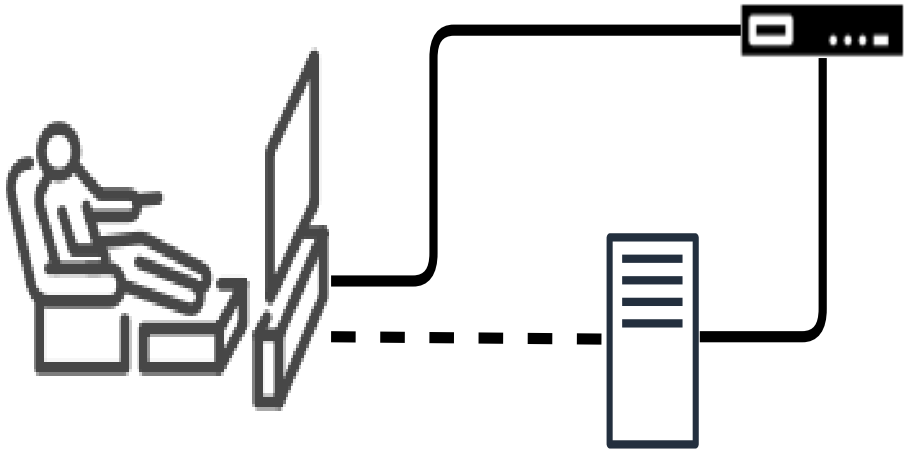
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de
Electrónica e Telecomunicações e de Computadores



Unicast Assisted System for Fast Channel Change on DVB-C

Pedro Tiago Fernandes Moreira

Licenciado em Engenharia Eletrotécnica

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática e de Computadores

Orientadores : Professor Doutor Tiago Miguel Braga da Silva Dias
Professor Doutor Nuno Carlos André Sebastião

Júri:

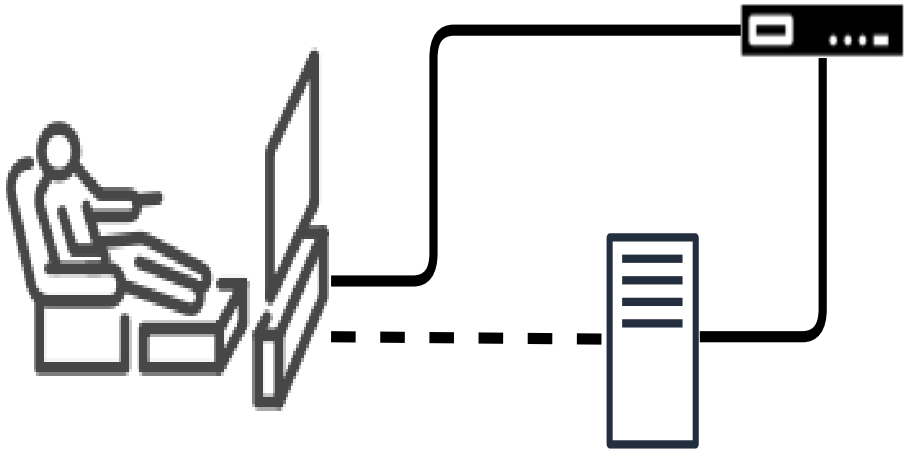
Presidente: Doutor Nuno Miguel Machado Cruz

Vogais: Doutor Nuno Filipe Valentim Roma
Doutor Tiago Miguel Braga da Silva Dias

Julho, 2021

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de
Electrónica e Telecomunicações e de Computadores



Unicast Assisted System for Fast Channel Change on DVB-C

Pedro Tiago Fernandes Moreira

Licenciado em Engenharia Eletrotécnica

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática e de Computadores

Orientadores : Professor Doutor Tiago Miguel Braga da Silva Dias
Professor Doutor Nuno Carlos André Sebastião

Júri:

Presidente: Doutor Nuno Miguel Machado Cruz

Vogais: Doutor Nuno Filipe Valentim Roma
Doutor Tiago Miguel Braga da Silva Dias

Julho, 2021

Resumo

Os sistemas de difusão de televisão digital (*Digital Video Broadcasting* (DVB) [1], em Inglês) apresentam um elevado tempo de mudança de canal (*zapping*). Este tempo resulta de vários fatores, por exemplo o tempo de bloqueio da frequência do sintonizador, o tempo de análise da sinalização do DVB, o tempo de inicialização do sistema de acesso ao conteúdo, o tempo de espera até ser recebido o início de um bloco de vídeo descodificável (tipicamente o início de um grupo de imagens, em Inglês *Group of Pictures* (GOP)), ou o tempo de atraso introduzido pelo *buffer*.

Vários métodos têm sido apresentados para reduzir o tempo de *zapping*. Uma das soluções mais recentes é o envio de partes da *stream*, a pedido, através de *unicast* sobre uma rede IP, que consiste na transmissão do canal que se pretende reproduzir de uma forma mais rápida, de modo a que o utilizador possa estar a ver o canal enquanto é feita a sintonização do canal recebido por DVB.

O objetivo deste trabalho é investigar uma técnica eficiente de mudança rápida de canal utilizando *streams unicast* sobre IP e desenvolver um componente *software* que a implemente, para ser integrada nos sistemas modernos de transmissão de vídeo em *broadcast*. Este componente compreende *software* no lado do servidor IP e do cliente, designadamente na *Set-Top Box* (STB) do utilizador.

Palavras-chave: DVB-C, Mudança Rápida de Canal, Protocolos de Transporte.

Abstract

Digital Video Broadcasting (DVB) systems typically suffer from a high channel change (*zapping*) time. This is due to multiple factors, namely: i) tuner frequency lock time; ii) parsing of DVB signaling; iii) content access system initialization; iv) delay until the beginning of a decodable video block is received (typically the start of a *Group of Pictures*); and v) *buffering* time.

Several methods have been exploited to reduce *zapping* time, including channel pretuning, in which other *streams* are tuned in the background and are ready to be displayed as soon as required, companion *streams*, where low quality *streams* with lower *buffer* requirements are also *broadcast*, and delivery of *stream* chunks, on request, via *unicast* on top of an IP network.

The goal of this work is to develop a software component to integrate in modern video *broadcast* systems to support efficient fast channel change using *unicast* streams over IP. Such component is comprised of both the server software and the client/set-top-box (STB) software. This system will also be used as a base to support and test the implementation of other features that aim at providing a smooth and fast channel change for DVB systems.

Keywords: Multimedia streaming, Digital Video Broadcast, Fast Channel Change, Transport Protocols.

Índice

Lista de Figuras **xi**

Lista de Abreviaturas e Siglas **xiii**

Glossário **xv**

1 Introdução **1**

1.1 Objetivos 2

1.2 Estrutura do documento 3

2 Conceitos **5**

2.1 Televisão Digital em DVB 5

2.2 Descodificação de um canal 6

2.3 *Transport Stream* 7

2.4 *Packetized Elementary Stream* 8

2.5 *Elementary Stream* 9

2.6 *Program Specific Information* 10

2.7 *Group of Pictures* 10

3 Trabalho Relacionado **13**

3.1 Métodos para uma mudança rápida de canal em DVB-C 13

3.2 Soluções híbridas 14

3.3 Métodos para uma mudança rápida de canal em IPTV 14
3.4 Análise das soluções 15

4 Solução Proposta **17**
4.1 Alterações ao sistema DVB-C 18
4.1.1 Servidor IP 18
4.1.2 Recetor 20
4.2 Mecanismo de sincronização 21
4.3 Comparação com o trabalho relacionado 21

5 Implementação **23**
5.1 *Gstreamer* 23
5.1.1 Funcionamento do *Gstreamer* 24
5.1.2 Bases do *Gstreamer* 24
5.1.2.1 *Pipeline* 24
5.1.2.2 Elementos 24
5.1.2.3 *Pads* 25
5.1.3 Aplicação *Gstreamer* 26
5.1.4 Obter uma SPTS 27
5.2 Modelação do sistema 29
5.2.1 Modelação do *headend* 29
5.2.2 Modelação do servidor IP 30
5.2.3 Modelação do recetor 31
5.2.4 Funcionamento da simulação 32
5.3 Desenvolvimento de um *Plugin* do *Gstreamer* 33
5.4 Resultados Experimentais 34

6 Conclusões **35**

Referências **37**

Lista de Figuras

2.1 Camadas de Sistema do MPEG-2 6
2.2 Processo de descodificação do sinal analógico 6
2.3 *Multi-program transport stream* 8
2.4 Pacote PES (representado na parte superior) dividido para formar múltiplos pacotes da TS 9
2.5 Exemplo de um *GOP* e do seu tipo de *frames* 11

4.1 Representação da solução proposta 18
4.2 Representação do funcionamento do servidor 19
4.3 Representação da constituição do servidor 19
4.4 Representação do funcionamento da STB 20

5.1 Exemplo de um *pipeline* para reprodução de um ficheiro com vídeo e áudio 24
5.2 Exemplo de diversos elementos e dos seu *pads* 25
5.3 *Pipeline* da aplicação para reprodução de um canal 27
5.4 Representação do *pipeline* utilizado para obter um ficheiro com a *SPTS* do programa que se pretende reproduzir 28
5.5 Representação do *pipeline* utilizado para obter e reproduzir uma *SPTS* . 28
5.6 Representação dos *pipelines* utilizados na aplicação do *headend* 29
5.7 Representação do *pipeline* do servidor IP 30
5.8 Representação do *pipeline* do recetor 31
5.9 Representação do *plugin* para sincronização das TS 33

Lista de Abreviaturas e Siglas

ADC	*Analog-to-Digital Converter*.
CAT	*Conditional Access Table*.
DOCSIS	*Data Over Cable Service Interface Specification*.
DTS	*Decode Time Stamp*.
DVB	*Digital Video Broadcasting*.
DVB-C	*Digital Video Broadcasting - Cable*.
DVB-H	*Digital Video Broadcasting - Handheld*.
DVB-S	*Digital Video Broadcasting - Satellite*.
DVB-T	*Digital Video Broadcasting - Terrestrial*.
ES	*Elementary Stream*.
FEC	*Forward Error Correction*.
GOP	*Group of Pictures*.
IPTV	*Internet Protocol Television*.
MPTS	*Multipe Program Transport Stream*.
NIT	*Network Information Table*.
PAT	*Program Association Table*.
PES	*Packetized Elementary Stream*.
PID	*Packet Identifier*.
PMT	*Program Map Table*.
PSI	*Program Specific Information*.
PTS	*Presentation Time Stamp*.
QAM	*Quadrature Amplitude Modulation*.
SPTS	*Single Program Transport Stream*.
STB	*Set-Top Box*.
TCP	*Transmissions Control Protocol*.

TDT Televisão Digital Terrestre.
TS *Transport Stream*.
UDP *User Datagram Protocol*.

Glossário

acesso condicionado	Proteção da *stream* para impedir o acesso à mesma por terceiros não autorizados ou por clientes que não a subscrevam.
access unit	Corresponde a uma *frame* ou bloco de áudio codificado - unidade de acesso.
broadcast	Comunicação de pacotes para todos os destinos (difusão).
buffer	Zona de memória que guarda dados temporariamente.
buffering	Processo de armazenamento em *buffer*.
GStreamer	Biblioteca para construir *pipelines* de componentes de processamento de informação utilizados para *streaming* de áudio/vídeo.
headend	Instalação responsável pela difusão dos canais de televisão por *Digital Video Broadcasting - Cable* (DVB-C).
multicast	Comunicação de pacotes para vários destinos simultaneamente.
parse	Ler e analisar dados para os dividir em componentes estruturados, que seguem uma sintaxe, para que possam ser facilmente guardados ou processados.
payload	Conteúdo de um pacote de dados.
scramble	Manipulação dos dados de uma *stream* antes da transmissão, semelhante à encriptação, para prevenir que a *stream* seja utilizável por entidades não autorizadas.
stream	Sequência de dados contínua disponibilizada ao longo do tempo.

streaming	Forma de distribuição digital, geralmente de áudio ou vídeo, sem necessidade de descarregar o conteúdo, permitindo a sua reprodução sem ocupar espaço no dispositivo do utilizador.
unicast	Comunicação de pacotes para um único destino.
timestamp	Marca temporal.
zapping	Mudança rápida e consecutiva de canal de televisão.

1

Introdução

Atualmente o consumo de conteúdos através da televisão continua a fazer parte do dia-a-dia do cidadão comum, seja para entretenimento, como novelas, séries, filmes, ou programas do quotidiano para informação ou para aprendizagem, como documentários ou programas culturais.

A transmissão de televisão digital está relacionada com o serviço fornecido, podendo ser transmitida por cabo (DVB-C), por satélite (*DVB-Satellite* (DVB-S)), por transmissão terrestre(*DVB-Terrestrial* (DVB-T)), conhecida como Televisão Digital Terrestre (TDT), ou transmitida para aparelhos móveis (*DVB-Handheld* (DVB-H)).

Em Portugal, tal como acontece na generalidade dos países desenvolvidos, a esmagadora maioria dos consumidores, no final de 2017 o número de subscritores de pacotes de serviços era 3,74 milhões [2], tem contratado serviços de televisão por cabo, que lhes disponibiliza um elevado número de canais com conteúdos muito diversificados. Todavia, com a introdução de serviços de fibra ótica, que face à transmissão por cabo permite ligações de alta velocidade, com uma menor degradação do sinal e menor latência, foi possível a adoção da transmissão de televisão por IP, conhecida por *Internet Protocol Television* (IPTV), pois permitem uma melhor experiência de utilização. Para ambas as tecnologias, dada a grande variedade de conteúdo disponível, verifica-se frequentemente a situação de se ter que percorrer muitos canais até se encontrar um programa que seja interessante para assistir, i.e. fazer *zapping*.

Nos serviços de distribuição de televisão por cabo, o transporte de áudio e vídeo é feito segundo a norma DVB-C [3]. Este sistema é composto por um *headend*, que faz

a difusão dos vários canais de televisão em *broadcast*, ou seja, os diversos canais são multiplexados formando uma *stream* que é transmitida para todos os utilizadores, e pelos recetores, que processam a *stream* de maneira a selecionar e apresentar o canal específico que se pretende visualizar. Estes recetores, devido à sua forma de funcionamento e aos requisitos das normas aplicáveis, podem apresentar um elevado tempo para mudança de canal (*zapping*), o que se deve a diversos fatores, de entre os quais se destacam os seguintes:

- o tempo para sintonizar a portadora na qual está disponibilizada a *Transport Stream* (TS), *stream* que pode conter apenas um ou vários canais, referente ao canal que se pretende;
- a necessidade de *buffering*, ou seja, guardar em memória, por tempo limitado, dados de vídeo e áudio para que a reprodução seja feita sem falhas, o que implica que mesmo que o recetor já tenha dados suficientes para começar a reprodução esta não é logo iniciada porque os dados podem não ser suficientes para que a reprodução seja feita sem falhas ao longo do tempo;
- a impossibilidade de iniciar a reprodução de vídeo em qualquer imagem (apenas é possível começar a reprodução de um novo canal quando o recetor recebe a primeira imagem de um novo grupo).

Do ponto de vista do utilizador, o tempo necessário para iniciar a reprodução de um novo canal é sinónimo de tempo perdido que gera desconforto e frustração, sendo que para os operadores conseguir reduzir o tempo de mudança de canal será uma melhoria na qualidade de experiência, que pode resultar numa vantagem comercial.

1.1 Objetivos

Os principais objetivos do trabalho são o estudo e o desenvolvimento de um sistema capaz de diminuir o tempo de *zapping* entre canais, através da utilização de *streams unicast* sobre IP. Em particular, este trabalho está direcionado à redução do tempo em sistemas DVB-C, visto a grande maioria destes sistemas estarem emparelhados com serviços de dados (via *Data Over Cable Service Interface Specification* (DOCSIS)) o que permite a existência de um canal de comunicação via IP.

O sistema a desenvolver baseia-se no sistema tradicional de DVB-C. Em comparação com o mesmo, para o consumidor apenas será necessário que o recetor tenha uma interface de rede IP, que permitirá obter a TS constituída apenas pelo novo canal a

visualizar por *unicast*. A emissão dessa TS será disponibilizada por um servidor IP, que também terá disponível as TS de outros canais de televisão, que enviará para o recetor apenas quando for pedida.

Este trabalho tem também como objetivo a modelação do sistema proposto usando a *framework GStreamer* [4], uma plataforma *open source* amplamente utilizada para desenvolvimento de aplicações que lidam com *streams* multimédia em diversos sistemas, em particular nos que se baseiam em Linux. Na indústria o *GStreamer* é utilizado nas STB, bem como para aplicações comerciais, tais como, o fornecedor de soluções multimédia Fluendo [5] e a aplicação Rhythmbox [6], o reprodutor de áudio disponibilizado de base em muitas das distribuições de Linux.

1.2 Estrutura do documento

O documento segue uma tipologia *Top-Down*, na qual se inicia no capítulo 2 pela introdução dos conceitos relacionados com o trabalho. Seguidamente, no capítulo 3 são apresentados trabalhos relacionados que exploram outros métodos para resolver o problema. No capítulo 4 é apresentada a solução proposta seguindo-se, no capítulo 5, a apresentação do trabalho realizado. Por fim, no capítulo 6 as conclusões são apresentadas bem como o trabalho futuro.

2

Conceitos

Este capítulo serve para apresentar os conceitos subjacentes ao *standard* DVB-C e que são explorados no âmbito deste trabalho.

2.1 Televisão Digital em DVB

A transmissão de televisão digital em DVB segue os *standards* desenvolvidos pelo consórcio DVB *Project* [7]. Os *standards* para DVB estão baseados na especificação do MPEG-2 [8], sendo também possível a utilização do MPEG-4 [9] para codificação de vídeo e áudio.

Tipicamente, um programa consiste num canal de vídeo e, possivelmente, múltiplos canais de áudio. A norma ISO-13818-1 [8] define métodos de multiplexar mais de uma *stream* (vídeo, áudio e dados) de modo a produzir um programa. Tem uma *framework* básica para serviços integrados de vídeo, áudio e dados. No entanto, podem existir várias *streams* de vídeo para o mesmo programa, como por exemplo na transmissão de um jogo de futebol com múltiplas câmaras. As *streams* de dados, se existirem, são apenas utilizadas para transmitir, em *broadcast*, dados relativos ao programa.

As camadas de sistema do MPEG-2 fornecem métodos normalizados de transmissão de vídeo, áudio e serviços de dados e estão representadas na Figura 2.1.

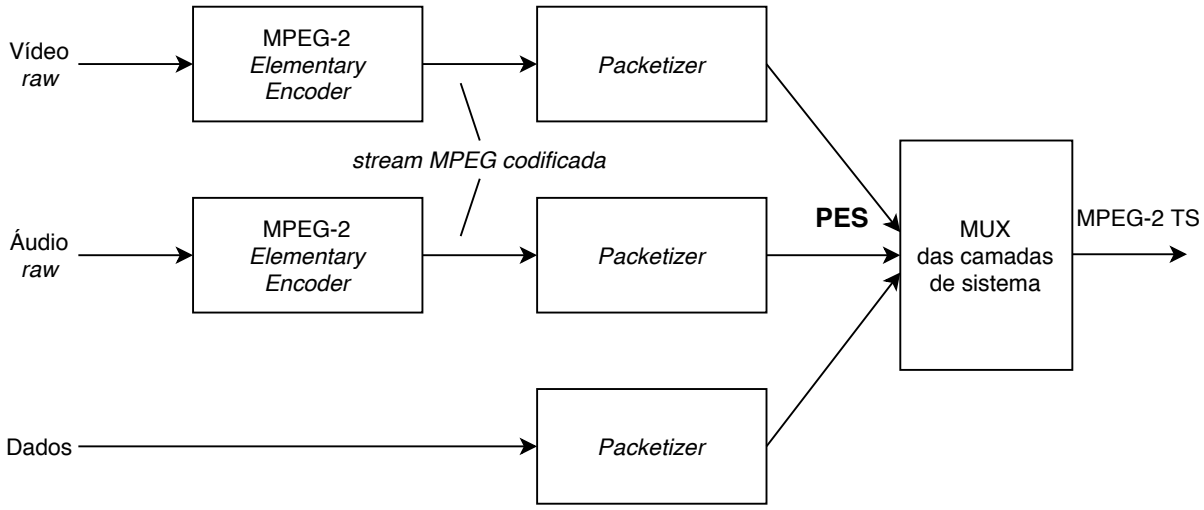


Figura 2.1: Camadas de Sistema do MPEG-2

Em DVB os dados dos canais são transmitidos por uma TS, que agrega todos os pacotes do tipo *Packetized Elementary Stream* (PES) e pela *Program Specific Information* (PSI).

2.2 Descodificação de um canal

Para fazer a reprodução de um canal o recetor tem que processar a *stream* de dados que contém a TS. No *headend* a TS é codificada, como primeira medida de resiliência, utilizando o código Reed-Solomon Seguidamente, para evitar longas sequências de erros, a *stream* é intercalada e volta a ser codificada, desta vez utilizando codificação diferencial. Por último, antes de ser convertida para sinal analógico e transmitida por radiofrequência, na onda portadora (onda que tem uma frequência específica), é modulada utilizando *Quadrature Amplitude Modulation* (QAM).

Deste modo, para obter a TS, o sinal analógico que chega ao recetor é convertido para digital, com a utilização de um *Analog-to-Digital Converter* (ADC), depois tem que passar por um desmodulador QAM e um descodificador diferencial. Posteriormente, tem que se reverter o processo de intercalação, com um desintercalador convolucional, e descodificar o código Reed-Solomon. Este processo está representado na Figura 2.2.

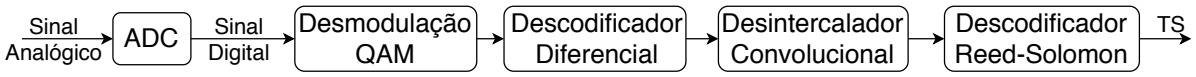


Figura 2.2: Processo de descodificação do sinal analógico

Após se obter a TS é necessário esperar por um pacote que contenha a *Program Association Table* (PAT), que contém uma lista de todos os canais presentes na TS. Adicionalmente, a TS contém informação, disponível na *Network Information Table* (NIT), sobre outras TS que façam parte do serviço prestado pelo operador e de onde podem ser encontradas (e.g. frequência e tipo de modulação da portadora). Se os canais forem *scrambled*, o recetor precisa de saber qual foi o sistema utilizado, para tal, tem que esperar por um pacote que contenha a *Conditional Access Table* (CAT), tabela que serve para gerir o sistema de *scrambling*, mas que tem pouca influência no tempo de *zapping*.

Depois de obtidas estas tabelas é possível sintonizar o canal que se pretende reproduzir, mas como a reprodução só se inicia numa *I-frame*, tem que se esperar por um pacote que contenha uma *I-frame*. Para prevenir falhas na reprodução de vídeo e áudio é necessário fazer-se *buffering*, para compensar a diferença de tamanho entre as *frames* codificadas, e sincronizar o vídeo com o áudio, só depois é que é possível começar a reproduzir o canal.

2.3 *Transport Stream*

Uma TS é uma *stream* de bits contínua na qual estão agregados vários serviços/programas. O propósito da TS é transportar todos os dados que um ou mais "canais de televisão" transmitem.

Uma TS é, portanto, composta pelas PES de áudio, vídeo e dados de um ou mais programas, que é multiplexada numa única *stream* em conjunto com a informação necessária para fazer a sincronização entre elas. Ademais, há um conjunto específico de *Elementary Stream* (ES) que é utilizado para sinalização global, na qual é descrita a TS, a rede, o operador, os serviços, entre outros.

Uma TS pode ser classificada como *Single Program Transport Stream* (SPTS) ou *Multipe Program Transport Stream* (MPTS), em que a primeira apenas contém as PES de um programa e a segunda de vários programas, como se ilustra na Figura 2.3.

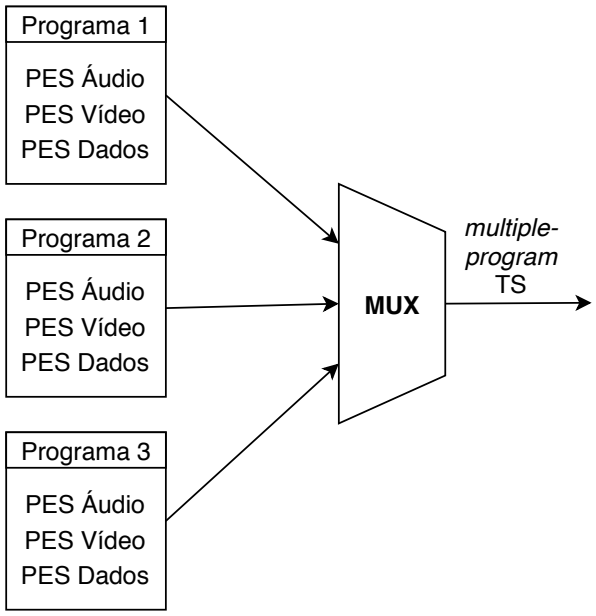


Figura 2.3: *Multi-program transport stream*

As TS foram desenhadas para ambientes onde é provável a perda de pacotes, logo, suportam a implementação de mecanismos de deteção e correção de erros, *Forward Error Correction* (FEC), que ajudam a corrigir os bits errados na *stream* de dados. Adicionalmente, cada pacote da TS inclui um contador de continuidade, que permite detetar a falta de pacotes na sequência de dados e assim tomar as ações necessárias, do lado do descodificador, para compensar esta falha. A TS inclui informação que indica se determinado pacote é o início de uma nova unidade da correspondente ES, permitindo desta forma a recuperação do descodificador na presença de erros na PES derivados da perda de pacotes na TS.

2.4 *Packetized Elementary Stream*

Uma PES é o resultado do processo de empacotamento, no qual o *payload* são os bytes extraídos sequencialmente da ES original.

Um pacote PES pode ser dividido em múltiplos pacotes, que são posteriormente encapsulados em pacotes da TS (com o acrescento de um *header* - o TS *packet header*), como representado na Figura 2.4.

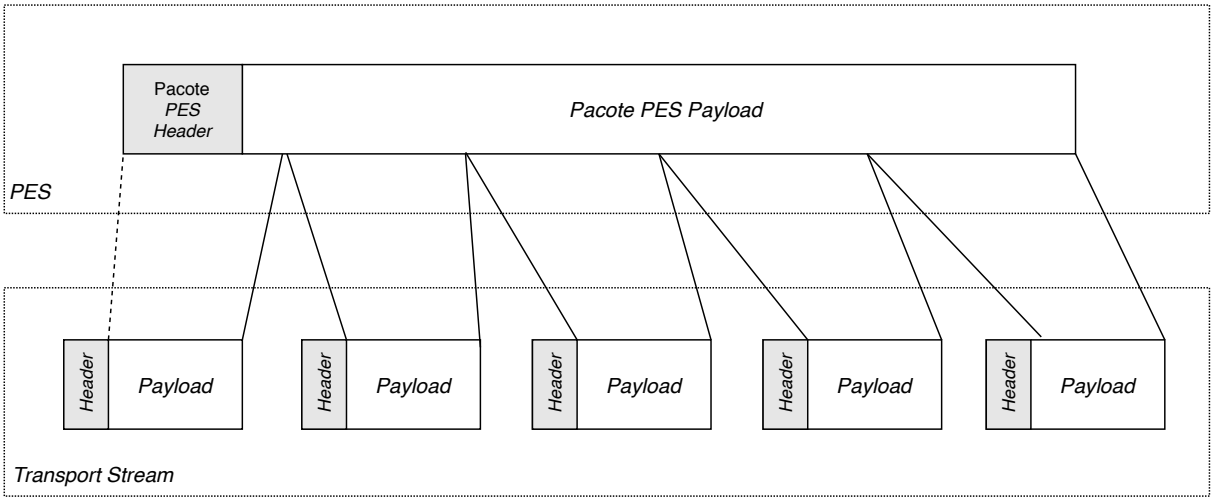


Figura 2.4: Pacote PES (representado na parte superior) dividido para formar múltiplos pacotes da TS

Cada pacote PES tem o seu próprio formato, que depende do conteúdo que se quer transmitir (áudio, vídeo ou dados) e que o seu tamanho é variável, mesmo dentro de cada ES. Por exemplo, um pacote de uma PES de vídeo pode ser maior que o seguinte, dependendo do codificador. Os cabeçalhos de PES distinguem os pacotes PES de várias *streams* e também contêm a informação temporal da ordem dos pacotes.

Para sincronizar diversas ES de um programa recorre-se ao *Presentation Time Stamp* (PTS) ou ao *Decode Time Stamp* (DTS), parâmetros do cabeçalho de um pacote PES. A PTS indica o instante temporal em que a *frame* descodificada ou o áudio descodificado deve ser apresentado no descodificador. A DTS indica o tempo em que uma *access unit* deve ser descodificada pelo descodificador. Este parâmetro é opcional, mas se for utilizado a PTS também deve ser fornecida.

2.5 *Elementary Stream*

Uma ES é o resultado de saída de um codificador contendo apenas um tipo de conteúdo, que pode ser vídeo, áudio referente a uma língua ou legendas para uma língua. A exceção são páginas teletexto que são uma junção de várias *streams* de texto.

Uma ES contém a informação mínima necessária para um descodificador conseguir produzir aproximadamente a mesma imagem ou áudio. Numa TS, uma ES é uma sequência de pacotes cujos cabeçalhos apresentam o mesmo valor para o *Packet Identifier* (PID).

2.6 *Program Specific Information*

O PSI consiste nos metadados que possibilitam a configuração automática do recetor para desmultiplexar a TS associada, não devendo ser alvo de *scramble*. Estes metadados são transportados em pacotes que têm PIDs únicos que são incluídos periodicamente na TS. O PID é um parâmetro do cabeçalho da TS que identifica uma ES ou uma tabela do PSI.

Numa TS, o PSI está classificado em cinco tabelas: a PAT, a CAT, a *Program Map Table* (PMT) e a NIT.

A PAT é sempre transmitida em pacotes com o PID 0 (zero) e contém uma lista completa de todos os programas presentes na TS e o PID para cada PMT de cada programa.

A PMT contém o PID de cada PES associado a um programa específico e a descrição técnica do serviço. O programa zero é sempre reservado para dados da rede e para a NIT.

A NIT é opcional e pode ser usada para providenciar informação útil relacionada com a rede física, tais como as frequências dos canais, o serviço de origem ou o nome do serviço.

A CAT é sempre incluída nos pacotes com PID 1 (um) e deve ser enviada quando a ES é *scrambled*. A CAT contém informação sobre o sistema de *scrambling* em uso e sobre os valores de PID dos pacotes de transporte que contêm a informação sobre o acesso condicionado.

2.7 *Group of Pictures*

Na codificação de vídeo, uma *stream* de vídeo consiste numa sequência de GOPs. Um GOP é um conjunto de imagens, usualmente designadas por *frames*, cuja descodificação depende apenas da informação contida no próprio GOP. As *frames* contidas num GOP são temporalmente consecutivas, ainda que a sua sequência no GOP possa ser diferente.

Um GOP, exemplificado na Figura 2.5, pode conter vários tipos de *frames*:

- *I-frames*: *frame* codificada de forma independente de outras *frames* e apenas contém informação relativa a si. Cada GOP inicia com uma *frame* deste tipo.

- *P-frames*: *frames* preditivas que estão relacionadas com a *I-frame* ou a *P-frame* anterior e utilizam compensação de movimento para apenas transmitirem as diferenças entre a informação presente nessa *frame* para a *frame* atual.
- *B-frames*: *frames* de previsão bidirecional que estão relacionadas com a *I-frame* ou *P-frame* anterior e com a *I-frame* ou *P-frame* posterior, utilizando a compensação de movimento para apenas transmitirem as diferenças entre a informação da última *frame* de referência para a nova *frame*, sendo que apenas são descodificadas depois da *frame* I ou P seguinte ser descodificada.

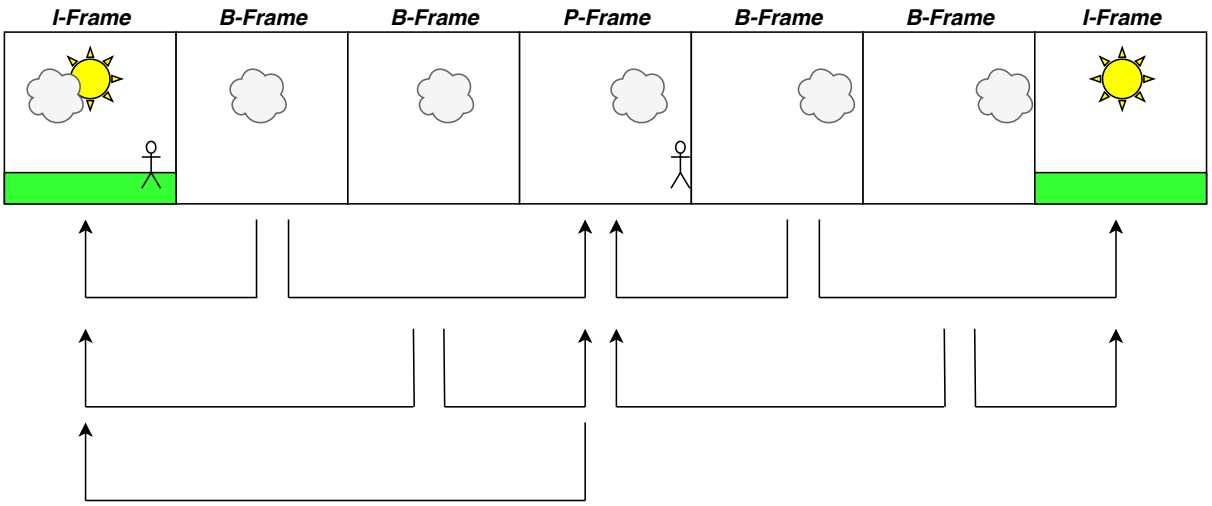


Figura 2.5: Exemplo de um *GOP* e do seu tipo de *frames*

3

Trabalho Relacionado

Neste capítulo são discutidos métodos já existentes que visam a redução do tempo de mudança de canal em DVB-C, bem como outros métodos que resolvam o mesmo problema noutro tipo de sistemas, designadamente em IPTV.

3.1 Métodos para uma mudança rápida de canal em DVB-C

Atualmente, já existem alguns métodos que possibilitam a redução do tempo de mudança de canal. Por exemplo, em [10] sugerem-se alterações que podem ser feitas na arquitetura do sistema para alcançar este objetivo baseadas em modelos de zap desenvolvidos para IPTV, mas adaptados para DVB, tais como, a introdução de dois novos módulos de software para fazerem gestão de recursos e a manipulação de notificações e de pedidos de retorno e a alteração na camada de abstração de hardware para reduzir o tempo de atraso introduzido pelo *buffer* no processo de sincronismo dos lábios com o som. Estas alterações possibilitam reduzir o tempo de mudança de canal até um quarto do valor inicial, tendo os autores conseguido reduzir o tempo de *zapping* em dois segundos, passando de 2648 ms para 650 ms. No entanto, trata-se de uma solução complexa e para conseguir estes resultados o novo canal a reproduzir tem que já estar pré-sintonizado.

Outros métodos para reduzir o tempo de *zapping* são:

- a sintonização preditiva [11], aplicável para DVB-C e também possível para IPTV, em que se faz *buffering* dos dados dos canais adjacentes ao canal a ser reproduzido para que a mudança para um destes canais seja mais rápida.
- a utilização de *streams* complementares, onde *streams* de menor qualidade que requerem buffers de menor dimensão são simultaneamente transmitidas em *broadcast*.

Um método interessante e que pode ser utilizado tanto para DVB-C como para IPTV é a utilização de *machine learning* [12], que utiliza o histórico de canais, para prever qual é o canal que o utilizador vai escolher.

3.2 Soluções híbridas

Na edição 34 da revista publicada pela DVB [13] são mencionadas soluções híbridas que são constituídas por sistemas de transmissão por *broadcast* com sistemas de IPTV com o objetivo de melhorar a experiência do utilizador. Para estas soluções são necessárias STBs mais complexas (como por exemplo, interface de rede) por terem mais funcionalidades e serviços que uma STB apenas capaz de receber e reproduzir uma TS transmitida por DVB-C. A empresa AXING AG desenvolveu um projeto para um hotel na Alemanha em que utiliza esta solução, o serviço de televisão é feito por DVB-C e os serviços interativos são disponibilizados por WiFi [14].

3.3 Métodos para uma mudança rápida de canal em IPTV

Em sistemas IPTV a difusão é feita em *multicast*, ou seja, cada canal tem a sua *stream multicast*. Para mudar de canal é necessário que a STB desconecte da *stream* que se estava a ver e conecte à *stream* do canal que se pretende assistir.

Dos métodos que existem para reduzir o tempo de mudança de canal, um deles é a utilização de uma *stream multicast*, de menor qualidade, complementar à qual o recetor se junta quando é pedida a mudança de canal enquanto o *buffer* de melhor qualidade correspondente à *stream* principal é preenchido [15].

Para redução do tempo de mudança de canal no artigo [16] são sugeridas as seguintes medidas:

- optimizações à arquitetura do sistema IPTV que não necessitam de mudanças no sistema geral.

- *streams* para sintonização na mudança de canal, que apenas necessitam de adaptação por parte do servidor e do cliente, sendo por isso agnósticas à rede.
- utilização de *edge servers* como extensão da arquitetura do sistema IPTV, sendo uma solução agnóstica para o utilizador.

3.4 Análise das soluções

A primeira solução apresentada para redução do tempo de mudança de canal em DVB-C baseia-se na aplicação de mudanças na arquitetura do sistema, uma solução complexa e que necessita da pré-sintonização do novo a canal a reproduzir, mas que possibilitam a redução do tempo de mudança de canal. Em relação à solução apresentada em [11] o problema é a necessidade de processar várias *streams* ao mesmo tempo e de fazer *buffering* de vários canais que poderão não ser utilizados, sendo um desperdício de recursos. A solução que utiliza *machine learning* é mais complexa e necessita de infraestruturas (servidores) para estudar o comportamento de cada utilizador para cada STB, sendo que se uma STB for utilizada por vários utilizadores com gostos diferentes, os canais previstos podem não corresponder aos que o utilizador atual deseja assistir.

As soluções híbridas são uma forma de complementar o sistema DVB-C, mas não garantem a redução do tempo de mudança de canal, pelo que, a maior vantagem apresentada é a inclusão de serviços interativos e a possibilidade de incorporar serviços de *streaming*, como a Netflix, o Youtube ou o Spotify.

Outros métodos para reduzir o tempo de mudança de canal em DVB-C passam pela adaptação das soluções apresentadas para os sistemas de IPTV. A optimização da arquitetura do sistema DVB-C, com base no sistema IPTV, apresentada em [10], resultou numa redução do tempo de mudança de canal, pelo que não será de excluir que outras alterações não tenham o mesmo resultado. Para um sistema DVB-C, a utilização de *streams* adicionais com menor qualidade [15], que em IPTV traz desvantagens quer ao nível da utilização extra de largura de banda quer da qualidade reduzida no início da reprodução, poderia ser adaptado, mas as desvantagens apresentadas manter-se-iam e seria necessário alterar o sistema DVB-C para se poder utilizar esta solução.

4

Solução Proposta

Neste capítulo é apresentada a solução que se pretende implementar para se alcançar uma redução no tempo de mudança de canal.

A solução proposta consiste na transmissão da TS do novo canal através de uma *stream unicast* sobre IP e que permitirá que o recetor comece a reproduzir o canal recebido por este método, enquanto a TS difundida por DVB-C é processada.

A solução proposta, ilustrada na Figura 4.1, é baseada no sistema normal de DVB-C, constituído por um *headend* e por um recetor (a STB do cliente), com a introdução de um servidor IP, que fará a transmissão da *stream unicast*. O princípio de funcionamento desta solução consiste em disponibilizar na STB a SPTS do novo canal a reproduzir, transmitida por IP, o que permite iniciar a reprodução mais rapidamente comparada com o sistema tradicional de DVB-C, e quando a *stream* DVB-C estiver pronta a ser reproduzida e de se encontrar um ponto em comum entre as duas *streams* fazer a transição para a *stream* DVB-C.

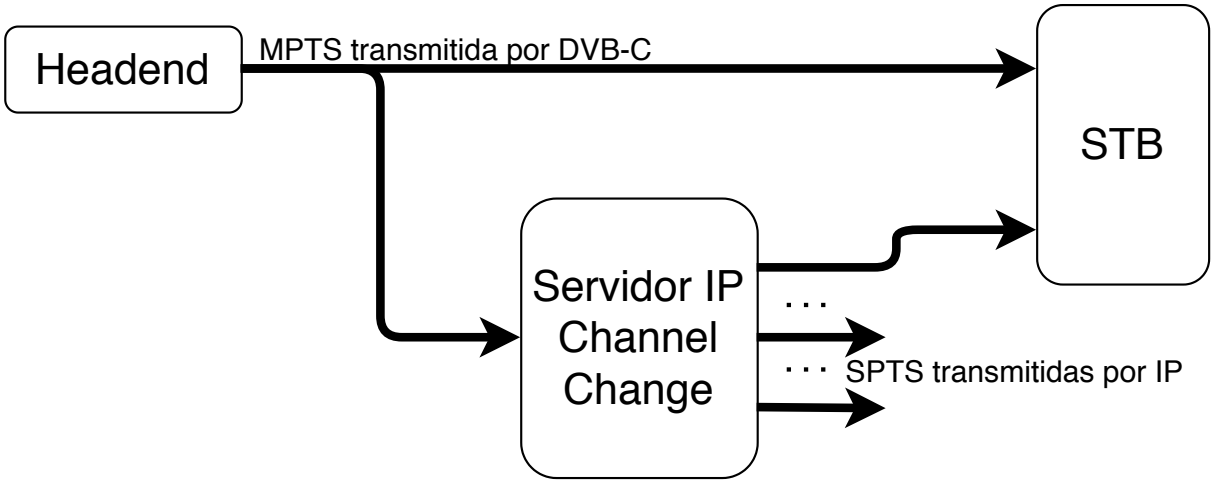


Figura 4.1: Representação da solução proposta

4.1 Alterações ao sistema DVB-C

Esta solução foca-se nas alterações necessárias a fazer nas infraestruturas do lado da rede e do lado do cliente. No lado da rede foca-se no desenvolvimento do servidor IP pois, tal como no sistema tradicional de DVB-C, o *headend* é responsável pela transmissão da MPTS por DVB-C e não irá necessitar de alterações. No lado do cliente foca-se no desenvolvimento da STB que vai passar obter duas TS de fontes diferente e sincronizar as duas.

4.1.1 Servidor IP

Na solução proposta a transmissão de uma *stream unicast* será feita através de um servidor IP e essa *stream* irá corresponder a uma SPTS para facilitar o processo de sincronização e diminuir a quantidade de dados a serem enviados porque na STB o canal que se pretende reproduzir corresponde a uma SPTS, que é obtido pela desmultiplexagem da MPTS recebida, e deste modo é possível iniciar a reprodução e, quando for feita a sincronização, a única ação necessária é mudar a fonte.

Deste modo, o servidor IP é responsável pela receção e desmultiplexagem da MPTS transmitida pelo *headend* por DVB-C e à disponibilização de uma SPTS quando houver uma mudança de canal, feita por uma STB, como representado na Figura 4.2. Para tal, o servidor tem que ter uma interface DVB-C para receber a MPTS e uma interface de rede para transmitir as SPTS por IP.

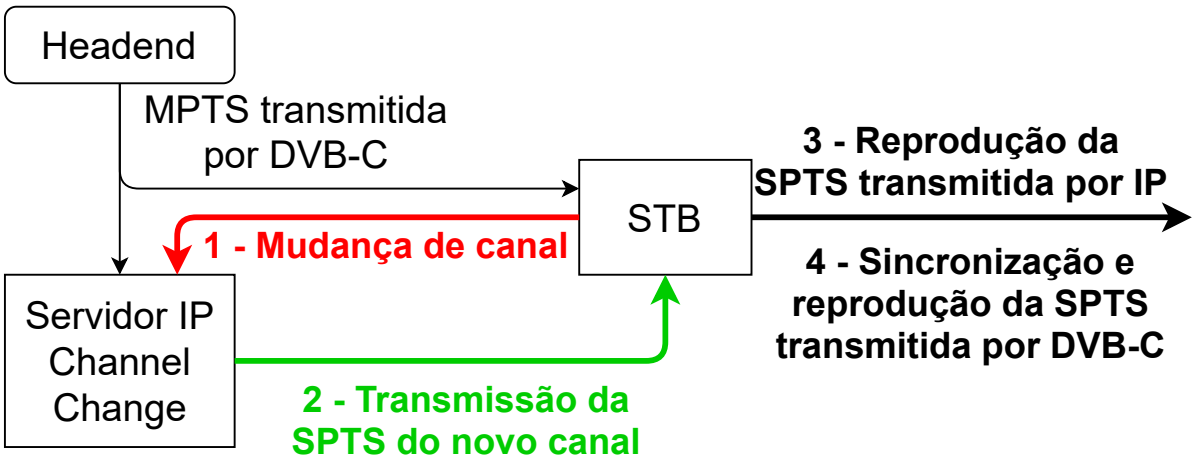


Figura 4.2: Representação do funcionamento do servidor

O servidor IP ao receber um pedido de mudança de canal feito por uma STB vai desmultiplexar a MPTS transmitida pelo *headend* para obter a SPTS referente ao canal pedido e logo que seja possível inicia-se a transmissão da mesma por IP.

Para prevenir que as *streams* transmitidas sejam acedidas por entidades não autorizadas as SPTS são protegidas utilizando o mesmo processo de *scramble* como se fossem transmitidas por DVB-C, sendo que a STB já terá os valores utilizados porque já foram recebidos na MPTS transmitida por DVB-C.

O servidor IP é constituído pela interface DVB-C que é responsável pela receção da MPTS proveniente do *headend* e pela interface de rede responsável pela transmissão da SPTS pedida pela STB, representado na Figura 4.3.

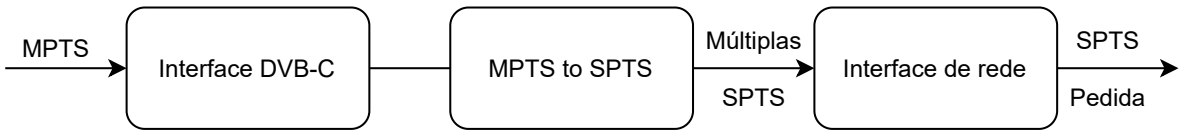


Figura 4.3: Representação da constituição do servidor

4.1.2 Recetor

Um dos requisitos técnicos desta solução é que a STB possua uma interface de rede para poder pedir e receber a *stream unicast* transmitida por IP correspondente à SPTS do novo canal a reproduzir.

Quando o utilizador muda de canal a STB comunica ao servidor IP qual é o novo canal a reproduzir e ao receber a SPTS correspondente inicia a reprodução. Paralelamente, a STB está também a receber a MPTS que chega por DVB-C e faz o processo normal para obter a SPTS do canal que se pretende reproduzir. Quando a STB tem as duas SPTS disponíveis será feita a mudança de fonte quando se encontrar um ponto de sincronismo. Este processo é ilustrado pela Figura 4.4.

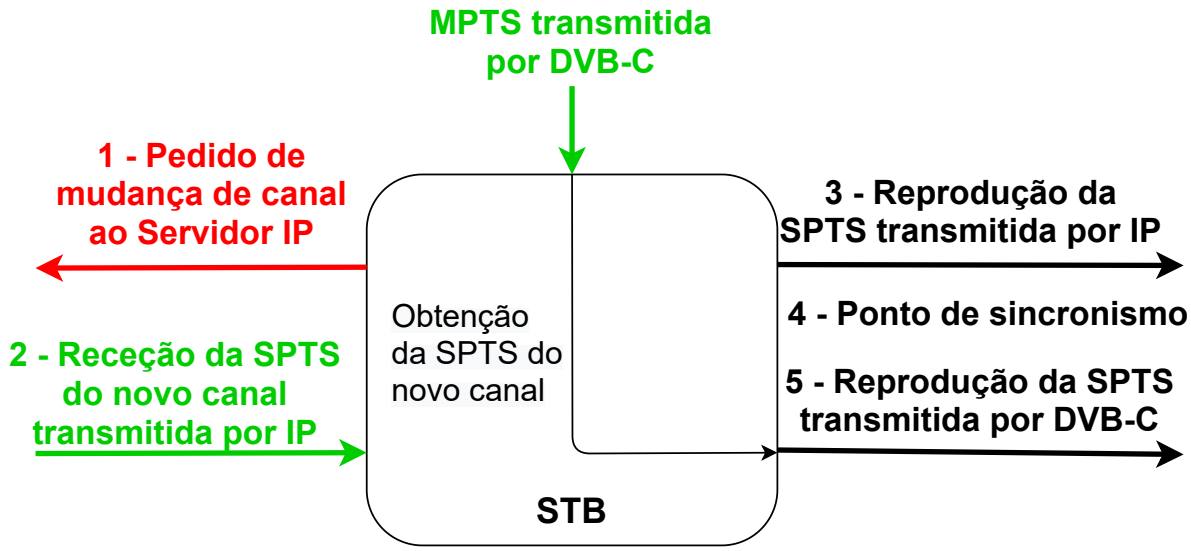


Figura 4.4: Representação do funcionamento da STB

O ponto de sincronismo corresponde ao momento em que o pacote recebido da SPTS, transmitida por IP, tem o mesmo PTS que um dos pacotes já recebidos da SPTS transmitida por DVB-C. Nesse ponto sabemos que os pacotes seguintes serão iguais e será possível mudar de fonte sem problemas e deixar de receber a SPTS transmitida por IP. Para verificar quando é que acontece o ponto de sincronismo faz-se *buffering* dos pacotes recebidos por DVB-C enquanto se reproduz a SPTS recebida por IP e vai-se verificando se o pacote recebido por IP já está contido no *buffer* com os pacotes DVB-C.

4.2 Mecanismo de sincronização

Após se obter o ponto de sincronismo é necessário sincronizar as duas SPTS para se poder mudar de fonte para garantir que não há problema na reprodução.

A sincronização é feita através dos PTS dos PES de cada SPTS. Quando se obtiver, por IP, um PES com o mesmo PTS que um PES que esteja no *buffer* dos pacotes recebidos por DVB-C pode-se admitir que os PES transmitidos por IP que chegarão depois já estão no *buffer* ou ainda não chegaram. Deste modo, quando se alcançar o ponto de sincronismo, o último PES proveniente do servidor IP a ser reproduzido é o que chegou imediatamente antes do PES com o mesmo PTS que já existia no *buffer*. A partir desse momento descartam-se os PES do *buffer* que chegaram antes do PES com o mesmo PTS e reproduzem-se os PES presentes no *buffer* e muda-se a fonte para a MPTS transmitida por DVB-C e a reprodução retorna ao sistema tradicional por DVB-C.

4.3 Comparação com o trabalho relacionado

A solução apresentada permite reduzir o tempo de mudança de canal porque ao receber diretamente a SPTS que se pretende reproduzir não é necessário esperar que a STB faça a sintonização com a portadora onde está a TS para iniciar a reprodução e não é preciso aguardar por um pacote que contenha as tabelas referentes ao PSI, pois estes valores já estão presentes na TS transmitida por DVB-C.

Das soluções apresentadas para a redução do tempo de *zapping* em DVB-C no capítulo 3 conclui-se que, possivelmente, se possam incorporar as alterações à arquitetura do sistema [10] em conjunto com a solução proposta, o que pode levar a uma redução no tempo de mudança de canal superior à apresentada.

Comparativamente com os métodos utilizados para a redução da mudança de canal em IPTV, apresentadas na secção, 3.3 não se justifica o método em que se utiliza uma *stream* de menor qualidade porque na solução apresentada a redução da mudança de canal obtém-se com a utilização de uma *stream unicast* em que se mantém a qualidade da *stream*. O método proposto para a utilização de *edge servers* quando implementado em conjunto com a solução proposta pode resultar num tempo de *zapping* inferior ao obtido através da solução proposta isolada.

5

Implementação

Neste capítulo é feita uma introdução à *framework GStreamer* [4], utilizada para simular a solução proposta, e descreve em detalhe a implementação da simulação correspondente à solução apresentada para a redução no tempo de mudança de canal no sistema DVB-C.

5.1 *Gstreamer*

Para fazer a simulação do sistema utilizou-se o *GStreamer* [4] que é uma *framework open source* que implementa a reprodução de conteúdos através da criação de *pipelines* para processamento de dados. A *framework GStreamer* é amplamente utilizada para desenvolvimento de aplicações que lidam com *streams* multimédia em diversos sistemas, em particular nos que se baseiam em Linux.

Como o *GStreamer* é utilizado como base na maioria das STB do sistema DVB-C, como também pelas aplicações que estas suportam, e por se tratar de uma plataforma *open source* apresenta-se como uma boa base para o desenvolvimento da solução a propor no âmbito deste projeto e permite, que em ambiente de simulação, se modele o sistema DVB-C e facilita a integração da solução proposta num sistema já existente.

Deste modo, o *GStreamer* é utilizado para simular o funcionamento de um sistema com um *headend* que faça o envio da TS, um servidor IP que receba essa TS, faça a sua desmultiplexagem e disponibilize cada canal individualmente na sua própria TS, e um

recetor que seja capaz de receber duas TS, uma que venha diretamente do *headend*, correspondente à *stream* obtida por DVB-C, e outra proveniente do servidor, que simulará a TS que chega por IP.

5.1.1 Funcionamento do *Gstreamer*

Para se compreender como funciona o *GStreamer* é preciso conhecer as suas bases [17], nomeadamente, saber que um *pipeline* é composto por elementos (também conhecidos como *plugins*) e que cada elemento tem pelo menos um *pad* e que os *pads* são a interface do elemento.

5.1.2 Bases do *Gstreamer*

As bases do *GStreamer* permitem compreender o seu funcionamento.

5.1.2.1 *Pipeline*

Um *pipeline* é um contentor de elementos, ligados entre si, e que é responsável pela sincronização entre eles. Um exemplo de um *pipeline* para reprodução de um ficheiro com vídeo e áudio está representado na Figura 5.1 [18].

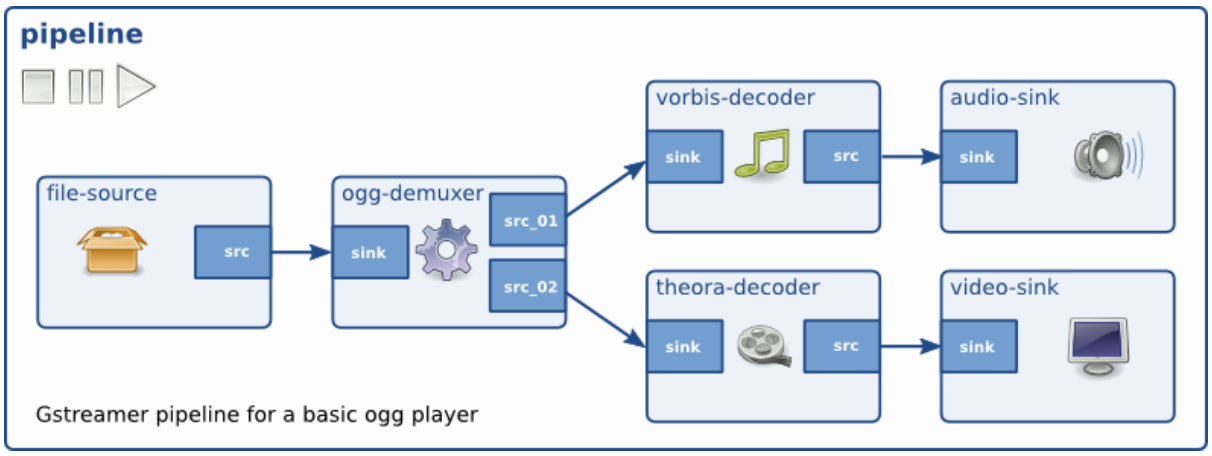


Figura 5.1: Exemplo de um *pipeline* para reprodução de um ficheiro com vídeo e áudio

5.1.2.2 Elementos

Os elementos são a base do *GStreamer*. Os elementos são constituídos por *pads*, representados na Figura 5.2 [18], e podem ter uma de três funções:

- Ler dados de um ficheiro ou da rede.
- Descodificar ou processar dados.
- Fazer o *output* dos dados, seja para reprodução de vídeo ou transmissão dos mesmos pela rede.

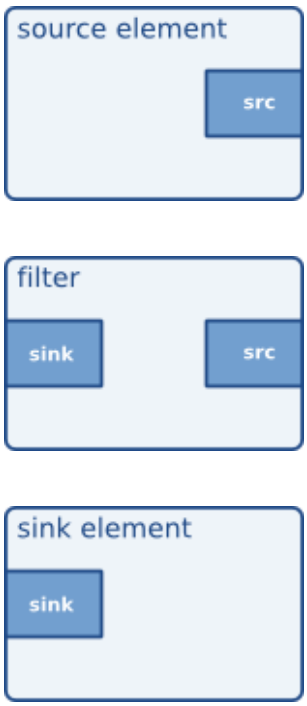


Figura 5.2: Exemplo de diversos elementos e dos seu *pads*

Geralmente são criadas cadeias de elementos onde se processam dados, denominadas como *pipelines*.

5.1.2.3 *Pads*

Os *pads* são a interface do elemento com o exterior. Existem dois tipos de *pads*, os *pads* que representam a saída do elemento, chamados de *source pads* (identificados como *src*) que são responsáveis por transmitir os dados para fora do elemento e os *pads* que representam a entrada do elemento, chamados de *sink pads* (identificados como *sink*) que recebem dados do exterior.

Os *pads* são caracterizados face à sua disponibilidade [19] podendo ser *always pads*, *request pads* ou *sometimes pads*.

Um *always pad* é um *pad* que existe sempre e está sempre disponível, são os mais comuns.

Um *request pad* é um *pad* que não está disponível a menos que seja pedido, são utilizados em elementos em que o número de entradas ou saídas é variável, como por exemplo um *multiplexer* ou um elemento que divide os dados recebidos por várias saídas.

Um *sometimes pad* é um *pad* que não está sempre disponível a quando a criação do elemento, podendo apenas ser criado quando recebe dados, são comuns em elementos com função de *demuxer* em que um dos *src pad* apenas é criado se os dados que transmitiria estiverem presentes na *stream* recebida.

Um *pad* pode ter uma ou várias ***probes*** [20] que são *callbacks* que informam a aplicação sobre o estado do fluxo de dados.

5.1.3 Aplicação *Gstreamer*

Como ponto de partida e introdução ao *GStreamer* foi desenvolvida uma aplicação para reprodução de um canal presente numa TS. Esta aplicação tem como finalidade apresentar alguns dos elementos/*plugins* utilizados na modelação do sistema e as suas funções.

A aplicação é composta por um *pipeline* que partindo de um ficheiro que contém uma MPTS obtém o vídeo e o áudio do canal escolhido e faz a sua reprodução corretamente. Esta aplicação é composta pelos seguintes elementos, que formam o *pipeline* apresentado na Figura 5.3:

- *Source*: define qual é o ficheiro a reproduzir;
- *Demuxer*: define o programa da TS a ser reproduzido;
- *Audio queue e Video queue*: fazem a sincronização do áudio e do vídeo;
- *Audio Parse e Video Parse*: fazem o *parse* do áudio e do vídeo;
- *Audio Decoder e Video Decoder*: fazem o *decode*, descodificação dos dados recebidos para *raw media*, do áudio e do vídeo;
- *Audio Convert e Video Convert*: fazem a conversão dos dados *raw* de áudio e de vídeo recebidos para o formato em que vão ser reproduzidos;
- *Audio Sink e Video Sink*: fazem a reprodução do áudio e do vídeo.

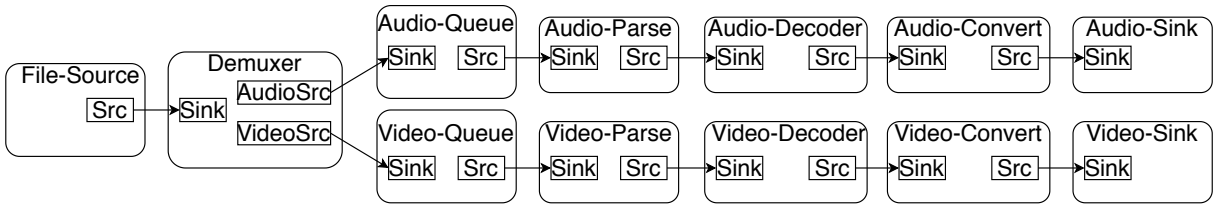


Figura 5.3: *Pipeline* da aplicação para reprodução de um canal

O elemento ou *plugin* utilizado como *source*, com a função de ler o ficheiro, foi o *filesrc* em que o caminho para o ficheiro é definido na propriedade *"location"*.

Para desmultiplexar a TS é utilizado o *plugin tsdemux* em que o programa a reproduzir é definido pela propriedade *"program-number"*.

A *audio queue* e a *video queue* utilizam ambas o *plugin queue*, no entanto, é iniciada uma *queue* para cada tipo de dados.

O *parse* do áudio é feito pelo *plugin mpegaudioparse* e o *parse* do vídeo pelo *plugin mpegvideoparse*.

Tal como acontece com as *queues*, o *audio decoder* e o *video decoder* utilizam o mesmo *plugin*, o *decodebin*.

A conversão dos dados é feita pelos *plugins audioconvert* e *videoconvert*, para o áudio e vídeo, respetivamente.

Por último, a reprodução do áudio é feita pelo *plugin autoaudiosink* e a reprodução do vídeo é feita pelo *plugin autovideosink*.

5.1.4 Obter uma SPTS

De modo a encontrar o ponto de sincronismo é necessário desmultiplexar uma MPTS nas SPTS que a constituem e, que para serem reproduzidas, devem conter algumas das tabelas referidas na secção 2.6, principalmente a PAT e a PMT.

A obtenção de uma SPTS começou pela análise do *plugin tsdemux*, utilizado no *pipeline* da secção 5.1.3. Este *plugin* recebe uma TS, independentemente de ser uma MPTS ou uma SPTS, e na sua saída apenas fornece as ES do canal a reproduzir, o que não permite obter uma SPTS. No entanto, analisando o código do *tsdemux*, verificou-se que para se obter as ES de cada SPTS era utilizado o *plugin tsparse*.

O *plugin tsparse* possui um *pad* de saída por pedido o "*program_%u*", no qual substituindo "*%u*"pelo número do programa que se pretende reproduzir obtém-se a SPTS desse programa.

A fim de comprovar que a utilização do *tsparse* é viável utilizou-se o *pipeline* ilustrado na Figura 5.4 para guardar a SPTS em ficheiro, utilizando o *plugin filesink*, e para fazer a reprodução utilizou-se o *pipeline* descrito na secção 5.1.3.

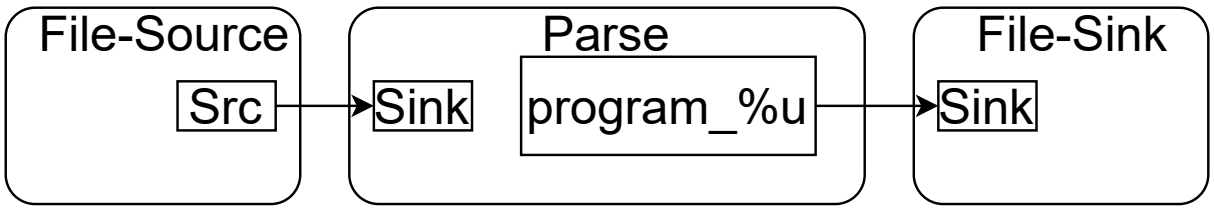


Figura 5.4: Representação do *pipeline* utilizado para obter um ficheiro com a *SPTS* do programa que se pretende reproduzir

Apesar da reprodução ter funcionado, para verificar que as tabelas necessárias para fazer a reprodução estavam presentes utilizou-se o programa *MPEG-2 Transport Stream packet analyser* [21] que confirmou que as mesmas estavam presentes, mas apenas uma vez, ao contrário do que seria espectável e que acontece na MPTS original que é a inclusão das tabelas em vários pacotes ao longo da TS.

Como na simulação a SPTS não será guardada em ficheiro, testou-se também a utilização do *tsparse* num único *pipeline*, adaptando o *pipeline* da secção 5.1.3 introduzindo o *tsparse* entre o *filesrc* e o *tsdemux*, que resulta no *pipeline* ilustrado na Figura 5.5. No entanto, para se conseguir que este *pipeline* funcionasse foi necessário editar o *plugin tsdemux*. Esta alteração permitiu a reprodução da SPTS sem causar problemas com a reprodução.

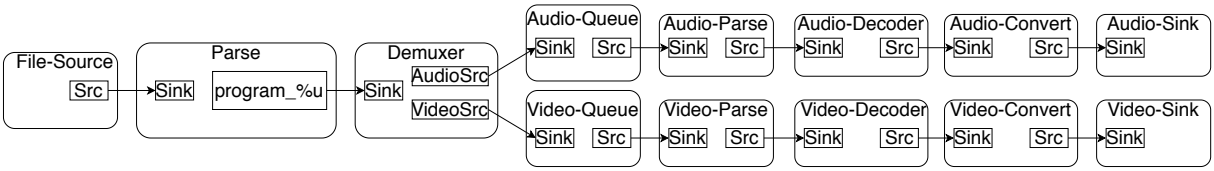


Figura 5.5: Representação do *pipeline* utilizado para obter e reproduzir uma *SPTS*

5.2 Modelação do sistema

De modo a simular a solução proposta no capítulo 4 foi necessário desenvolver três aplicações para representar os componentes do sistema, o *headend*, o servidor IP e o recetor.

5.2.1 Modelação do *headend*

Para modelação do *headend* foi desenvolvida uma aplicação *GStreamer*, representada na Figura 5.6, constituída por dois *pipelines* idênticos responsáveis pela transmissão de uma MPTS. É necessária a utilização de dois *pipelines* porque é necessário introduzir um atraso na transmissão por IP e o início da transmissão da MPTS tem que ser iniciada ao mesmo tempo.

O primeiro *pipeline* simula a difusão de uma TS por DVB-C e o segundo *pipeline* simula a transmissão da TS que vai chegar ao servidor IP. Este segundo *pipeline* apenas é utilizado para criar um atraso temporal na entrega dos dados no servidor IP e por consequência no recetor. A forma que se optou para introduzir esse atraso em simulação foi a criação de um *pipleine* idêntico ao primeiro, mas que inicia a transmissão mais tarde.

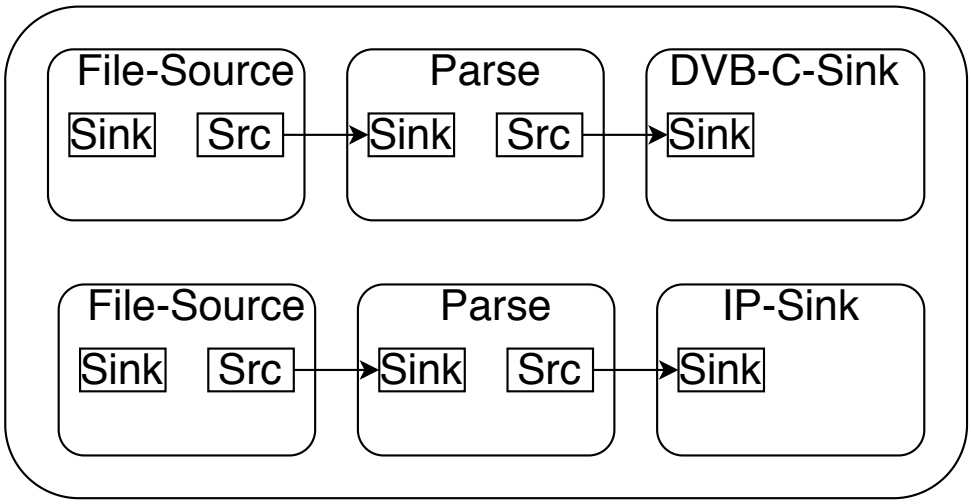


Figura 5.6: Representação dos *pipelines* utilizados na aplicação do *headend*

Para fazer transmissão de *streams* de dados por IP o *GStreamer* possui os protocolos da camada de transporte mais utilizados, o *Transmissions Control Protocol* (TCP) e o *User Datagram Protocol* (UDP).

Para fazer a transmissão utilizando UDP são necessários os *plugins udpsink*, que é o último *plugin* da *pipeline* do *headend* e serve para enviar a *stream* pela rede, e o *udpsrc*, que é o primeiro *plugin* da *pipeline* do recetor que recebe os dados da *stream* transmitidos pela rede. No entanto, para fazer a transmissão de uma TS por UDP é, por limitação dos *plugins* do *GStreamer*, necessário encapsular ou codificar a *stream* antes de ser transmitida e consequentemente desencapsular ou descodificar depois de ser recebida.

Para fazer a transmissão utilizando o TCP são utilizados os *plugins tcpserversink* ou *tcpclientsink* para transmitir dados pela rede, ou os *plugins tcpserversrc* ou *tcpclientsrc* para receber dados pela rede e, ao contrário do que acontece em UDP, não é necessário codificar ou encapsular dados.

No âmbito do trabalho, o protocolo de transmissão escolhido foi o TCP de modo a não se introduzir nenhum tipo de processamento extra proveniente do encapsulamento ou codificação de dados que poderiam ter impacto na solução proposta.

Deste modo, nos *pipelines* do *headend*, representados na Figura 5.6, o "DVB-C-Sink"e o "IP-Sink"correspondem ao *plugin tcpserversink* e para se conseguir fazer a transmissão é necessário a utilização do *plugin tsparse*, entre a *source* e o *sink*, com a propriedade *"set-timestamps"* a *TRUE* de modo a definir *timestamps*, que permitem a transmissão da TS, sendo que estes *timestamps* não têm influência nos PTS nem nos DTS da TS.

5.2.2 Modelação do servidor IP

A modelação do servidor IP é feita por uma aplicação formada por um *pipeline*, representado na Figura 5.7, que recebe uma MPTS transmitida por IP e disponibiliza as SPTS dos canais presentes na MPTS.

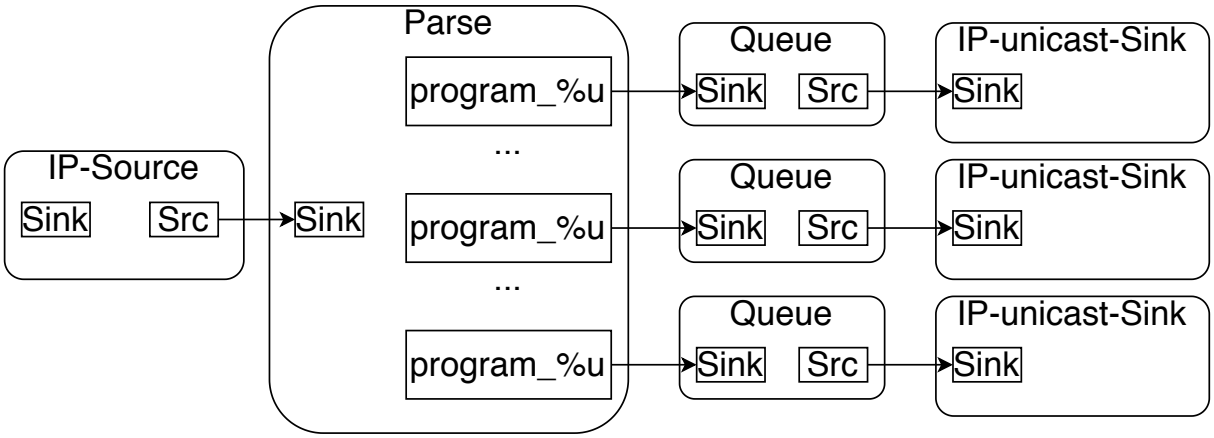


Figura 5.7: Representação do *pipeline* do servidor IP

A receção da MPTS é feita utilizando o *plugin tcpclientsrc*. Posteriormente utiliza-se o *tsparse* que disponibiliza um *pad* para cada uma das SPTS presentes na MPTS. De forma a que as SPTS estejam sincronizadas, utiliza-se uma *queue* antes de serem transmitidas por TCP, utilizando o *tcpserversink*, de modo a estarem disponíveis mais depressa quando for solicitada a mudança de canal.

5.2.3 Modelação do recetor

O recetor é modelado por uma aplicação constituída por um *pipeline*, representado na Figura 5.8, que recebe duas TS, começa por reproduzir a TS que chega por IP e ao alcançar o ponto de sincronismo inicia a reprodução da TS que chega por DVB-C.

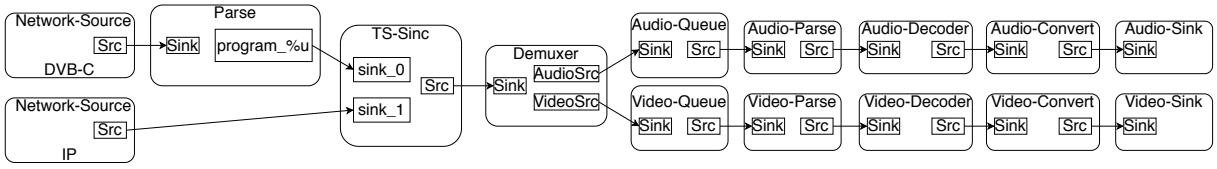


Figura 5.8: Representação do *pipeline* do recetor

Tal como na aplicação do servidor, a receção das TS é feita pelo *plugin tcpclientsrc*. Neste pipeline temos um *tcpclientsrc* que recebe a MPTS proveniente do *headend* e outro *tcpclientsrc* que recebe a SPTS proveniente do servidor IP.

Como no sistema real a *stream* transmitida por IP é processada no servidor IP irá chegar ao recetor depois da *stream* transmitida por DVB-C. Todavia, a MPTS transmitida por DVB-C ainda tem que ser processada e a SPTS já está pronta a ser reproduzida.Assim sendo, na simulação introduzimos este tempo para processamento da MPTS ignorando os pacotes que chegam por DVB-C, introduzindo o *plugin identity*, depois do *plugin tcpclientsrc* que recebe a SPTS proveniente do servidor IP, que tem uma *probe* que quando começa a receber dados faz um *callback* a uma *probe* introduzida no *pad* do *plugin tcpclientsrc* que recebe a MPTS proveniente do *headend* e sinaliza que os pacotes que chegam são para descartar enquanto se aguarda pela receção de dados no *tcpclientsrc* que recebe a SPTS proveniente do servidor IP.

Deste modo, a SPTS proveniente do servidor chega primeiro, mas está atrasada em relação à TS proveniente do *headend*, e inicia-se a sua reprodução, utilizando o *plugin tsdemux* modificado, sem se indicar o programa a reproduzir, e os elementos *queue*, *mpegaudioparse*, *mpegvideoparse*, *decodebin*, *audioconvert*, *videoconvert*, *autoaudiosink*, como descritos na secção 5.1.3, e o *plugin ximagesink* para reprodução do vídeo.

Já com a SPTS transmitida por IP em reprodução, obtém-se a SPTS do canal a reproduzir utilizando o *plugin tsparse* para desmultiplexar a MPTS que chega proveniente do *headend* e o *plugin* "TS-Sinc", que foi desenvolvido para fazer a sincronização entre duas SPTS, faz *buffering* dos pacotes que da SPTS transmitida por DVB-C e verifica se os pacotes que estão a chegar por IP já estão presentes no *buffer*. Se o pacote mais recente já estiver presente no *buffer* está encontrado o ponto de sincronismo e passa-se a reproduzir a *stream* transmitida por DVB-C.

5.2.4 Funcionamento da simulação

As aplicações são iniciadas através da linha de comandos do Linux.

A primeira aplicação a ser iniciada é a do *headend* que começa logo a transmitir as TS. Seguidamente, inicia-se a aplicação referente ao servidor IP que recebe a MPTS transmitida por IP, separa-a nas SPTS referentes a cada canal e inicia a transmissão de cada uma pela rede. Por último, inicia-se a aplicação do recetor em que o programa a reproduzir está definido no código da aplicação.

As aplicações têm que ser iniciadas pela ordem em que foram apresentadas porque o *plugin* utilizado para fazer a transmissão das TS, o *tcpserversink*, inicia a transmissão automaticamente, mesmo que o *pipeline* que irá receber os dados ainda não tenha sido criado, e como o *plugin tsparse* apenas inclui a informação necessária para sincronização das ES no início da TS, logo, o recetor tem que ser iniciado em último porque se começar a receber a *stream* sem obter o princípio da TS não irá conseguir fazer a reprodução da mesma.

5.3 Desenvolvimento de um *Plugin* do *Gstreamer*

Nesta secção é explicado o *plugin*, representado na Figura 5.9, desenvolvido para fazer a sincronização entre duas SPTS que se encontram desfasadas temporalmente.

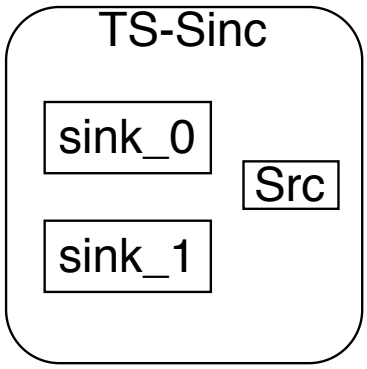


Figura 5.9: Representação do *plugin* para sincronização das TS

O *plugin* é constituído por dois *sink pads* e um *source pad*, todos caracterizados como *always pads*.

No *pad* "sink_0"recebe-se a TS proveniente do *headend* e no *pad* "sink_1"recebe-se a TS proveniente do servidor IP. O *plugin* "Src"faz o *output* da TS a reproduzir, se ainda não foi alcançado o ponto de sincronismo esta TS corresponde à SPTS proveniente do servidor IP, se já se alcançou o ponto de sincronismo a TS corresponde à SPTS que foi desmultiplexada a partir da MPTS que é transmitida por DVB-C.

O *plugin* tem as mesmas fundações que os *plugins tsparse* e *tsdemux*.

O funcionamento do *plugin* é o seguinte:

1. Se o *plugin* não está a receber nenhum *buffer* do *GStreamer* comunica que a *stream* acabou;

2. Se o *plugin* apenas estiver a receber buffers de uma fonte faz *output* desses mesmos buffers;

3. Se o *plugin* estiver a receber buffers de ambas as fontes:

   (a) Se já foi alcançado o ponto de sincronismo ignora-se os buffers que chegam por IP e faz-se *output* dos buffers que chegam por DVB-C;

   (b) Se ainda não foi alcançado o ponto de sincronismo:

i. Verifica-se se ainda não existe uma lista com os buffers que chegam por DVB-C, cria-se a lista, se a lista existir, mas estiver no seu tamanho máximo apaga-se a lista e cria-se uma nova. Depois introduz-se o *buffer* na lista;

ii. Com o *buffer* que chega por IP forma-se um pacote da TS e verifica-se se o PTS desse pacote é igual ao PTS de algum dos pacotes já existentes na lista de buffers (com os buffers da lista formam-se os pacote da TS correspondentes):

   A. Se não se encontrar nenhum PTS igual faz-se *output* do *buffer* que chega por IP;

   B. Quando se encontrar o mesmo PTS, significa que o ponto de sincronismo foi alcançado, apaga-se a lista até ao *buffer* com o mesmo PTS e faz-se *output* do resto da lista.

5.4 Resultados Experimentais

De modo a testar o ambiente de simulação montado foi utilizado um ficheiro constituído por uma MPTS com diversos canais da televisão portuguesa.

Os testes realizados consistiram na múltipla reprodução das SPTS presentes no ficheiro utilizado para garantir que a aplicação referente ao recetor e o *plugin* desenvolvidos funcionavam.

Para se saber quando ocorreu a mudança de fonte o *plugin* escreve na linha de comandos que agora se está a reproduzir a *stream* que chega por DVB-C e para comprovar este facto desliga-se a aplicação do servidor IP e a reprodução continua sem problema.

O facto de não ser percetível quando ocorre a mudança de fonte é um bom indicador que o *plugin* funciona.

6

Conclusões

Em suma, consegui-se com este trabalho sincronizar duas TS desfasadas temporalmente que soluciona a sincronização entre uma *stream* difundida por DVB-C e uma *stream unicast* transmitida sobre IP.

Em termos de simulação a mudança de canal é simulada no início da transmissão das TS com o atraso introduzido na transmissão da MPTS com destino ao servidor. Por limitação dos *plugins* TCP para transmissão e receção das TS não foi possível exibir uma transmissão *unicast* sobre IP, logo, as transmissões das TS começam quando as aplicações são iniciadas. Também como o *plugin tsparse* apenas inclui a informação que permite a reprodução da *stream* nos primeiros pacotes da TS, a mudança de canal no recetor não é possível pois se apenas começar a receber pacotes a meio da TS não é possível processar a *stream* para ser reproduzida. Para superar esta limitação teria-se que alterar o *plugin tsparse* existente de forma a serem guardadas as tabelas necessárias e posteriormente enviadas como parte da SPTS do programa. Só assim seria possível o recetor fazer mudar de canal dinamicamente.

Como resultado deste trabalho foi desenvolvido um *plugin GStreamer*, com o objetivo de ser incorporado nos recetores do sistema DVB-C, que faz a sincronização de duas TS, uma recebidas por DVB-C e outra recebida por IP.

Como trabalho futuro é necessário alterar o *plugin tsparse*, pedir e transmitir a SPTS do canal a reproduzir por *unicast* e implementar uma mudança de canal dinâmica na aplicação do recetor.

De modo a comprovar que a redução de mudança de canal obtida no ambiente simulado se refletiriam no sistema real seria necessário fazer testes práticos, mas tais não foram possíveis devido à necessidade de equipamentos próprios para obter estes resultados.

Referências

[1] ETSI EN 301 192. Digital Video Broadcasting (DVB); DVB specification for data broadcasting, 2016. URL https://dvb.org/wp-content/uploads/2019/12/a027_dvb-data_spec.pdf.

[2] ANACOM. Número de clientes de pacotes de serviços aumenta mais de 6% em 2017. URL https://www.anacom.pt/render.jsp?contentId=1434151.

[3] ETSI EN 300 429. Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for cable systems, 1998. URL https://www.etsi.org/deliver/etsi_en/300400_300499/300429/01.02.01_60/en_300429v010201p.pdf.

[4] Gstreamer. URL https://gstreamer.freedesktop.org/.

[5] Fluendo. What we do at Fluendo. URL https://fluendo.com/en/multimedia-codecs-gstreamer/.

[6] GNOME. Rhythmbox Basic information. URL https://wiki.gnome.org/Apps/Rhythmbox/FAQ#Basic_information.

[7] DVB. About. URL https://dvb.org/about/.

[8] ISO/IEC 13818-1:2019. Information technology - Generic coding of moving pictures and associated audio information - Part 1: Systems, 2019. URL https://www.iso.org/standard/75928.html.

[9] ISO/IEC 14496-1:2010. Information technology - Coding of audio-visual objects - Part 1: Systems, 2010. URL https://www.iso.org/standard/55688.html.

[10] N. Fimic, I. Basicevic, and N. Teslic. Reducing channel change time by system architecture changes in dvb-s/c/t set top boxes. IEEE Transactions on Consumer Electronics, 65(3):312–321, 2019. URL https://ieeexplore.ieee.org/document/8700296.

[11] Chunglae Cho, Intak Han, Yongil Jun, and Hyeongho Lee. Improvement of channel zapping time in iptv services using the adjacent groups join-leave method. In The 6th International Conference on Advanced Communication Technology, 2004., volume 2, pages 971–975, 2004. URL https://ieeexplore.ieee.org/document/1293012.

[12] I. Basicevic, D. Kukolj, S. Ocovaj, G. Cmiljanovic, and N. Fimic. A fast channel change technique based on channel prediction. IEEE Transactions on Consumer Electronics, 64(4):418–423, 2018. URL https://ieeexplore.ieee.org/document/8488560.

[13] Fast Channel Change Solutions. DVB-SCENE, 34. URL https://dvb.org/wp-content/uploads/2019/12/DVB-SCENE34.pdf.

[14] IPTV Hybrid Solution for Hotel K99 in Radolfzell, Germany. URL https://axing.com/en/referenzen/iptv-hybrid-solution/.

[15] Damodar Banodkar, K. K. Ramakrishman, Shivkumar Kalyanaraman, Alexandre Gerber, Oliver Spatscheck, Rensselaer Polytechnic Institute (RPI), and AT&T Labs Research. Multicast Instant Channel Change in IPTV Systems. IEEE, pages 370–379, 2008. URL https://ieeexplore.ieee.org/document/4554442.

[16] Harald Fuchs and Nikolaus Färber. Optimizing channel change time in IPTV applications. IEEE, pages 1–8, 2008. URL https://ieeexplore.ieee.org/document/4536619.

[17] Gstreamer. Foundations, . URL https://gstreamer.freedesktop.org/documentation/application-development/introduction/basics.html?gi-language=c.

[18] Gstreamer. Dynamic pipelines, . URL https://gstreamer.freedesktop.org/documentation/tutorials/basic/dynamic-pipelines.html?gi-language=c.

[19] Gstreamer. Pads and capabilities, . URL https://gstreamer.freedesktop.org/documentation/application-development/basics/pads.html?gi-language=c.

[20] Gstreamer. Probes, . URL https://gstreamer.freedesktop.org/documentation/additional/design/probes.html?gi-language=c#usecases.

[21] MPEG-2 Transport Stream packet analyser. URL http://www.pjdaniel.org.uk/mpeg/.