FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO
Departamento de Engenharia Electrotécnica e de Computadores

Suporte de Serviços MPEG-2 em Redes de Banda Larga

Joaquim Fernando Fernandes da Silva

Licenciado em Engenharia Electrotécnica e de Computadores pela
Faculdade de Engenharia da Universidade do Porto

Dissertação submetida para satisfação parcial dos requisitos
do grau de mestre
em
Engenharia Electrotécnica e de Computadores
(Área de especialização de Telecomunicações)

Dissertação realizada sob a supervisão de
Professor Doutor José Ruela Fernandes,
do Departamento de Engenharia Electrotécnica e de Computadores
da Faculdade de Engenharia Universidade do Porto.

Porto, Setembro de1997

Resumo

Assiste-se actualmente a rápidas e profundas transformações nas redes de comunicações. A comunicação analógica de sons e imagens deu lugar ao digital e este ao multimédia.

O objectivo central da dissertação foi o estudo de duas tecnologias emergentes no mundo das comunicações audiovisuais, o MPEG-2 e o ATM.

O MPEG-2 é uma norma destinada à codificação e transmissão de informação audiovisual, enquanto a tecnologia ATM é particularmente adequada à transmissão de sinais multimédia, visto que permite efectuar transporte integrado de tráfego multi-débito, suportando com eficiência e flexibilidade diferentes requisitos de uma grande diversidade de aplicações.

Neste âmbito a tese teve como objectivos práticos a criação de um Selector de programas MPEG-2 e a transmissão de tramas MPEG-2 num canal ATM.

O modo de descodificar multi-programas, a escolha do canal de adaptação, o método do encapsulamento dos pacotes MPEG-2 em pacotes AAL foram as questões abordadas.

A descodificação de multi-programas baseou-se na Norma MPEG-2 Sistema em particular a utilização das tabelas PSI (Programme Specific Information), enquanto a escolha do canal de adaptação ATM recaiu sobre o AAL5 visto ser o adoptado pelo ATM Forum para transmissão VBR MPEG-2 em redes ATM.

Palavras Chave:

MPEG-2, MPEG-2 Camada de Sistemas, redes ATM, Camada de Adaptação ATM.

Abstract

Nowadays we are assisting to fast changes in communications. The sound and images transmitted by analogue communications have changed to digital and this one to the multimedia.

The main goal of this thesis was the study of two emergent technologies in the audio-visual communication's world, the MPEG-2 (Motion Pictures Expert Group) and the ATM (Asynchronous Transfer Mode).

The MPEG-2 is a standard designed to the codification and transmission of audio-visual information, whereas ATM technology is specifically suitable to multimedia signals transmission, since it allow integrated transport of multi-rate traffic, supporting with efficiency and flexibility different requirements of a vast range of applications.

The work carried out aimed at the creation of MPEG-2 programme Selector and MPEG-2 stream transmission in an ATM channel.

The way to demultiplex multi programmes, the ATM adaptation layer to select and the procedure of packing MPEG-2 in AAL PDUs were the main areas addressed.

The MPEG-2 standard (Systems Part) was used for multi programme demultiplexing, in particular the PSI (Programme Specific Information) tables. As far as the ATM adaptation layer protocol, AAL5 has been selected, since this was the ATM Forum choice for the transport of VBR MPEG-2 over ATM.

Keywords:

MPEG-2, MPEG-2 Systems Layer, ATM networks, ATM Adaptation layers.

Résumé

On assiste aujourd'hui à des transformations profonds et rapides dans les réseaux de communications. La communication analogique de sones et images a rendu place au digital et ce-ci au multimédia.

Le travail s'est basé sur deux technologies émergentes dans le Monde des Communications audiovisuelles, le MPEG-2 et l'ATM.

Le MPEG-2 est une norme destinée à la codification et transmission d'information audiovisuelle, tandis que la technologie ATM est particulièrement appropriée pour la transmission de signaux multimédia, puisqu'elle permet d'effectuer le transport integré de traffic multidébit et supporter avec efficacité et flexibilité un grand nombre d'applications très différents.

Dans ce contexte, la thèse eut comme objectives pratiques la création d'un sélecteur de programmes MPEG-2 et la transmission de trames MPEG-2 dans un canal ATM.

Le mode de décodifier multiprogrammes, le choix du canal d'adaptation, le méthode d'encapsulement des paquets MPEG-2 en AAL étaient les principaux questions addressées.

Par décodifier multiprogrammes on a utilizé la norme MPEG-2 système, en particulier les tables PSI (*Programme Specific Information*), tandis que le choix du canal d'adaptation ATM fut le AAL5 car il a été adopté par l'ATM Forum pour la transmission VBR MPEG-2 en réseaux ATM.

Agradecimentos

Agradeço ao meu orientador, Professor Doutor José Ruela Fernandes, a disponibilidade e colaboração prestadas durante a realização desta dissertação.

À Engenheira Teresa Andrade pela pronta disponibilidade nas explicações e nas sugestões que muito contribuíram para o desenvolvimento desta tese.

Ao grupo de Codificação de Áudio e Vídeo do INESC do PORTO pelo apoio prestado, durante a realização deste trabalho.

Ao INESC PORTO pelas condições de trabalho que me proporcionou, sem as quais não seria possível este trabalho.

Não posso também de deixar expressar o meu reconhecimento a todas as pessoas que de alguma forma tornaram possível a realização desta Tese.

Expresso aqui também o apoio prestado pelo Sub-Programa Ciência e Tecnologia do 2º Quadro Comunitário de Apoio durante a edição da tese.

Finalmente, agradeço à FEUP a possibilidade deste Mestrado.

Porto, 30 de Setembro de 1997

Joaquim Fernando Fernandes Silva

Índice de Conteúdos

Índice de Figuras

Índice de Tabelas

Glossário

Reúnem-se neste "Glossário" os termos utilizados neste trabalho.

Este glossário foi dividido em três partes. A primeira parte é relativa a termos utilizados na Norma MPEG-2, a segunda parte corresponde aos termos referentes ao ATM e a terceira a termos gerais.

MPEG-2

DSM-CC – Digital Storage Media-Command and Control
DTS – Decoding Timestamp
HDTV - High Definition Television
MPEG – Motion Pictures Expert Group
MPTS – Multi Program Transport Stream
PES – Packetised Elementary Stream
PAT – Program Association Table
PCR – Program Clock Reference
PMT – Program Map Table
PS – Program Stream
PTS – Presentation Time stamp
PSI – Program Specific Information
STC - System Time Clock
TS – Transport Stream

ATM

AAL – ATM Adaptation Layer
ABR – Available Bit Rate
ATM – Asynchronous Transfer Mode
B-ISDN – Broadband Integrated Services Digital Network
CBR - Constant Bit Rate
CER – Cell Error Ratio
CDV – Cell Delay Variation
CLP – Cell Loss Priority
CLR – Cell Loss Ratio
CRC – Cyclic Redundancy Check
CPCS – Common Part Convergence Sublayer
CS – Convergence Sublayer
MBS – Maximum Burst Size

MCR – Minimum Cell Rate
MCTD – Maximum Cell Transfer Delay
NRT-VBR – Non Real-Time Variable Bit Rate
PCR – Peak Cell Rate
PDU – Protocol Data Unit
PVC – Permanent Virtual Connection
PVP – Permanent Virtual Path
RM – Resource Management
RT-VBR – Real-Time Variable Bit Rate
SAR – Segmentation And Reassembly
SCR – Sustainable Cell Rate
SVC – Switched Virtual Connection
UBR – Unspecified Bit-Rate
UNI – User-Network-Interface
VBR – Variable Bit Rate
VC – Virtual Connection, Virtual Channel
VCC – Virtual Channel Connection
VCI – Virtual Channel Identifier
VP – Virtual Path
VPC – Virtual Path Connection
VPI – Virtual Path Identifier

Geral

ADSL – Asymmetrical Digital Subscriber Loop
CATV – Common Antenna Television
DCT – Discrete Cosine Transform
DQDB - Distributed Queue Dual Bus
FDDI - Fibber Distributed Data Bus
HDSL – High Speed Digital Subscriber Line
IEC – International Elecrotechnical Commission
ISO – International Organisation for Standardisation
ITU-T – International Telecommunications Union-Telecommunication Sector
JPEG – Joint Pictures Expert Group
PC – Personal Computer
QoS – Quality of Service
RFC – Request For Comments
SMDS – Switched Multimegabit Data Service
SONET – Synchronous Optical Network
STT - Set Top Terminal
VoD - Video-on-Demand

Capítulo 1

Introdução

A norma MPEG-2 destina-se à codificação e transmissão de informação audiovisual. Esta norma tem vindo a ser alvo de interesse por parte da indústria e operadores de telecomunicações, sendo de esperar que venha a ser utilizada em aplicações que envolvam a transmissão de sequências de vídeo de alta qualidade e som associado.

A tecnologia ATM é particularmente adequada à transmissão de sinais multimédia, visto que permite efectuar o transporte integrado multi-débito, suportando com eficiência e flexibilidade diferentes requisitos de uma grande diversidade de aplicações.

MPEG e ATM são assim duas tecnologias que apresentam grandes potencialidades para fornecerem as mais promissoras soluções para as telecomunicações do futuro.

1.1 O estado da arte

Temos assistido a um aumento crescente, à escala mundial, do interesse em aplicações multimédia. Por outro lado as redes de comunicações existentes não eram capazes de assegurar a largura de banda necessária aos novos serviços audiovisuais agora emergentes.

Desta forma estavam criados os requisitos para o desenvolvimento de uma Rede de Banda Larga com Integração de Serviços (RDIS-BL).

O modo de transferência assíncrono, juntamente com o protocolo de adaptação são considerados uma tecnologia ideal para a nova rede de comunicações. Permitem satisfazer serviços e aplicações com requisitos de largura de banda e qualidade de serviço (QoS) muito distintos para cada ligação.

O MPEG-2 é uma norma para compressão de áudio e vídeo, vocacionada para a televisão digital, capaz de explorar a redundância temporal e espacial para conseguir compressões até 200:1 e codificar uma fonte de som ou imagem com diferentes níveis de qualidade.

O standard MPEG-2 é capaz de codificar sequências audiovisuais, com definição televisiva a taxas de 4-9 Mbit/s e com televisão de alta definição a taxas de 15-25 Mbit/s.

Por outro lado, a tecnologia ATM tem vindo já a ser utilizada para suportar a distribuição de aplicações multimédia, baseados em serviços de vídeo e áudio de acordo com a norma MPEG-2, nomeadamente pelo ATM-Forum para o transporte de serviços multimédia audiovisuais em redes ATM (AMS: VoD Specification 1.0) e pela ITU para a multiplexagem, sincronização e transmissão de sinais audiovisuais em redes de banda larga (recomendações H.222.1 e J.82).

No entanto o desempenho do transporte de serviços de vídeo em redes ATM continua em aberto dado as funcionalidades existentes na camada de adaptação ainda não estarem definidas.

Desta feita o estudo do transporte do vídeo MPEG-2 em redes ATM torna-se uma das mais interessantes e prementes áreas de estudo.

1.2 Objectivos do trabalho

Esta tese teve como objectivos mais importantes a criação de um Selector de programas e a subsequente transmissão do programa seleccionado num canal ATM.

O modo de descodificar multi-programas, a escolha do canal de adaptação e o método do encapsulamento dos pacotes MPEG-2 em pacotes AAL foram as questões abordadas inicialmente.

O modo de descodificar multi-programas requer a utilização da Norma MPEG-2, parte de Sistema, enquanto a escolha do canal de adaptação terá forçosamente de ser analisado mais profundamente.

A escolha do canal de adaptação envolve várias alternativas. O uso do protocolo de adaptação do tipo 1 (AAL1) elimina os problemas de sincronização associados ao MPEG-2, mas apenas pode ser utilizado com débito constante. No caso geral de débito variável (VBR) terá que ser utilizado outro protocolo de adaptação, visto o

anterior não servir. Foi escolhido o protocolo do tipo 5 (AAL5), inicialmente proposto para suportar tráfego de dados nas redes ATM.

Existem duas soluções alternativas para encapsular as tramas MPEG-2 em pacotes AAL5. A primeira solução designa-se por *PCR-aware*, sendo o empacotamento feito de modo a garantir que quando um pacote contém o valor do PCR este é o último pacote a ser encapsulado. Reduz-se assim o *jitter* existente nos PCR durante o empacotamento. A segunda solução designa-se por *PCR-unaware*, não se verificando a existência do valor do PCR na trama de transporte; logo induz o aparecimento de *jitter* nos valores do PCR durante esta fase do encapsulamento. Finalmente surge um novo protocolo de adaptação (AAL2), embora ainda não esteja estandardizada, se revela como uma alternativa ao transporte do tráfego variável gerado de acordo com a norma MPEG-2. O AAL2 irá fornecer suporte para a sincronização do relógio e superior controlo de erros.

1.3 Organização da tese

A tese está organizada em seis capítulos:

- Capítulo 1 – Introdução;
- Capítulo 2 – A norma MPEG-2;
- Capítulo 3 – As redes de Banda Larga;
- Capitulo 4 – MPEG-2 em redes ATM;
- Capítulo 5 - Trabalho experimental;
- Capítulo 6 – Conclusões.

No Capítulo 1 é feita uma introdução à temática da tese.

O Capítulo 2 apresenta a norma MPEG-2, em particular a norma de vídeo, responsável pela representação codificada de sinais de vídeo, a norma de áudio, responsável pelos sinais de áudio e a norma de sistema, responsável pela mistura, sincronização e protecção dos fluxos elementares (áudio, vídeo e dados) codificados.

O Capítulo 3 aborda as redes de banda larga, dando particular atenção ao modo de transferência assíncrono, o ATM.

No Capítulo 4 é apresentado um estudo da transferência de informação audiovisual codificada de acordo com a norma MPEG-2 em redes baseadas na tecnologia ATM.

No Capítulo 5 é apresentado o trabalho realizado, quer baseado na norma MPEG-2 Sistemas, quer baseado na camada de adaptação do ATM.

Por fim no Capítulo 6 são apresentadas as conclusões.

Capítulo 2
A norma MPEG-2

A tecnologia multimédia lida com a combinação de sinais de meios diferentes, nomeadamente o som, a imagem e o texto. A combinação destas diferentes formas de informação cria uma nova entidade, multimédia, com maior potencial comunicativo e com um largo espectro de aplicações.

O uso da mesma tecnologia num leque alargado de aplicações e a possibilidade de inter-operação entre elas graças à utilização do mesmo núcleo comum, permitirá obter consideráveis economias de escala e flexibilidade de operação acrescida.

Para que estas oportunidades não sejam perdidas é necessário um esforço importante na normalização, com o objectivo de evitar a proliferação de standards de facto. Esta normalização deverá ser efectuada a dois níveis: ao nível dos sinais componentes e ao nível da forma de combinação entre eles.

O comité MPEG foi criado em Janeiro de 1988 com o propósito de desenvolver standards para a representação de imagens com movimento, áudio e a sua combinação. Opera integrado no comité técnico ISO/IEC (JTC1) e é designado WG11 do SC29.

A norma MPEG-2 foi o segundo standard desenvolvido pelo comité MPEG e é particularmente orientada para televisão digital.

Embora o MPEG-2 só por si não seja capaz de fornecer interactividade num ambiente de difusão, é possível utilizar um conjunto de técnicas de modo a tornar um sistema

interactivo. Por exemplo ao multiplexar mais do que um canal de áudio associado a um canal de vídeo, o utilizador poderá escolher qual a língua que pretende; ou ao multiplexar mais do que uma trama de vídeo, esta pode ser utilizada para, por exemplo, colocar ou retirar o repórter numa reportagem de vídeo e o envio de dados adicionais pode permitir ao utilizador escolher os subtítulos de uma dada língua que pretende ler. Desta forma é possível, com base na norma MPEG-2, criar um sistema interactivo.

Nesta fase existe uma bifurcação na tecnologia. Por um lado existe a opção de manter a especificação MPEG-2 apenas para difusão, mas por outro lado encontra-se a opção de avançar na criação de um standard, com diferentes níveis de interactividade, em que o nível zero seria a difusão.

Com este propósito, no início de 1994 surgiu o DAVIC (*Digital Audio-Visual Council*), cujo objectivo era fornecer uma solução neutral aceitável nos meios da difusão, das telecomunicações e dos computadores. Infelizmente tal não foi totalmente conseguido, dado que se encontrou uma solução dupla, o uso do MPEG-2 e do MHEG [Chiariglione2].

Desta feita criou-se uma linha entre a tecnologia utilizada para fazer comunicações multimédia interactivas e não interactivas.

No presente capítulo é apresentado, em pormenor, o desenrolar dos acontecimentos que levaram ao surgimento da norma MPEG-2. De seguida é feita uma abordagem à norma MPEG-2, que está organizada em várias partes:

A **norma de vídeo** que é responsável pela representação codificada de sinais de vídeo.

A **norma de áudio** que é responsável pelos sinais de áudio.

A **norma de sistema** que é responsável pela mistura, sincronização e protecção dos fluxos elementares (áudio, vídeo e dados) codificados.

Para além destas três partes existem ainda mais três. Uma para testes de conformidade da implementação da norma, uma outra composta por um conjunto de programas de simulação e teste que implementam os aspectos fundamentais da descodificação e uma última com as especificações dos protocolos de gestão e controlo dos sistemas por parte do utente (DSM-CC).

2.1 Historial

O desenvolvimento da tecnologia digital no anos 80 tornou prático o uso de sinais de vídeo digitais em produtos de consumo.

Estava assim aberta a possibilidade para o surgimento de uma nova gama de serviços e produtos. Mas para transformar esta possibilidade em realidade era essencial que a tecnologia de compressão se tornasse barata e largamente utilizada. A normalização tem sido muito importante no encorajamento de investigação em circuitos integrados e em sistemas baratos para o mercado de consumo.

Em 1988 a necessidade da criação de um standard para compressão de vídeo era evidente. Havia na altura o standard ITU-T quase completo, hoje denominado H.261, para Vídeo-Conferência e Vídeo-Telefonia e o grupo de trabalho JPEG (*Joint Pictures Expert Group*) que estudava a possibilidade de compressão de imagens paradas, utilizando um algoritmo baseado em DCT (*Discrete Cosine Transform*).

O grupo de trabalho MPEG (*Moving Pictures Expert Group*) foi assim criado, com o intuito de definir os standards para a compressão de sinais audiovisuais.

O primeiro objectivo era definir um algoritmo de codificação de vídeo, para aplicações de armazenamento (DSM, *Digital Storage Media*). Rapidamente a necessidade de codificar o áudio se tornou evidente e o objectivo era encontrar um algoritmo genérico capaz de ser virtualmente usado por todas as aplicações, desde sistemas de armazenamento, difusão de televisão e até em aplicações de comunicações como VoD (*Video on Demand*) , Vídeo-Telefonia entre outros.

O primeiro projecto foi publicado em 1993 como ISO/IEC 11172 e denominou-se MPEG-1. É um standard dividido em três partes, definindo os métodos de codificação para compressão de áudio e vídeo e o sistema de multiplexar o áudio e o vídeo de modo ser possível a reprodução com sincronização das componentes. Foi aplicado no sistema CD-I e no Vídeo-CD, com o CD-ROM como suporte físico.

Esta norma suporta codificações da ordem dos 1,5 Mbit/s fornecendo qualidade vídeo semelhante ao VHS e áudio com qualidade estéreo de 192 kbit/s/canal e foi optimizado para sinais de vídeo não entrelaçado.

Em 1990 o grupo MPEG verificou a necessidade para um segundo standard, relativo à codificação de vídeo com maiores débitos e num formato entrelaçado. Esta é a principal diferença técnica entre o MPEG-1 e o MPEG-2. Enquanto o MPEG-1 foi especificado para codificar imagens progressivas, o MPEG-2 foi especificado para optimizar imagens intrelaçadas (de salientar o facto de as tramas MPEG-1 serem descodificados pelos descodificadores MPEG-2).

O standard MPEG-2 é capaz de codificar sequências audiovisuais, com definição televisiva a taxas de 4-9 Mbit/s e com televisão de alta definição a taxas de 15-25 Mbit/s.

O MPEG-2 é mais eficiente para sinais de vídeo entrelaçados codificados a uma taxa de 3 Mbit/s. O áudio do MPEG-2 é compatível com o áudio do MPEG-1, mas é extensível à codificação de um sistema multi-canal envolvente.

A compressão dos sinais pode variar entre 30:1 e 100:1 de acordo com o nível de qualidade. Em termos de largura de banda podemos observar na tabela 2.1 alguns exemplos.

Tabela 2.1 – Qualidade e largura de banda do MPEG-2.

Nível de qualidade MPEG-2	Largura de banda utilizada
VHS	1,5 Mbit/s
Difusão	5,0 Mbit/s
Estúdio	7,0 Mbit/s

Ambos os standards MPEG-1 e MPEG-2 foram divididos em três partes: Codificação de Áudio, Codificação de Vídeo e Sistema. Assim o grupo MPEG dividiu-se em três subgrupos, um responsável por cada parte e adicionalmente criaram-se outros subgrupos para aconselhar na implementação, para efectuar testes e para estudar os requisitos a serem suportados.

Cada subgrupo desenvolveu-se de uma forma semelhante. Inicialmente foram analisadas as especificações do sistema, o que contribuiu para o aparecimento de várias propostas para algoritmos de compressão de áudio e de vídeo. De seguida surgiu uma fase competitiva, cujo objectivo era identificar um conjunto de técnicas viáveis de serem implementadas. De seguida avançou-se para a fase colaborativa, em que se definiram as técnicas a adoptar na versão final da norma, de que resultou um primeiro documento com especificações.

Actualmente continuam os testes de conformidade, interoperabilidade e demonstração da possibilidade de realizar codecs em *hardware* e *software*, implementando a codificação, mistura e descodificação de fluxos audiovisuais.

Apesar de alguns Gurus terem preconizado que a convergência nas telecomunicações, entretenimento e computadores iria surgir, tal facto ainda não sucedeu.

O comité MPEG tem como objectivo criar standards que vão ao encontro das necessidades de inúmeras indústrias, na área das comunicações.

Em Julho de 1993 iniciaram o projecto MPEG-4 e espera-se que em Novembro de 1998 o MPEG-4 se torne um standard Internacional em comunicações multimédia.

Comunicação multimédia é a possibilidade de comunicar informação audiovisual que:

1- É natural, sintética ou mista;
2- Funciona em tempo real ou diferido;
3- Suporta diferentes funcionalidades de acordo com as necessidades do utilizador;
4- Comuta simultaneamente entre diferentes fontes;
5- O utilizador não precisa de se preocupar com as especificações do canal de comunicação, mas a tecnologia utilizada é sensível ;
6- Possibilita ao utilizador interagir com diferentes elementos de informação;
7- Permite ao utilizador apresentar o resultado das sua interacção de acordo com as suas necessidades;

O projecto MPEG-4 será especialmente adequado às aplicações da Internet, que são independentes da rede física, interactivos e capazes de fazer o *Download* [Chiariglione2].

2.2. A codificação MPEG-2 Vídeo

A norma MPEG-2 Vídeo específica uma forma de apresentação codificada, para comunicação no espaço ou no tempo, de imagens em formato digital bem como o processo de descodificação associado, distinguindo-se os seguintes aspectos (que se relacionam entre si):

Sintaxe de mistura do débito binário comprimido – com a qual se estabelecem as regras de mistura (hierárquica) e representação binária dos parâmetros utilizados pelas técnicas de processamento (informação de cabeçalho) e do sinal de imagem propriamente dito, após compressão.

Algoritmo de compressão – no qual se incluem técnicas de processamento de sinal para a redução de redundância temporal e espacial de vídeo.

A norma MPEG-2 Vídeo estabelece duas categorias para a representação dos sinais vídeo que correspondem à sintaxe/algoritmo do tipo escalável. No primeiro caso (não escalável), a representação codificada de uma sequência de imagens consiste num único débito binário elementar. No segundo caso uma sequência de imagens é obtida através da combinação de um conjunto de débitos binários elementares.[1]

A codificação de sinais de vídeo é baseada em algoritmos da família MC/DCT que foi também adoptada como referência pelas normas JPEG, H.261, MPEG-1.

[1] Representações em escala (de qualidade, espacial ou temporal) do sinal de vídeo.

A norma MPEG-2 Vídeo garante a possibilidade de implementar funções que se adaptam às exigências das várias aplicações, entre as quais se destacam:

- Possibilidade de utilizar formatos 4:2:0, 4:2:2 e 4:4:4;
- Codificação de formatos de imagem entrelaçados e progressivos;
- Conversão dos sinais provenientes dos sistemas de televisão existentes (PAL; SECAM; NTSC; MAC);
- Conversão de formato filme (24 imagens por segundo em formato progressivo) para formato TV (25 imagens por segundo, formato entrelaçado);
- Codificação e débito binário constante (CBR) ou variável (VBR);
- Modo de codificação com atraso reduzido para comunicações interpessoais;
- Edição de informação codificada, em modos especiais (*trick modes*);
- Resiliência a erros;

2.2.1 Aspectos sintácticos da norma MPEG-2 Vídeo

Na compressão baseada em técnicas MC/DCT é importante referir o conceito de escalabilidade.

O conceito de escalabilidade refere-se à possibilidade de obter informação útil do sinal de vídeo comprimido descodificando partes da informação comprimida.
A escalabilidade pode ser de diversas formas.

A **escalabilidade espacial** tem que ver com a codificação do vídeo original em várias versões de resolução espacial crescente. No caso de dois níveis a informação comprimida é organizada em dois fluxos codificados correspondendo ao nível base (*base layer*) e ao nível melhorado (*enhanced layer*).

A **escalabilidade em qualidade** (SNR) é um caso particular da anterior, em que a resolução se mantém igual nos dois níveis.

A **escalabilidade temporal** considera a possibilidade de codificar o sinal em dois níveis, mas variando a resolução temporal, e a **escalabilidade em frequência** resulta da organização da informação codificada por subconjuntos dos coeficientes DCT dos blocos 8x8 (sub-bandas DCT).

Existe ainda a escalabilidade híbrida, que considera a possibilidade de combinar dois tipos de escalabilidade em três níveis de fluxo codificados.

2.2.2 Organização hierárquica da norma MPEG-2 Vídeo

A norma MPEG-2 Vídeo utiliza níveis hierárquicos para organizar a informação de vídeo codificada (ver Figura 2.1). A cada nível correspondem funções específicas quer sob o ponto de vista do codificador quer do descodificador.

Sob o ponto de vista do codificador, os níveis hierárquicos correspondem a uma divisão feita no sinal de vídeo original, correspondendo a cada nível hierárquico do sinal de vídeo uma funcionalidade específica do codificador (robustez a erros, detecção e compensação do movimento, DCT , etc.).

Sob o ponto de vista do descodificador, os níveis hierárquicos correspondem a uma divisão do débito binário codificado, correspondendo uma funcionalidade específica do descodificador (sincronização, acesso aleatório, compensação do movimento, DCT inversa, etc.).

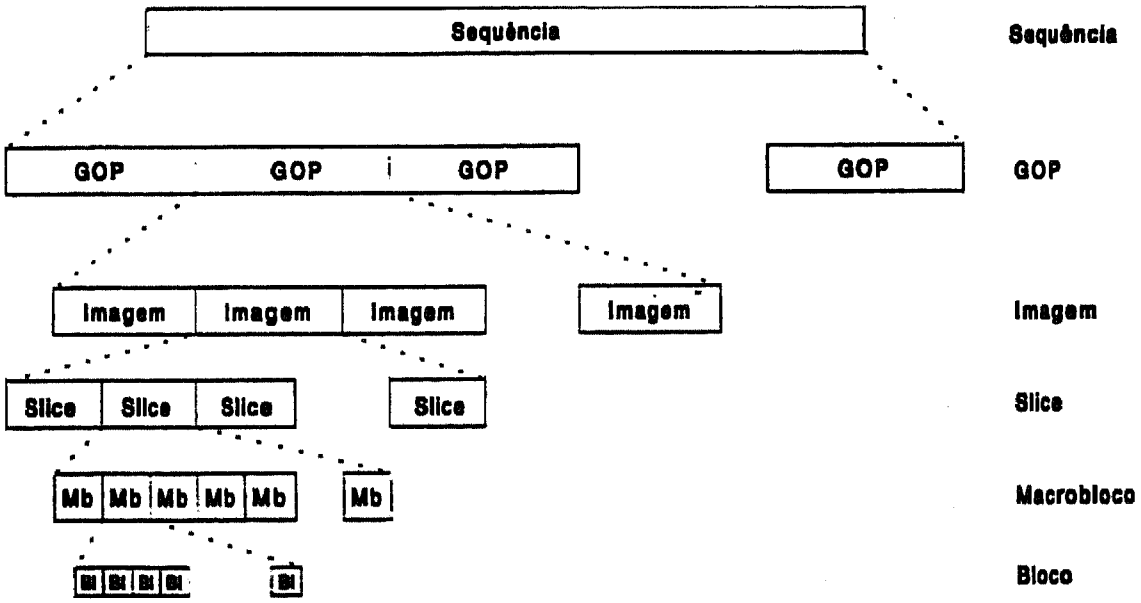


Figura 2.1 – A norma MPEG2 organiza a informação do sinal de vídeo em seis níveis hierárquicos.

A organização da norma MPEG-2 Vídeo divide-se em duas partes. A sintaxe compatível com MPEG-1, com seis níveis hierárquicos base, definidos na norma MPEG-1, e a sintaxe não compatível com MPEG-1, com extensões aos níveis hierárquicos base.

2.2.3 Aspectos do algoritmo de compressão vídeo

O algoritmo MC/DCT introduz distorção no sinal descodificado uma vez que certas operações aplicadas ao sinal de imagem não são completamente reversíveis (transformada DCT calculada com precisão finita ou com quantificação).

Este algoritmo compreende duas fases, a análise e a compressão.

Na fase de **análise** detecta-se o movimento e analisam-se as características, gerais e locais, da imagem.

Na fase de **compressão,** o codificador utiliza a informação auxiliar, da fase anterior, para obter a melhor compromisso entre a qualidade final das imagens e os recursos disponíveis (quantidade de bits por unidade de tempo). Nesta fase são implementadas as funções de codificação com compensação do movimento e transformada (MC/DCT – *Motion Conpensated/Discrete Cosine Transform*), quantificação e codificação entrópica.

A compressão do sinal de vídeo consiste numa sucessão de decisões e de funções que procuram optimizar a relação qualidade – ritmo binário após compressão.

Nas decisões ao nível da imagem, o codificador decide quanto ao modo de codificação, classificando cada imagem segundo o modo de predição admitido (Figura 2.2).

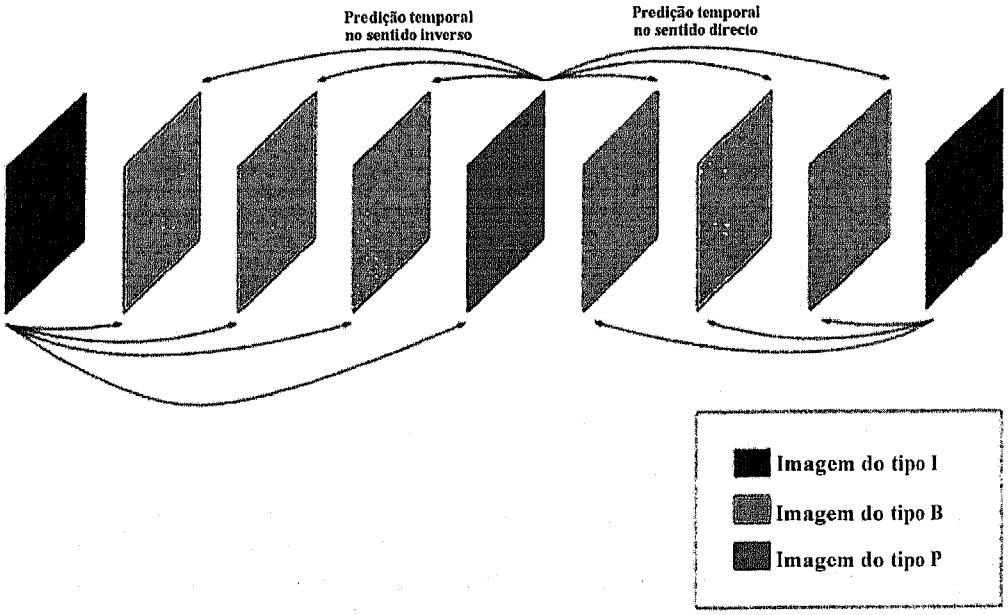


Figura 2.2 - Tipologia das imagens segundo o modo de predição utilizado: (I –Intra; P – Preditiva; B – Bidireccional).

Intra (I) – em que não é utilizado nenhum tipo de predição temporal.

Preditivo (P) – em que é utilizada a predição temporal no sentido directo do tempo (*forward*).

Bidireccional (B) – em que se utiliza a predição temporal no sentido directo do tempo (*forward*), no sentido inverso (*backward*), ou ambos simultaneamente (*bidireccional*).

As decisões ao nível do macrobloco do codificador são condicionadas pelas decisões do nível hierárquico superior.

Nas imagens de tipo **I**, os macroblocos podem ser codificados apenas no modo intra (não diferencial).

Nas imagens de tipo **P** e **B** o codificador deve decidir entre codificar cada macrobloco em modo diferencial ou não diferencial. No caso diferencial, o codificador deve escolher, ainda, se usar a predição simples ou com movimento compensado (eixo do tempo ou eixo do movimento).

Enquanto que nas imagens do tipo **P** cada macrobloco diferencial só pode obter a sua predição de uma imagem de referência no passado (predição do tipo *forward*), no tipo **B** os macroblocos de predição podem ser obtidos de imagens temporalmente precedentes e/ou subsequentes (predição do tipo *forward* e *backward*). Neste tipo de imagem existe, ainda, a possibilidade de obter a predição bidireccional através da interpolação combinando os macroblocos de predição *forward* e *backward*.

Nas decisões ao nível do bloco, é aplicada a transformada DCT, seguindo-se a quantificação dos coeficientes, a geração de símbolos a partir destes e a sua codificação entrópica através de códigos de Huffman.

2.2.4 Organização da norma MPEG-2 Vídeo em Perfis e Níveis

Tendo em conta a flexibilidade e a complexidade variável dos modos de codificação propostos, a norma MPEG-2 estabelece critérios com os quais se pode garantir um elevado grau de interoperabilidade entre codificadores, redes de transmissão e descodificadores.

A gestão do compromisso custo-benefício é feita através da classificação dos codificadores-descodificadores de vídeo em Perfis e Níveis.

Estes subconjuntos garantem a eficiência compressão-qualidade requerida por uma classe de aplicações e introduzem parcimónia, limitando a complexidade e os custos associados aos sistemas codificadores e descodificadores.

O conceito de perfil associa-se a um conjunto de opções de codificação que se adaptam às funcionalidades requeridas por um conjunto de classes de aplicações. Por exemplo, um perfil não escalável não prevê nenhum tipo de escalabilidade e poderá ser adoptado por aplicações que não requerem essa funcionalidade.

O conceito de nível estabelece, para cada perfil, vínculos sobre valores que certos parâmetros de codificação devem tomar. Por exemplo, podem estabelecer um limite para a máxima resolução espacial do sinal de vídeo a codificar ou para o débito binário máximo.

Os perfis relacionam-se entre si de uma forma hierárquica, ou seja, a sintaxe que define um dado perfil contém, como subconjunto, todos os elementos sintácticos necessários aos perfis que lhe são inferiores. É também garantido que para uma dado perfil os elementos sintácticos não variam em função do nível adoptado.

Também os níveis se relacionam hierarquicamente, isto é, a gama de variabilidade para os parâmetros associados cresce com o nível. Os vínculos impostos por um determinado nível são mais limitativos dos que os de um nível hierarquicamente superior. Por exemplo, para um dado perfil, um descodificador que descodifica um nível principal (*Main*) também descodifica um nível baixo (*Low*).

Se um codificador produz um débito binário que supera os limites sintácticos pré-definidos para um determinado perfil e/ou nível é-lhe atribuído o perfil e/ou o nível imediatamente superior.

Se a capacidade do descodificador está aquém daquela pré-definida para um determinado perfil e/ou nível é-lhe atribuído o perfil e/ou o nível imediatamente inferior.

Tabela 2.2 – Perfis e Níveis definidos na norma MPEG-2 Vídeo.

Perfil	**Nível**
Simples	Baixo
Principal	Principal
Escalável em qualidade	Elevado 1440
Escalável no espaço	Elevado
Elevado	--

Tabela 2.3 – Algumas das características dos Perfis definidos.

Sintaxe	Perfil				
	Simples	Principal	Escalável Qualidade	Escalável Espaço	Elevado
Tipos de imagem	I e P	I,P e B	I,P e B	I,P e B	I,P e B
Sub-amostragem da Crominância	4:2:0	4:2:0	4:2:0	4:2:0	4:2:0, 4:2:2
Escalabilidade	Não	Não	Sim	Sim	Sim

Na tabela 2.2 indicam-se os Perfis e os Níveis determinados pela norma. A título de exemplo apresentam-se, na tabela 2.3, algumas das características associadas aos perfis definidos e, na tabela 2.4, algumas restrições impostas pelos vários níveis.

Tabela 2.4 – Vínculos estabelecidos em alguns parâmetros para o Perfil Principal.

Parâmetro Sintáctico	Nível			
	Baixo	Principal	Elevado-1440	Elevado
Resolução espacial-temporal máxima.	352x288 30Hz	720x576 30 Hz	1440x1152 60 Hz	1920x1152 60 Hz
Débito Binário Máximo (Mbit/s)	4	15	60	80

Desta forma está estabelecida uma interoperabilidade entre sistemas conformes à norma MPEG-2.

Presentemente já foram definidos os Perfis e os Níveis que cobrem uma vasta gama de aplicações, mas encontram-se em fase de proposta novo Perfis e Níveis que a norma MPEG-2 Vídeo prevê incluir no futuro (por exemplo os perfis *multi view*, escalabilidade em frequência, etc.).

2.3 A codificação MPEG-2 Áudio

A parte da norma MPEG-2 Áudio está completamente estabilizada, embora incluindo importantes extensões, mantem basicamente a mesma estrutura e os mesmos algoritmos da fase precedente dos trabalhos MPEG.

Os esquemas de codificação MPEG-2 Áudio fazem parte duma classe designada por algoritmos híbridos psico-acústicos. Baseiam-se em técnicas clássicas de processamento de sinal no domínio da frequência (codificação por divisão em sub-bandas espectrais), mas a característica mais saliente é a adaptação dos parâmetros de codificação (quantificação, distribuição de bits pelas sub-bandas, etc.) às características locais do sinal e à sensibilidade do aparelho humano à distorção e ruído.

O codificador áudio está organizado de modo hierárquico sendo previstos três níveis de complexidade e eficiência crescente:

Níveis I e II – Codificação baseada em técnicas de sub-banda, com quantificação e "mascaramento" do ruído baseados em modelos psico-acústicos. As diferenças entre os níveis I e II encontram-se na estruturação do sinal de entrada e nos modelos utilizados para o sinal. Nenhum destes níveis utiliza técnicas de codificação de entropia.

Nível III – Codificação híbrida baseada numa adaptação do modelo psico-acústico utilizado no nível II à zona de alta frequência, melhorando a eficiência de compressão nesta zona do espectro. A complexidade deste modo de codificação é superior à dos

níveis I e II e, neste caso, é usada também a codificação de entropia (Huffman) para a geração do fluxo comprimido a partir dos símbolos produzidos na fase de análise em sub-bandas e quantificação.

A estrutura hierárquica do codificador áudio obriga a que os receptores que descodificam um determinado nível devam descodificar todos os níveis de ordem inferior. O sinal áudio é estruturado em tramas que correspondem a unidades de apresentação de som (aspecto importante na fase de sincronização mútua no caso de aplicações audiovisuais).

As tramas áudio têm dimensões que dependem do nível de codificação (Tabela 2.5). Por exemplo, para o nível I a trama áudio contém 384 amostras do sinal original. Nos restantes níveis (II e III), cada trama contém 1152 amostras do sinal áudio.

Tabela 2.5 – Número de amostras em cada trama áudio MPEG (depende do nível).

	Modo Monofónico	Modo Estereofónico
Nível I	384	2x 384
Nível II	1152	2x 1152
Nível III	1152	3x 1152

As frequências de amostragem definidas no âmbito da norma MPEG-1 são 32 kHz, 44,1 kHz e 48 kHz. A duração temporal áudio de uma trama áudio depende da frequência de amostragem e do número de amostras por trama considerado em cada nível (Tabela 2.6).

Tabela 2.6 – Duração temporal de uma trama áudio para as frequências de amostragem previstas na norma MPEG-2 áudio.

	32 kHz	44.1 kHz	48 kHz
Nível I	12 ms	8,7 ms	8 ms
Nível II	36 ms	26,12 ms	24 ms
Nível III	36 ms	26,12 ms	24 ms

A norma MPEG-2 Áudio considera algumas extensões importantes relativamente à fase MPEG-1:

Maior eficiência a baixo débito binário (< 64 kbit/s) – são consideradas três frequências de amostragem adicionais (16 kHz, 22,05 kHz e 24 kHz) além das previamente referidas;

Hiper-estereofonia para qualidade (multi-canal) – em relação à fase MPEG-1 (onde eram previstos os modos mono-canal ou bi-canal estéreo), a norma MPEG-2 prevê um aumento de número de canais codificados até um máximo de 5+1 (este último para aplicações particulares com efeitos especiais).

Os cinco canais do sistema hiper-estéreo permitem obter som de muito alta qualidade. Este sistema designado por 3+2 uma vez que aos dois altifalantes usuais do sistema estéreo bi-canal (esquerdo e direito) se juntam um terceiro altifalante frontal e dois altifalantes de ambiente.

Hiper-estereofonia para transmissão multi-língua – os mesmos conceitos apresentados anteriormente podem ser utilizados para transmissão de programas em várias línguas e, ainda para sistemas de comunicação (*clean dialogue*) destinados a indivíduos com deficiências auditivas.

2.4. Especificação da camada de Sistema do MPEG-2

2.4.1 Introdução

A norma MPEG-2 Sistema estabelece um conjunto de técnicas e sintaxe associada para a combinação de uma ou mais fluxos elementares numa única trama de nível hierárquico superior.

Uma trama de transporte (*TS – Transport Stream*) pode conter vários fluxos elementares correspondentes a diversas partições de um sinal de vídeo ou pode conter a mistura de tramas elementares pertencentes a vários sinais de vídeo, áudio ou outro tipo de dados que têm de ser apresentados de modo síncrono.

Ao nível de sistema a função de *multiplexagem* garante a representação síncrona de sinais multimédia, a mistura e descrição da interdependência entre fluxos elementares que contém a informação estratificada de uma sequência de vídeo, a adaptação ao meio de transmissão prevendo a protecção contra erros, o dimensionamento e funcionamento estável dos *buffer* utilizados pelos codificadores e descodificadores que não devem transbordar ou ficar vazios (Figura 2.3).

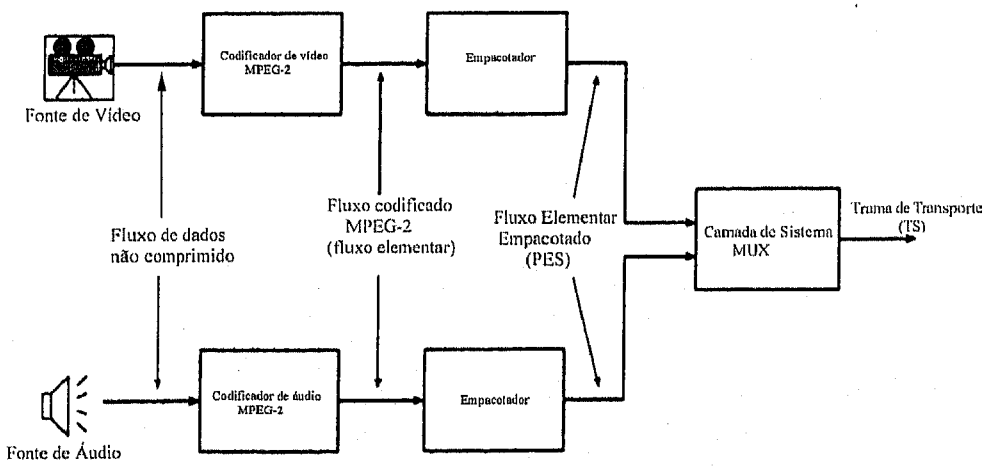


Figura 2.3 – Estrutura genérica do MPEG-2 Sistema (caso da trama de transporte TS).

A gestão da interacção utente-descodificador realiza-se através de comandos de controlo para os modos especiais (por exemplo: mudança de canal durante uma transmissão televisiva, movimento lento e movimento acelerado).

2.4.2 Unidades de Apresentação e Unidades de Acesso

A Figura 2.4 representa uma sequência de vídeo não comprimida a ser codificada através do codificador de vídeo MPEG-2. Cada imagem não comprimida é denominada por unidade de apresentação[2] . O codificador comprime cada unidade de apresentação de acordo com o standard e agora é denominada por unidade de acesso[3] . É de salientar o facto de as unidades de acesso não serem do mesmo tamanho. O seu tamanho depende do tipo de imagem ser do tipo P- Preditivo, do tipo I- Intra, ou do tipo B- bidireccional.

Como resultado da codificação MPEG-2 obtemos uma sucessão de unidades de acesso, que compõem o fluxo elementar[4] de vídeo.

De modo análogo o resultado de uma codificação MPEG-2 de um fluxo de áudio não comprimido, é uma sucessão de unidades de acesso de áudio, que compõem o fluxo elementar de áudio. Cada unidade de acesso de áudio contém tipicamente algumas centenas de milisegundos de áudio comprimido [Sarginson2].

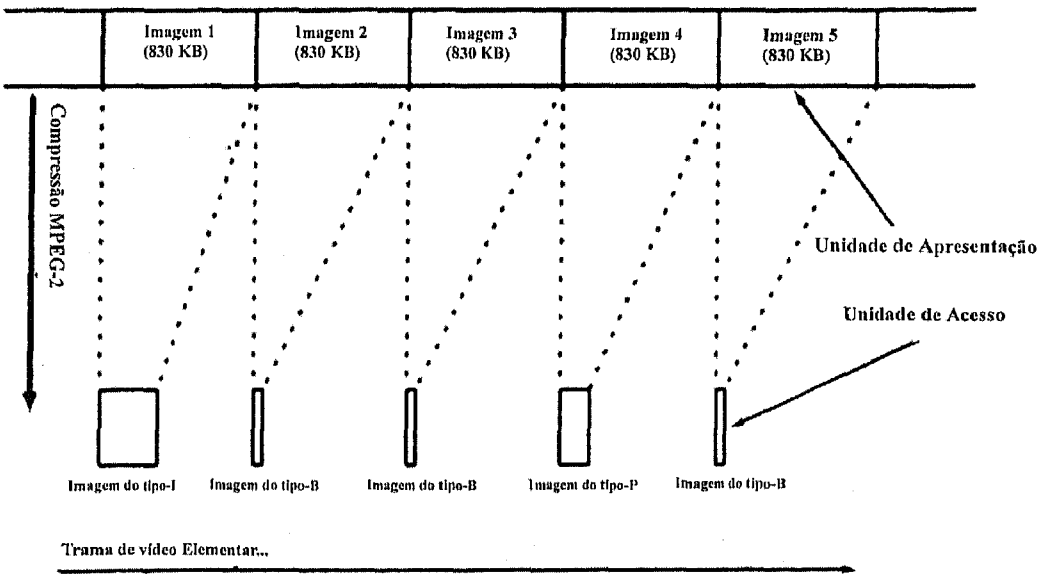


Figura 2.4 Criação de uma trama elementar a partir de dados não comprimidos.

[2] Do inglês *presentation unit*
[3] Do inglês *acess unit*
[4] Do inglês *elementary stream*

2.4.3 A camada PES

Após a criação do fluxo elementar, a etapa seguinte é o seu empacotamento. Como resultado obtemos um fluxo elementar empacotado (**PES** - *Packetised Elementary Stream*) e os pacotes são denominados pacotes PES (Figura 2.5).

A camada PES (contendo um fluxo elementar) pode ser gerada pelo codificador elementar ou pelo misturador de sistema.

Os pacotes do nível PES são constituídos pelo cabeçalho e pela informação do fluxo elementar codificado designado por carga [5].

O comprimento dos pacotes PES se for variável é codificado ao nível do cabeçalho (com um máximo de 2^{16} bytes). O seu início é identificado por um conjunto de códigos *start_code* pré-definidos que permitem distinguir o fluxo elementar contido na carga (vídeo, áudio, DSM-CC, etc.). Neste caso cada pacote PES contém apenas uma unidade de acesso (ver Figura 2.5).

Mas o empacotamento pode ser feito utilizando pacotes PES de tamanho fixo. Neste caso o processo de empacotamento é mais fácil.

Os pacotes PES possuem identificadores especiais, que permitem distinguir os pacotes PES pertencentes a outras fluxos elementares. Podem transportar dois marcos[6] de tempo diferentes, que são usados pelo descodificador para a correcta sincronização dos fluxos elementares relacionados: o marco de apresentação (PTS – *Presentation Timestamp*) e o marco de descodificação (DTS – *Decoding Timestamp*).

A partir do fluxo elementar empacotado (**PES**), são construídos de forma hierárquica, dois modos de mistura, que se designam por **Modo Programa** (**PS** - *Program Stream*) e **Modo Transporte** (**TS** - *Transport Stream*).

No modo de programa os pacotes PES não podem ser maiores que 64 kbytes, enquanto no modo de programa o seu tamanho é arbitrário.

No caso do modo de transporte (TS) a informação PES é empacotada em pacotes de tamanho fixo (*transport packets*).

[5] Do inglês *payload*
[6] Do inglês *timestamps*

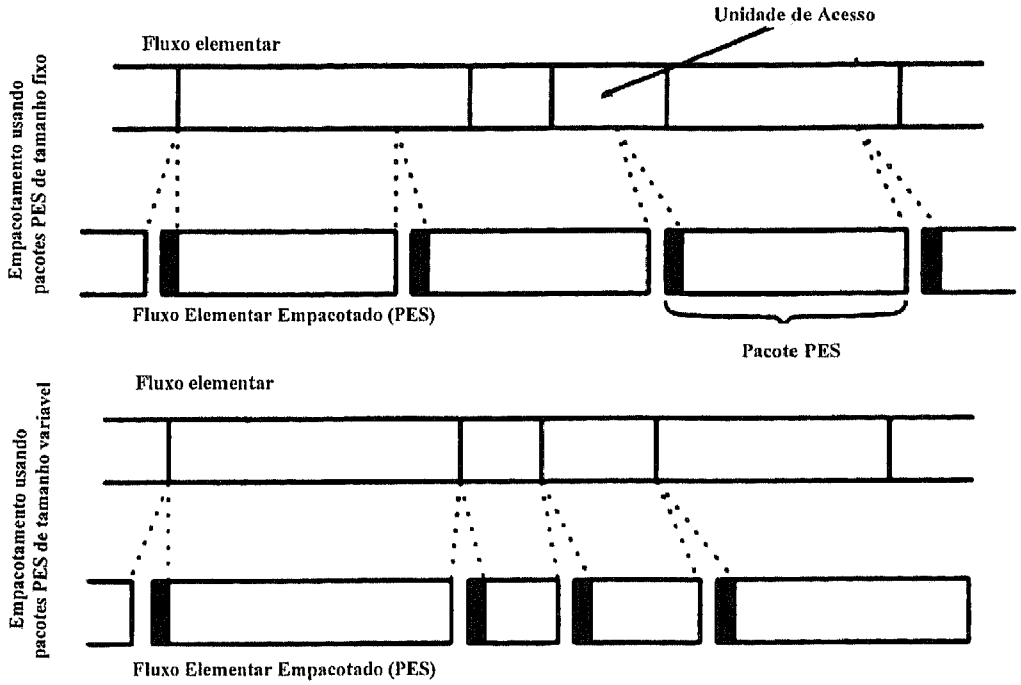


Figura 2.5 – Geração de um fluxo elementar empacotado (PES) a partir de uma trama elementar.

2.4.4 Sintaxe PS

A sintaxe PS (*Program Stream*) ou Modo de Programa, é um modo de mistura que inclui técnicas que são uma evolução da norma MPEG1. Um fluxo binário de sistema construído no modo PS é organizado em pacotes de dimensão variável e adapta-se a aplicações onde o meio de transmissão e/ou memorização seja isento de erros.

O Modo de Programa visa a sua utilização em ambientes sem erros, porque este método é mais susceptível a erros.

Isto sucede devido a duas razões.

Primeiro, a trama de programa é composta por uma sucessão relativamente longa de pacotes de tamanhos diferentes. Cada pacote começa por um *Cabeçalho* (Header) de Pacote e um erro neste pacote origina a perda do pacote inteiro. Como um pacote da Trama de Programa pode conter muitos kilobytes de dados, a perda de um único pacote representa a perda ou a corrupção de uma imagem inteira de vídeo.

Segundo, a variabilidade do tamanho nos pacotes representa uma dificuldade, pois o descodificador não pode prever quando um pacote acaba e o outro começa. Assim o descodificador para conhecer o tamanho do pacote tem que ler e interpretar o campo com o tamanho, que está contido no cabeçalho de cada pacote. Se este campo contiver um erro o descodificador perde o sincronismo com o fluxo, resultando igualmente na perda de pelo menos um pacote.

Este modo presta-se a aplicações que requerem a manipulação de dados multimédia gravados em suporte digital, uma vez que contém mecanismos que facilitam a interacção directa com o fluxo elementar da informação codificada.

2.4.5 Sintaxe TS e MPTS

Os pacotes PES dos vários fluxos elementares são divididos pela carga[7] de vários pacotes TS. No entanto existem duas restrições:

1. O primeiro byte do pacote PES tem de ser o primeiro byte da carga do pacote TS.

2. Cada pacote de transporte tem de conter dados de apenas um pacote PES, i. e., um único pacote TS refere-se apenas a um fluxo elementar codificado (PES) de uma trama elementar.

Devido às duas restrições acima enumeradas, o pacote de transporte poderá não ser completamente preenchido, dado que é improvável que um pacote PES caiba exactamente num número inteiro de pacotes TS. Os bytes usados para completar o pacote são colocados no campo de adaptação, excepto no caso de o pacote de transporte conter informação específica do programa (PSI – *Program Specific Information*) em que é colocado no fim do pacote. A quantidade de *padding* pode ser minimizado se for feita uma prévia selecção no comprimento dos pacotes PES. Normalmente os pacotes PES compridos são melhores em termos de eficiência de largura de banda, mas são mais susceptíveis a problemas de sincronização. A escolha é deixada ao cargo do projectista (ver Figura 2.6).

[7] Do inglês *payload*

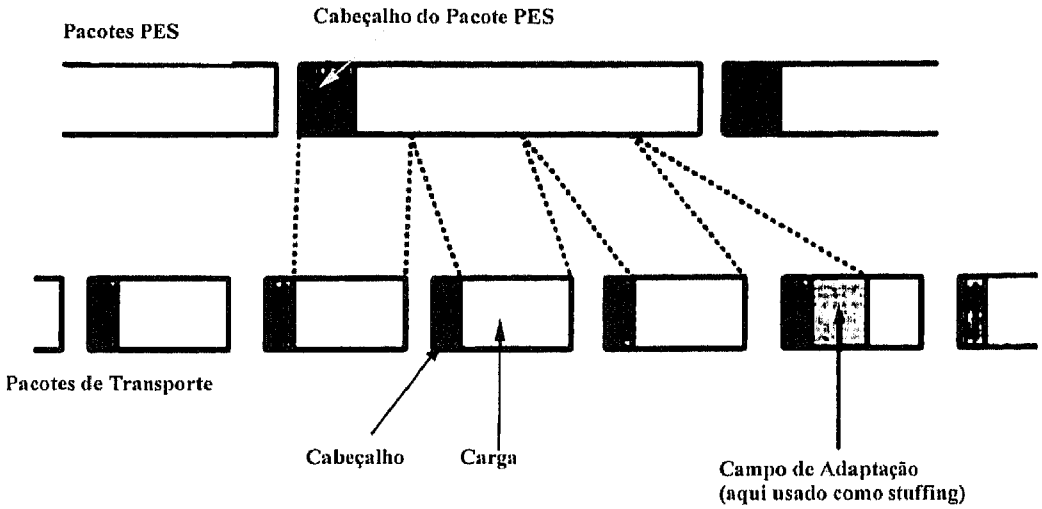


Figura 2.6 – Geração de uma pacotes de transporte (TS) a partir de fluxos elementares empacotados (PES).

Não existe nenhuma restrição na ordem que os pacotes TS, preenchidos a partir de diferentes fluxos de dados elementares, surgem no *multiplexer*, excepto na preservação da ordem cronológica dos pacotes pertencentes ao mesmo fluxo elementar.

Os pacotes de transporte podem conter tabelas de informação específica do programa (PSI), usadas para o mapeamento de programas e da associação com fluxos elementares. Existem também pacotes de transporte nulos, que devem ser colocados na trama de modo a preencher a capacidade de um débito binário constante (CBR) nas tramas de transporte.

Na sintaxe TS (*Transport Stream*) ou Modo de Transporte, o fluxo de sistema está dividido em pacotes de 188 bytes (Figura 2.7).

A sua dimensão (188 bytes) resulta de um compromisso entre a quantidade de informação de cabeçalho, de carga, a resistência a erros ou perda de pacotes e à compatibilidade com outras normas (ATM).

Uma parte do cabeçalho TS (presente em todos os pacotes) gere a ligação ponto-a-ponto e designa-se por um cabeçalho de nível de ligação (*link*). Esta parte do cabeçalho caracteriza-se por um byte de sincronismo 47 (Hex.), um identificador de programa (PID), de 13 bits e alguns bits de indicação (*flags*) relacionados com a indicação de erros, prioridades, acesso aleatório e cifra.

Um dos bits do nível de ligação indica a presença ou não de um cabeçalho (*adaptation field*) contendo informação relacionada com os fluxos elementares misturados.

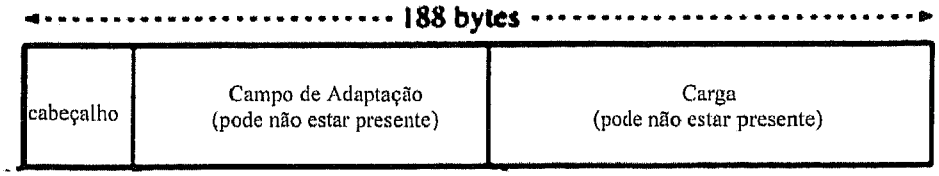


Figura 2.7 Estrutura do pacote TS.

Na norma MPEG, um programa é definido como sendo um canal televisivo, assim por exemplo a SIC seria um programa, a RTP1 seria outro programa e a TVI outro programa. Se nós pretendessemos receber estes três programas televisivos, estes seriam multiplexados em conjunto originando uma trama de transporte com múltiplos programas (MPTS – *Multi Program Transport Stream*). Ou seja de três tramas de transporte TS resultava uma única trama múltipla MPTS (Figura 2.8).

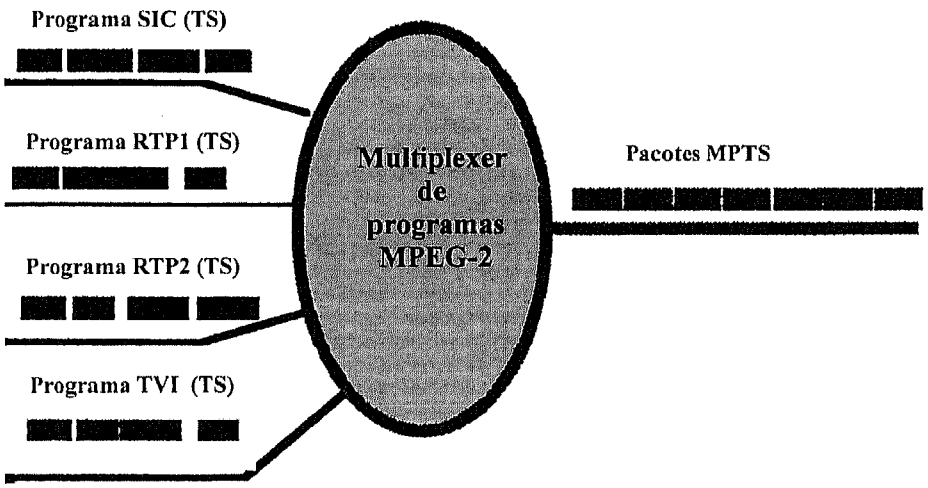


Figura 2.8 – Multiplexagem de tramas de transporte (TS) numa trama multi-programa (MPTS).

Na sintaxe TS (*Transport Stream*) ou Modo de Transporte procuram-se resolver problemas como a transmissão em ambientes com erros, a mistura de fluxos heterogéneos quanto à base de tempo, a transmissão em redes de comutação rápida, re-sincronização rápida (mudança de canal), inserção de programas (por exemplo, para a inserção de publicidade), acesso condicionado, etc.

Um aspecto importante do nível de sistema é a sincronização. Um exemplo imediato é o da sincronização mútua dos sinais de vídeo e áudio pertencentes à mesma sequência mas este problema é mais vasto e engloba os seguintes aspectos:

- Controlo do fluxo de dados entre codificador (emissor) e descodificador (receptor), designada por sincronização para gestão dos *buffers*.

- Sincronização mútua de fluxos elementares que compõem o programa.

- Condições iniciais do receptor.

Este problema da sincronização assume aspectos particulares, de acordo com o tipo do canal de transmissão, ou seja:

- Canais síncronos de débito binário constante (CBR, *Constant Bit Rate*), cujos exemplos são as redes de transmissão síncronas de circuito dedicado e a leitura do disco compacto (CD).

- Canais assíncronos com débito binário variável e retroacção (*VBR with feedback*), como por exemplo os discos dos computadores.

- Canais assíncronos com débito binário variável sem retroacção (*VBR with no feedback*), de que são exemplo as redes ATM.

Capítulo 3

Redes de Banda Larga

As redes de Comunicação têm sofrido uma profunda evolução ao longo dos últimos anos com o consequente aparecimento de um número elevado de novas tecnologias e serviços.

As redes de banda larga são um passo na evolução das actuais redes de telecomunicações e surgem como uma necessidade de suportar ainda um certo tipo de serviços como a transmissão de vídeo e dados de débitos elevados.

O Modo de Transferência Assíncrono ATM (Asynchronous Transfer Mode) surge como uma tecnologia capaz de dotar a rede de banda larga com os requisitos necessários à satisfação dessas necessidades, fornecendo uma boa eficiência e flexibilidade da largura de banda, devido essencialmente à multiplexagem estatística.

No presente capítulo é apresentada uma breve descrição da sua evolução, são introduzidos alguns modos de transferência de informação e de seguida é apresentado, com algum detalhe o modo de transferência assíncrono - ATM, referindo as categorias de serviço existentes.

3.1 Breve descrição da sua evolução

Um século após a descoberta do telefone, as redes de telecomunicações sofreram uma profunda transformação com o consequente aparecimento de um grande número de novos e atractivos serviços. A evolução tecnológica no domínio da microelectrónica permitiu a introdução de técnicas digitais ao nível da transmissão e comutação. A introdução das fibras ópticas permitiu a qualidade das ligações, aumentar o seu débito, transportar um maior e diversificado números de serviços e alargar a área geográfica abrangida.

As actuais redes públicas de comunicação são caracterizadas por uma especialização de acordo com o tipo de serviço transportado. Vejamos alguns exemplos de redes públicas:

- Rede telefónica pública:

Esta rede, também conhecida como rede pública de comutação de circuitos telefónicos (PTSN, *Public Switched Telephone Network*), oferece aos utilizadores o sistema clássico de transmissão de voz bidireccional;

- Redes públicas de dados:

A informação é transportada através de uma rede de comutação de pacotes (PSDN, *Packet Switched Data Network*) baseada em protocolos X.25 ou, em alguns países, através de uma rede digital de comutação de circuitos (CSDN, *Circuit Switched Digital Network*) baseada em protocolos X.21;

- Redes de TV:

Os sinais de televisão podem ser transportados de três maneiras: difundidos através de ondas rádio usando antenas terrestres, por uma rede de transmissão por cabo (CATV, *Community Antenna TV*) e via satélite por meio do sistema DBS (*Direct Broadcast System*).

Cada uma destas redes foi especialmente concebida para transportar um serviço específico e geralmente não pode ser usada para transportar de forma eficiente um serviço diferente.

No entanto, existe um número limitado de casos em que uma rede, destinada a transportar determinado tipo de serviço, consegue transportar outro diferente. É o caso, por exemplo, da rede telefónica que consegue transportar dados de débito limitado quando usa *modems* em ambos os extremos da linha.

Uma consequência imediata desta especialização em função do serviço é a existência de um elevado número de redes independentes, cada uma requerendo a sua fase de projecto, implementação e manutenção. Além disso, o dimensionamento de cada rede é feito individualmente para cada serviço específico o que impossibilita o uso dos recursos disponíveis de uma rede por um outro serviço para o qual a rede não foi projectada.

Neste contexto tornou-se atraente a criação de uma rede universal, capaz de integrar todo o tipo de serviços.

O primeiro passo neste sentido foi dado com a introdução da Rede Digital com Integração de serviços (RDIS[8]). Esta rede, embora capaz de integrar serviços de voz e dados num único meio de transmissão, não permite o transporte de sinais TV devido às suas capacidades limitadas em termos de largura de banda. Por outro lado, embora havendo um interface de acesso normalizado entre o utilizador e a rede, coexistem internamente e de uma forma temporária, duas redes especializadas em voz e dados.

Uma outra desvantagem da RDIS é a incapacidade desta poder beneficiar com a evolução dos algoritmos de codificação de voz e imagem. Uma vez que o transporte de informação RDIS é baseado em canais de débito fixo, os recursos da rede serão usados de uma forma ineficiente, já que o débito necessário para transmitir determinado serviço será cada vez mais reduzido para além de alguns serviços poderem gerar tráfego de débito variável (VBR, *Variable Bit Rate*).

Assim evoluiu-se para uma nova rede, com os seguintes requisitos:

- Flexibilidade e capacidade de evolução:

Os avanços no estado da arte dos algoritmos de codificação e da tecnologia VLSI (*Very Large Scale Integration*) poderão reduzir a largura de banda necessária ao transporte dos serviços existentes. Uma rede capaz de transportar todo o tipo de serviços será também capaz de se adaptar às novas necessidades que se lhe apresentarão.

- Uso eficiente dos recursos disponíveis:

Os recursos da rede poderão ser partilhados por todos os serviços, de modo a obter uma óptima partilha estatística de recursos.

- Baixos custos:

Uma vez que uma só rede necessita de ser projectada, implementada, explorada e mantida, os custos globais serão naturalmente reduzidos. Este factor permitirá a criação de standards universais, atrair novos mercados e suscitar o interesse dos fabricantes.

A União Internacional de Telecomunicações (ITU[9], *International Telecommunication Union*) e o ETSI (*European Telecommunications Standards Institute*) têm vindo a desenvolver esforços no sentido de criar as recomendações necessárias à implementação de uma rede universal e integradora de todos os serviços de telecomunicações, definindo também um modo único de transferência de informação – o modo de transferência assíncrono (ATM).

[8] Ou ISDN – *Integrated Services Digital Network.*

[9] ITU antigo CCITT, *Comité Consultative International de Teléphone et Telégraphe.*

A Rede Digital com Integração de Serviços de Banda Larga (RDIS-BL[10]) permite um número elevado de serviços com requisitos diversos e novos serviços, associados à crescente solicitação de novos serviços pelo consumidor.

Dentre esses serviços encontramos a Televisão de alta definição (*HDTV- High definition TV*), Vídeo-Conferência, Transferência de dados de alto débito, Vídeo-telefone, Conferência multimédia e Vídeo-a-pedido. Cada um destes serviços exige da rede a satisfação de requisitos muito diferentes (por exemplo, débito, atraso, taxas de perdas) dos exigidos às redes tradicionais, o que introduz à necessidade de criação de uma rede universal, concebida segundo princípios diferentes, capaz de suportar este grande número de serviços.

3.2. Modos de Transferência de Informação

O Modo de Transferência Assíncrono (ATM), juntamente com o Protocolo de Adaptação (AAL), são referidos como a tecnologia capaz de dotar a Rede Digital com Integração de Serviços de Banda Larga (RDIS-BL) com os requisitos necessários ao transporte integrado de todo o tipo de serviços. Através do uso do ATM, serviços como a voz, vídeo e dados poderão ser transmitidos, multiplexados, juntamente, num único formato universal. A rede resultante será por isso mais simples, fácil de gerir, de administrar e usar os recursos de uma forma eficiente.

Existem, no entanto, outros modos de transferência de informação, passíveis de serem utilizados na RDIS-BL.

O modo ATM, foi desenvolvido nos anos 80 com o objectivo de fornecer um modo de transferência capaz de suportar elevados débitos, com flexibilidade, escalabilidade e tecnologia suporte para as LAN's, MAN's e WAN's. É baseado no princípio de transferência rápida de pacotes, que consiste num desenvolvimento da convencional transferência de pacotes.

O sistema ATM foi desenvolvimento graças ao aparecimento de novas tecnologias digitais, nomeadamente fibras ópticas e electrónica de comutação rápida.

A tecnologia ATM permite efectuar uma gestão eficiente e flexível dos recursos de rede, permitindo o transporte integrado de diferentes tipos de dados e aplicações com diferentes requisitos. Alia as vantagens da flexibilidade e eficiência das redes tradicionais de computadores, às garantias de qualidade de serviço necessárias à transmissão fiável de serviços de vídeo com requisitos de tempo real.

O modo de transferência assíncrona permite satisfazer serviços e aplicações com requisitos de largura de banda e qualidade de serviço (QoS) muito distintos, já que define um conjunto diversificado de categorias de serviço e que permite a negociação

[10] Ou IBCN – Integrated Broadband Communications Network.

dos parâmetros de transmissão para cada ligação. É assim possível optimizar os recursos da rede face aos requisitos específicos de cada aplicação com a flexibilidade necessária para acompanhar variações ao longo do tempo desses mesmos requisitos.

Nas redes ATM a informação é partida em pacotes de dimensão fixa e pequena designados por células, as quais são encaminhadas e comutadas através da rede até ao seu destino. Cada célula tem um comprimento de 53 bytes, sendo 5 bytes reservados para um cabeçalho com informação de encaminhamento, prioridades, etc., e os restantes 48 bytes destinados ao campo de dados da aplicação.

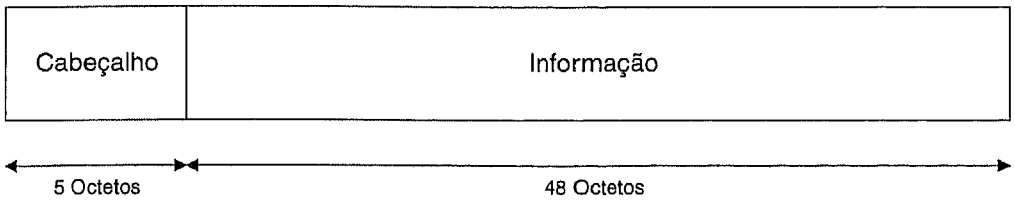


Figura 3.1 – Formato da célula ATM.

3.2.1 Modelo de Referência Protocolar

À semelhança do modelo OSI (*OSI - Open Systems Interconnection*) da ISO (ISO - *International Standard Organization*), o modelo de referência protocolar (*PRM -Protocol Reference Model*) descrito na Rec. 1.321 do ITU, satisfaz os princípios da modularidade e independência da tecnologia, definindo os serviços e os protocolos por camadas e as respectivas primitivas de serviço.

O modelo adoptado na RDIS-BL está esquematizado na figura seguinte:

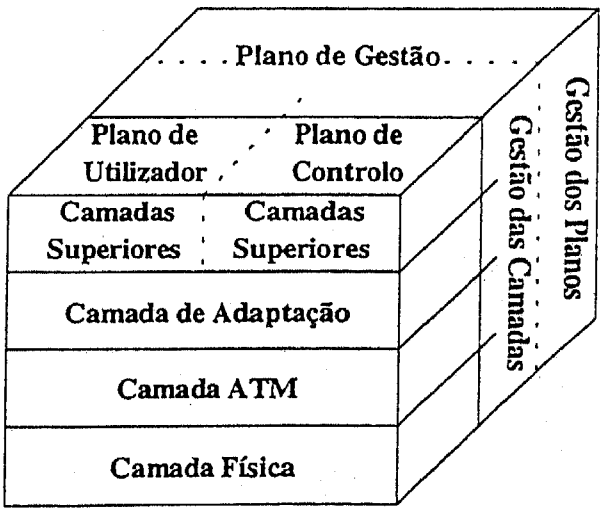


Figura 3.2 - Modelo de Referência Protocolar.

Tal como no PRM da RDIS, existem três planos: o plano de utilizador, onde é efectuado o transporte da informação a ele associado, o plano de controlo, que lida principalmente com informação de sinalização, e o plano de gestão, usado para fazer a coordenação entre os planos e executar funções operacionais.

De acordo com o ITU, as camadas podem ser divididas tal como se apresenta na Figura 3.2. As funções das camadas superiores do PRM são dependentes dos serviços e ainda não estão completamente definidas. A camada de adaptação (*AAL - ATM Adaptation Layer*) executa funções dependentes dos serviços suportados pelas camadas superiores.

A camada ATM é independente dos serviços e da camada física associada ao meio de transmissão.

Na camada AAL, as funções dependentes dos serviços constituem a sub-camada de convergência (*CS- Convergence sublayer*) e as funções de adaptação à camada formam a sub-camada de segmentação e reassemblagem (*SAR - Segmentation and Reassembly Sublayer*).

Na camada Física PL (*PL - Physical Layer*), as funções dependentes do meio físico constituem a sub-camada de Meio Físico (*PM - Physical Medium*) e as funções de adaptação à camada ATM fornam a sub-camada de convergência de transmissão (*TC - Transmission Convergence*).

Tabela 3.1 - Funções e níveis do PRM.

Camadas Superiores		
Convergência	CS	AAL
Segmentação e Reassemblagem	SAR	
Controlo global de fluxo		
Geração e extracção do cabeçalho da célula		ATM
Comutação		
Multiplexagem e desmultiplexagem de células		
Desacoplamento do débito de célula		
Verificação/Criação da sequência do HEC do cabeçalho		
Delineação de células	TC	
Adaptação para transmissão de tramas		PL
Inserção e extracção de células em tramas		
Extracção do relógio de bit	PM	
Meio físico		

3.2.2 A camada ATM

As características da camada ATM são independentes do sistema de transmissão e do meio de transmissão utilizados. A adaptação da camada ATM ao sistema de transmissão é feita ao nível da camada Física. Esta separação, entre a camada ATM e a transmissão, permite que comutadores e multiplexers ATM, possam ser introduzidos na rede e evoluir independentemente dos aspectos de transmissão da rede.

As funções da camada ATM aparecem reflectidas na estrutura do cabeçalho da célula. Os 5 octetos do cabeçalho são repartidos por diversos campos, conforme mostra a figura 3.3.

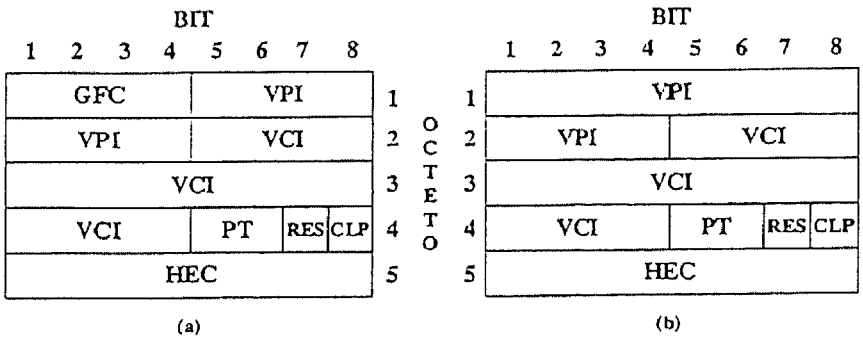


Figura 3.3- Estrutura do cabeçalho da célula ATM nas interfaces de utilizador (a) e de rede (b).

A estrutura do cabeçalho é diferente na interface utilizador-rede (*UNI - User Network interface*) e nas interfaces internas da rede, designadas interfaces entre nós da rede (*NNI - Network Node interface*).

A diferença consiste somente na existência do campo para controlo de fluxo (*GFC - Generic Flow Control*), que é utilizado na interface utilizador-rede, no caso de existirem configurações com múltiplos utilizadores, para controlar o acesso destes utilizadores à rede. Como o campo GFC é desnecessário nas interfaces internas da rede, é aproveitado nas interfaces entre os nós para aumentar o comprimento do campo VPI.

A identificação de um canal lógico ATM está dividida em duas entidades hierárquicas: caminho virtual (*VP - Virtual Path*) e canal virtual (*VC - Virtual Channel*).

Estas entidades são identificadas no cabeçalho da célula pelo identificador de caminho virtual (*VPI - Virtual Path identifier*) e pelo indicador de canal virtual (VCI, Virtual Channel identifier).

Numa dada interface, um canal de comunicação é identificado pelo campo (VPI+VCI) completo. A relação hierárquica entre caminhos e canais virtuais é indicada na Figura 3.4.

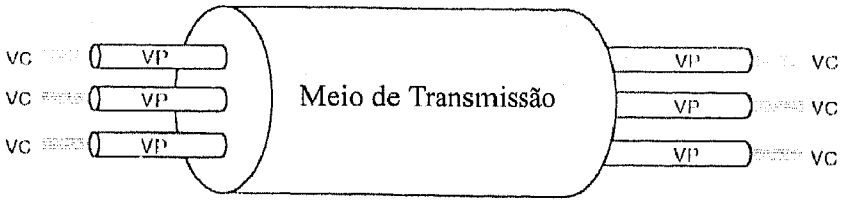


Figura 3.4 - Relação entre canais virtuais e caminhos virtuais.

A existência de caminhos virtuais permite que a rede suporte ligações semi-permanentes entre utilizadores, comutando caminhos virtuais, através de sistemas *cross-connect*, tratando de um modo global todos os canais virtuais pertencentes a um caminho virtual.

Por outro lado, as ligações na rede são estabelecidas através de comutadores ATM, que comutarão caminhos e canais virtuais individualmente. A comutação é feita através de uma tabela que mapeia portas de entrada em portas de saída baseada em valores do VPI e VCI da célula, como podemos ver na figura 3.5 .

O bit CLP (*CLP - Cell Loss Priority*) indica a prioridade de perda de célula. No caso de congestão da rede, as células de prioridade mais baixa serão as primeiras a ser eliminadas.

Os dois bits PT (*PT - Payioad Type*) indicam o tipo de informação contida na célula. A célula pode conter informação do utilizador ou informação para gestão e manutenção da rede.

O campo HEC (*HEC- Header Error Control*) é um campo para controlo de erros no cabeçalho. Devido ao mecanismo de controlo de erros ser também usado para determinar a delimitação de células, considera-se que este pertence à camada Física.

Existe ainda um bit (RES - Reserved Field) reservado para futura normalização.

3.2.3 A camada de adaptação

O objectivo do modo de transferência ATM é permitir uma comutação/encaminhamento simples e flexível. Na realidade, as funções da camada ATM satisfazem os requisitos de simplicidade e flexibilidade, atendendo a que a camada ATM fornece um serviço que é independente da estrutura da unidade de informação e do débito dos serviços suportados. Porém, estas características conduzem aos seguintes inconvenientes ao nível da camada ATM:

- não há informação quanto à frequência de relógio correspondente ao serviço suportado;
- existe uma variação no atraso de propagação das diferentes células, devido aos diferentes tempos de propagação das células nos comutadores da rede;
- não existe informação sobre a delimitação das unidades de informação ao nível do serviço de telecomunicações suportado;
- não são detectadas as células perdidas na rede, como resultado de um enchimento nos *buffers* ou erro no identificador da célula;
- células mal inseridas, isto é, células que são entregues ao destino errado sem serem detectadas.

Toma-se assim necessário garantir, para cada serviço de telecomunicações, que a qualidade de serviço não seja degradada, por causa dos referidos inconvenientes. O papel da camada AAL é garantir as soluções apropriadas para cada caso, de modo a alcançar-se a qualidade de serviço requerida.

As características da camada AAL estão dependentes do serviço. Porém, em geral, as funções que a camada AAL deve executar para complementar o serviço da camada ATM são as seguintes:

- recuperação da frequência do relógio correspondente ao serviço suportado, quando requerido pelas características do serviço;
- compensação da variação do atraso das células, ainda quando requerido pelas características do serviço;
- mapeamento das unidades de informação da camada acima da AAL nos campos de informação das células;
- detecção da ocorrência de células perdidas e o desencadear das medidas necessárias para diminuir o impacto desta situação na qualidade de serviço;
- detecção da ocorrência de células mal inseridas e sua correspondente eliminação.

Embora a maneira precisa como a camada a AAL executa as funções referidas dependa do serviço é um objectivo claro reduzir, tanto quanto possível, o número de diferentes protocolos AAL. Como há algumas funções comuns entre os diferentes serviços, eles podem ser agrupados num pequeno número de classes, correspondendo cada classe a um determinado tipo de AAL, com um conjunto bem definido de funções básicas.

Para cada tipo de AAL estas funções básicas são realizadas numa subcamada de AAL, chamada subcamada de segmentação e reassemblagem (*SAR - Segmention And Reassembly sublayer*). A subcamada acima desta é chamada subcamada de convergência (*CS - Convergence Sublayer*) e implementa as funções AAL complementares às básicas, requeridas para um serviço específico dessa classe. Haverá, portanto, para cada tipo de AAL um CS distinto para cada tipo de serviço.

Tabela 3.2 - Classificação das diferentes classes de serviços.

	Classe A	Classe B	Classe C	Classe D
Relação Temporal entre Fonte e Destino	Necessária		Não Necessária	
Débito	Constante	Variável		
Modo de Operação	Com Conexão			Sem conexão

Os serviços transportados pela camada ATM são classificados em 4 classes, possuindo cada uma requisitos específicos na camada AAL. De modo a definir estas 4 classes, os serviços são classificados de acordo com os seguintes parâmetros (Tabela 3.2):

• Relação Temporal entre fonte e destino:

Vários serviços requerem que seja preservada a relação temporal entre a fonte e o destino. Por exemplo, em canais de voz PCM a 64 kbits, existe urna relação temporal bem definida entre fonte e destino. A informação transferida entre computadores não requer esta relação.

• Débito:

Alguns serviços produzem informação com um ritmo constante e outros com um ritmo variável.

• Modo de conexão:

Os serviços podem ser orientados à conexão (*connection-oriented*) ou sem conexão (*conection-less*).

Para cada uma destas classes foram previstos inicialmente quatro tipos de AAL, a que corresponde um protocolo específico ao nível da subcamada SAR. Posteriormente, foi introduzido um novo tipo (AAL 5), e os níveis AAL 3 e AAL 4 foram fundidos num único (AAL 3/4). No entanto, um mesmo AAL pode ser usado para transportar serviços de classes diferentes.

De seguida apenas serão referenciados os tipos relevantes para a tese. O AAL1 utilizado na transmissão de serviços de débito constante (CBR), o AAL5 para débito variável (VBR) e o AAL2 como possível solução no futuro para VBR.

AAL - Tipo 1

A Figura 3.7 representa o formato da unidade de dados do protocolo (*PDU - Protocol Data Unit*) para a subcamada de segmentação e reassemblagem (SAR-PDU) na camada de adaptação do tipo 1. Neste tipo de AAL, a informação transferida entre a fonte e o destino tem um ritmo constante.

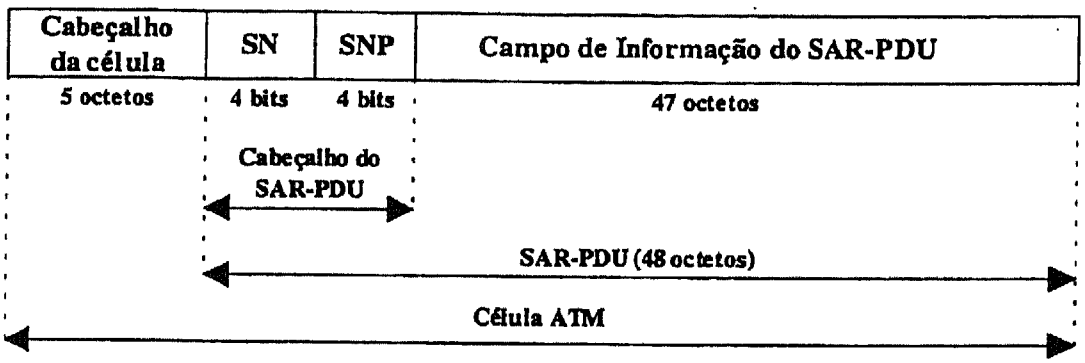


Figura 3.7 - Unidade de dados do protocolo para a sub-camada SAR do AAL-1.

O primeiro octeto do campo de informação da célula é utilizado para a inserção de um número de sequência (*SN - Sequence Number*) e do respectivo campo de protecção (*SNP - Sequence Number Protection),* baseado no polinómio gerador $G(x) = x^3 + x + 1$ com um bit extra de paridade. A existência do campo SN permite detectar, na recepção, se houve células perdidas ou mal inseridas durante a transmissão. Os restantes 47 octetos transportam a informação da subcamada superior CS. É ao nível da subcamada CS que os diversos serviços do tipo 1 são adaptados. Para alguns serviços específicos a subcamada CS suporta algumas funções especiais, de que são exemplos:

- Áudio e Vídeo de alta qualidade e de ritmo constante:

Neste caso, é necessário implementar um mecanismo de correcção de erros ao nível do campo de informação. Este processo pode ser combinado com um método em que os bits são entrelaçados antes de serem colocados nas células.

- Voz:

A função principal a realizar ao nível da sub-camada CS é a recuperação de relógio na recepção baseada na sequência de chegada de células. Esta técnica pode ser realizada através do seguimento do nível de enchimento do *buffer* de recepção.

AAL - Tipo 2

O AAL tipo 2 possibilita transferência de informação com débito variável. Deste modo, é necessário transferir informação temporal entre a fonte e o destino. Uma vez que a fonte gera informação com um ritmo variável, é possível que as células não sejam completamente preenchidas, o que leva a incluir funções adicionais na sub-camada SAR.

O formato proposto (mas ainda não normalizado pelo ITU) para à unidade de dados do protocolo da sub-camada SAR do AAL-2 é mostrado na Figura 3.8.

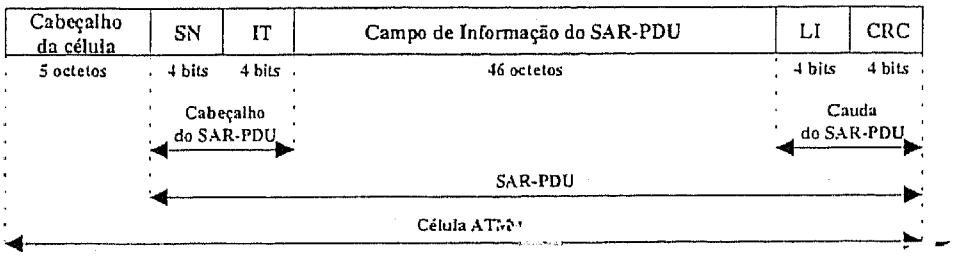


Figura 3.8 - Unidade de dados do protocolo proposto para a sub-camada SAR do AAL-2.

O campo SN (*SN - Sequence Number*) tem as mesmas funções do descrito para o tipo 1.

O campo IT (*IT - Information Type*) indica o início de uma mensagem (*BOM - Beginning Of Message*), a continuação de uma mensagem (*COM - Continuation Of Message*), o fim de uma mensagem (*EOM - End Of Message*), ou se a célula transporta informação temporal ou dum outro tipo. BOM, COM e EOM indicam, respectivamente, que a célula é a primeira, intermédia ou a final de uma mensagem. O campo LI (*LI - Length Indicator*) indica o número útil de bytes nas células preenchidas parcialmente, e é protegido pelo campo CRC (*CRC - Cyclic Redundancy Code*).

Na subcamada CS é necessário implementar as seguintes funções:

- recuperação do sinal do relógio através da inserção de uma indicação explicita de sincronização;
- tratamento de células perdidas ou mal inseridas;
- implementação de um mecanismo de FEC (*FEC - Forward Error Correction*) para serviços de áudio e vídeo.

AAL - Tipo 5

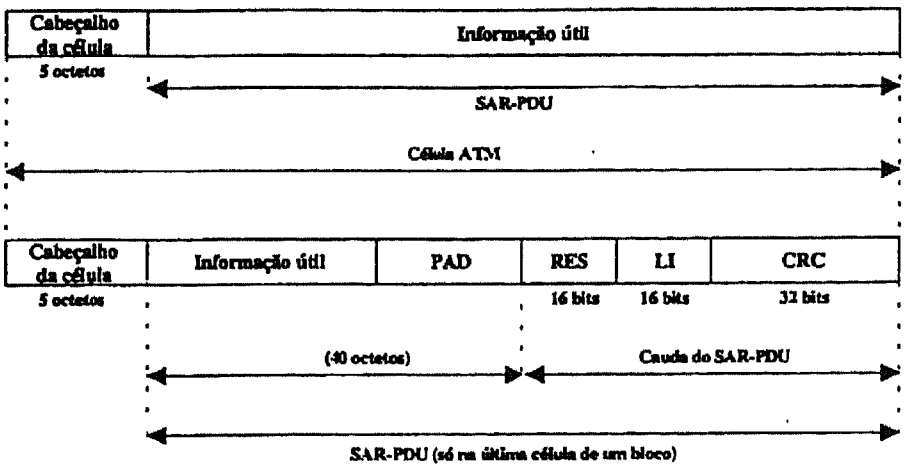


Figura 3.9 - Unidade de dados do protocolo para a subcamada SAR do AAL-5.

Os serviços caracterizados pela necessidade de transmissão de grandes blocos de informação podem utilizar a totalidade da capacidade da unidade de dados do protocolo (SAR-PDU). A última célula do bloco de informação tem uma cauda de 8 octetos, sendo 4 para protecção do bloco com um código cíclico redundante (CRC), 2 octetos para indicação do comprimento (LI) e, finalmente, 2 octetos (*RES - Reserved Field*) para aplicações não especificadas. A última célula inclui ainda um campo de comprimento variável de 0 a 48 octetos (*PAD - Padding*), para ajustar o comprimento do bloco de informação a um múltiplo de 48 octetos.

Na Figura 3.9 representa-se a unidade de dados do protocolo da subcamada de segmentação e reassemblagem (SAR-PDU) da camada de adaptação do tipo AAL - 5. A delimitação de PDUs é efectuada através de um campo específico no cabeçalho da célula (*PT- Payload Type*).

3.2.4. Categorias de serviços

A tecnologia ATM propõe-se efectuar de modo integrado e flexível a transmissão de qualquer tipo de aplicação com requisitos específicos. Por esse motivo foram definidas várias categorias de serviço ATM, cada uma delas com características homogéneas de tráfego, de requisitos de qualidade de serviço e de mecanismos de controlo. Cada categoria é adequada a um determinado tipo de alocação de recursos, relacionando requisitos de qualidade e características de tráfego da aplicação com procedimentos e parâmetros da rede.

As categorias de serviço existentes foram definidas de acordo com a normalização ITU-T e especificadas no ATM-Forum [TMS4], [Perth].

As categorias de serviço ATM definidas pelo ATM Forum ou pelo ITU-T embora tenham nomes diferentes possuem características homogéneas em termos de padrões

de tráfego, requisitos de qualidade de serviço (QoS - *Quality of Service*) e mecanismos de controlo.

Tabela 3.3 - Relação entre os serviços do ATM Forum e os do ITU-T ATM.

Categoria de Serviço ATM (ATM Forum TM4.0)	Capacidade de Transferência ATM (ITU-T I.371)	Uso típico
CBR - Constant Bit Rate	DBR – Deterministic Bit Rate	Tempo real, garantias QoS
RT-VBR – Real Time Variable Bit Rate	Em estudo	Multiplexagem estatística em tempo real
NRT-VBR – Non Real Time Variable Bit Rate	SBR - Statistical Bit Rate	Multiplexagem estatística
ABR – Available Bit Rate	ABR – Available Bit Rate	Exploração de recursos, utilização de canal de retorno para controlo
UBR – Unspecified Bit Rate	Sem equivalente	Sem exigências de garantia de qualidade de serviço
Sem equivalente	ABT - ATM Block Transfer	Nível de *burst* com canal de retorno para controlo

Daqui em diante os nomes das categorias de serviço serão baseados no ATM Forum [TMS4].

3.2.4.1 Débito binário Constante (CBR- *Constant Bit Rate)*

A categoria CBR é utilizada em ligações que requeiram uma quantidade fixa de largura de banda, caracterizada pelo débito máximo de emissão na fonte, ou seja pelo (PCR- *Peak Cell Rate*), que é disponibilizada durante a ligação. A fonte pode emitir células abaixo do PCR durante a ligação, ou até mesmo parar de emitir. Esta categoria é utilizada para aplicações em tempo real, isto é, aquelas que exigem garantias no serviço (CTD- *Cell Transfer Delay*) e (CDV- *Cell Delay Variation*), mas não é restrita a estas aplicações. Pode ser apropriada para aplicações de vídeo e áudio, bem como serviços com emulação de circuitos (CES- *Circuit emulation Services*).

A rede assegura, após o estabelecimento da ligação a qualidade de serviço (QoS) negociada previamente a todas as células de acordo com os testes de conformidade. É assumido que todas as células cujo atraso seja maior que o CTD sejam de importância menor para a aplicação.

3.2.4.2 Débito binário variável com requisitos de tempo real (rt-VBR – R*eal-Time Variable Bit Rate*)

O débito binário variável com requisitos de tempo real é vocacionado para aplicações sensíveis a atrasos temporais. Como exemplos podemos citar aplicações de voz e

vídeo. As fontes transmitem a um débito variável com o tempo. Exigem garantias no serviço, nomeadamente atrasos, perdas e desvio (*jitter*).

Os parâmetros de tráfego são : o *Peak Cell Rate* (PCR), *Sustainable Cell Rate* (SCR), *Maximum Burst Size* (MBS). As células cujo atraso é maior que o CTD são consideradas de importância menor para a aplicação. O serviço RT-VBR pode suportar multiplexagem estatística de fontes em tempo real.

3.2.4.3 Débito binário variável sem requisitos de tempo real (nrt-VBR – *Non Real-Time Variable Bit Rate*)

O débito binário variável é utilizado em aplicações que geram tráfego em rajadas, onde o controlo do jitter não é necessário, mas é exigido um máximo no atraso. Como parâmetros de tráfego encontramos : PCR, SCR e MBS. Para aquelas células que são transferidas dentro do contrato de tráfego, a aplicação espera um baixo número de células perdidas (*CLR - Cell Loss Ratio*). Para as restantes é esperado um limite no atraso de transferência das células (*Cell Transfer Delay – CTD*). O serviço nrt-VBR pode suportar multiplexagem estatística nas ligações.

3.2.4.4 Débito binário disponível (ABR - *Available Bit Rate*)

O débito binário disponível é utilizado em aplicações tolerantes a atrasos, em que a reserva de recursos da rede varia no tempo, de acordo com a disponibilidade e utiliza um canal de retorno para controlar o potencial congestionamento da rede.

Existem inúmeras aplicações que possuem requisitos vagos no que diz respeito ao débito necessário. Ou seja em vez de um valor médio, é definido uma gama de valores aceitáveis, por exemplo um máximo e um mínimo. Assim quando ao estabelecer um serviço ABR o sistema define uma largura de banda máxima e uma mínima, que pode ser zero. Estes valores são definidos, respectivamente, como *Peak Cell Rate* (PCR) e *Minimum Cell Rate* (MCR).

A largura de banda disponível na rede pode variar dado que é a soma do MCR com um débito variável, que resulta da partilha da capacidade disponível com todas as conexões ABR através de uma política definida. Neste serviço é especificado um mecanismo de controlo de fluxo, que suporta vários tipos de retroacção para controlar o débito da fonte.

Apesar de não ser negociado nenhum parâmetro QoS no serviço ABR, é esperado que um sistema que adapte o seu tráfego de acordo com a resposta no canal de retorno, possua uma baixa perda de células *(CLR – Cell Loss Ratio)* e uma razoável largura de banda disponível.

O serviço ABR não pressupõe a sua utilização em aplicações com requisitos de tempo real.

3.2.4.5 Débito binário não especificado (UBR - *Unspecified Bit Rate*)

O débito binário não especificado é utilizado em aplicações que não tenham exigências de garantia de qualidade de serviço.

As fontes UBR transmitem rajadas não contínuas de células, suportando um elevado grau de multiplexagem estatística entre as fontes. O serviço UBR não inclui a noção de uma negociação da largura de banda na ligação.

Na tabela seguinte podemos sintetizar, de um modo geral, os parâmetros de qualidade existentes nas redes ATM.

Tabela 3.4 – Parâmetros na Qualidade de serviço no ATM.

Característica	CBR	rt-VBR	nrt-VBR	UBR	ABR
Variação no atraso da célula (CDV)	Especificável	Não especificável	Não especificável	Não especificável	Não especificável
Máximo atraso da célula (MCTD)	Especificável	Especificável	Especificável	Não especificável	Em estudo
Percentagem da perda de células(CLR)	Especificável	Especificável	Especificável	Não especificável	Especificável
Percentagem de células com erros (CER)	Especificável	Especificável	Especificável	Não especificável	Especificável

Capítulo 4

MPEG-2 em redes ATM

Embora a norma não especifique como o fluxo de transporte MPEG-2 deve ser enviado numa rede de comunicações, certas condições devem ser asseguradas de modo a garantir qualidade satisfatória no receptor.

O grupo de trabalho SAA (SAA – *Service Aspects and Applications*) do ATM Forum votou um mecanismo para suportar o MPEG2 em AAL5. Este mecanismo é activado através duma camada de convergência AVSSCS (AVSSCS – *Audio-Visual Service-Specific Convergence Sublayer*), que se encontra no estado de especificação. Esta subcamada irá ser projectada para outras compressões de vídeo e protocolos de codificação. Esta solução foi votada por duas razões. Primeiro o AAL1 fornece marcos de tempo (SRTS – *Synchronous Residual Time Stamp*) e é usado nas conexões CBR. Mas como o MPEG-2 também precisa de recuperar a informação temporal e utiliza o PCR (PCR- *Program Clock Reference*) nas suas tramas elementares, torna-se redundante enviar informação temporal na camada MPEG-2 e na camada AAL. Adicionalmente a existência de SRTS e da sobrecarga[11] adicional no AAL1 demonstram que o AAL5 é de facto uma melhor escolha.

O AAL5 possui menos sobracarga e o comprimento do PDU é múltiplo de 48 bytes, coincidindo com o tamanho dos pacotes MPEG-2. Adicionalmente a proliferação de sistemas com suporte AAL5, garantem baixos custos na transmissão de serviços MPEG-2 em redes ATM.

[11] Do inglês *overhead*

Encontra-se em desenvolvimento a criação de uma categoria de débito binário variável imediato (rt-VBR) para UNI 4.0, pelo reconhecimento das características do tráfego de Vídeo e o controlo do *jitter* e da latência.

O ATM introduz latência e *jitter*, que serão controlados pelo AAL5, permitindo ao utilizador negociar com a rede os seus níveis. Adicionalmente a taxa de perdas de células e células com erros também são negociáveis, fornecendo ao utilizador um nível aceitável na qualidade do serviço[Cisco1].

No presente capítulo são apresentadas algumas opções na implementação da norma MPEG-2 nas redes ATM, nomeadamente a camada de adaptação e a categoria de serviço a escolher. No fim são apresentados alguns serviços normalizados, como por exemplo o Vídeo-a-pedido.

4.1. Escolha da camada de adaptação

A camada de adaptação é responsável por tornar o comportamento da rede transparente para a aplicação.

A escolha de uma camada de adaptação para o transporte das tramas MPEG-2 pressupõe certas condicionantes como por exemplo a remoção do *jitter*, a detecção e/ou correcção de erros, a minimização do atraso das transmissões em tempo real e o suporte de aplicações CBR e VBR.

4.1.1 A Camada de Adaptação tipo 1 (AAL 1)

A camada de adaptação do tipo 1 é especialmente utilizada para emulação de circuitos em redes ATM. AAL1 é ideal para o transporte de débito binário constante (CBR) dado que o atraso na rede é constante e são utilizados mecanismos na recepção que removem o *jitter*.

AAL1 fornece dois modos de sincronizar os relógios e entregar na recepção um relógio sem desvios, dependendo do estado do relógio de serviço CBR.

No caso de haver sincronismo entre o relógio da rede e o relógio de serviço, a sua recuperação é feita directamente a partir do relógio da rede.

No caso assíncrono o AAL1 fornece duas técnicas para a recuperação do relógio na recepção: o método SRTS (SRTS - *Synchronous Residual Time Stamp*) e o método adaptativo.

O método SRTS só pode ser usado se existir um relógio de referência comum na emissão e na recepção, enquanto o adaptativo não necessita de qualquer relógio de referência, sendo feita a sincronização através dos níveis dos buffers. Além do mais o AAL1 fornece a opção da correcção de erros (FEC – *Forward Error Correction*) que pode esconder da aplicação os efeitos da perda de células na rede.

AAL1 é actualmente a escolha natural para o transporte de CBR MPEG-2 em redes ATM dado que o tráfego tem um débito constante e precisa de um atraso constante. No entanto existem inúmeras desvantagens com o uso do AAL1 para o transporte de tramas MPEG-2:

1. O AAL1 não pode ser usado para transportar tramas MPEG-2 com débito binário variável (VBR), que é o método mais provável no futuro.

2. A técnica SRTS não pode ser usada se o relógio da rede não estiver disponível como referência. Ou seja o AAL1 não pode ser usado em redes compostas por diversos relógios dessincronizados.

3. O método adaptativo para recuperar o relógio precisa de uma PLL (PLL – *Phase Locked Loop*) para determinar como o *buffer* no descodificador está a ser esvaziado. Dado que é precisa uma PLL para recuperar o relógio de sistema do MPEG-2 a anterior torna-se redundante.

4. Dado que a sinalização é feita em AAL5, a interface da rede ATM iria precisar de suportar os dois tipos de camadas de adaptação (AAL1 e AAL5), o que se revelaria uma escolha cara.

4.1.2 A Camada de Adaptação tipo 5 (AAL 5)

A camada de adaptação do tipo 5 foi projectado para o tráfego de dados, em tempo real, em ATM. A sua camada de convergência consiste em duas subcamadas: a CPCS (*CPCS - Common Part Convergence Sublayer*) e a SSCS (*SSCS - Service-Specific Convergence Sublayer*).

O CPCS do AAL5 pode fazer uso do tamanho variável das unidades de dados do protocolo (PDUs), que podem variar entre 1 e 65536 bytes. Isto significa que o tamanho de unidade de dados do protocolo (AAL5 PDU) não é fixo.

O SSCS fornece a flexibilidade de haver uma subcamada específica para os diferentes serviços que precisam utilizar o AAL5. O SSCS pode ser nulo no caso de não ser preciso executar nenhuma tarefa específica.

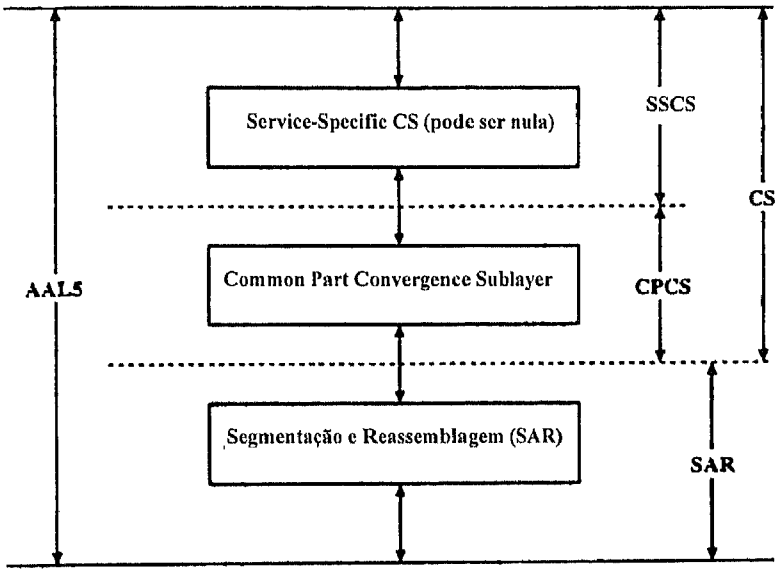


Figura 4.1 – Estrutura da camada AAL5.

O CPCS em conjunto com a camada de segmentação e reassemblagem (SAR) fornece as capacidades de enviar e receber as unidades de dados de serviço (*CPAAL5 SDU-Common Part AAL5 service Data Unit*) de uma rede ATM.

No caso de uma unidade de dados de serviço (SDU) for perdida ou corrompida, uma indicação é enviada para o SSCS (ou para a camada de serviço se o SSCS for nulo). Mas neste caso os SDUs perdidos ou corrompidos não são recuperados como no caso do AAL1, visto não haver correcção de erros (FEC). Uma unidade de dados de serviço (SDU) corrompida pode opcionalmente ser direccionada para a camada SSCS, deixando a esta a responsabilidade de corrigir o erro. Um SDU corrompido pode ser detectado à custa do CRC de 32 bit no fim do SDU e pela verificação do comprimento do SDU no campo do seu cabeçalho.

No caso de serviços audiovisuais, uma subcamada de convergência de serviços vídeo e áudio (*VASSCS - Video Audio Service Specific Convergence Sublayer*) é proposta de modo a recuperar o relógio e o atraso constante que o MPEG-2 assume para a rede.

O VASSCS pode ser projectado para obter serviços MPEG-2 com CBR e VBR. Para implementar um atraso constante, o VASSSCS utiliza um mecanismo *time stamp* e um buffer de remoção do *jitter* na camada de adaptação do receptor. Adicionalmente o VASSCS duplica funções presentes na camada de sistemas do MPEG-2. Mas não é verosímil a necessidade de aumentar a complexidade e os custos em detrimento de uma melhor qualidade. Num estudo [Perkins] demonstra-se que com um pequeno buffer (cerca de 50 kbytes), a PLL do descodificador MPEG-2 é capaz de recuperar a fonte do relógio, na base do pressuposto que o *jitter* introduzido pela rede é menor que 1ms. Assim sendo isto torna a opção VASSCS praticamente redundante. Mas na prática como é difícil negociar com a rede um *jitter* de 1ms, a remoção do *jitter* na camada AAL5 torna-se útil.

O AAL5 tem várias vantagens em relação a outras alternativas:

1. O AAL5 é actualmente a camada de adaptação mais utilizada pela indústria.

2. Com a opção do CS nulo, o suporte de hardware é minimizado e a complexidade é transferida para a camada de serviço.

3. O AAL5 pode no futuro suportar tráfego VBR MPEG-2, enquanto o AAL1 pode ser usado apenas com tráfego CBR.

No entanto também possui desvantagens:

1. A CPCS não suporta correcção de erros (FEC), mas apenas detecção de erros e no caso de haver erro uma unidade de dados de serviço (SDU) pode ser completamente rejeitada, aumentando assim o efeito do erro nas aplicações audiovisuais.

   Contudo, dado que as perdas por congestionamento na rede ATM são prováveis de acontecer em rajadas, a eficácia do FEC é limitada de qualquer forma.

2. Embora o AAL5 (como está definido actualmente) detecte erros num SDU, este não envia os SDUs corrompidos para a camada da aplicação. O que significa, no caso das tramas de transporte do MPEG-2 (MPEG-2 TS) uma perda de um ou mais pacotes na camada de serviço. Assim sendo uma perda excessiva torna as técnicas de recuperação de erros do MPEG muito ineficiente, dado que a sincronização dos macroblocos ao nível da camada de vídeo é posta em risco.

3. As diferentes formas de encapsular as tramas de transporte MPEG-2 em AAL5 SDUs pode afectar o *jitter* dos pacotes de transportem, resultando numa afectação do processo de recuperação do relógio de sistema.

4.2. O Encapsulamento da trama de transporte

Quando o AAL5 é escolhido como a camada de adaptação para transportar Tramas de Transporte MPEG-2, um esquema deve ser projectado para encapsular os pacotes de transporte MPEG-2 nos SDUs do AAL5.

Se cada pacote de transporte, cujo tamanho é de 188 bytes, for encapsulado num AAL5 SDU distinto, então serão precisas cinco células na camada ATM para cada pacote. Desta forma é perdida uma largura de banda significativa em cabeçalhos. O AAL1 é diferente na medida em que um pacote pode ser transportado em apenas quatro células AAL1 SDU. Para reduzir a largura de banda do cabeçalho, mais de um pacote de transporte deve ser encapsulado num AAL5 SDU.

Partindo do princípio que um SDU vai ser usado para encapsular N pacotes de transporte, haverá um *jitter* de empacotamento associado a cada pacote de transporte encapsulado no SDU, que contenha o PCR (*Program Clock Reference*) necessário para a recuperação do relógio no destino. Neste caso o *jitter* de empacotamento reflecte-se numa variação de atraso no descodificador, afectando a qualidade da recuperação do relógio. Para resolver este problema surgiram dois esquemas de empacotamento no AAL5, propostos pelo ATM Forum: o *PCR-unaware* e o *PCR-aware*.

No esquema *PCR-unaware* a camada de adaptação forma um AAL5 SDU a partir de N pacotes de transporte consecutivos, sem examinar a camada de transporte. Como todos os AAL5 SDU possuem um tamanho fixo, o resultado é uma trama de débito constante e o débito máximo de células pode ser calculado com exactidão. Neste caso o *jitter* de empacotamento depende do número de pacotes de transporte por AAL5 SDU, bem como do débito de transporte.

No esquema *PCR-aware* a camada de adaptação forma um AAL5 SDU a partir de N pacotes de transporte consecutivos, excepto quando o pacote de transporte contém um valor de PCR. Ao receber um pacote com o valor de PCR, o AAL5 SDU é enviado imediatamente para a subcamada SAR para segmentação. Isto é, no esquema *PCR-aware* o pacote de transporte que contém o valor de PCR aparece sempre como o último pacote no AAL5 SDU. Assim sendo o *jitter* de empacotamento para os pacotes que transportam os valores de PCR é essencialmente zero. No entanto esta abordagem exige que a fonte (o codificador ou o servidor de vídeo) seja capaz de detectar os pacotes de transporte com os valores PCR. Dado que o débito das tramas de transporte normalmente excede 1 Mbit/s, isto tem de ser feito por *hardware*, adicionando complexidade aos servidores de vídeo.

O ATM Forum recentemente escolheu o esquema *PCR-unaware* com N=2 como o modo preferido no transporte de MPEG-2 em ATM [VOD1]. Neste caso um pacote de transporte contendo o valor de PCR pode ocupar a primeira ou a segunda posição dentro do AAL5 SDU.

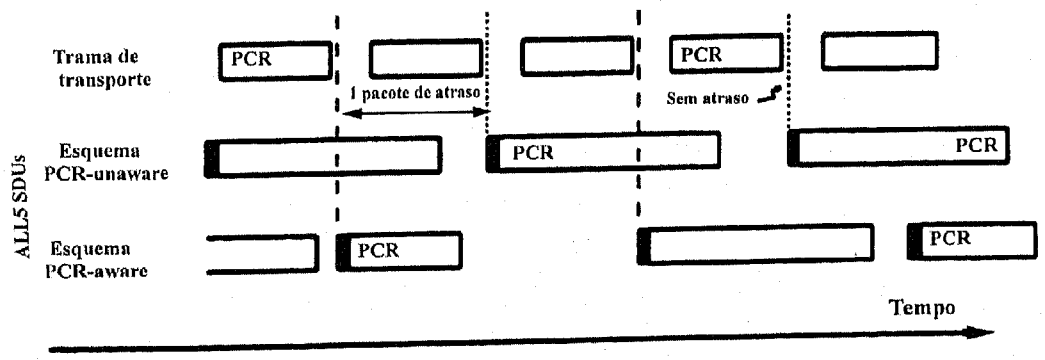


Figura 4.2 – Esquema de empacotamento para N=2.

Num estudo recente foi proposto um método alternativo [Tryfonas] que pode ser considerado como uma versão do esquema *PCR-unaware*. Neste método alternativo existem as vantagens do *PCR-aware* sem as respectivas desvantagens. Este método basicamente consiste em :

1. Controlar a geração dos pacotes de transporte que contém os PCRs: os PCRs podem ser colocados nos pacotes par ou impar de transporte.

2. Atraso no pacote de destino: o *jitter* de empacotamento ocorre quando o valor de PCR estiver presente no primeiro pacote de um AAL5 SDU. Neste caso o *jitter* é compensado no destino, ao permitir-se uma espera do segundo pacote na camada de adaptação enquanto não for transferido o primeiro pacote para a camada de serviço, usando a informação do débito de transporte.

3. Melhoria da PLL: dado que a quantidade de *jitter* é conhecido a priori, o atraso é eliminado se subtrairmos do valor do PCR um dado valor fixo.

No entanto este método possui um senão, dado que se houver uma perda de um pacote de transporte MPEG-2 é difícil determinar o alinhamento dos pacotes (par ou ímpar) para executar a acção apropriada de correcção.

4.3. A escolha da categoria de serviço

4.3.1 Débito Binário Constante (CBR)

Outro problema que surge no transporte de MPEG-2 em redes ATM é a selecção da categoria de serviço. Existem várias soluções propostas na literatura.

Numa primeira proposta surge a de débito binário constante (CBR) determinístico.

Nesta proposta o fluxo MPEG-2 é considerado de débito constante (CBR) e a rede é tratada como tal. O débito constante é previamente calculado, no caso de existir um fluxo MPEG, ou estimado no caso de uma aplicação real. Qualquer compensação[12] necessária para entregar uma trama MPEG-2 CBR tem de ser feita no codificador através de *buffers*.

Se o armazenamento for limitado, usa-se um esquema de retroacção para ajustar o débito produzido pelo codificador (factor – Q) ou deixa-se que o *buffer* não transborde. Mas ambas as soluções resultam numa degradação da qualidade, sendo a degradação diferente em ambos os casos [Tryfonas].

[12] do inglês *smoothing*

Uma alteração à proposta anterior é a ausência de compensação[12], sendo a largura de banda efectiva[13] calculada da mesma forma, mas o tráfego na rede é de débito variável (VBR). Neste caso é preciso uma grande quantidade de armazenamento nos comutadores para ser possível multiplexagem estatística. Se houver armazenamento limitado, as perdas irão irremediavelmente levar a uma degradação na qualidade.

Outras propostas apontam no sentido de um serviço CBR, com renegociação (RCBR – *Renegotiated Constant Bit Rate*). Neste caso os autores pretendem combinar a simplicidade do CBR, em termos de largura de banda usada e admissão de ligação, com as vantagens do VBR, em termos de ganho na multiplexagem. Mas o ganho na multiplexagem é conseguido não na rede pois há partilha de armazenamento, mas sim na fonte.

A ideia subjacente ao RCBR é manter o tráfego na rede próximo ao CBR e reduzir o congestionamento da rede. A fonte, com o esquema RCBR, negoceia o seu débito em escalas através de sinalização. Durante a renegociação o débito é assumido CBR e igual a um valor previamente estipulado para esse intervalo.

Um outro esquema apresentado como solução é o RED-VBR (*Renegotiation Delay Variable Bit Rate*) em que embora se tente eliminar o excesso de débito com renegociação, se usa o serviço determinístico VBR (D-VBR- *Deterministic Variable Bit Rate*), enquanto o RCBR utiliza um serviço CBR. Assim sendo, o RED-VBR consegue uma melhor utilização da rede para o mesmo nível de probabilidades de bloqueio.

Ambas as propostas fornecem um serviço estatístico, no sentido de que um pedido de renegociação pode ser recusado. Neste caso a fonte tem de agir de modo a evitar a perda de pacotes e a subsequente degradação de qualidade.

O serviço baseado na retroacção, do tipo "melhor esforço", com ou sem reserva de recursos, também foi proposto. Neste caso a fonte ajusta o seu débito com base na informação que recebe periodicamente da rede, variando o débito de codificação. Este esquema apesar de permitir uma utilização eficiente da largura de banda, pode resultar numa qualidade inaceitável para o utilizador.

No caso do serviço estatístico sem garantias, a trama é transportada na rede num modo de "melhor esforço", sem retroacção. A qualidade no receptor depende do nível de congestionamento na rede [Tryfonas].

Ao seleccionarmos um serviço específico para o transporte de vídeo é preciso efectuar um compromisso entre duas especificações conflituosas: a garantia da qualidade de serviço e a utilização da rede. O CBR determinístico certamente é a melhor opção para tramas de transporte MPEG-2 com débito constante. É preciso seleccionar um processo de escalonamento[14] para a rede ATM em que a trama de transporte MPEG-2 CBR irá ser transportada. Este escalonamento precisa garantir não só a largura de banda, mas também garantir que os atrasos máximos são majorados, para garantir uma boa qualidade.

---

[13] ou effective bandwith
[14] ou *scheduling*

4.3.2 Débito Binário Variável (VBR)

No caso das tramas de transporte MPEG-2 com débito binário variável (VBR), a renegociação parece ser a escolha mais apropriada, dado que tenta capturar a natureza VBR do tráfego enquanto mantém uma boa qualidade.

O MPEG-2 é CBR durante um certo intervalo de tempo e em certa altura são negociados os pontos do novo débito. Uma solução apresentada [Hodgins1], propõe propagar as alterações no débito de transporte até ao descodificador, filtrando a variação no atraso das células (CDV) à custa do actual débito, ou dos limites PCR (pelo menos cada 0,1s).

Neste esquema é proposto um indicador de alteração de débito[15], que é enviado na trama de transporte, no momento exacto em que o débito é alterado no codificador. Com esta proposta a camada de adaptação tem de incluir esta informação. Mas, neste caso, a rede ATM não se adapta ao novo débito da fonte e continua com o mesmo tratamento.

São precisos algoritmos para calcular os pontos de renegociação que não sobrecarreguem a sinalização e maximizem a utilização da rede, enquanto a qualidade permanece elevada.

A largura de banda efectiva, entre os dois pontos de negociação, precisa de ser calculada pela camada TS do MPEG-2 e não pela trama de vídeo, dado que as propriedades estatísticas da fonte de vídeo podem ser alteradas depois do processamento e da multiplexagem que ocorre na obtenção do fluxo de transporte (TS) MPEG-2.

No caso da codificação MPEG-2 escalável, as combinações das técnicas focadas anteriormente funcionarão eficientemente. Por exemplo, a camada base de uma trama elementar de vídeo pode ser enviada por um serviço determinístico CBR, usando a largura de banda efectiva, enquanto a camada melhorada[16] pode ser enviada por um serviço de débito variável. No entanto surgem várias questões no que diz respeito à sincronização.

A prioridade é outro aspecto que é preciso ter em conta. Os parâmetros que precisam ter prioridade nas tramas de transporte MPEG-2 são:

1. Os pacotes de transporte que contém a informação de relógio, i. e., pacotes que contém valores PCR.
2. Os pacotes de transporte da camada base, no caso de uma trama de vídeo escalável.
3. Os pacotes de transporte que possuem o princípio e o fim de uma unidade PES.
4. A combinação de todos os de cima.

[15] ou *Rate Change Indicator*
[16] do inglês *enhancement*

No entanto, em qualquer caso, a continuidade da trama deve ser preservada e a prioridade pode apenas afectar diferentes estratégias de armazenamento nos comutadores ATM durante os períodos de sobrecarga.

4.4 Sincronismo de relógio

Ao nível da rede não é desejável haver variação no atraso das células (*CDV - Cell Delay Variation*) dado que introduz problemas de sincronização entre a fonte (ou codificador) e o descodificador. Uma solução é obviamente utilizar um mecanismo que absorva o *jitter* imposto pela rede.

Foi proposta uma solução [Hodgins1] para o caso de tramas de transporte CBR, em que se utiliza uma técnica de pré-filtragem adaptativa para esvaziar o *buffer*. Mas a rede deve ser capaz de garantir que os atrasos máximos são majorados e manter o *jitter* entre certos limites. Quanto menor forem esse limites menos complexa será a PLL no descodificador, o que minimizará os custos globais, dado que também será preciso menos memória para o *buffer*.

Mas no caso mais geral, o débito variável (VBR), torna-se praticamente impossível fornecer um atraso constante ao longo da rede dado que desconhecemos com precisão o débito de transporte, verificando-se variações de ocupação do *buffer* do longo do tempo.

A solução passa por uma selecção do método de escalonamento para a rede ATM, de modo a garantir um *jitter* pequeno e confinado em limites pequenos.

Adicionalmente em ambientes de distribuição é imperioso que haja retroacção. Nestes casos a informação do relógio dos descodificadores deve ser retornada para a fonte de modo a garantir que todos os descodificadores se encontram sincronizados uns dos outros.

4.5. Serviços Normalizados

Os Sistemas Multimédia Interactivos (AMS) representam uma tecnologia chave que tem sido desenvolvida com amplas perspectivas de sucesso comercial.

4.5.1 Serviço Video-on-Demand (VOD)

O Serviço Video-on-Demand (VOD) é essencialmente um serviço unidirecional assimétrico que transfere informação codificada e comprimida de vídeo de uma fonte, um servidor de vídeo, para um destino, um cliente (um PC ou um STT). No descodificador as tramas são reconstituídas, descodificadas, convertidas de digital para analógico e apresentadas no monitor.

Este serviço é tendencialmente vocacionado para fins lúdicos, permitindo o acesso de subscritores a uma biblioteca de filmes digitais, através de uma ligação ponto-a-ponto. Esta ligação permite algum controlo ao utilizador, como por exemplo, pausa, avanço, recuo, etc. Uma ligação ponto-a-multiponto é também possível num cenário de difusão.

O serviço VOD é classificado em diversas categorias de acordo com o tipo de interactividade disponível. Temos assim quatro categorias em que vai aumentando a interactividade. O serviço de Difusão (No-VOD), o serviço Pagar para ver (PPV), o serviço quase Vídeo-a-pedido (Q-VoD) e finalmente o serviço Vídeo-a-pedido pleno (T-VoD).

O VOD fornece comunicação terminal-a-terminal com qualidade de vídeo de acordo com o standard de compressão de vídeo (exemplo, MPEG1, MPEG2, etc.) e qualidade de áudio de acordo com os respectivos standards do MPEG áudio. A transferência de informação requer sincronização nas tramas de vídeo e áudio no STT. A descodificação e a recuperação dos *Timestamp* é considerada crítica.

4.5.2 Serviço Multimedia Desktop

O serviço *Multimedia Desktop* é um teleserviço bidireccional simétrico em tempo real, que fornece uma comunicação pessoa-a-pessoa, com transferência de Voz (som), Vídeo e opcionalmente dados entre as duas pessoas.

Este serviço está vocacionalmente preparado para melhorar o serviço telefónico, permitindo uma comunicação multimédia. Adicionalmente é possível acrescentar serviços suplementares, como por exemplo conferência multimédia permitindo o serviço a multi-utilizadores.

Este serviço fornece comunicações de qualidade vídeo de acordo com os standards de compressão de vídeo (por exemplo JPEG, MJPEG, MPEG1, MPEG2, H.262, etc.) e qualidade de som correspondente aos respectivos standards para 3,1; 7,5 e 15 kHz MPEG2/H.262. Adicionalmente é fornecida a possibilidade de transferir dados, por exemplo ficheiros, imagens, gráficos, texto e mensagens de controlo.

4.5.3 Serviço Video Conferencing

O serviço *Video Conferencing* fornece a possibilidade dos participantes numa conferência em grupo transferirem entre si diferentes tipos de informação, incluindo voz, vídeo com qualidade televisiva, som (previamente gravado) e adicionalmente imagens digitalizadas, documentos, ou outro tipo de suporte à conferência entre duas ou mais localizações.

Para a interconectividade entre os participantes em três ou mais localizações, é necessária uma funcionalidade específica de interconectividade para manter a correcta distribuição dos vários sinais entre as ligações e assegurar os procedimentos correctos dentre os subscritores deste serviço.

4.5.4 Serviço Interactive Distance Learning (IDL)

Interactive Distance Learning é um serviço audiovisual baseado na presença virtual do formador em vários sítios. Pressupõe suficiente qualidade de vídeo, fidelidade de som, ausência de latência distractiva para assegurar um correcto funcionamento, semelhante à natureza do processo educativo.

O ensino à distância fornece a possibilidade de educação especializada, como por exemplo Arte, Medicina e temas técnicos, poderem ser ensinados através de um modo explícito, permitindo uma participação efectiva dos alunos e dos professores. Só através de vídeo e som, com qualidade televisiva, em tempo real, é que se torna viável pensar em ensino à distância.

Uma característica do IDL é o uso de múltiplos lugares, com uma variedade de aulas. É exigido um controlo especializado para que os participantes façam directamente as suas perguntas para o formador, que o formador interaja com várias salas de aulas simultaneamente e que haja a possibilidade de por controlo remoto seleccionar a câmara, rodar, focar, e fazer zoom. Todas estas tarefas pressupõem acções em tempo real, de contrário todo o serviço ficaria comprometido.

4.5.5 Serviço Post-Production Editing

Este serviço por sua vez requer um ambiente em tempo real, um meio unidirecional e serviços audiovisuais que forneçam elevada qualidade vídeo, com as faixas de som associadas, e opcionalmente a entrega de mensagens de controlo.

A edição Pós-Gravação exige que a distribuição intermediária ou final do vídeo e das faixas de áudio sejam de elevada qualidade tanto na fonte como no destino.
Uma trama de vídeo pode ser uma das seguintes:

- Uma componente vídeo sem compressão de 270 Mb/s
- Uma sequência de vídeo MPEG-1 entre 24 a 40 Mb/s
- Uma sequência de vídeo MPEG-2 entre 10 a 16 Mb/s

Existem tipicamente quatro canais de áudio, amostrados a 48 kHz, até 20 bits por amostra, havendo adicionalmente a possibilidade de um canal de controlo em tempo real.

Capítulo 5

Trabalho experimental

No presente capítulo é apresentado o trabalho desenvolvido em torno da área do suporte de serviços MPEG-2 em redes ATM. A abordagem consistiu no desenvolvimento de software em duas vertentes.

A abordagem inicial centrou-se no estudo e desenvolvimento de um Selector de programas, cujo principal objectivo é a possibilidade de permitir ao utilizador a escolha de um programa existente num fluxo de programas multiplexados.

O trabalho foi faseado em três partes consecutivas. Numa primeira fase o objectivo foi a detecção dos programas existentes, a partir das tabelas PSI existentes no fluxo de dados MPEG-2. A segunda fase consistiu no interface com o utilizador, nomeadamente na visualização dos programas existentes e na escolha do programa pretendido. A terceira fase teve como propósito tornar o processo mais autónomo e inteligente na medida em que realiza as seguintes tarefas:

- Se existir um único programa, este é indicado ao utilizador, mas começa imediatamente a armazená-lo;

- Na segunda fase o utilizador só podia escolher um novo programa quando surgissem alterações, mas nesta fase pode alterar a qualquer altura;

- Caso haja alterações nos programas existentes e o utilizador não tomar nenhuma opção, então o programa continua a guardar o programa previamente escolhido, ou no caso de esse já não estar presente, então é guardado o primeiro programa.

A outra vertente do trabalho prendeu-se no estudo da escolha da camada de adaptação a ser utlizada como suporte ao MPEG-2 em redes ATM. Em particular foram analisados os efeitos da variação da Referência de Relógio de Programa (PCR) no caso da utilização da camada de adaptação AAL5.

5.1. A nível da camada de Sistema
Selecção de Programas MPEG-2 TS

No capítulo 2 fez-se uma breve introdução à camada de Sistema definida na norma MPEG-2. Realçam-se de seguida os aspectos mais importantes da camada de sistema que directamente foram utilizados na realização do Selector de Programas MPEG-2.

5.1.1 O nível de transporte

A especificação da camada de Sistema da norma MPEG-2 descreve como as tramas de vídeo comprimido MPEG e os dados de som devem ser multiplexados para formar uma única trama de dados.

A multiplexagem da trama de transporte é constituída por um número fixo e pequeno de pacotes de transporte. Esses pacotes de transporte têm uma dimensão fixa de 188 bytes divididos em cabeçalho e espaço de transporte de dados (*payload*) representados na figura 5.1.

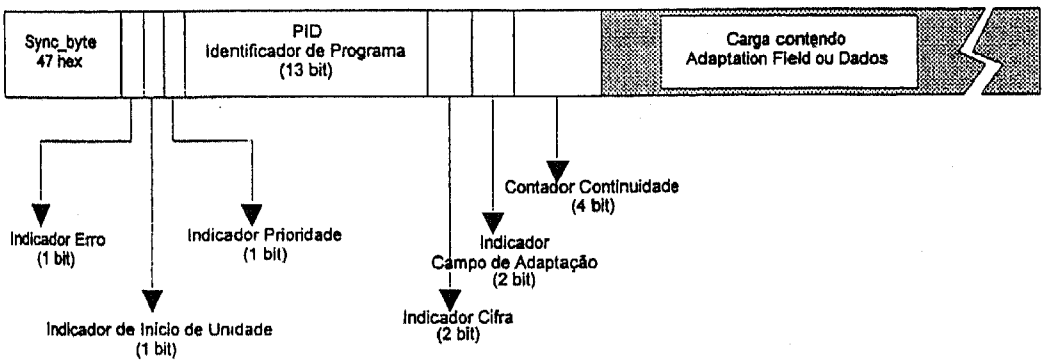


Figura 5.1 – Estrutura do pacote de transporte.

O cabeçalho dos pacotes de transporte divide-se em duas partes com funcionalidades distintas:

- Um cabeçalho de ligação formado por quatro bytes.
- Um campo de adaptação que contém informação utilizada na descodificação de alto nível (Figura 5.2).

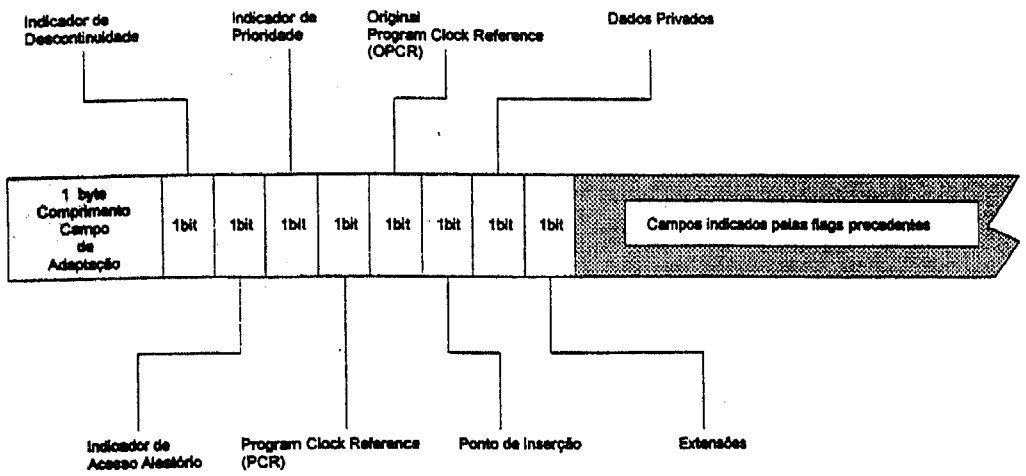


Figura 5.2 – Estrutura do campo de adaptação transmitido nos pacotes de transporte.

O cabeçalho de ligação é composto por:

1 byte de sincronismo com código 47 (Hex);
1 bit de erro indicando que a informação do pacote contém erros não corrigíveis e deve ser rejeitado pelo sistema;
1 bit indicando o início de uma pacote do nível hierárquico inferior (pacotes PES);
1 bit indicador de prioridade na transmissão;
13 bits identificadores do programa (PID, *Packet Identification*);
2 bits indicando a utilização de sistemas de cifra;
2 bits indicando a presença do campo de adaptação no espaço reservado de dados;
4 bits de um contador de continuidade para controlo da sequência de chegada de informação.

O campo de adaptação, quando presente, é composto por seu turno por:

1 byte que determina o comprimento do campo de adaptação;
1 bit que indica a presença de descontinuidade;
1 bit indicando a presença de acesso aleatório, ou seja, a carga do pacote pode ser descodificada sem referência a informação precedente.
1 bit de prioridade;
1 bit indicando a presença do PCR;
1 bit indicando a presença do OPCR;
1 bit indicador do ponto de inserção;
1 bit de dados privados;
1 bit de extensões.

De salientar o parâmetro PID que serve para identificar, de forma indirecta através de informação em tabelas, o tipo de informação contida na carga do pacote.

Dos 2^{13} valores possíveis 17 são reservados para tarefas específicas, donde resulta 8175 valores possíveis de serem utilizados para distinguir as diferentes tramas elementares.

Essa identificação é conseguida através da existência de informação específica dos programas (PSI – *Program Specific Information*) em quatro tabelas especificadas na norma MPEG-2 Sistemas.

Uma tabela associada ao programa (PAT – *Program Association Table*) é transmitida nos pacotes de transporte com o PID reservado (PID=0) e através dela são especificados os PID associados aos pacotes que transportam a tabela de mapa de programas e informação de rede. Cada programa é listado com um PID associado, que representa os pacotes de transporte que possuem uma tabela de Mapeamento desse programa (PMT).

Uma tabela de mapa de programa (PMT – *Program Map Table*) permite conhecer os PID dos pacotes de transporte de informação associados a cada programa.

Uma tabela de acesso condicionado (CAT - *Conditional Acess Table*) é transportada em pacotes com um PID reservado (PID=1) e a sua utilização não é especificada pela norma.

Uma tabela de informação de rede (NIT - *Network Information table*) contém informação reservada não normalizada.

O funcionamento do Selector de programa assenta toda a sua estrutura nestas tabelas, dado que é através delas que se faz a apresentação dos programas e a selecção do programa especificado pelo utilizador.

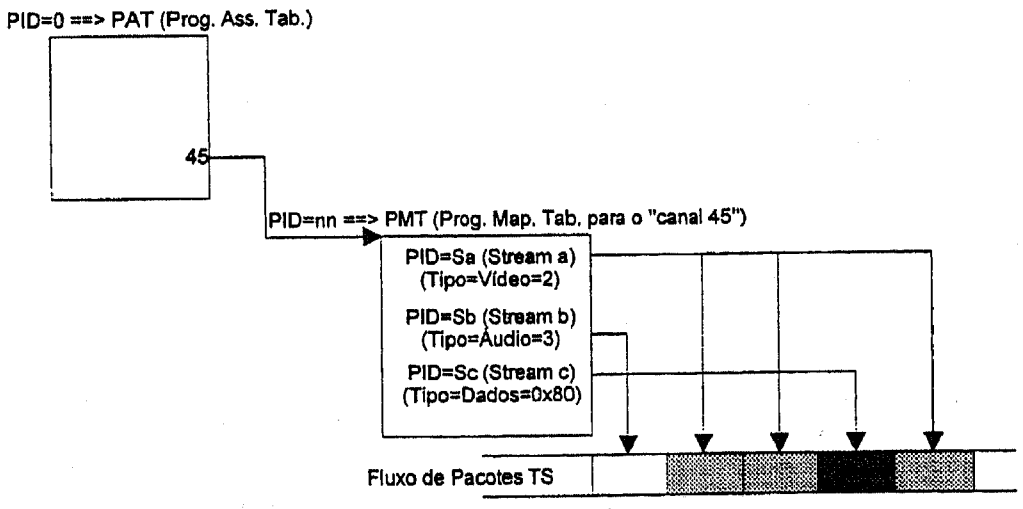


Figura 5.3 – Mecanismo para selecção de canal utilizando a informação PSI definida na sintaxe de transporte MPEG-2.

Assim, numa primeira análise, é lida a tabela da associação PAT (com o PID=0) e são apresentados ao utilizador os programas existentes. De seguida consoante a escolha do utilizador o editor determina o PID dos pacotes de transporte associados à tabela de mapa de programa, PMT, lê a informação da PMT e reconhece os pacotes que transportam os fluxos elementares que compõem o programa. Esta última informação é dada no parâmetro tipo de fluxo (stream_type) codificado na PMT (ver Figura 5.3).

5.1.2 Programa desenvolvido

O programa foi implementado em C, de acordo com a norma MPEG-2 de Sistemas, [ISO/IEC1] sendo capaz de desmultiplexar um MPTS num único SPTS escolhido pelo utilizador.

No processo de desmultiplexagem foi utilizado, como teste, um MPTS gerado previamente e que contém inicialmente apenas um programa, mas a partir de uma certa altura possui dois programas. Neste caso é dada inicialmente a informação ao utilizador da existência de apenas um programa e o programa Selector começa a guardar num ficheiro os pacotes respeitantes a esse programa.

Um objectivo futuro será enviar para um buffer de entrada e entregar os pacotes para apresentação.

Após alterações na tabela PAT, indicando alterações nos programas presentes no MPTS, o utilizador é avisado novamente dos programas existentes, sendo-lhe pedido uma acção de escolha. Caso o utilizador opte, o programa Selector continua o seu processo de armazenamento dos TS respeitantes do programa escolhido, caso o utilizador não execute nenhuma operação, então se o programa anterior estiver presente continua a armazená-lo, caso contrário guarda o primeiro programa encontrado.

De salientar o facto de em qualquer altura o utilizador poder escolher o programa a armazenar, alterando a versão da PAT.

O processo de desmultiplexagem utilizado consistiu na leitura das PSIs. Primeiro é feita a leitura da PAT e detectados os programas existentes. De seguida selecciona-se o programa escolhido, é refeita e copiada a PAT para o ficheiro de saída e finalmente a PMT e as respectivas tramas indicadas na PMT referentes ao programa escolhido são copiadas para o ficheiro de saída.

Na figura 5.4 é descrita, de um modo global, a arquitectura do programa desenvolvido, realçando-se a interligação existente entre os diferentes blocos.

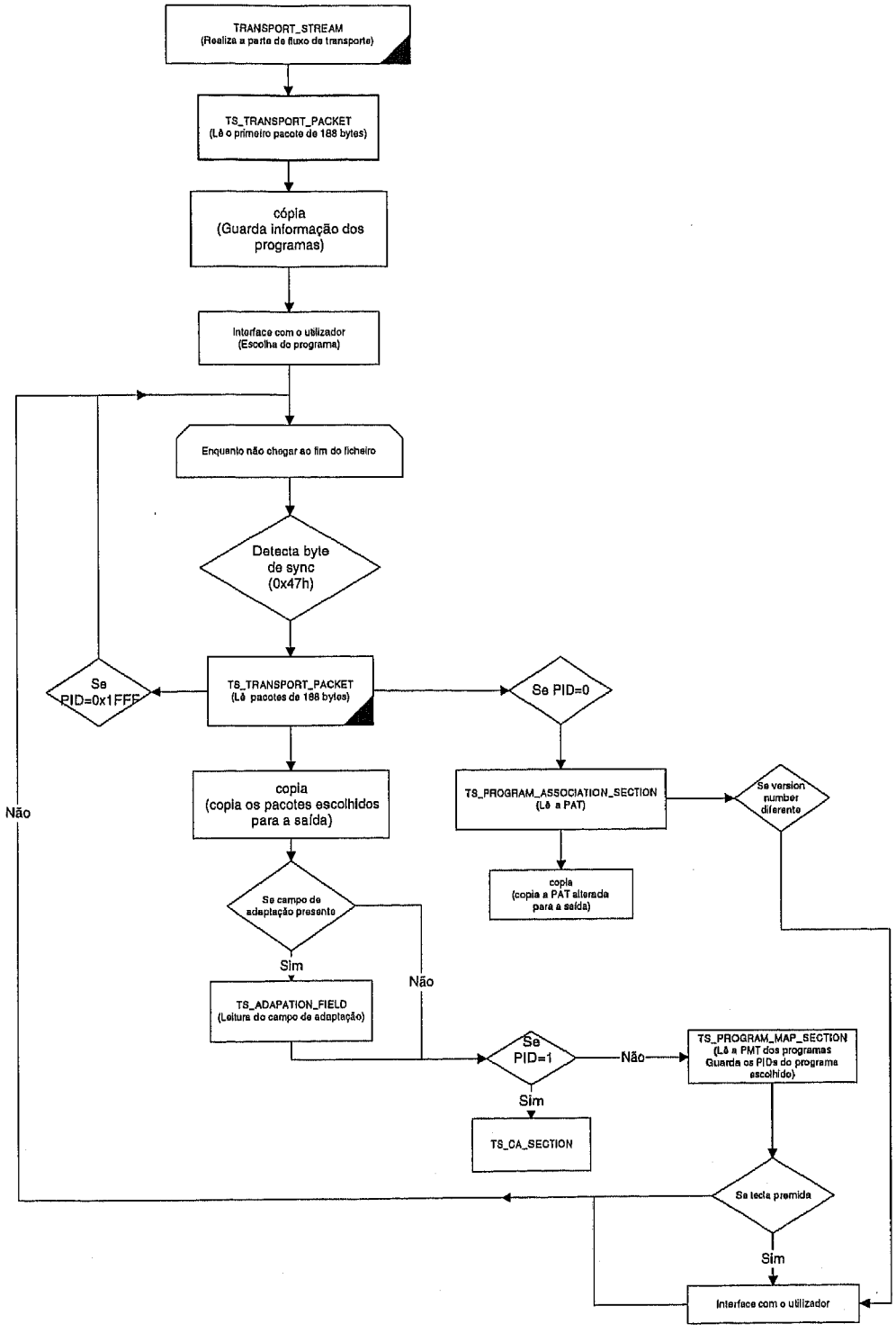


Figura 5.4 – Arquitectura funcional do Selector de programas desenvolvido.

O programa começa por chamar `TRANSPORT_STREAM` que vai ler o primeiro pacote de transporte e de seguida após detectar o byte de sincronismo vai lendo sucessivamente todos os pacotes de transporte até ao fim do ficheiro:

```
TS_TRANSPORT_PACKET(in, out);
n_TPs = 1;
while( !feof(in) ) if(getc(in) == 0x47)
                    {
                    TS_TRANSPORT_PACKET(in, out);
                    tcsetattr(STDIN_FILENO, TCSADRAIN, &new_term);
                    read(STDIN_FILENO, &option, 1);
                    }
```

No procedimento `TS_TRANSPORT_PACKET`, foi criada a seguinte estrutura:

```
{
  struct header_bits
  {
    unsigned transport_error_indicator    :  1;
    unsigned payload_unit_start_indicator :  1;
    unsigned transport_priority           :  1;
    unsigned PID_12_8                     :  5;
    unsigned PID_7_0                      :  8;
    unsigned transport_scrambling_control :  2;
    unsigned adaptation_field_control     :  2;
    unsigned continuity_counter           :  4;
  };
```

que representa os quatro bytes do cabeçalho do pacote TS.

O pacote TS tem dois bits do campo *adaptation field control* que identificam na carga do pacote a existência do campo de adaptação, de dados, ou de ambos da seguinte forma:

adaptation field control =1 apenas dados;
=2 apenas o campo de adaptação;
=3 dados e campo de adaptação;

ou na Linguagem C,

```
...
i = header.bits.adaptation_field_control;
  if( i == 2  ||  i == 3 )  TS_ADAPTATION_FIELD(i, &bytes_read, in,
out);
  /* payload */
  if( header.bits.adaptation_field_control == 1 ||
      header.bits.adaptation_field_control == 3     )
  {
     ...
    {
      i = header.bits.payload_unit_start_indicator;
      if(PID == 0)
              { ....
                    TS_PROGRAM_ASSOCIATION_SECTION(i, &bytes_read,
in, out);
          }
       ...
```

Neste pequeno extracto do programa encontramos igualmente a referência utilizada no caso de se encontrar uma tabela PAT, identificada com o PID=0. Neste caso é chamado o procedimento `TS_PROGRAM_ASSOCIATION_SECTION`.

Neste procedimento foram construídas as seguintes estruturas:

```
struct header_bits
{
  unsigned section_syntax_indicator :  1;
  unsigned no_name                  :  1;
  unsigned reserved1                :  2;
  unsigned section_length_11_8      :  4;
  unsigned section_length_7_0       :  8;
  unsigned transport_stream_id_15_8 :  8;
  unsigned transport_stream_id_7_0  :  8;
  unsigned reserved2                :  2;
  unsigned version_number           :  5;
  unsigned current_next_indicator   :  1;
};
```

sendo de realçar nesta estrutura o `version_number` responsável pela versão da PAT e outra estrutura, que é composta por quatro bytes, que contém a informação relativa a cada programa.

```
struct table_bits
  {
    unsigned program_number_15_8              :  8;

    unsigned program_number_7_0               :  8;
    unsigned reserved                         :  3;
    unsigned network_or_program_map_PID_12_8  :  5;
    unsigned network_or_program_map_PID_7_0   :  8;
  };
```

Neste caso os mais importantes são claramente os dois bytes que possuem o número do programa.

For fim, encontramos o `TS_PROGRAM_MAP_SECTION`, que é responsável pela informação referente à tabela PMT.

Novamente foi construída uma estrutura de acordo com a norma:

```
struct header_bits
  {
    unsigned section_syntax_indicator :  1;
    unsigned no_name                  :  1;
    unsigned reserved1                :  2;
    unsigned section_length_11_8      :  4;
    unsigned section_length_7_0       :  8;
    unsigned program_number_15_8      :  8;
    unsigned program_number_7_0       :  8;
    unsigned reserved2                :  2;
    unsigned version_number           :  5;
    unsigned current_next_indicator   :  1;
    unsigned section_number           :  8;
    unsigned last_section_number      :  8;
    unsigned reserved3                :  3;
```

```
    unsigned PCR_PID_12_8              :  5;
    unsigned PCR_PID_7_0               :  8;
    unsigned reserved4                 :  4;
    unsigned program_info_length_11_8  :  4;
    unsigned program_info_length_7_0   :  8;
  };
```

Note-se que é neste bloco que é feita a leitura e armazenamento dos PIDs referentes às tramas onde se encontram os dados.

```
...
/* elementary_PID */
        i = ES.bits.elementary_PID_12_8 * 256
           +ES.bits.elementary_PID_7_0;
        fprintf(out,"| elementary_PID = 0x%04X\n", i);
          if ( programaPMT)
        {
          PIDPMT[PMTPIDcont]=i;
          PMTPIDcont++;
          progPIDPMT=1;
        }
...
```

O programa vai lendo os TSs e simultaneamente vai fazendo as alterações necessárias e armazenando a informação num novo ficheiro.

Primeiro são armazenados os quatro bytes do cabeçalho:

```
....

   Bytes_lidos=0;
   comp=bytes_read;
   pos=ftell(in)- comp;
   if (PATII){ trama=fopen(sda,"wb");
               PATII=0;
             }
   else trama=fopen(sda,"ab");
        fseek(in,pos,SEEK_SET);
        aux1=comp;
        while(aux1--)putc(getc(in),trama);
        bytes_lidos=comp;
        maisf++;
...
```

de seguida é necessário alterar a PAT, de modo a referir apenas a existência de apenas um programa. Para tal é necessário alterar os seguintes campos:

- O section_length que indica o comprimento da PAT,
- O version number que indica a versão da PAT,
- O current_next_indicator que indica se a PAT já é considerada,

```
/* copia para a nova PAT 8 bytes */
        putc(PA_section.buffer[0],trama);
        putc(PA_section.buffer[1],trama);
        putc(PA_section.buffer[2],trama);
      /* tamanho do pacote- section length */
        putc( 0x0D,trama);
        putc(PA_section.buffer[4],trama);
        putc(PA_section.buffer[5],trama);
      /* Muda o version number e o current_next_indicator */
        header.bits.version_number=vactual;
        header.bits.current_next_indicator=1;
    if ((!novo && agora ) || (option!=0))
        {
       printf("\n------------------------//-----------------------
");
         printf("\n|  Nova PAT          ");
         if (option!=0) printf("\n| Escolheu o programa %c
",option);
         if (option!=0) prog=option-48;
         if (vactual>31) vactual=0;
         printf("       VERSAO    %d    "
,header.bits.version_number);
         if (option!=0)header.bits.current_next_indicator=1;
         option=0;

         header.bits.version_number=vactual;
         vactual=vactual+1;
         printf(" \n|    Nova Versao    %d  \n",vactual);
         }
        PA_section.buffer[6]=header.byte[4];
        putc(PA_section.buffer[6],trama);
        putc(PA_section.buffer[7],trama);
        putc(PA_section.buffer[8],trama);
         bytes_lidos=bytes_lidos+9;
  }
```

por fim acrescenta-se o CRC e preenche-se o resto da PAT com *padding*

```
if (NPAT)
{
 /* copia relativo ao programa */
 for (i=0;i<4;i++)
 putc(PA_section.buffer[4*prog+5+i],trama);
 /* mantem o CRC */
 for(i=PA_section.p-3;i<=PA_section.p;i++)
 putc(0,trama);
 bytes_lidos=bytes_lidos+8;
 for(i=bytes_lidos;i<=188;i++)
 putc(0xFF,trama);
}
```

Todas as tramas, identificadas pelo PID, referentes ao programa escolhido são guardadas da seguinte forma:

```
/* copia o PID da PMT */
    if ( NovaPMT)
    if (PID==PIDPROG[prog])
       {
/* copia os 4 bytes de cabecalho */
         pos=ftell(in);
```

```
            novapos=pos-4;
            copia(in,sda,novapos);
            fseek(in,pos,SEEK_SET);
        }

/* copia os PID que vem na PMT..*/
            if (progPIDPMT)
                for (c=1;c<PMTPIDcont;c++)
                 {
                   if (PID==PIDPMT[c])
                   {
                     pos=ftell(in);
                    pos=pos-4;
                    copia(in,sda,pos);
                    pos=pos+4;
                    fseek(in,pos,SEEK_SET);
                   }
                 }
   /* fim da copia */
```

O programa permite ao utilizador alterar o programa que pretende, em qualquer altura, dado que no fim da leitura e tratamento de um pacote de 188 bytes é verificado se alguma tecla foi premida. Se o utilizador premir uma tecla válida, essa tecla é guardada para depois ser processada.

```
...
tcgetattr(STDIN_FILENO, &old_term);
      new_term=old_term;
      new_term.c_lflag &= ~(ECHO | ECHOE | ECHOK | ECHOCTL | ECHONL |ICANON);
      new_term.c_cc[VMIN] =0;
      new_term.c_cc[VTIME]=0;

.....
tcsetattr(STDIN_FILENO, TCSADRAIN, &new_term);
                        read(STDIN_FILENO, &option, 1);
                        }
tcsetattr(STDIN_FILENO, TCSADRAIN, &old_term);
...
```

No anexo A apresenta-se a listagem completa do programa implementado, tendo sido aqui referidos apenas alguns aspectos considerados relevantes.

5.2. A nível da camada ATM

Estudo da variação da Referência de Relógio de Programa (PCR), ou Jitter, no empacotamento AAL5.

Actualmente os avanços nas tecnologias de transmissão apresentam baixas taxas de erros; no entanto os serviços de débito variável (VBR) são muito sensíveis a erros de transmissão.

Num sistema de transmissão baseado na comutação ATM, a taxa de células perdidas para débito variável pode não ser desprezável. O problema na realidade torna-se evidente quando existem tramas que são perdidas durante a transmissão e

infelizmente os serviços multimédia em tempo real, transmitidos na Rede de Banda Larga, tendem a ser muito sensíveis a essas perdas.

Por outro lado estas aplicações também são sensíveis a atrasos e a variações nos atrasos de tal forma que se certos dados chegarem depois de um limiar (MCTD), são considerados como perdidos para a aplicação.

Existem alguns mecanismos baseados em correcção de erros (FEC), e algumas das suas variantes revelam-se com interesse [Stock], [Verscheure], [Garcia].

5.2.1 O desempenho na transmissão de MPEG-2 em redes ATM

O ATM foi definido como a tecnologia de redes capaz de suportar transmissão de dados, voz e vídeo. Em particular o ATM suporta inúmeras aplicações, como por exemplo as aplicações de vídeo baseadas nos standards MPEG-1 e MPEG-2.

Enquanto o standard predominante para a Vídeo-Conferência, é actualmente o H.320, a principal razão na adopção do ATM para o MPEG foi a capacidade de maior largura de banda e de fornecer controlo na latência e no *jitter* da rede. É importante detectar precisamente qual a latência e o *jitter* tolerável pelo vídeo.

Vários estudos têm sido feitos para determinar exactamente esses níveis de latência e de *jitter*. Para suportar a transmissão de vídeo os comutadores ATM devem assegurar os níveis de latência ou de *jitter* exigidos. Para além disso é normal que em redes locais os atrasos de transmissão sejam menores que os exigidos o que permite um maior controlo no equipamento de codificação e descodificação bem como da própria rede [Cisco1].

Em particular as comunicações em dois sentidos requerem um maior controlo na latência que uma transmissão num só sentido.

5.2.2 O modelo do Relógio de Sistema (STC – System Time Clock)

O MPEG-2 possui um modelo de temporização, no qual o atraso total do sinal desde o codificador até ao descodificador é constante. Este atraso é a soma da codificação, do armazenamento no codificador, da multiplexagem, da comunicação, da desmultiplexagem, do armazenamento no descodificador, da descodificação e dos atrasos de apresentação.

A trama de sistema possui, no entanto, informação temporal que pode ser usada para implementar sistemas que garantem esse atraso constante. Toda a temporização é definida em termos de um modelo de relógio de sistema, referido na nomenclatura por STC- *System Time Clock.*

O conceito de relógio de sistema é baseado num oscilador de referência utilizado para a aquisição dos sinais de vídeo no codificador (ITU-T- 601, 656 e 657) e que funciona como uma base de tempo do sistema codificador-descodificador. A especificação do oscilador no codificador é:

Frequência do relógio do Sistema = 27 MHz +/– 810 Hz
Vagueio da Frequência do relógio do Sistema <= 75 x 10^{-3} Hz/sec

Por exemplo, se o relógio (oscilador) local do descodificador remoto, devido a vagueio intrínseco[17], se tornar mais lento que o do codificador, gera-se uma situação em que o descodificador tem que acumular informação nas suas memória tampão com o consequente risco de transbordamento.

No caso de descodificadores de vídeo, a sincronização codificador-descodificador traduz-se na associação da frequência de varrimento do monitor (na recepção) à frequência de varrimento da câmara (na emissão).

A norma MPEG-2 Sistema fornece um mecanismo para sincronização (Figura 5.5).

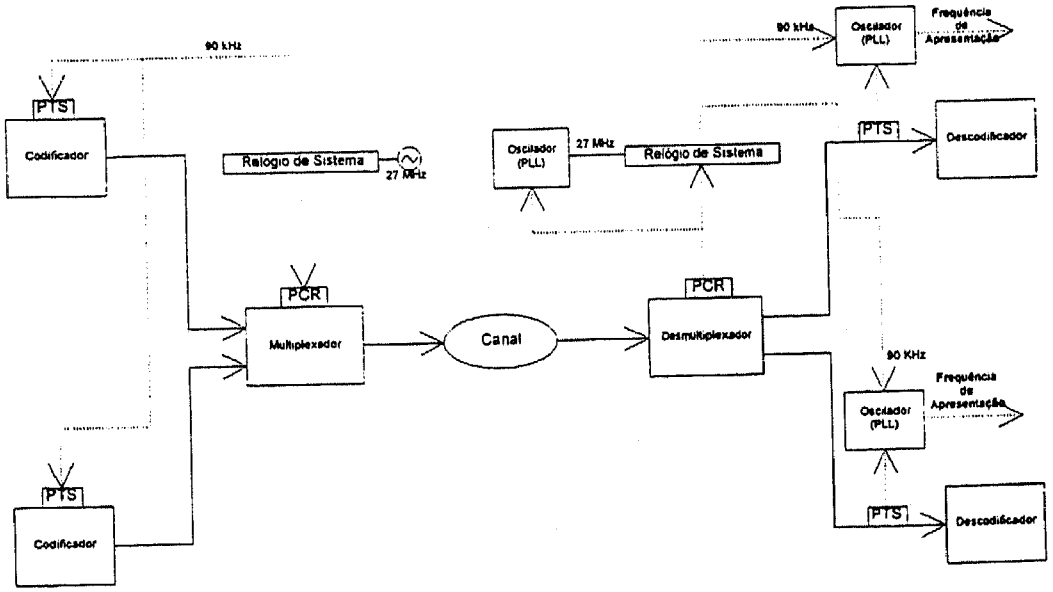


Figura 5.5 – Sincronização codificador-descodificador conforme as especificações da norma MPEG-2 sistema.

Antes de continuar é imperioso referir que os conceitos SCR (*System Clock Reference*), PCR (*Program Clock Reference*) embora tenham o mesmo significado, possuem algumas diferenças. O termo SCR é utilizado pela norma MPEG-1 Sistema e manteve-se no MPEG-2 Sistema, no contexto de mistura no modo programa (*Program Stream*). O termo PCR é utilizado no contexto da sincronização de fluxos misturados no modo de transporte (*Transport Stream*) da norma MPEG-2 Sistema.

[17] Os osciladores de referência devem cumprir especificações precisas de tolerância de vagueio, definida na norma MPEG-2 Sistema.

O conceito de programa está subjacente ao conceito de canal nos sistemas de TV analógica. É constituído por um conjunto de sinais partilhando a mesma base de tempo. Na norma, o relógio de sistema é designado em geral por STC (*System Time Clock*).

Cada vez que um codificador elementar inicia a aquisição de uma Unidade de Apresentação (*Presentation Unit*), o relógio de sistema (STC) é amostrado e o seu valor armazenado no parâmetro PTS (*Presentation Time Stamp*).

Por outro lado em algumas tramas existe o DTS (*Decoding Time Stamp*), que representa o tempo que a unidade de acesso tem para ser enviada para o descodificador, de modo a esta ser descodificada mas não apresentada.

Normalmente o DTS é encontrado nas imagens I ou P das tramas elementares de vídeo.

A cada DTS tem que estar associado um PTS, de valor superior, que representa o tempo de apresentação.

Cada codificador elementar passa os respectivos PTS e DTS ao primeiro nível de mistura de nível PES. Os parâmetros PTS e DTS são transportados no cabeçalho dos pacotes PES, em unidades de 90kHz, associados aos respectivos fluxos codificados elementares.

Na recepção, o descodificador de sistema distribui os fluxos elementares PES pelos descodificadores elementares que extraem a informação PTS utilizada para determinar os instantes de apresentação. Para que os valores PTS tenham significado, é necessário reproduzir no descodificador a base de tempo (STC) utilizada no codificador o que é conseguido através da transmissão da informação do próprio relógio de sistema.

O multiplexer de sistema transmite as amostras de um contador de 42 bits (STC[18]), que são enviadas ao nível dos pacotes de transporte. O *desmultiplexer*, na recepção recebe essas amostras sucessivas que alimentam um oscilador controlado do tipo PLL (*Phase Locked Loop*), que reproduz a frequência utilizada no emissor (STC).

As amostras STC são enviadas em intervalos regulares que não podem exceder os 100 ms permitindo, com a tecnologia actual de osciladores controlados, obter resultados de precisão dentro dos requisitos de tolerância definidos na norma MPEG-2 Sistema para a reconstrução da frequência de referência. Ao nível do desmultiplexer do sistema, os valores das amostras STC são comparados com o valor do contador local e a diferença é utilizada para corrigir o oscilador controlado.

Por seu lado o PTS, ou DTS, deve ocorrer a intervalos de pelo menos 0,7s, no caso de pacotes PES de vídeo ou áudio.

[18] Nota o oscilador de 27MHz faz avançar o contador PCR e o de 90kHz o contador PTS. A conversão entre as diferentes frequências de oscilação é feita através de um divisor por 300.

Os dois parâmetros (STC e PTS) permitem resolver o problema da sincronização mútua. Para a garantir, os descodificadores elementares devem reproduzir a frequência de apresentação e os instantes absolutos (fase) na base de tempo reconstruída através de PCR. Essa informação está contida nos parâmetros PTS uma vez que são obtidas por amostragem, em emissão, do relógio de sistema.

No caso de redes assíncronas, onde o atraso é intrinsecamente variável, o mecanismo de sincronização é intrinsecamente variável, o mecanismo de sincronização é afectado pelas flutuações nos tempos de chegada dos parâmetros PCR e PTS.

Nestes casos o vagueio do oscilador local é eliminado pela malha de seguimento de fase (filtro +PLL).

No caso de uma rede do tipo ATM introduzir um atraso variável[19] ao vagueio entre as frequências de oscilação vai-se juntar o *jitter* dos instantes das chegadas dos parâmetros PCR.

Uma solução é utilizar a informação do fluxo de sistema, *Mux-rate*, ou seja o débito binário médio da mistura do nível de transporte e considerar-se uma memória tampão antes do desmultiplexer que acumula a informação que provém do canal a débito variável e a fornece ao desmultiplexer à velocidade indicada pelo *Mux_rate*. Desta forma a flutuação introduzida tende a anular-se pois o atraso variável entre dois PCR consecutivos tende para o valor médio do atraso de transmissão, conforme podemos verificar pelos testes efectuados, para diferentes valores de N no caso do *PCR-unaware* (Figura 5.9).

$$\text{Mux\_rate} = \frac{(\text{número de bytes entre PCRs}) \times \text{Frequência do relógio do Sistema}}{\text{PCR}_{\text{novo}} - \text{PCR}_{\text{antigo}}}$$

É de salientar que este Mux_rate é fixo ou variável, no entanto entre dois PCRs é obrigatoriamente constante (ver Figura 5.6).

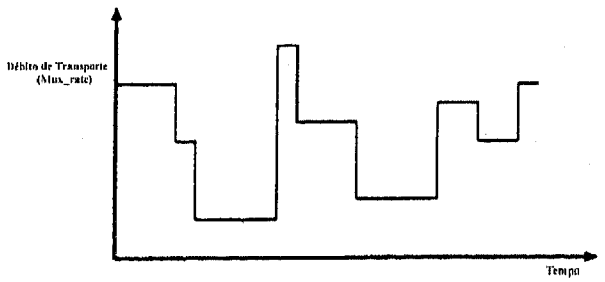


Figura 5.6 – Linearidade no Débito de Transporte.

No caso da trama de Transporte conter múltiplos programas, com bases de tempo independentes, haverá necessariamente um grupo de PCRs associado a cada programa.

[19] Partindo do pressuposto que a velocidade média de transmissão na rede não é inferior ao débito binário nominal requerido pelo descodificador.

Dado que o débito de transporte tem a ver invariavelmente com os PCRs de cada programa, se o débito de transporte for variável, então só poderá variar entre os PCRs do programa em consideração. Quando os PCRs dos programas e os pontos no fluxo onde o débito varia não são iguais, o débito de transporte irá variar consoante o programa que está a ser descodificado. Assim sendo não é possível construir um fluxo com múltiplos programas (MPTS), a não ser se recorrermos ao débito constante, introduzindo pacotes de *stuffing*.

A recuperação do relógio de sistema no descodificador é conseguida eliminando a flutuação residual (componente de vagueio do oscilador local e das flutuações de velocidade na rede), que é feita através de filtragem associada à malha de seguimento de fase [Pires] [Tryfonas].

Os valores de PCR encontrados, de um modo regular, na Trama de Transporte, são utilizados na reconstrução do relógio, no descodificador. As referências de relógio que o descodificador recebe no TS, são os PCRs de um programa. Como na transmissão MPEG-2 em redes ATM os atrasos não são constantes e os relógios do descodificador e do codificador não estão sincronizados, então é preciso garantir a sincronização. O relógio do descodificador precisa de ficar sincronizado com os PCRs encontrados no TS. Este processo é conseguido com base numa PLL, como podemos ver na Figura 5.7.

Esta PLL inicialmente espera pela recepção de um nova base de tempo. Este valor (o primeiro PCR recebido pelo programa que está a ser descodificado) é carregado no contador STC e a PLL começa a funcionar em ciclo fechado. Quando um novo PCR chega à PLL, então este valor é comparado com o do STC local. A diferença entre os dois origina um erro ***e***. Este erro é enviado para um filtro passa-baixo. O resultado deste filtro vai actuar no VCO (*Voltage Controlled Oscillator*) com a frequência central igual aos 27MHz. É de salientar o facto do contador STC possuir dois contadores. Um para os PTS, à frequência central de 90kHz e outro para os PCR à frequência central de 27MHz. Quando o erro ***e*** converge para um valor constante, então a PLL fica sincronizada e nós temos a garantia de ambos os relógios, do codificador e do descoficador, estarem sincronizados.

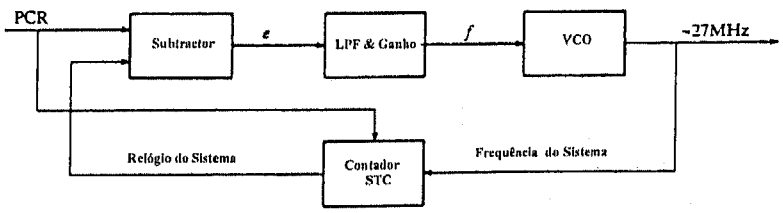


Figura 5.7 – Recuperação do relógio usando uma PLL (*Phase-Locked Loop).*

É de salientar o facto mais importante que é o desenho do filtro passa-baixo, responsável em grande medida pela rapidez com que a PLL fica sincronizada.

No caso dos PCRs possuirem *jitter* , então é preciso remover este *jitter* através de um buffer adicional[20][Hodgins2].

[20] Ou *Dejittering Buffer*

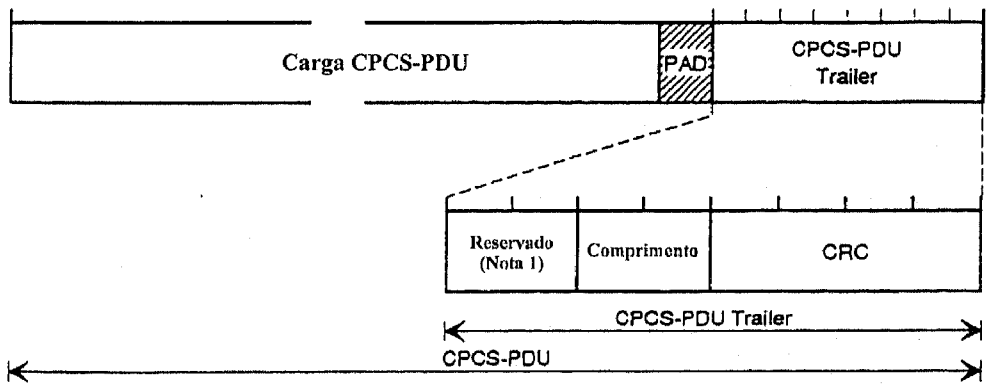
No caso de o débito de transporte TS for CBR, então é suficiente ir enchendo o buffer com os TS que vão chegando e ir retirando os TS ao débito da Trama de Transporte, garantindo que o buffer não esvazie ou transborde. Mas o problema não é tão fácil de resolver no caso VBR, dado que os atrasos e os débitos de transporte não são constantes.

5.2.3 A Camada de Adaptação ATM (AAL)

A camada de adaptação ATM está localizada entre a camada ATM e as camadas Superiores.

O AAL é subdividido em duas subcamadas: a subcamada de Segmentação e Reassemblagem (SAR) e a subcamada de convergência (CS). A principal função da SAR é a segmentação da informação, no lado do transmissor, num tamanho múltiplo de 48 bytes e a reassemblagem, no lado do receptor, dos pacotes recebidos. O CS é responsável pelas funções necessárias para assegurar a qualidade de serviço entregue às camadas superiores. Assim considera-se que o CS é mais um serviço específico que a SAR.

No caso do AAL5, na Subcamada de Convergência (CS ou CPCS- *Common Part CS*) ao PDU é acrescentado um *trailer* de 8 octetos. Este *trailer* é composto por um campo reservado de 2 octetos, por 2 octetos com o comprimento do PDU e 4 octetos com o CRC. Adicionalmente é necessário garantir o alinhamento dos 48 octetos, ou seja o tamanho do PDU tem de ser um múltiplo de 48 octetos. O *padding* é utilizado para assegurar esse alinhamento (Figura 5.8).



PAD:	*Padding*	(0.. 47 octetos)
Reservado:	Em estudo	(2 octetos)
Comprimento:	Comprimento do CPCS-PDU	(2 octetos)
CRC:	Cyclic Redundancy Check	(4 octetos)

Nota 1: Estão em estudo funções adicionais para além do alinhamento de 32-bit.

Figura 5.8 – Formato do AAL5 CPCS-PDU (I.363).

5.2.4 Programa desenvolvido

Neste caso o objectivo era analisar as consequências do empacotamento das tramas MPEG-2 no AAL5, nos casos do *PCR-aware* e do *PCR-unaware,* em particular a localização dos PCRs nas tramas e a sua influência no *jitter* na desmultiplexagem.

O programa implementado, baseou-se no já existente - o programa Selector, para detectar a presença dos PCRs necessários no caso de *PCR-aware* [I.363].

De salientar o facto de nesta abordagem terem sido analisadas ambas as tipologias de empacotamento (*PCR-aware* ou *PCR-unaware*) com diferentes números de pacotes TS a empacotar por cada PDU da camada AAL5.

Foram utilizadas nos testes duas tramas, um MPTS com dois programas e um TS. Em particular, o caso a seguir apresentado é o do TS gerado, que possui 4041 tramas, com 53 PCRs.

Como o débito era de 6Mbps, calculado através do programa Mpegchecker, então um atraso de um pacote de 188 bytes no PCR representa um atraso de 0,25ms.

No caso *PCR-unaware* foram efectuadas simulações variando N entre 1 e 14, dado que entre estes valores se verificar um ciclo no número de *Padding* (Tabela 5.1).

Tabela 5.1 – Relação entre o número de pacotes TS com o CS-PDU.

Número de pacotes TS (N)	Bytes de Padding	Tamanho do CS-PDU (bytes)
1	44	**5***48=240
2	0	**8***48=384
3	4	**12***48=576
4	8	**16***48=768
5	12	**20***48=960
6	16	**24***48=1152
7	20	**28***48=1344
8	24	**32***48=1536
9	28	**36***48=1728
10	32	**40***48=1920
11	36	**44***48=2112
12	40	**48***48=2304
13	44	**52***48=2496
14	0	**55***48=2640

De um modo geral o atraso médio aumentou e o número de pacotes CS-PDU diminuiu, o que seria de esperar. Nesta estrutura podemos verificar que de facto a camada AAL5 é muito simples originando um pequeno aumento da largura de banda utilizada na transmissão.

Tabela 5.2 – Relação entre o número de pacotes TS com o atraso médio.

Número de pacotes TS (N)	Número de CS-PDU	Atraso médio (ms)
1	4041	0,00
2	2021	0,12
3	1347	0,26
4	1011	0,38
5	809	0, 46
6	674	0,58
7	578	0,81
8	506	0,93
9	449	1,02
10	405	1,12
11	368	1,38
12	337	1,41
13	311	1,53
14	289	1,67

Graficamente podemos observar a distribuição dos atrasos, em números de pacotes dos PCRs no esquema *PCR-unaware* (Figura 5.9).

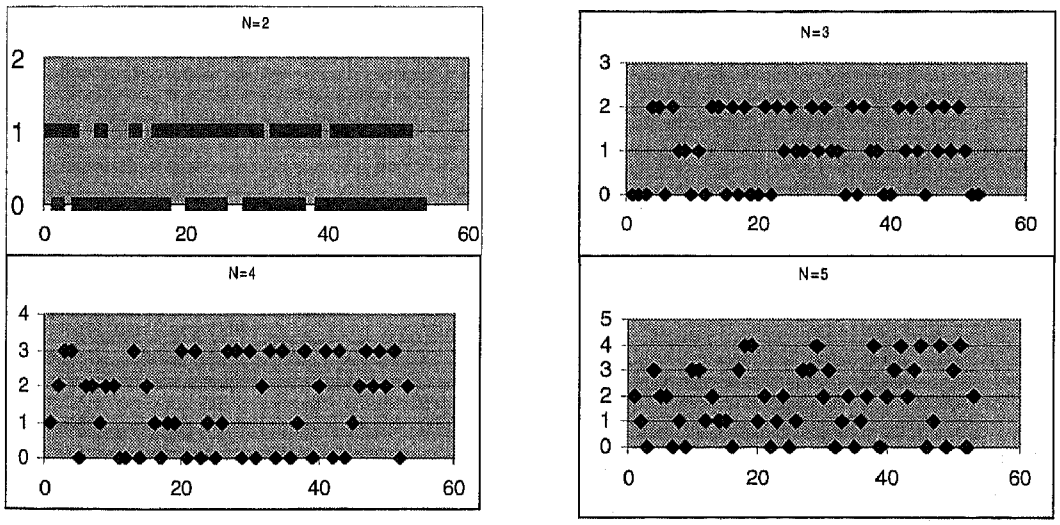


Figura 5.9 – Resultados obtidos para diferentes valores de N (de 2 a 5) na transmissão *PCR-unaware*.

Nestes quatro casos podemos constatar que de facto os PCRs se localizam de uma forma aleatória, havendo diversos atrasos não determinísticos. Se observarmos para os restantes casos o resultado seria idêntico dado que no *PCR-unaware* a localização dos PCRs não é considerada. Dos testes efectuados constatou-se que a melhor alternativa é de facto o N=2, dado que a largura de banda é toda utilizada e o atraso afecta apenas alguns PCRs.

De um modo geral, no caso do PCR-aware os atrasos eram nulos, mas a largura de banda aumentava consideravelmente.

O caso do PCR-aware foi posto de lado dado que a atribuição de banda não é deterministica, ao contrário do PCR-unaware.

Se a informação da localização do PCR for resolvida, através de vários processos já referidos, então para N=2 ou N=14 a largura de banda é toda aproveitada para o envio de pacotes TS na camada AAL5. De salientar que apesar de N=14 ser uma opção à primeira vista interessante, o facto de se perder um pacote no ATM, implicar a perda de 14 pacotes TS, seria inaceitável. Assim sendo o N=2 foi o tamanho escolhido.

Capítulo 6

Conclusões

6.1 Do trabalho

Este trabalho consistiu no estudo da transmissão de serviços MPEG-2 em redes de Banda Larga, em particular as redes ATM. Deu-se especial ênfase à camada MPEG-2 Sistema, responsável pela multiplexagem de tramas elementares de dados MPEG.

Foi desenvolvido um Selector de programas e o estudo das diferentes formas de transmissão MPEG-2 num canal ATM.

No que diz respeito à norma MPEG-2 Sistema, dado o seu carácter estável, foi possível elaborar um editor de programas, à custa da leitura e alteração das tabelas, consoante a escolha do utilizador, mas no que diz respeito à sua transmissão no meio ATM muito ainda há que definir.

Actualmente duas camadas de adaptação estão disponíveis para transportar dados Multimedia: AAL1 e AAL5. O AAL1 é basicamente utilizado para serviços de emulação de circuitos. Ou seja, oferece à aplicação um débito binário constante. O AAL1 oferece a vantagem de permitir detectar e corrigir erros através de FEC combinado com entrelaçamento[21]. No entanto este entrelaçamento introduz atrasos proporcionais ao tamanho do bloco FEC, tanto no emissor como no receptor. O esquema FEC é baseado nos códigos Reed-Solomon, mas é capaz de corrigir erros.

[21] Do inglês *interleaving*.

Por outro lado o AAL5 é mais simples e introduz menos "overhead". Esta simplicidade não fornece protecção contra erros, para além da sua detecção, e foi especialmente projectada para aplicações que implementam mecanismos robustas de controlo de erros. Devido à ausência de funcionalidades mais sofisticadas na detecção de erros, no caso de células perdidas, o AAL5 é incapaz de detectar a posição do erro no PDU, mesmo em camadas superiores, e o pacote é rejeitado. Se não houver retransmissão, o mais provável de acontecer, a rejeição do pacote leva à perda de dados, ao nível da aplicação.

Assim sendo é evidente que esta camada, apesar de ser a utilizada para aplicações em tempo real, ainda apresenta duas grandes limitações. Primeiro uma simples célula perdida causa a perda de vários dados que podiam continuar a ser utilizados pelo descodificador com técnicas como a re-sincronização. Segundo, o AAL5 não avisa as camadas superiores da ocorrência de erros. Desta forma o descodificador pode não detectar a corrupção dos dados e não aplicar um dos múltiplos mecanismos de correcção de erros.

6.2 Pontos em aberto e perspectivas de evolução

Alguns aspectos foram deixados em aberto até à altura da edição desta tese. Por um lado o interface terminal-utente não foi muito desenvolvido, uma vez que se tratava de um trabalho demonstrador das capacidades da norma MPEG-2. A desmultiplexagem para armazenamento revelou-se pouco optimizada, na medida em que para ser eficiente seria conveniente alterar os valores do PCR de acordo com o novo débito binário.

Por outro lado seria interessante efectuar testes numa rede ATM onde fosse possível simular diferentes situações específicas, com a introdução de erros e testar a resposta das diferentes técnicas aqui abordadas.

Acompanhar o desenvolvimento dos estudos na transmissão de serviços MPEG-2 em redes ATM, para além das opções da utilização das camadas de adaptação um e cinco (AAL1 e AAL5).

Surgem propostas alternativas à camada AAL5, cujo objectivo é exactamente fornecer informação adicional para correcção de erros. Estas variantes baseiam-se na utilização de pacotes de 47 bytes, o que liberta um octeto por cada célula ATM, que é utilizado para inserir uma sequência numerada e informação de controlo de um mecanismo FEC. Adicionalmente ao nível dos pacotes TS, como estes possuem um comprimento de 188 bytes, sendo 188 múltiplo de 47, cada pacote TS origina exactamente 4 pacotes. Esta técnica não introduz os cabeçalhos iniciais do PDU, pois considera que uma vez estabelecida a ligação o tamanho dos PDU's não irão ser alterados. A sequência introduzida, por sua vez, irá ser utilizada para detectar o número e a posição do erro, informação essa que pode ser passada para as camadas superiores. Os resultados obtidos, apesar de apresentar melhores valores que o AAL5 não se revelam satisfatórios [Adanez].

Por outro lado encontra-se ainda em aberto a utilização dum protocolo de adaptação AAL2 (ainda não normalizado) como uma alternativa possivel no transporte do tráfego variável gerado pela norma MPEG-2. Prevê-se que o AAL2 irá fornecer suporte para a sincronização do relógio e superior controlo de erros.

Neste momento assistimos ao surgimento de uma nova norma o MPEG-4, que constitui a fase mais activa no âmbito das organizações de normalização de algoritmos e instrumentos para a comunicação e armazenamento de imagem.

A norma MPEG-2, para além de ter conseguido um elevado grau de flexibilidade, apresenta algoritmos de codificação e uma sintaxe estabilizada para a solução de alguns problemas a enfrentar no âmbito do MPEG-4.

Bibliografia

- [ISO/IEC1] ISO/IEC 11138-1 " Generic Coding of Pictures and Associated Audio Information: System", Nov. 1994 (Recommendation ITU-T H.222.0).

- [ISO/IEC2] ISO/IEC 11138-2 " Generic Coding of Pictures and Associated Audio Information: Video", Nov. 1994 (Recommendation ITU-T H.262).

- [ISO/IEC3] ISO/IEC 11138-3 " Generic Coding of Pictures and Associated Audio Information: Audio", Nov. 1994 .

- [Bertin] Christian Bertin / CCETT, Amaro F. Sousa / IT, Gonzalo Figuera / Telefonica, Manuel Fridich / DeTeBerkon, Mario Guglielmo - " Experimentation of Multimedia Interactive Services in the Race MARS project" - CSELT.

- [Wright] Steven A. Wright - "SAA Audio Visual Multimedia Services (AMS) Implementation Agreement" , ATM Forum contribution AF95-00012 .

- [Morris] O.J. Morris -" MPEG-2 Where did it como from and What is it?", Philips Research Laboratories.

- [Gaspar] António J. T. Gaspar e Artur P. Alves - " Multiplexer MPEG-2 –One Multiplexer Multimédia Multiaplicação" .

- [Pires] Carlos G. Ferrreira M. Pires - " Co-Dec MPEG2 Vídeo: da Análise Funcional à realização de um Protótipo" .

- [Silva] Joaquim Fernando Silva -" A Norma MPEG-2 A Camada de Sistemas", Junho 1996.

- [Sarginson1] P. A. Sarginson - " MPEG-2 – A tutorial introduction to the system layer", BBC R&D.

- [Sarginson2] P. A. Sarginson - " MPEG-2 Overview of the Systems Layer", BBC R&D.

- [Nilsson] M. Nilsson - " MPEG-2 over ATM" ,BT Laboratories..

- [Rasheed] Yasser M. Rasheed - " ATM Adaptation Layers for Video Applications", Master Thesis by University of Toronto 1995.

- [Tryfonas] Christos Tryfonas - " MPEG-2 Transport over ATM Networks", Master thesis by University of California Santa Cruz 1996.

- [Perkins] M. Perkins, P. Skelly – " A hardware MPEG clock recovery experiment in the presence of Jitter" ATM Forum /94-0434, Abril 1994.

- [Handel] Rainer Handel, Manfred N. huber, Stefan Schroder – " ATM Networks Concepts, Protocols, Applications", Second Edition Maio 1994, Addison-Wesley.

- [Key] Peter Key, Robin Mscfadyen, Mike Nilson – " VBR Video, Policing and Dimension", ATM Forum /97-308 , Maio 1997.

- [Khasnabish] B. Khasnabish, Vijay Samalam – " Traffic Shaping for Carrying Variable Bit Rate MPEG-2 over rt-VBR", ATM Forum /97-0118, Fevereiro 1997.

- [Gringeri] S. Gringeri, B. Khasnabish, V. Salamam, A. Lewis – "Traffic Analysis of variable rate MPEG1 and MPEG2 Video", ATM Forum /97-0549, Julho 1997.

- [VOD1] " Audiovisual Multimedia Services: Video on Demand Specification 1.0" – ATM Forum Technical Committee af-saa-0049.000, Janeiro 1996

- [VBR1] " Audiovisual Multimedia services: VBR MPEG-2 Baseline Text" - ATM Forum Technical Committee BTD-SAA-AMS-VBRMPEG2-02.00, Fevereiro 1997.

- [VBR2] " Audiovisual Multimedia services: VBR MPEG-2 Living List" - ATM Forum Technical Committee LTD-SAA-AMS-VBRMPEG2-02.00, Fevereiro 1997.

- [BMS1] " Broadband Multimedia Services Living List " - ATM Forum Technical Committee LTD-SAA-AMS-BMS-02.00, Janeiro 1997.

- [TMS4] " Traffic Management Specification" Version 4.0, ATM Forum Fevereiro 1996.

- [I.363] " I.363 – B-ISDN Adaptation Layer Type 5 (AAL5) Specification", Swiss PTT.

- [Hodgins1] P. Hodgins, E. Itakura – " VBR MPEG-2 over AAL5", ATM Forum /94-1052, Dezembro 1994.

- [Hodgins2] P. Hodgins, E. Itakura – " The Issues of Transportation of MPEG over ATM", ATF Forum/94-0570, Julho 1994.

- [Goldman] Mathew S. Goldman, DiviCom " Variable Bit Rate MPEG-2 over ATM: Definitions and Recommendations", ATM Forum/96-1433.

- [Kushida] Takayuki Kushida -" VBR encoded MPEG-2 over AAL5 as a work item for AMS Phase 2" , ATM Forum/95-0444.

- [Seetharam] Srini Seetharam -" VBR Class for MPEG-2 ", ATM Forum/96-0665.

- [Cabral] José Manuel Vieira Cabral -" Emulação de Circuitos em Redes ATM", Tese de Mestrado 1995.

- [Perth] Perth -" Traffic Control and Congestion Control in B-ISDN" Draft Release 2, , ITU-T Grupo de trabalho 13 Recomendação I.371, Novembro 1995.

- [Lambarelli] Livio Lambarelli -" ATM Services Categories: The benefits to the User", CSELT, Torino Italy, 1996.

Nota: Diversas Fontes bibliográficas foram consultadas na internet por WWW.

• Endereço da página CISCO http://cio.cisco.com/

- [Cisco1] "Video Over ATM and Existing Networks", Novembro 1995.

- [Cisco2] "Asynchronous Transfer Mode", Outubro 1996.

- [Cisco3] "ATM Internet working", White Paper, Maio 1995.

- [Cisco4] " Performance Issues for high-End Video Over ATM", Agosto 1995.

- [Cisco5] " Glossary of ATM Terms and Acronyms".

- [Alles] Anthony Alles - " ATM Internetworking", ATM Product Line Manager, Cisco Systems , 1995 .
http://cio.cisco.com/warp/public/614/12.htm

• Endereço da Página ATM Forum http://www.atmforum.com/atmforum/

- [ATM1] "ATM Services Categories: The benefits to the user" , Agosto 1996.

- [ATM2] "Wide Area ATM Deployment in Europe", White Paper.

• Endereço da Página DSE http://www-dse.doc.ic.ac.uk/

- [Derbyshire] A. Derbyshire & K. C. Rajh " Video Transmission Over Broadband Networks", DSE Junho 1996.
  http://www-dse.doc.ic.ac.uk/~nd/surprise_96/vol4/arad/report.htm

• Endereço da Página CSELT http://drogo.cselt.stet.it/

- [Chiariglione1] Leonardo Chiariglione - "MPEG-2 FAQs" ISO/IEC JTC1/SC29/WG11 N1390, CSELT Junho 1996.
  http://drogo.cselt.stet.it/mpeg/faq_mpeg-2.htm

- [Chiariglione2] Leonardo Chiariglione – " MPEG and multimedia communications" , CSELT Agosto 1996.

• Endereço da Página EPFL http://tcomwww.epfl.ch/

- [Stock] Thomas Stock, Xavier Garcia – " On the Potencials of Error Correction Mechanisms applied to Real-Time Services over B-ISDN", Alcatel STR AG 1997.

- [Verscheure] Olivier Verscheure, Xavier Garcia Adanez – " Perceptual Quality Metric as a Performance Tool for ATM Adaptation of MPEG-2 based Multimedia Applications", Telecommunications Services Group, TCOM Laboratory Swiss Federal Institute of Technology. Lausanne, Switzerland.

- [Adanez] Xavier Garcia Adanez, Verscheure, Jean-Pierre Hubaux – " New Network and ATM Adaption Layers for Real-Time Multimedia Applications: A Performance Study Based on Psychophysics", Telecommunications Services Group, TCOM Laboratory Swiss Federal Institute of Technology. Lausanne, Switzerland.

Anexo A
Listagem do código implementado no Selector de Programas

Neste apêndice apresenta-se a listagem do programa, realizado em C, que foi implementado e que testa as capacidades de desmultiplexagem de um MPTS em TS.

Listagem do código implementado:

```
/*************************************
*************************************
 MPEG2 system bitstreams parser:
ISO/IEC 13818-1
30/03/97

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

#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "ctype.h"
#include <termios.h>
#include <unistd.h>
#define PES_BUFFER_SIZE 470 /*Maximum
buffer size (+1) for the PES_packets
                            header (in
PMT). Could include data_bytes. */

/* /// Compilation problem in UNIX
///// */

#ifndef SEEK_SET
  #define SEEK_SET 0
#endif
#ifndef SEEK_CUR
  #define SEEK_CUR 1
#endif

/* /// Type definitions
///////////////// */

unsigned long n_TPs; /* TP counter.
Initialized in PROGRAM_STREAM() (due to
                        Warn()).
                        Initialized and
used in TRANSPORT_STREAM(). Used in

typedef enum { NO, YES } boolean;
typedef struct PA_table *PAT_ptr;
typedef struct PM_table *PMT_ptr;
typedef struct CA_table *CAT_ptr;

/* /// Global variables
///////////////// */

boolean extract,
       log, /* Creation of a log file
with errors found in the analysed
stream
                Will also have some
statistics (e.g. overhead).
                See FindSwitch(),
Main(), Warn(). */
       unconditional;/* Type of PES
extraction:
                          1 - all PIDs
                          0 - after
having the PSI tables.
                        See
TRANSPORT_STREAM(), FindSwitch().
                          In case 1 the
new table may require much memory and
                          make the parsing
to abort.*/

unsigned char detail;/* Detail level in
the parsing file.
                       Initialized in
FindSwitch. */


                       Warn(); */

char no_extension[80];

struct buffer
```

```
{
  unsigned char buffer[1024]; /*must be
unsigned !*/
  int p; /* buffer pointer */
  int descriptor_start; /* buffer
pointer; start of most recent
descriptor */
  int bytes_left; /* Remaining bytes to
read in order to complete the section*/
}
  PA_section, /*Used in
TS_PROGRAM_ASSOCIATION_SECTION() */
  CA_section, /*Used in TS_CA_SECTION()
*/
  PM_section; /*Used in
TS_PROGRAM_MAP_SECTION() */

struct
{
  char buffer[4096]; /* Maximum size of
a private section */
  int p; /* buffer pointer */
  int bytes_left; /* Remaining bytes to
complete the section */
}
  private_section; /*Used in
PRIVATE_SECTION() */
struct PA_table
{
  int program_number;
  int PID;
  PAT_ptr next;
};

struct PM_table
{
  int elementary_PID; /*Initialized in
TS_PROGRAM_MAP_SECTION()*/
  char continuity_counter;/*Used in
TS_TRANSPORT_PACKET(); Init.in
TS_PROGRAM_MAP_SECTION()*/
  boolean old_stream_removed;
/*Activated in PES_PACKET(); Init.in
TS_PROGRAM_MAP_SECTION()*/
  unsigned char
buffer[PES_BUFFER_SIZE];/*PES_packet
header buffer */
  int p, info_start, info_end; /*
buffer pointers */
  int field; /*Which part of PES_packet
is being read*//*Init. in
TS_PROGRAM_MAP_SECTION()*/
  unsigned flag[7]; /*PES_packet flags
buffer (helps deciding field)*/
  int packet_length; /* If null then
length is unknown */
  int bytes_left; /* Remaining bytes to
complete PES_packet*/
  int header_data_length; /* To confirm
stuffing_bytes */
  PMT_ptr next;
};

struct CA_table
{
  int CA_PID;
  CAT_ptr next;
};
PAT_ptr PAT,
        new_PAT;
PMT_ptr PMT,
        new_PMT,
        ALL;/*Used in PES extraction
without PSI tables*/
CAT_ptr CAT,
        new_CAT;

int PAT_version; /* version_number of
new_PAT */
int PMT_version; /* version_number of
new_PMT */
int CAT_version; /* version_number of
new_CAT */

/* New variables*/
int prog;
int
vant,vnova,vactual,auxnum,bytes_lidos;
long pos, novapos,comp,aux1;
int c,mais,mais4112,mais4113,mais100,

mais200,maisf,maisf4112,maisf4113,maisf
100,maisf200;
int PIDPMT[10];
int PMTPIDcont;
FILE *trama;
char *sda;
int PIDPROG[10];
boolean
novo,inicio,NovaPAT,NovaPMT,programaPMT
,progPIDPMT,NPAT,PATII,agora;
boolean novoaux,vr,fim;
int countprog;
int PIDesc,i;
struct termios new_term, old_term;
char crr ;
int jj, ii=0;
char option;
/* /// Function prototypes
////////////// */

void    Program_Use
(void);
void    Expands_meta_ch
(int *, char ***);
char   *Obtains_path
(char *);
void    NameOut
(char *, char *, char *);
void    FindSwitch
(int *, char ***);
boolean PROGRAM_STREAM
(FILE *, FILE *);
void    STARTCODE_prefix
(FILE *);
void    PACK
(FILE *, FILE *);
void    SYSTEM_HEADER
(FILE *, FILE *);
void    PACKET
(int, FILE *, FILE *);
/* Nova funcao */
void    copia
(FILE *,char *, int);
boolean TRANSPORT_STREAM
(FILE *, FILE *);
void    TS_TRANSPORT_PACKET
(FILE *, FILE *);
PAT_ptr PA_PID
(PAT_ptr, int, int *);
PMT_ptr Elementary_PID
(PMT_ptr, int);
PMT_ptr ALL_PID
(PMT_ptr, int);
```

```
void    TS_ADAPTATION_FIELD
(int, int *, FILE *, FILE *);
void    TS_PROGRAM_ASSOCIATION_SECTION
(int, int *, FILE *, FILE *);
void    TS_CA_SECTION
(int, int *, FILE *, FILE *);
void    TS_PROGRAM_MAP_SECTION
(int, int *, FILE *, FILE *);
void    DESCRIPTOR
(struct buffer *, FILE *);
void    SYSTEM_CLOCK_DESCRIPTOR
(struct buffer *, FILE *);
void
MULTIPLEX_BUFFER_UTILIZATION_DESCRIPTOR
(struct buffer *, FILE *);
void    MAXIMUM_BITRATE_DESCRIPTOR
(struct buffer *, FILE *);
void    PES_PACKET
(PMT_ptr, int, int *, FILE *, FILE *);
void    Warn
(FILE *, char *);

/*
////////////////////////////////////
*/
void
Program_Use()
/************************************
 ISO 13818-1 streams parser.

************************************/
{
  fprintf(stderr,"--------------------
----------------------------------------
-\n");
  fprintf(stderr,"MPEG2 system
bitstreams parser: ISO/IEC 13818-1
\n");
  fprintf(stderr,"\nOutputs:\n");
  fprintf(stderr,"The output file with
the program specified is prog.spts\n");
  fprintf(stderr,"--------------------
----------------------------------------
-\n");
  exit(0);
}

void
NameOut (char *output, char *input,
char *executable_name)
/************************************
 Obtains the output filename by
adding/modifying an extension to the
input.
 The extension is made from the first 3
(or less) letters of the program name
 without the path.

************************************/
{
    register int i, j;
    char extension[80];
    strcpy(extension, executable_name);
    for(i = strlen(extension) - 1;
        i>=0 && extension[i]!='\\' &&
extension[i]!='/' && extension[i]!=':';
        i--) ;
    extension[4] = '\0';
    extension[0] = '.';
    for(j=1; j<4; j++)
    {
      extension[j] =
executable_name[i+j];
      if(extension[j]=='.')
      {
        extension[j]='\0';
        break;
      }
    }
    strcpy(output, input);
    for(i=strlen(output)-1; i>=0; i--)
    {
      if(output[i]=='.')
      {
        output[i]='\0';
        break;
      }
    }
    /*Global variable initialization.*/
    strcpy(no_extension, output);
    strcat(output, extension);
}
void
FindSwitch(int *argc, char **argv[])
/************************************
 Finds (and removes), in the command
line arguments, instructions to define
the
 program behaviour.

************************************/
{
  register int i,
               new_argc=0;
  char **argv_buffer = (char
**)malloc(*argc * sizeof(char *));
  /* Switches initialization (defaults)
*/
  extract = YES;
  unconditional = NO;
  log = NO;
  detail = 1;
  /* Search */
  for(i=0; i < *argc; i++)
  {
    if( (*argv)[i][0] == '-') /* switch
indicator */
    {
        fprintf(stderr,"\nInvalid
switch !");
        Program_Use();
    }
    else
    {
      argv_buffer[new_argc] = (char
*)malloc(80 * sizeof(char));
      strcpy(argv_buffer[new_argc++],
(*argv)[i]);
    }
  }
  *argc = new_argc;
  *argv = argv_buffer;
}

main(int argc, char *argv[])
{
  FILE *in, *out ;
  char str[80], s[80];
  int file;
  FindSwitch(&argc, &argv); /*
Initializes global variables according
to the
                                command
line.*/
  if(argc < 2) Program_Use();
  for(file=1; file < argc; file++)
```

```
  {
    if((in=fopen(argv[file],"rb")) ==
NULL)
    {
      sprintf(str,"\nrb - can't open
%s", argv[file]);
      perror(str);
      continue;
    }
    NameOut(str, argv[file], argv[0]);
   sda="prog.spts";
   if((out=fopen(str,"wt")) == NULL)
    {
      sprintf(s,"\nwt - can't open %s",
str);
      perror(s);
      fclose(in);
      continue;
    }
    if (setvbuf(in, NULL, _IOFBF,
15360) != 0)
      fprintf(stderr,"\n(failed to set
up buffer for input file)");
    if (setvbuf(out, NULL, _IOFBF,
15360) != 0)
      fprintf(stderr,"\n(failed to set
up buffer for output file)");
    fprintf(stdout,"\n A processar a
informacao de %s para %s\n",
argv[file], str);
    fprintf(out,"%s
parse:",argv[file]);
    if( log ) /* file with errors and
stream statistics */
    {
      sprintf(s,"%s.LOG",
no_extension);
      if(remove(s))
      {
       sprintf(str,"(can't remove %s
(log file))", s);
       perror(str);
      }
    }

mais=mais4112=mais4113=maisf4112=maisf4
113=mais100=mais200=maisf100=maisf200=m
aisf=0;

novo=novoaux=vant=vactual=inicio=NPAT=N
ovaPAT=programaPMT=progPIDPMT=agora=0;
     prog=1;
      PATII=1;
      tcgetattr(STDIN_FILENO,
&old_term);
      new_term=old_term;
      new_term.c_lflag &= ~(ECHO |
ECHOE | ECHOK | ECHOCTL | ECHONL
|ICANON);
      new_term.c_cc[VMIN] =0;
      new_term.c_cc[VTIME]=0;

    if( !TRANSPORT_STREAM(in, out) )
      fprintf(stderr,"\n%s isn't a
valid ISO13818-1 bitstream
!\n",argv[file]);
    fclose(in);
    fclose(out);
  }
printf("\n-----------------------//---
---------------------");
printf("\n They were written in file
prog.spts  %d PATs,%d PMTs e %Streams
\n ",maisf,maisf4112,maisf100);
  return 0;
}

void
copia(FILE *in,char *saida,int pos)
{
  char str[80], s[80];
  FILE *stream;
  int comp;
  long aux;

 if (fim && NPAT)
 {
 if ((stream=fopen(saida,"ab")) ==
NULL)
        {
          sprintf(str,"\nab - can't
open %s (elementary stream)",s);
          perror(str);
          return;
        }
  aux=pos;
  fseek(in,aux,SEEK_SET);
  comp=188;
  while(comp--)  putc(getc(in),stream);
  fclose(stream);
}
}

boolean
TRANSPORT_STREAM (FILE *in, FILE *out)
/*************************************
 In the Transport Packets (TPs)
identification I assume that there can
be trash
 between packets and without
synchronism byte emulation.
 Returns YES or NO depending on a valid
stream parsing.
 Counts the number of processed
packets.
 Initializes the PSI (Program Specific
Information) linked list pointers
 (global variables). An empty table is
a one element list.

*************************************/
{
  boolean aux; PAT_ptr pat_aux; PMT_ptr
pmt_aux; CAT_ptr cat_aux;/*Auxiliary*/
  while(getc(in) != 0x47)  if(feof(in))
return NO; /* Luck ? */
  if(
(new_PAT=(PAT_ptr)malloc(sizeof(struct
PA_table))) == NULL )
   {puts("TRANSPORT_STREAM(): malloc
error"); exit(1);}
  new_PAT->next=NULL;
  if(
(new_PMT=(PMT_ptr)malloc(sizeof(struct
PM_table))) == NULL )
   {puts("TRANSPORT_STREAM(): malloc
error"); exit(1);}
  new_PMT->next=NULL;
  if(
(new_CAT=(CAT_ptr)malloc(sizeof(struct
CA_table))) == NULL )
   {puts("TRANSPORT_STREAM(): malloc
error"); exit(1);}
  new_CAT->next=NULL;
```

```
  PAT = new_PAT; /* The current table
is new */
  PMT = new_PMT;
  CAT = new_CAT;
  PAT_version = 0;
  PMT_version = 0;
  CAT_version = 0;
  if( unconditional )
  {
    if(
(ALL=(PMT_ptr)malloc(sizeof(struct
PM_table))) == NULL )
    {puts("TRANSPORT_STREAM(): malloc
error"); exit(1);}
    ALL->next=NULL;
  }
  else ALL = NULL;
  TS_TRANSPORT_PACKET(in, out);
  n_TPs = 1;
  while( !feof(in) ) if(getc(in) ==
0x47)
                        {

TS_TRANSPORT_PACKET(in, out);

tcsetattr(STDIN_FILENO, TCSADRAIN,
&new_term);

read(STDIN_FILENO, &option, 1);
                        }
 tcsetattr(STDIN_FILENO, TCSADRAIN,
&old_term);

   printf("\a");
  /* Memory release */
  if(ALL != NULL)
  {
    while(ALL->next != NULL)
    {
      pmt_aux = ALL->next;
      free(ALL);
      ALL = pmt_aux;
    }
    free(ALL);
  }
  if(PAT == new_PAT) aux = NO;
  else aux = YES;
  if(PAT != NULL)
  {
    while(PAT->next != NULL)
    {
      pat_aux = PAT->next;
      free(PAT);
      PAT = pat_aux;
    }
    free(PAT);
  }
  if(aux && new_PAT != NULL)
  {
    while(new_PAT->next != NULL)
    {
      pat_aux = new_PAT->next;
      free(new_PAT);
      new_PAT = pat_aux;
    }
    free(new_PAT);
  }
  if(PMT == new_PMT) aux = NO;
  else aux = YES;
  if(PMT != NULL)
  {
    while(PMT->next != NULL)
    {
      pmt_aux = PMT->next;
      free(PMT);
      PMT = pmt_aux;
    }
    free(PMT);
  }
  if(aux && new_PMT != NULL)
  {
    while(new_PMT->next != NULL)
    {
      pmt_aux = new_PMT->next;
      free(new_PMT);
      new_PMT = pmt_aux;
    }
    free(new_PMT);
  }
  if(CAT == new_CAT) aux = NO;
  else aux = YES;
  if(CAT != NULL)
  {
    while(CAT->next != NULL)
    {
      cat_aux = CAT->next;
      free(CAT);
      CAT = cat_aux;
    }
    free(CAT);
  }
  if(aux && new_CAT != NULL)
  {
    while(new_CAT->next != NULL)
    {
      cat_aux = new_CAT->next;
      free(new_CAT);
      new_CAT = cat_aux;
    }
    free(new_CAT);
  }
  return YES;
}
void
TS_TRANSPORT_PACKET (FILE *in, FILE
*out)
/*************************************
 sync_byte = 1 byte (already read).

 transport_error_indicator = 1 bit
\
 payload_unit_start_indicator = 1 bit
\
 transport_priority = 1 bit
\
 PID = 13 bits
> 3 bytes.
 transport_scrambling_control = 2 bits
/
 adaptation_field_control = 2 bits
/
 continuity_counter = 4 bits
/

 adaptation_field and/or data_bytes =
until 184 bytes.


*************************************/
{
  struct header_bits
  {
    unsigned transport_error_indicator
:  1;
    unsigned
payload_unit_start_indicator :  1;
```

```
    unsigned transport_priority
:  1;
    unsigned PID_12_8
:  5;
    unsigned PID_7_0
:  8;
    unsigned
transport_scrambling_control :  2;
    unsigned adaptation_field_control
:  2;
    unsigned continuity_counter
:  4;
  };

  union
  {
    struct header_bits bits;
    char byte[3];
  } header;

  int PID;
  register int i;
  char str[80];
  int bytes_read;
  int PM_flag;
  PAT_ptr PA_ptr;
  PMT_ptr PES_ptr;
  /*Read 3 bytes of TP header.*/
  for(i=0; i<3; i++)
header.byte[i]=(char)getc(in);
  if( feof(in) )
  {
    Warn(out,"Premature EOF in TP
header");
    return;
  }
  bytes_read = 4;
  /* Packet Identifier */
  PID = header.bits.PID_12_8 * 256
      +header.bits.PID_7_0;
  if(PID == 0) fprintf(out,"(Program
Association Table)");

/* copy  PID of PMT */
    if ( NovaPMT)
    if (PID==PIDPROG[prog])
       {
         maisf4112++;
     /* copy 4 bytes of header */
         pos=ftell(in);
         novapos=pos-4;
         copia(in,sda,novapos);
         fseek(in,pos,SEEK_SET);
        }

  /* copy PIDs referenced in PMT..*/
        if (progPIDPMT)
            for (c=1;c<PMTPIDcont;c++)
             {
               if (PID==PIDPMT[c])
               {
                 maisf100++;
                pos=ftell(in);
                pos=pos-4;
                copia(in,sda,pos);
                pos=pos+4;

fseek(in,pos,SEEK_SET);
               }
             }
   /* End of copy */

  if(PID == 1)
fprintf(out,"(Conditional Access
Table)");
  if(PID == 0x1FFF)
  {
    i = 188 - bytes_read;
    fseek(in, i, SEEK_CUR); /*Ignores
rest of packet*/
    bytes_read = 188;
    return;
  }

  /* adaptation_field */
  i =
header.bits.adaptation_field_control;
  if( i == 2  ||  i == 3 )
TS_ADAPTATION_FIELD(i, &bytes_read, in,
out);
  /* payload */
  if(
header.bits.adaptation_field_control ==
1 ||

header.bits.adaptation_field_control ==
3    )
  {
    if(bytes_read >= 188) Warn(out,"No
space for payload");
    else
    {
      i =
header.bits.payload_unit_start_indicato
r;
      if(PID == 0)
                {
                 mais++;
                 /* copy header*/
                                  fim=0;

bytes_lidos=0;

comp=bytes_read;

pos=ftell(in)-comp;
                                  if
(PATII){ trama=fopen(sda,"wb");

PATII=0;

}
                                  else
trama=fopen(sda,"ab");

fseek(in,pos,SEEK_SET);

aux1=comp;

while(aux1--)putc(getc(in),trama);

bytes_lidos=comp;

maisf++;

TS_PROGRAM_ASSOCIATION_SECTION(i,
&bytes_read, in, out);
          }
      else
      if(PID == 1)  TS_CA_SECTION(i,
&bytes_read, in, out);
      else
      {
        PA_ptr = PA_PID(PAT, PID,
&PM_flag);
```

```
        if(PA_ptr != NULL)
        {
          if(!PM_flag)
fprintf(out,"|\n| (Network Information
Table)\n");
          else
           {
             fprintf (out,
"|PID=0x%04x",PID);
            TS_PROGRAM_MAP_SECTION(i,
&bytes_read, in, out);
             programaPMT=0;
          }
        }
        else
        {
          /*If there's a table with all
PIDs then PES extraction is
            independent of PMT */
          if(ALL != NULL)
          {
            PES_ptr = ALL_PID(ALL, PID);
            if(PES_ptr == NULL)
            {
              /*Linked list new
element.*/
              if(
(PES_ptr=(PMT_ptr)malloc(sizeof(struct
PM_table))) == NULL )

{puts("TS_TRANSPORT_PACKET(): malloc
error"); exit(1);}
              /*Insert new element in
table. At the list start, just after
the
               head */
              PES_ptr->next = ALL->next;
              ALL->next = PES_ptr;
              PES_ptr->elementary_PID =
PID;
              PES_ptr-
>old_stream_removed = NO;
              PES_ptr->field = 0;
/*PES_PACKET() ignored if there's no
unit_start*/
            }
            else /* duplicate packets */
            if(PES_ptr-
>continuity_counter ==
header.bits.continuity_counter)
            {
              i = 188 - bytes_read;
              sprintf(str,"Duplicate
packets: payload ignored (%d
bytes)",i);
              Warn(out, str);
              fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
              bytes_read = 188;
              return;
            }
            PES_ptr->continuity_counter
= header.bits.continuity_counter;

PES_PACKET(PES_ptr,i,&bytes_read,in,out
);
          }
          else
          {
            PES_ptr =
Elementary_PID(PMT, PID);
            if( PES_ptr != NULL &&
                PES_ptr-
>continuity_counter !=
header.bits.continuity_counter )
              {
                PES_ptr-
>continuity_counter =
header.bits.continuity_counter;
                PES_PACKET(PES_ptr, i,
&bytes_read, in, out);
              }
              else
              {
                i = 188 - bytes_read;
                fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
                bytes_read = 188;
                if(PES_ptr == NULL)
                  sprintf(str,"Unknown
PID: payload ignored (%d bytes)",i);
                else
                  sprintf(str,"Duplicate
packets: payload ignored (%d
bytes)",i);
                Warn(out, str);
              }
            }
          }
        }
      }
    }
  }
}


PAT_ptr
PA_PID (PAT_ptr table, int PID, int
*PM_flag)
/*************************************
 Searches PA_table (GLOBAL) looking for
the PID given as an argument.
 If found it's relative (except for
program_number = 0, Network) to packets
 with PM_table (the flag is activated).
 Returns a pointer to the found table
entry (or NULL).
 The first element of the linked isn't
part of the table information.

*************************************/
{
  PAT_ptr ptr;

  *PM_flag = 0;
  if(table != NULL) /*active table*/
  {
    ptr = table->next;
    while(ptr != NULL)
    {
      if(ptr->PID == PID)
      {
        if(ptr->program_number != 0)
*PM_flag = 1;
        return ptr;
      }
      ptr = ptr->next;
    }
  }

  return NULL;
}


PMT_ptr
Elementary_PID (PMT_ptr table, int PID)
```

```
/*************************************
 Searches PM_table (GLOBAL) looking for
the PID given as an argument.
 If found it's relative to packets with
elementary streams.
 Returns a pointer to the found table
entry (or NULL).
 The first element of the linked isn't
part of the table information.

*************************************/
{
  PMT_ptr ptr;

  if(table != NULL) /*active table*/
  {
    ptr = table->next;
    while(ptr != NULL)
    {
      if(ptr->elementary_PID == PID)
return ptr;
      ptr = ptr->next;
    }
  }
  return NULL;
}

PMT_ptr
ALL_PID (PMT_ptr table, int PID)
/*************************************
 Searches ALL table (GLOBAL) looking
for the PID given as an argument.
 If found it could be relative to
packets with elementary streams.
 Returns a pointer to the found table
entry (or NULL).
 The first element of the linked isn't
part of the table information.

*************************************/
{
  PMT_ptr ptr;

  if(table == NULL)
  { puts("error!"); exit(3); }

  ptr = table->next;
  while(ptr != NULL)
  {
    if(ptr->elementary_PID == PID)
return ptr;
    ptr = ptr->next;
  }

  return NULL;
}

void
TS_ADAPTATION_FIELD (int control, int
*bytes_read, FILE *in, FILE *out)
/*************************************
 adaptation_field_length = 1 byte

 descontinuity_indicator = 1 bit
\
 random_access_indicator = 1 bit
\
 elementary_stream_priority_indicator =
1 bit   \
 PCR_flag = 1 bit
> 1 byte.
 OPCR_flag = 1 bit
/
 splicing_point_flag = 1 bit
/
 transport_private_data_flag = 1 bit
/
 adaptation_field_extension_flag = 1
bit     /

 program_clock_reference_base = 33 bits
\
 reserved = 6 bits
> 6 optional bytes.
 program_clock_reference_extension = 9
bits /

 original_program_clock_reference_base
= 33 bits     \
 reserved = 6 bits
> 6 optional bytes.

original_program_clock_reference_extens
ion = 9 bits /

 splice_countdown = 1 optional byte.

 transport_private_data_length = 1
optional byte.
 private_data_byte = n optional bytes.

 adaptation_field_extension_length = 8
bits \
 ltw_flag = 1 bit
\
 piecewise_rate_flag = 1 bit
> 2 optional bytes.
 seamless_splice_flag = 1 bit
/
 reserved = 5 bits
/

 stuffing_byte = n optional bytes.


*************************************/
{
  struct flag_bits
  {
    unsigned descontinuity_indicator
:  1;
    unsigned random_access_indicator
:  1;
    unsigned
elementary_stream_priority_indicator :
1;
    unsigned PCR_flag
:  1;
    unsigned OPCR_flag
:  1;
    unsigned splicing_point_flag
:  1;
    unsigned
transport_private_data_flag          :
1;
    unsigned
adaptation_field_extension_flag      :
1;
  };

  union
  {
    struct flag_bits bits;
    char byte;
  } flags;
```

```
  struct PCR_bits /* Used for PCR and
OPCR */
  {
    unsigned PCR_base_32_25    :  8;

    unsigned PCR_base_24_17    :  8;

    unsigned PCR_base_16_9     :  8;

    unsigned PCR_base_8_1      :  8;
    unsigned PCR_base_0        :  1;
    unsigned reserved          :  6;
    unsigned PCR_extension_8   :  1;
    unsigned PCR_extension_7_0 :  8;
  };

  union
  {
    struct PCR_bits bits;
    char byte[6];
  } PCR;

  struct extension_flag_bits
  {
    unsigned ltw_flag              :
1;
    unsigned piecewise_rate_flag   :
1;
    unsigned seamless_splice_flag  :
1;
    unsigned reserved              :
5;
  };

  union
  {
    struct extension_flag_bits bits;
    char byte;
  } extension;

  struct ltw_bits
  {
    unsigned ltw_valid_flag  :  1;
    unsigned ltw_offset_14_8 :  7;
    unsigned ltw_offset_7_0  :  8;
  };

  union
  {
    struct ltw_bits bits;
    char byte[2];
  } ltw;

  struct piecewise_rate_bits
  {
    unsigned reserved             :  2;
    unsigned piecewise_rate_21_16 :  6;
    unsigned piecewise_rate_15_8  :  8;
    unsigned piecewise_rate_7_0   :  8;
  };

  union
  {
    struct piecewise_rate_bits bits;
    char byte[2];
  } piecewise_rate;

  struct seamless_splice_bits
  {
    unsigned splice_type       :  4;
    unsigned DTS_next_au_32_30 :  3;
    unsigned marker_bit_1      :  1;
    unsigned DTS_next_au_29_22 :  8;
    unsigned DTS_next_au_21_15 :  7;
    unsigned marker_bit_2      :  1;
    unsigned DTS_next_au_14_8  :  8;
    unsigned DTS_next_au_7_0   :  7;
    unsigned marker_bit_3      :  1;
  };

  union
  {
    struct seamless_splice_bits bits;
    char byte[5];
  } splice;

  int    adaptation_field_length;
  double PCR_base, PCR_extension,
         OPCR_base, OPCR_extension;
  int    splice_countdown,
         transport_private_data_length,

adaptation_field_extension_length,
         stuffing_bytes;
  char str[80];
  register int i,
               n_bytes;/*counter of
bytes in the adaptation_field*/
  fprintf(out,"|ADAPTATION
FIELD*****************************\n")
;
  /*Read adaptation_field_length.*/
  adaptation_field_length = getc(in);
  if( feof(in) )
  {
    Warn(out,"Premature EOF in
adaptation_field_length");
    exit(1);
  }
  n_bytes = 1;
  fprintf(out,"|
adaptation_field_length =
%d\n",adaptation_field_length);
  if(control == 3 &&
(adaptation_field_length > 183 ||
adaptation_field_length < 0) ||
     control == 2 &&
adaptation_field_length != 183 )
  {
    i = 188 - *bytes_read -n_bytes;
    sprintf(str,"Invalid length: %d
bytes ignored",i);
    Warn(out, str);
    fseek(in, i, SEEK_CUR); /*Ignores
rest of packet*/
    *bytes_read = 188;
    return;
  }
  if(!adaptation_field_length) /*
single stuffing_byte*/
  {
    *bytes_read += n_bytes;
    return;
  }

  /*Read flags byte.*/
  flags.byte = (char)getc(in);
  if( feof(in) )
  {
    Warn(out,"Premature EOF in flags");
    exit(1);
  }
  n_bytes++;

  /* descontinuity_indicator */
```

```
  i =
flags.bits.descontinuity_indicator;
  if(detail < 2 || i)
  {
    fprintf(out,"|
descontinuity_indicator = %d ", i);
    if(i) fprintf(out,"(new time base
(PCR) in this or next packet(s) with
the same PID)");
    fprintf(out,"\n");
  }
  /* PCR_flag */
  i = flags.bits.PCR_flag;
  if(detail < 2 || i)
    fprintf(out,"| PCR_flag
= %d\n", i);
  /* OPCR_flag */
  i = flags.bits.OPCR_flag;
  if(detail < 2 || i)
    fprintf(out,"| OPCR_flag
= %d\n", i);
  /* Program Clock Reference */
  if( flags.bits.PCR_flag )
  {
    /*Read 6 bytes of PCR.*/
    for(i=0; i<6; i++)
PCR.byte[i]=(char)getc(in);
    if( feof(in) )
    {
      Warn(out,"Premature EOF in PCR");
      exit(1);
    }
    n_bytes += 6;

    PCR_base =
(double)PCR.bits.PCR_base_32_25 *
33554432

+(double)PCR.bits.PCR_base_24_17 *
131072

+(double)PCR.bits.PCR_base_16_9  * 512

+(double)PCR.bits.PCR_base_8_1   * 2

+(double)PCR.bits.PCR_base_0;
    fprintf(out,"|\n| PCR_base = %g\n",
PCR_base);
    PCR_extension =
(double)PCR.bits.PCR_extension_8 * 256

+(double)PCR.bits.PCR_extension_7_0;
    fprintf(out,"| PCR_extension = %g
", PCR_extension);

    fprintf(out,"(PCR = %g s)\n",
(PCR_base*300+PCR_extension)/27000000);
  }

  /* Original Program Clock Reference
*/
  if( flags.bits.OPCR_flag )
  {
    /*Read 6 bytes of OPCR.*/
    for(i=0; i<6; i++)
PCR.byte[i]=(char)getc(in);
    if( feof(in) )
    {
      Warn(out,"Premature EOF in
OPCR");
      exit(1);
    }
    n_bytes += 6;

    OPCR_base =
(double)PCR.bits.PCR_base_32_25 *
33554432

+(double)PCR.bits.PCR_base_24_17 *
131072

+(double)PCR.bits.PCR_base_16_9  * 512

+(double)PCR.bits.PCR_base_8_1   * 2

+(double)PCR.bits.PCR_base_0;
    fprintf(out,"|\n| OPCR_base =
%g\n", OPCR_base);
    OPCR_extension =
(double)PCR.bits.PCR_extension_8 * 256

+(double)PCR.bits.PCR_extension_7_0;
    fprintf(out,"| OPCR_extension = %g
", OPCR_extension);

    fprintf(out,"(OPCR = %g s)\n",
(OPCR_base*300+OPCR_extension)/27000000
);
  }

  if( flags.bits.splicing_point_flag )
  {
    splice_countdown = getc(in);
    if( feof(in) )
    {
      Warn(out,"Premature EOF in
splice_countdown");
      exit(1);
    }
    n_bytes++;
  }
  /*Read private data.*/
  if(
flags.bits.transport_private_data_flag
)
  {
    /*Read length*/
    transport_private_data_length =
getc(in);
    if( feof(in) )
    {
      Warn(out,"Premature EOF in
transport_private_data_length");
      exit(1);
    }
    n_bytes++;
    if((transport_private_data_length +
n_bytes -1) > adaptation_field_length)
    {
      i = 188 - *bytes_read -n_bytes;
      sprintf(str,"Invalid length: %d
bytes ignored",i);
      Warn(out, str);
      fseek(in, i, SEEK_CUR); /*Ignores
rest of packet*/
      *bytes_read = 188;
      return;
    }

    /*Read data*/
    fseek(in,
transport_private_data_length,
SEEK_CUR);
    if( feof(in) )
    {
```


```
      Warn(out,"Premature EOF in
private data");
      exit(1);
    }
    n_bytes +=
transport_private_data_length;
  }

  /*Read adaptation_field extension.*/
  if(
flags.bits.adaptation_field_extension_f
lag )
  {
    /*Read length*/
    adaptation_field_extension_length =
getc(in);
    if( feof(in) )
    {
      Warn(out,"Premature EOF in
adaptation_field_extension_length");
      exit(1);
    }
    n_bytes++;

if((adaptation_field_extension_length +
n_bytes -1) > adaptation_field_length)
    {
      i = 188 - *bytes_read -n_bytes;
      sprintf(str,"Invalid length: %d
bytes ignored",i);
      Warn(out, str);
      fseek(in, i, SEEK_CUR); /*Ignores
rest of packet*/
      *bytes_read = 188;
      return;
    }


if(adaptation_field_extension_length >
0)
    {
      /*Read extension flags.*/
      extension.byte = getc(in);
      if( feof(in) )
      {
       Warn(out,"Premature EOF in
extension flags");
       exit(1);
      }
      n_bytes++;

adaptation_field_extension_length--;
      /* Legal Time Window */
      if(extension.bits.ltw_flag)
      {
       /*Read 2 bytes of ltw*/
       ltw.byte[0] = getc(in);
       ltw.byte[1] = getc(in);
       if( feof(in) )
       {
         Warn(out,"Premature EOF in
ltw");
         exit(1);
       }
       n_bytes += 2;
       adaptation_field_extension_lengt
h -= 2;

      }

      /* Piecewise Rate */

if(extension.bits.piecewise_rate_flag)
      {
       /*Read 3 bytes of
piecewise_rate*/
       for(i=0; i<3; i++)
piecewise_rate.byte[i]=(char)getc(in);
       if( feof(in) )
       {
         Warn(out,"Premature EOF in
piecewise_rate");
         exit(1);
       }
       n_bytes += 3;
       adaptation_field_extension_lengt
h -= 3;

      }

      /* Seamless Splice */

if(extension.bits.seamless_splice_flag)
      {
       /*Read 5 bytes.*/
       for(i=0; i<5; i++)
splice.byte[i]=(char)getc(in);
       if( feof(in) )
       {
         Warn(out,"Premature EOF
prematuro in Seamless Splice");
         exit(1);
       }
       n_bytes += 5;
       adaptation_field_extension_lengt
h -= 5;

       if( !splice.bits.marker_bit_1
||
           !splice.bits.marker_bit_2
||
           !splice.bits.marker_bit_3
)
       {
         i = 188 - *bytes_read -
n_bytes;
         sprintf(str,"Splice: invalid
marker(s) - %d bytes ignored",i);
         Warn(out, str);
         fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
         *bytes_read = 188;
         return;
       }
      }

if(adaptation_field_extension_length <
0)
      {
       i = 188 - *bytes_read -n_bytes;
       sprintf(str,"adaptation_field_ex
tension length doesn't fit: %d bytes
ignored",i);
       Warn(out, str);
       fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
       *bytes_read = 188;
       return;
      }
      fseek(in,
adaptation_field_extension_length,
SEEK_CUR);
      if( feof(in) )
      {
       Warn(out,"Premature EOF in
reserved bytes");
```

```
        exit(1);
      }
      n_bytes +=
adaptation_field_extension_length;
    }
  }

  for(stuffing_bytes=0, i=0; i <
adaptation_field_length +1 - n_bytes;
i++)
    if(getc(in) == 0xFF)
stuffing_bytes++;

  *bytes_read +=
(adaptation_field_length +1);
}
void
TS_PROGRAM_ASSOCIATION_SECTION(int
unit_start,int *bytes_read,FILE
*in,FILE *out)
/*************************************
 pointer_field = 1 optional byte
depending on 'unit_start'.

 table_id = 1 byte

 section_syntax_indicator = 1 bit \
 no_name = 1 bit                   \
 reserved1 = 2 bits                 \
 section_length = 12 bits            >
5 bytes.
 transport_stream_id = 16 bits      /
 reserved2 = 2 bits                /
 version_number = 5 bits          /
 current_next_indicator = 1 bit  /

 section_number = 1 byte;

 last_section_number = 1 byte;

 program_number = 16 bits            \
 reserved = 3 bits
> N * 4 bytes.
 network_or_program_map_PID = 13 bits /

 CRC_32 = 4 bytes;

 Uses a buffer for the section, defined
in a global struct. This way one can
 reconstruct the section from packet to
packet and do the final CRC.
 descriptor_start is not used in this
struct.


*************************************/
{
  struct header_bits
  {
    unsigned section_syntax_indicator :
1;
    unsigned no_name                  :
1;
    unsigned reserved1                :
2;
    unsigned section_length_11_8      :
4;
    unsigned section_length_7_0       :
8;
    unsigned transport_stream_id_15_8 :
8;
    unsigned transport_stream_id_7_0  :
8;
    unsigned reserved2                :
2;
    unsigned version_number           :
5;
    unsigned current_next_indicator   :
1;
  };
  union
  {
    struct header_bits bits;
    char byte[5];
  } header;

  struct table_bits
  {
    unsigned program_number_15_8
:  8;

    unsigned program_number_7_0
:  8;
    unsigned reserved
:  3;
    unsigned
network_or_program_map_PID_12_8 :  5;
    unsigned
network_or_program_map_PID_7_0  :  8;
  };

  union
  {
    struct table_bits bits;
    char byte[6];
  } table;

  int    pointer_field,
         table_id,
         section_number,
         last_section_number,
         stuffing_bytes;
  double CRC_32;
  char str[80];
  register int i,
               n_bytes = 0;/* counter of
PA_section bytes */
  int j, k;    /*Auxiliary*/
  PAT_ptr ptr; /*Auxiliary: free of PAT
and new_PAT; insertion of new elements
                 in table*/
  fprintf(out,"|PROGRAM ASSOCIATION
SECTION******************\n");
  if( unit_start )
  {
    /*Read pointer_field.*/
    pointer_field = getc(in);
    if( feof(in) )
    {
      Warn(out,"Premature EOF in
pointer_field");
      exit(1);
    }
    n_bytes = 1;
    if(pointer_field + *bytes_read + 1
> 188)
    {
      i = 188 - *bytes_read -n_bytes;
      sprintf(str,"Invalid length: %d
bytes ignored",i);
      Warn(out, str);
      fseek(in, i, SEEK_CUR); /*Ignores
rest of packet*/
      *bytes_read = 188;
      return;
    }
```

```
    if(pointer_field == 0) PA_section.p
= 0; /*Byte 1: PA_section.p = 1*/
  }
  while(*bytes_read + n_bytes < 188)
/*Verifies end of packet.*/
  {
    /*Read a section byte to a
buffer.*/
    PA_section.buffer[PA_section.p+1] =
(char)getc(in);
    if( feof(in) )
    {
      Warn(out,"Premature EOF in
program_association_section.");
      exit(1);
    }
    n_bytes++;
    PA_section.p++;
    PA_section.bytes_left--;
    if(PA_section.p == 8) /*Header is
in buffer.*/
    {
      /* table_id */
      table_id = PA_section.buffer[1];
      fprintf(out,"| table_id = %d\n",
table_id);
      /*Read header.*/
      for(i=0; i<5; i++)
header.byte[i] =
PA_section.buffer[i+2];
      /* section_syntax_indicator */
      fprintf(out,"|
section_syntax_indicator =
%d\n",header.bits.section_syntax_indica
tor);
      if(  table_id != 0
||

!header.bits.section_syntax_indicator
||
           header.bits.no_name
)
      {
       i = 188 - *bytes_read -n_bytes;
       sprintf(str,"Invalid PA_section
header: %d bytes ignored",i);
       Warn(out, str);
       fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
       *bytes_read = 188;
       return;
      }
      if(detail < 1)
       fprintf(out,"| reserved = %d\n",
header.bits.reserved1);
      /*Read section length.*/
      /*Initialization of
'PA_section.bytes_left'.*/
      PA_section.bytes_left =
header.bits.section_length_11_8 * 256

+header.bits.section_length_7_0;
      fprintf(out,"| section_length =
%d\n", PA_section.bytes_left);
      if(PA_section.bytes_left > 1021)
/*See buffer size*/
      {
       i = 188 - *bytes_read -n_bytes;
       sprintf(str,"Invalid length: %d
bytes ignored",i);
       Warn(out, str);
       fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
       *bytes_read = 188;
       return;
      }
      PA_section.bytes_left -= 5;
      /* transport_stream_id */ /*JUST
ONE TS ASSUMED !*/
      fprintf(out,"|
transport_stream_id = %d\n",

header.bits.transport_stream_id_15_8 *
256

+header.bits.transport_stream_id_7_0
);
      if(detail < 1)
       fprintf(out,"| reserved = %d\n",
header.bits.reserved2);
      /* version_number */
      fprintf(out,"| version_number =
%d\n", header.bits.version_number);

vnova=(int)header.bits.version_number;
      if(vnova!=vant)
                              {
                               novo=0;
                               vant=vnova;
                               }
      i =
(int)header.bits.version_number -
PAT_version;
      if(i != 0) /* Table version
management */
      {
       if(i != 1 && i != -31)
       {
        i = 188 - *bytes_read -
n_bytes;
        sprintf(str,"version_number
not in sequence: %d bytes ignored",i);
        Warn(out, str);
        fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
        *bytes_read = 188;
        return;
       }
       if(PAT != new_PAT)
       {
        Warn(out,"Previous version not
used");
        /*free new_PAT*/
        ptr = new_PAT;
        while(ptr != NULL)
        {
         ptr = ptr->next;
         free(new_PAT);
         new_PAT = ptr;
        }
       }
       if(
(new_PAT=(PAT_ptr)malloc(sizeof(struct
PA_table))) == NULL )

{puts("TS_PROGRAM_ASSOCIATION_SECTION()
: malloc error"); exit(1);}
       new_PAT->next=NULL;
       PAT_version =
header.bits.version_number;
      }
      /* current_next_indicator */
      fprintf(out,"|
current_next_indicator =
%d\n",header.bits.current_next_indicato
r);
```

```
    if
(!header.bits.current_next_indicator)
agora=0;

if(header.bits.current_next_indicator)
/* Current table */
      {
       agora=1;
       if(PAT != new_PAT)
       {
         /*free PAT*/
         ptr = PAT;
         while(ptr != NULL)
         {
           ptr = ptr->next;
           free(PAT);
           PAT = ptr;
         }
         PAT = new_PAT; /* New table is
the current one */
       }
      }
      else if(PAT == new_PAT)  PAT =
NULL; /* Desactivate current table */
      /* section_number */
      section_number =
PA_section.buffer[7];
      fprintf(out,"| section_number =
%d\n", section_number);
      /* last_section_number */
      last_section_number =
PA_section.buffer[8];
      fprintf(out,"|
last_section_number = %d\n",
last_section_number);
      /* copia para a nova PAT 8 bytes
*/

putc(PA_section.buffer[0],trama);

putc(PA_section.buffer[1],trama);

putc(PA_section.buffer[2],trama);
       /* length of packet-section left
*/
         putc( 0x0D,trama);

putc(PA_section.buffer[4],trama);

putc(PA_section.buffer[5],trama);
       /* Muda o version number e o
current_next_indicator */

header.bits.version_number=vactual;

header.bits.current_next_indicator=1;
     if ((!novo && agora ) ||
(option!=0))
         {
        printf("\n----------------------
---//-------------------------");
          printf("\n|  New PAT
");
          if (option!=0) printf("\n|
The program chosen was: %c ",option);
          if (option!=0) prog=option-
48;
          if (vactual>31) vactual=0;
          printf("       VERSion    %d
" ,header.bits.version_number);
          if
(option!=0)header.bits.current_next_ind
icator=1;
          option=0;


header.bits.version_number=vactual;
          vactual=vactual+1;

PA_section.buffer[6]=header.byte[4];
          printf(" \n|
Nova Versao    %d  \n",vactual);
          }

PA_section.buffer[6]=header.byte[4];

putc(PA_section.buffer[6],trama);

putc(PA_section.buffer[7],trama);

putc(PA_section.buffer[8],trama);
          bytes_lidos=bytes_lidos+9;
  }
    else /*Verifies if table element (4
bytes) is in buffer.*/
    if(PA_section.p > 8 &&
!(PA_section.p % 4))
    {
      if(PA_section.bytes_left) /*Read
one element.*/
       {
        for(i=0; i<4; i++)
table.byte[i]=PA_section.buffer[PA_sect
ion.p-3+i];
        /* program_number */
        i=table.bits.program_number_15_8
*256 + table.bits.program_number_7_0;
        fprintf(out,"|\n|
program_number = %d\n", i);
       tcsetattr(STDIN_FILENO,
TCSADRAIN, &old_term);

        if (!novo && agora)
{printf("|\n");
                    printf("| Program
present %d   ",i);
                    countprog= (int) i;
                   }
        if(detail < 1)
          fprintf(out,"| reserved =
%d\n", table.bits.reserved);
        /* network_PID or
program_map_PID */
        if(i)
          fprintf(out,"| program_map_PID
= ");
        else
          fprintf(out,"| network_PID =
");
        j =
table.bits.network_or_program_map_PID_1
2_8 * 256

+table.bits.network_or_program_map_PID_
7_0;
        fprintf(out,"0x%04X\n", j);
        if (!novo && agora) {
                    printf ("PID =
0x%04X \n",j);

PIDPROG[countprog]=j;
                    }
        /* Verifies if PID is in table
*/
        ptr = PA_PID(new_PAT, j, &k);
        if(ptr == NULL)
```

```
        {
          /*New linked list element*/
          if(
(ptr=(PAT_ptr)malloc(sizeof(struct
PA_table))) == NULL )

{puts("TS_PROGRAM_ASSOCIATION_SECTION()
: malloc error"); exit(1);}

          /*Insert new element in table.
At the list start, just after the
head.*/
          ptr->next = new_PAT->next;
          new_PAT->next = ptr;
        }
        ptr->program_number = i;
        ptr->PID = j;
      }
      else /*Section END.*/
      {
        /* Read CRC_32.*/
        CRC_32 =
(double)PA_section.buffer[PA_section.p-
3] * 16777216 /* 2^24 */

        +(double)PA_section.buffer[PA_se
ction.p-2] * 65536     /* 2^16 */

        +(double)PA_section.buffer[PA_se
ction.p-1] * 256       /* 2^8  */

        +(double)PA_section.buffer[PA_se
ction.p];
        fprintf(out,"|\n| CRC_32 =
%g\n", CRC_32);

        if (!novo && agora) {
                printf("|\n|   Choose
the program " );
               ii=0;
        tcsetattr(STDIN_FILENO,
TCSAFLUSH, &new_term);
              while((read(STDIN_FILENO,
&crr, 1)==0) && ii<150000)
                ii++;
       tcsetattr(STDIN_FILENO,
TCSADRAIN, &old_term);
                if (crr!=0)
                   prog=crr-48;
                printf("\n|  -->
Programa, %d ",prog);
                printf("  PID,%d
\n",PIDPROG[prog]);
                printf("--------------
----------//-----------------------");
      tcsetattr(STDIN_FILENO,
TCSADRAIN, &new_term);

                NPAT=1;
                NovaPMT=1;
                novo=1;
                agora=0;
                  }
       if (NPAT)
              {
                /* copy of program */
                for (i=0;i<4;i++)
putc(PA_section.buffer[4*prog+5+i],tram
a);
                /* CRC */
                for(i=PA_section.p-
3;i<=PA_section.p;i++) putc(0,trama);

bytes_lidos=bytes_lidos+8;

for(i=bytes_lidos;i<=188;i++)
putc(0xFF,trama);

                }
        /*Stuffing bytes*/
        if((n_bytes - 1) <
(pointer_field +1))
        {
          /*One section ends and another
one starts.*/

          fprintf(out,"|\n| %d bytes
left - ",pointer_field +1 - n_bytes);
          for(stuffing_bytes=0, i=0; i <
pointer_field +1 - n_bytes; i++)
            if(getc(in) == 0xFF)
stuffing_bytes++;
          n_bytes = pointer_field + 1;
          PA_section.p = 0; /*Byte 1:
PA_section.p = 1*/
        }
        else  /*The section ended and
rest of the packet is garbage.*/
        {
          for(stuffing_bytes=0, i=0; i <
188 - n_bytes - *bytes_read; i++)
            if(getc(in) == 0xFF)
stuffing_bytes++;

          n_bytes = 188 - *bytes_read;
        }
      }
    }
  }
  fclose(trama);
 fim=1;
*bytes_read = 188;
}


void
TS_CA_SECTION(int unit_start,int
*bytes_read,FILE *in,FILE *out)
/*************************************
 NOT TESTED !!!         (adapted from
PA_section)

 pointer_field = 1 optional byte
depending on 'unit_start'.

 table_id = 1 byte

 section_syntax_indicator = 1 bit \
 no_name = 1 bit                   \
 reserved1 = 2 bits                 \
 section_length = 12 bits            >
5 bytes.
 reserved2 = 18 bits                /
 version_number = 5 bits           /
 current_next_indicator = 1 bit   /

 section_number = 1 byte;

 last_section_number = 1 byte;

 N decritores of variable length;

 CRC_32 = 4 bytes;
```

```
 Uses a buffer for the section, defined
in a global struct. This way one can
 reconstruct the section from packet to
packet and do the final CRC.
 It's assumed that in a section there's
at least 1 descriptor.

 Linked list with CA_table still not
implemented !


**************************************/
{
  struct header_bits
  {
    unsigned section_syntax_indicator :
1;
    unsigned no_name                  :
1;
    unsigned reserved1                :
2;
    unsigned section_length_11_8      :
4;
    unsigned section_length_7_0       :
8;
    unsigned reserved2                :
8;
    unsigned reserved3                :
8;
    unsigned reserved4                :
2;
    unsigned version_number           :
5;
    unsigned current_next_indicator   :
1;
  };

  union
  {
    struct header_bits bits;
    char byte[5];
  } header;


  int    pointer_field,
         table_id,
         section_number,
         last_section_number,
         stuffing_bytes;
  double CRC_32;

  char str[80];
  register int i,
               n_bytes = 0; /* counter
of CA_section bytes */
  static int   needed_bytes;/*Needed
bytes to read descriptor info from
buffer.*/


  fprintf(out,"|CONDITIONAL ACCESS
SECTION*******************\n");

  if( unit_start )
  {
    /*Read pointer_field.*/
    pointer_field = getc(in);
    if( feof(in) )
    {
      Warn(out,"Premature EOF in
pointer_field");
      exit(1);
    }

    n_bytes = 1;
    if(pointer_field + *bytes_read + 1
> 188)
    {
      i = 188 - *bytes_read -n_bytes;
      sprintf(str,"Invalid length: %d
bytes ignored",i);
      Warn(out, str);
      fseek(in, i, SEEK_CUR); /*Ignores
rest of packet*/
      *bytes_read = 188;
      return;
    }

    if(pointer_field == 0) CA_section.p
= 0; /*Byte 1: CA_section.p = 1*/
    /*! Maybe should verify if previous
section was completely read.*/
  }

  while(*bytes_read + n_bytes < 188)
/*Verifies end of packet.*/
  {
    /*Read a section byte to a
buffer.*/
    CA_section.buffer[CA_section.p+1] =
(char)getc(in);
    if( feof(in) )
    {
      Warn(out,"Premature EOF in
conditional_access_section");
      exit(1);
    }
    n_bytes++;
    CA_section.p++;
    CA_section.bytes_left--;
    needed_bytes--;

    /*Section and first descriptor
header is in buffer.*/
    if(CA_section.p == 10)
    {
      /* table_id */
      table_id = CA_section.buffer[1];
      fprintf(out,"| table_id = %d\n",
table_id);

      /*Read section header.*/
      for(i=0; i<5; i++)
header.byte[i] =
CA_section.buffer[i+2];

      /* section_syntax_indicator */
      fprintf(out,"|
section_syntax_indicator =
%d\n",header.bits.section_syntax_indica
tor);

      if(  table_id != 1
||

!header.bits.section_syntax_indicator
||
          header.bits.no_name
)
      {
        i = 188 - *bytes_read -n_bytes;
        sprintf(str,"Invalid CA_section
header: %d bytes ignored",i);
        Warn(out, str);
        fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
        *bytes_read = 188;
```

```
        return;
      }

      if(detail < 1)
        fprintf(out,"| reserved = %d\n",
header.bits.reserved1);

      /*Read section length.*/
      /*Initialization of
'CA_section.bytes_left'.*/
      CA_section.bytes_left =
header.bits.section_length_11_8 * 256

+header.bits.section_length_7_0;
      fprintf(out,"| section_length =
%d\n", CA_section.bytes_left);
      if(CA_section.bytes_left > 1021)
/*See buffer size*/
      {
        i = 188 - *bytes_read -n_bytes;
        sprintf(str,"Invalid length: %d
bytes ignored",i);
        Warn(out, str);
        fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
        *bytes_read = 188;
        return;
      }
      CA_section.bytes_left -= 7;

      if(detail < 1)
        fprintf(out,"| reserved = %d %d
%d\n", header.bits.reserved2,

header.bits.reserved3,

header.bits.reserved4 );

      /* version_number */
      fprintf(out,"| version_number =
%d\n", header.bits.version_number);

      /* current_next_indicator */
      fprintf(out,"|
current_next_indicator =
%d\n",header.bits.current_next_indicato
r);

      /* section_number */
      section_number =
CA_section.buffer[7];
      fprintf(out,"| section_number =
%d\n", section_number);

      /* last_section_number */
      last_section_number =
CA_section.buffer[8];
      fprintf(out,"|
last_section_number = %d\n",
last_section_number);
      /*First descriptor header.*/

      /*Needed bytes in buffer are the
descriptor size and, if there is more,
        next's header (giving the
descriptor size).*/
      if(CA_section.buffer[10] <
(CA_section.bytes_left - 4))
        /* Size of descriptor plus
next's header */
        needed_bytes =
CA_section.buffer[10] + 2;
      else
        /* Size of descriptor plus CRC
*/
        needed_bytes =
CA_section.buffer[10] + 4;

      if(needed_bytes >
CA_section.bytes_left)
      {
        fprintf(out,"|\n| descriptor_tag
= %d\n", CA_section.buffer[9]);
        fprintf(out,"| descriptor_length
= %d\n", CA_section.buffer[10]);
        i = 188 - *bytes_read -n_bytes;
        sprintf(str,"Invalid descriptor
length: %d bytes ignored",i);
        Warn(out, str);
        fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
        *bytes_read = 188;
        return;
      }

      /*Start of last descriptor kept
in buffer.*/
      CA_section.descriptor_start = 9;
    }
    else /*Checks if there's enough
info in buffer to read descriptor and
what follows.*/
    if(CA_section.p > 8 &&
!needed_bytes)
    {
      /*Read descriptor.*/
      DESCRIPTOR(&CA_section, out);

      if(CA_section.bytes_left)
/*There's more descriptors.*/
      {
        i = CA_section.p;

        /*Next descriptor header.*/

        /*Needed bytes in buffer are the
descriptor size and, if there is more,
          next's header (giving the
descriptor size).*/
        if(CA_section.buffer[i] <
(CA_section.bytes_left - 4))
          /* Size of descriptor plus
next's header */
          needed_bytes =
CA_section.buffer[i] + 2;
        else
          /* Size of descriptor plus CRC
*/
          needed_bytes =
CA_section.buffer[i] + 4;

        if(needed_bytes >
CA_section.bytes_left)
        {
          fprintf(out,"|\n|
descriptor_tag = %d\n",
CA_section.buffer[i-1]);
          fprintf(out,"|
descriptor_length = %d\n",
CA_section.buffer[i]);
          sprintf(str,"Invalid
descriptor length: %d bytes
ignored",188-*bytes_read-n_bytes);
          Warn(out, str);
```

```
          fseek(in, 188 - *bytes_read -
n_bytes, SEEK_CUR); /*Ignores rest of
packet*/
          *bytes_read = 188;
          return;
        }

        /*Start of last descriptor kept
in buffer.*/
        CA_section.descriptor_start = i-
1;
      }
      else /*Section END.*/
      {
        /* Read CRC_32.*/
        CRC_32 =
(double)CA_section.buffer[CA_section.p-
3] * 16777216 /* 2^24 */

        +(double)CA_section.buffer[CA_se
ction.p-2] * 65536     /* 2^16 */

        +(double)CA_section.buffer[CA_se
ction.p-1] * 256       /* 2^8  */

        +(double)CA_section.buffer[CA_se
ction.p];
        fprintf(out,"|\n| CRC_32 =
%g\n", CRC_32);

        /*! Here one would VERIFY THE
CRC over the buffer*/

        /*Stuffing bytes*/
        if((n_bytes - 1) <
(pointer_field +1))
        {
          /*One section ends and another
one starts.*/

          fprintf(out,"|\n| %d bytes
left - ",pointer_field +1 - n_bytes);
          for(stuffing_bytes=0, i=0; i <
pointer_field +1 - n_bytes; i++)
            if(getc(in) == 0xFF)
stuffing_bytes++;

          n_bytes = pointer_field + 1;
          CA_section.p = 0; /*Byte 1:
CA_section.p = 1*/
        }
        else  /*The section ended and
rest of the packet is garbage.*/
        {
          for(stuffing_bytes=0, i=0; i <
188 - n_bytes - *bytes_read; i++)
            if(getc(in) == 0xFF)
stuffing_bytes++;

          n_bytes = 188 - *bytes_read;
        }
      }
    }
  }

  *bytes_read = 188;
}


void
TS_PROGRAM_MAP_SECTION(int
unit_start,int *bytes_read,FILE
*in,FILE *out)
```

```
/*************************************
 pointer_field = 1 optional byte
depending on 'unit_start'.

 table_id = 1 byte

 section_syntax_indicator = 1 bit \
 no_name = 1 bit                   \
 reserved1 = 2 bits                 \
 section_length = 12 bits            \
 program_number = 16 bits             \
 reserved2 = 2 bits
\
 version_number = 5 bits
> 11 bytes.
 current_next_indicator = 1 bit
/
 section_number = 8 bits              /
 last_section_number = 8 bits        /
 reserved3 = 3 bits                 /
 PCR_PID = 13 bits                 /
 reserved4 = 4 bits               /
 program_info_length = 12 bits   /

 N descriptors of varying length;

 stream_type = 8 bits      \
 reserved = 3 bits          \
 elementary_PID = 13 bits    > N * 5
bytes.
 reserved = 4 bits          /
 ES_info_length = 12 bits /

 N descriptors of varying length;

 CRC_32 = 4 bytes;

 Uses a buffer for the section, defined
in a global struct. This way one can
 reconstruct the section from packet to
packet and do the final CRC.

 Depending on table_id private data can
be transported.
 TO DO ! (see table 2-35, pg55 - Stream
Type Assignments: 0x05)

*************************************/
{
  struct header_bits
  {
    unsigned section_syntax_indicator :
1;
    unsigned no_name                  :
1;
    unsigned reserved1                :
2;
    unsigned section_length_11_8      :
4;
    unsigned section_length_7_0       :
8;
    unsigned program_number_15_8      :
8;
    unsigned program_number_7_0       :
8;
    unsigned reserved2                :
2;
    unsigned version_number           :
5;
    unsigned current_next_indicator   :
1;
    unsigned section_number           :
8;
```

```
    unsigned last_section_number      :
8;
    unsigned reserved3                :
3;
    unsigned PCR_PID_12_8             :
5;
    unsigned PCR_PID_7_0              :
8;
    unsigned reserved4                :
4;
    unsigned program_info_length_11_8 :
4;
    unsigned program_info_length_7_0  :
8;
  };

  union
  {
    struct header_bits bits;
    char byte[11];
  } header;

  struct ES_bits
  {
    unsigned  stream_type         :  8;
    unsigned reserved1            :  3;
    unsigned elementary_PID_12_8  :  5;
    unsigned elementary_PID_7_0   :  8;
    unsigned reserved2            :  4;
    unsigned ES_info_length_11_8  :  4;
    unsigned ES_info_length_7_0   :  8;
  };

  union
  {
    struct ES_bits bits;
    char byte[5];
  } ES;

  int         pointer_field,
             table_id,
             stuffing_bytes;
  static int field,     /* control of
aplicable syntax */
             program_info_length,
             ES_info_length;
  double      CRC_32;

  /* struct {} PM_section;
     PMT_ptr new_PMT, PMT;
     int PMT_version;       GLOBAL VAR.!
*/

  char str[80];
  register int i,
               n_bytes = 0; /* counter
of PM_section bytes */
  static int
descriptor_needed_bytes,/*Needed bytes
to read descriptor info

from buffer.*/
               info_start; /*Buffer
pointer.*/
  PMT_ptr       ptr;  /*Auxiliary: free
of PMT and new_PMT; insertion of new
elements
                in table*/
  fprintf(out,"|PROGRAM MAP
SECTION*************************\n");
  if( unit_start )
  {
    /*Read pointer_field.*/
    pointer_field = getc(in);
    if( feof(in) )
    {
      Warn(out,"Premature EOF in
pointer_field");
      exit(1);
    }
    n_bytes = 1;
    fprintf(out,"| pointer_field = %d
bytes\n", pointer_field);
    if(pointer_field + *bytes_read + 1
> 188)
    {
      i = 188 - *bytes_read -n_bytes;
      sprintf(str,"Invalid length: %d
bytes ignored",i);
      Warn(out, str);
      fseek(in, i, SEEK_CUR); /*Ignores
rest of packet*/
      *bytes_read = 188;
      return;
    }
    if(pointer_field == 0) PM_section.p
= 0; /*Byte 1: PM_section.p = 1*/
  }

  while(*bytes_read + n_bytes < 188)
/*Verifies end of packet.*/
  {
    /*Read a section byte to a
buffer.*/
    PM_section.buffer[PM_section.p+1] =
(char)getc(in);
    if( feof(in) )
    {
      Warn(out,"Premature EOF in
program_map_section");
      exit(1);
    }
    n_bytes++;
    PM_section.p++;
    PM_section.bytes_left--;
    descriptor_needed_bytes--;
    if(PM_section.p == 1) /* table_id
is in buffer */
    {
      /* table_id */
      table_id = PM_section.buffer[1];
      fprintf(out,"| table_id
= %d", table_id);
      if(table_id != 2)
      {
        if(table_id == 0xFF)
          fprintf(out," ->
forbidden\n");
        if(table_id >= 0x40 && table_id
<= 0xFE)
          fprintf(out," -> User
private\n");     /*
TS_PRIVATE_SECTION(); !!! */
        if(table_id >= 0x03 && table_id
<= 0x3F)
          fprintf(out," -> ISO/IEC 13818
reserved\n");
        i = 188 - *bytes_read -n_bytes;
        fprintf(out,"| (%d bytes
ignored)\n",i);
        fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
        *bytes_read = 188;
        return;
      }
    }
```

```
    else
    if(PM_section.p == 12) /*Header is
in buffer.*/
    {
      /*Read header.*/
      for(i=0; i<11; i++)
header.byte[i] =
PM_section.buffer[i+2];

      /* section_syntax_indicator */
      fprintf(out,"\n|
section_syntax_indicator =
%d\n",header.bits.section_syntax_indica
tor);
      if(
!header.bits.section_syntax_indicator
||
           header.bits.no_name
)
      {
       i = 188 - *bytes_read -n_bytes;
       sprintf(str,"Invalid PM_section
header: %d bytes ignored",i);
       Warn(out, str);
       fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
       *bytes_read = 188;
       return;
      }

      if(detail < 1)
       fprintf(out,"| reserved = %d\n",
header.bits.reserved1);

      /*Read section length.*/
      /*Initialization of
'PM_section.bytes_left'.*/
      PM_section.bytes_left =
header.bits.section_length_11_8 * 256

+header.bits.section_length_7_0;
      fprintf(out,"| section_length =
%d\n", PM_section.bytes_left);
      if(PM_section.bytes_left > 1021)
/*See buffer size*/
      {
       i = 188 - *bytes_read -n_bytes;
       sprintf(str,"Invalid length: %d
bytes ignored",i);
       Warn(out, str);
       fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
       *bytes_read = 188;
       return;
      }
      PM_section.bytes_left -= 9;

      /* program_number */
      fprintf(out,"| program_number xxv
= %d\n",

header.bits.program_number_15_8 * 256

+header.bits.program_number_7_0
);
    auxnum=
header.bits.program_number_15_8 * 256

+header.bits.program_number_7_0 ;
    if (prog==auxnum)
      {
       PMTPIDcont=1;
       programaPMT=1;
      }

     if(detail < 1)
       fprintf(out,"| reserved = %d\n",
header.bits.reserved2);

      /* version_number */
      fprintf(out,"| version_number
= %d\n", header.bits.version_number);
      i =
(int)header.bits.version_number -
PMT_version;
      if(i != 0) /* Table version
management */
      {
       if(i != 1 && i != -31)
       {
         i = 188 - *bytes_read -
n_bytes;
         sprintf(str,"version_number
not in sequence: %d bytes ignored",i);
         Warn(out, str);
         fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
         *bytes_read = 188;
         return;
       }

       if(PMT != new_PMT)
       {
         Warn(out,"Previous version not
used");
         /*free new_PMT*/
         ptr = new_PMT;
         while(ptr != NULL)
         {
           ptr = ptr->next;
           free(new_PMT);
           new_PMT = ptr;
         }
       }

       if(
(new_PMT=(PMT_ptr)malloc(sizeof(struct
PM_table))) == NULL )

{puts("TS_PROGRAM_MAP_SECTION(): malloc
error"); exit(1);}
       new_PMT->next=NULL;

       PMT_version =
header.bits.version_number;
      }

     /* current_next_indicator */
     fprintf(out,"|
current_next_indicator =
%d\n",header.bits.current_next_indicato
r);

if(header.bits.current_next_indicator)
/* Current table */
      {
       if(PMT != new_PMT)
       {
         /*free PMT*/
         ptr = PMT;
         while(ptr != NULL)
         {
           ptr = ptr->next;
           free(PMT);
           PMT = ptr;
         }
```

```
        PMT = new_PMT; /* New table is
the current one */
       }
      }
      else if(PMT == new_PMT)  PMT =
NULL; /* Desactivate current table */

      /* section_number */
      fprintf(out,"| section_number
= %d\n", header.bits.section_number);
      if(header.bits.section_number)
       Warn(out,"Invalid
section_number");

      /* last_section_number */
      fprintf(out,"|
last_section_number    =
%d\n",header.bits.last_section_number);

if(header.bits.last_section_number)
       Warn(out,"Invalid
last_section_number");

      if(detail < 1)
       fprintf(out,"| reserved = %d\n",
header.bits.reserved3);

      /* PCR_PID */
      fprintf(out,"| PCR_PID =
0x%04X\n", header.bits.PCR_PID_12_8 *
256

+header.bits.PCR_PID_7_0       );

      if(detail < 1)
       fprintf(out,"| reserved = %d\n",
header.bits.reserved4);

      /* program_info_length */
      program_info_length =
header.bits.program_info_length_11_8 *
256

+header.bits.program_info_length_7_0;
      fprintf(out,"|
program_info_length = %d\n",
program_info_length);
      if(program_info_length+4 >
PM_section.bytes_left)
      {
       i = 188 - *bytes_read -n_bytes;
       sprintf(str,"Invalid length: %d
bytes ignored",i);
       Warn(out, str);
       fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
       *bytes_read = 188;
       return;
      }

      if(program_info_length)  field =
1; /* Program descriptors */
      else
      {
       field = 2; /* ES info. */
       info_start = PM_section.p;
      }
    }
    else
    if(PM_section.p > 12) /* flag is
defined */
    {
      switch(field)
      {
       case 1: /* Program descriptors
*/

       if(PM_section.p == 14) /*First
descriptor header.*/
        {
         program_info_length -= 2;

         /*Needed bytes in buffer are
the descriptor size and, if there is
more,
          next's header (giving the
descriptor size).*/
         descriptor_needed_bytes =
PM_section.buffer[14];
         if(PM_section.buffer[14] <
program_info_length)
          /* Size of descriptor plus
next's header */
          descriptor_needed_bytes +=
2;

         if(descriptor_needed_bytes >
program_info_length)
         {
          fprintf(out,"|\n|
descriptor_tag = %d\n",
PM_section.buffer[13]);
          fprintf(out,"|
descriptor_length = %d\n",
PM_section.buffer[14]);
          i = 188 - *bytes_read -
n_bytes;
          sprintf(str,"Invalid
descriptor length: %d bytes
ignored",i);
          Warn(out, str);
          fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
          *bytes_read = 188;
          return;
         }

         /*Start of last descriptor
kept in buffer.*/
         PM_section.descriptor_start =
13;
        }
        else
        if(PM_section.p > 14 &&
!descriptor_needed_bytes)
        {
         program_info_length-=
PM_section.buffer[PM_section.descriptor
_start+1];

         /*Read descriptor.*/
         DESCRIPTOR(&PM_section, out);

         if(program_info_length)
/*There's more descriptors.*/
         {
          program_info_length -= 2;

          i = PM_section.p;

          /*Next descriptor header.*/

          /*Needed bytes in buffer are
the descriptor size and, if there is
more,
```

```
            next's header (giving the
descriptor size).*/
            descriptor_needed_bytes =
PM_section.buffer[i];
            if(PM_section.buffer[i] <
program_info_length)
              /* Size of descriptor plus
next's header */
              descriptor_needed_bytes +=
2;

            if(descriptor_needed_bytes >
program_info_length)
            {
              fprintf(out,"|\n|
descriptor_tag = %d\n",
PM_section.buffer[i-1]);
              fprintf(out,"|
descriptor_length = %d\n",
PM_section.buffer[i]);
              i = 188 - *bytes_read -
n_bytes;
              sprintf(str,"Invalid
descriptor length: %d bytes
ignored",i);
              Warn(out, str);
              fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
              *bytes_read = 188;
              return;
            }

            /*Start of last descriptor
kept in buffer.*/
            PM_section.descriptor_start
= i-1;
          }
          else
          {
            field = 2; /* ES info. */
            info_start = PM_section.p;
          }
        }
        break;

       case 2: /* ES info. */

        if(PM_section.bytes_left < 4)
field = 4; /* CRC and end */
        else
        if(PM_section.p == (info_start +
5)) /* ES header in buffer */
        {
          /*Read ES header.*/
          for(i=0; i<5; i++)
ES.byte[i]=PM_section.buffer[PM_section
.p-4+i];

          /* stream_type */
          fprintf(out,"|\n| stream_type
= %d   -> ", ES.bits.stream_type);
          switch(ES.bits.stream_type)
          {
           case 0x00:
                    fprintf(out,"ITU-T
| ISO/IEC Reserved\n");
                    break;
           case 0x01:

fprintf(out,"ISO/IEC 11172 Video\n");
                    break;
           case 0x02:
                    fprintf(out,"ITU-T
Rec.H262 | ISO/IEC 13818-2 Video\n");
                    break;
           case 0x03:

fprintf(out,"ISO/IEC 11172 Audio\n");
                    break;
           case 0x04:

fprintf(out,"ISO/IEC 13818-3 Audio\n");
                    break;
           case 0x05:
                    fprintf(out,"ITU-T
Rec.H222.0|ISO/IEC 13818-1
private_section\n");
                    break;
           case 0x06:
                    fprintf(out,"ITU-T
Rec.H222.0|ISO/IEC 13818-1 PES packets
containing private data\n");
                    break;
           case 0x07:

fprintf(out,"ISO/IEC 13522 MHEG\n");
                    break;
           case 0x08:
                    fprintf(out,"ITU-T
Rec.H222.0|ISO/IEC 13818-1 DSM CC\n");
                    break;
           case 0x09:
                    fprintf(out,"ITU-T
Rec.H222.0|ISO/IEC 13818-1/11172-1
Auxiliary\n");
                    break;
           default  :

if(ES.bits.stream_type>=0x0A &&
ES.bits.stream_type<=0x7F)
                      fprintf(out,"ITU-
T Rec.H222.0|ISO/IEC 13818-1
Reserved\n");
                    else

if(ES.bits.stream_type>=0x80 &&
ES.bits.stream_type<=0xFF)
                      fprintf(out,"User
Private\n");
          }


          /* elementary_PID */
          i =
ES.bits.elementary_PID_12_8 * 256

+ES.bits.elementary_PID_7_0;
          fprintf(out,"| elementary_PID
= 0x%04X\n", i);
          if ( programaPMT)
        {
          PIDPMT[PMTPIDcont]=i;
          PMTPIDcont++;
          progPIDPMT=1;
        }
          /* Verifies if PID is in table
*/
          ptr = Elementary_PID(new_PMT,
i);

          if(ptr == NULL)

          {
            /*New linked list element*/
```

```
          if(
(ptr=(PMT_ptr)malloc(sizeof(struct
PM_table))) == NULL )

{puts("TS_PROGRAM_MAP_SECTION(): malloc
error"); exit(1);}

          /*Insert new element in
table. At the list start, just after
the head.*/
          ptr->next = new_PMT->next;
          new_PMT->next = ptr;

          ptr->elementary_PID = i;
          ptr->continuity_counter = -
1;/*Impossible value. See
TS_TRANSPORT_PACKET()*/
          ptr->old_stream_removed =
NO;
          ptr->field = 0;
/*PES_PACKET() ignored if there's no
unit_start*/
        }

        /*ptr->stream_type =
ES.bits.stream_type; NOT USED!*/
/*IMPROVE ROBUSTNESS !*/
        if(ES_info_length+4 >
PM_section.bytes_left)
        {
          i = 188 - *bytes_read -
n_bytes;
          sprintf(str,"Invalid length:
%d bytes ignored",i);
          Warn(out, str);
          fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
          *bytes_read = 188;
          return;
        }

        if(ES_info_length)  field = 3;
/* ES descriptors */

        info_start = PM_section.p;
      }
      break;

    case 3: /* ES descriptors */

      if(PM_section.p == info_start+2)
/*First descriptor header.*/
      {
        ES_info_length -= 2;

        /*Needed bytes in buffer are
the descriptor size and, if there is
more,
          next's header (giving the
descriptor size).*/
        descriptor_needed_bytes =
PM_section.buffer[PM_section.p];

if(PM_section.buffer[PM_section.p] <
ES_info_length)
          /* Size of descriptor plus
next's header */
          descriptor_needed_bytes +=
2;

        if(descriptor_needed_bytes >
ES_info_length)
        {
          fprintf(out,"|\n|
descriptor_tag = %d\n",
PM_section.buffer[13]);
          fprintf(out,"|
descriptor_length = %d\n",
PM_section.buffer[14]);
          i = 188 - *bytes_read -
n_bytes;
          sprintf(str,"Invalid
descriptor length: %d bytes
ignored",i);
          Warn(out, str);
          fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
          *bytes_read = 188;
          return;
        }

        /*Start of last descriptor
kept in buffer.*/
        PM_section.descriptor_start =
PM_section.p - 1;
      }
      else
      if(PM_section.p>info_start+2 &&
!descriptor_needed_bytes)
      {
        ES_info_length-=
PM_section.buffer[PM_section.descriptor
_start+1];

        /*Read descriptor.*/
        DESCRIPTOR(&PM_section, out);

        if(ES_info_length) /*There's
more descriptors.*/
        {
          ES_info_length -= 2;

          i = PM_section.p;

          /*Next descriptor header.*/

          /*Needed bytes in buffer are
the descriptor size and, if there is
more,
            next's header (giving the
descriptor size).*/
          descriptor_needed_bytes =
PM_section.buffer[i];
          if(PM_section.buffer[i] <
ES_info_length)
            /* Size of descriptor plus
next's header */
            descriptor_needed_bytes +=
2;

          if(descriptor_needed_bytes >
ES_info_length)
          {
            fprintf(out,"|\n|
descriptor_tag = %d\n",
PM_section.buffer[i-1]);
            fprintf(out,"|
descriptor_length = %d\n",
PM_section.buffer[i]);
            sprintf(str,"Invalid
descriptor length: %d bytes
ignored",188 - *bytes_read -n_bytes);
            Warn(out, str);
            fseek(in, 188 -
*bytes_read -n_bytes, SEEK_CUR);
/*Ignores rest of packet*/
```

```
            *bytes_read = 188;
            return;
          }

          /*Start of last descriptor
kept in buffer.*/
          PM_section.descriptor_start
= i-1;
        }
        else
        {
          field = 2; /* ES info.
(verifies again if there's more ES)*/
          info_start = PM_section.p;
        }
      }
      break;

    case 4: /* CRC and section END
*/

      if(PM_section.p == info_start+4)
/* CRC */
      {
        if( PM_section.bytes_left )
          {
            i = 188 - *bytes_read -
n_bytes;
            sprintf(str,"Invalid CRC
length: %d bytes ignored",i);
            Warn(out, str);
            fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
            *bytes_read = 188;
            return;
          }

        /* Read CRC_32.*/
        CRC_32 =
(double)PM_section.buffer[PM_section.p-
3] * 16777216 /* 2^24 */

+(double)PM_section.buffer[PM_section.p
-2] * 65536     /* 2^16 */

+(double)PM_section.buffer[PM_section.p
-1] * 256       /* 2^8   */

+(double)PM_section.buffer[PM_section.p
];

        /*Stuffing bytes*/
        if((n_bytes - 1) <
(pointer_field +1))
        {
          /*One section ends and
another one starts.*/

          fprintf(out,"|\n| %d bytes
left - ",pointer_field +1 - n_bytes);
          for(stuffing_bytes=0, i=0; i
< pointer_field +1 - n_bytes; i++)
            if(getc(in) == 0xFF)
stuffing_bytes++;

          n_bytes = pointer_field + 1;
          PM_section.p = 0; /*Byte 1:
PM_section.p = 1*/
        }
        else  /*The section ended and
rest of the packet is garbage.*/
        {

          for(stuffing_bytes=0, i=0; i
< 188 - n_bytes - *bytes_read; i++)
            if(getc(in) == 0xFF)
stuffing_bytes++;

          n_bytes = 188 - *bytes_read;
        }
      }
      break;
    }
  }

  *bytes_read = 188;
}


void
DESCRIPTOR(struct buffer
*stream_buffer, FILE *out)
/*************************************
 With the descriptor_tag, indexed by
stream_buffer->descriptor_start,
present
 in the buffer (see global struct)
which contains the descriptor that will
be
 parsed, this routine does the routing
to the appropriate sub-routine.

*************************************/
{
  register int i;

          if(i>=16 && i<=63 || !i ||
i==1) fprintf(out,"Reserved\n");
          else
          if(i>=64 && i<=255)
fprintf(out,"User Private\n");
          else
          fprintf(out,"ERROR!\n");
}

void
SYSTEM_CLOCK_DESCRIPTOR(struct buffer
*stream_buffer, FILE *out)
/*************************************
 Reads the descriptor info contained in
the buffer (global struct).

*************************************/
{
  register int i;

  /* descriptor_length */
  i = stream_buffer-
>buffer[stream_buffer->descriptor_start
+ 1];
  fprintf(out,"| descriptor_length =
%d\n", i);
  if(i != 2)
  {
    Warn(out,"Invalid descriptor
length");
    return;
  }

  /* Read 1 byte */
  i = stream_buffer-
>buffer[stream_buffer->descriptor_start
+ 2];
```

```
  /* external_clock_reference_indicator
*/
  fprintf(out,"|
external_clock_reference_indicator =
%d\n",(i&128) >> 7);/*mask: 1000 0000*/

  if(detail < 1)
    fprintf(out,"| reserved =
%d\n",(i&64) >> 6);/*mask: 0100 0000*/

  /* clock_accuracy_integer */
  fprintf(out,"| clock_accuracy_integer
= %d\n", i&63);/*mask: 0011 1111*/

  /* Read 1 byte */
  i = stream_buffer-
>buffer[stream_buffer->descriptor_start
+ 3];

  /* clock_accuracy_exponent */
  fprintf(out,"|
clock_accuracy_exponent = %d\n",(i&224)
>> 5);/*mask: 1110 0000*/

  if(detail < 1)
    fprintf(out,"| reserved = %d\n",
i&31);/*mask: 0001 1111*/
}

void
MULTIPLEX_BUFFER_UTILIZATION_DESCRIPTOR
(struct buffer *stream_buffer, FILE
*out)
/*************************************
 Reads the descriptor info contained in
the buffer (global struct).

*************************************/
{
  register int i;

  /* descriptor_length */
  i = stream_buffer-
>buffer[stream_buffer->descriptor_start
+ 1];
  fprintf(out,"| descriptor_length =
%d\n", i);
  if(i != 3)
  {
    Warn(out, "Invalid descriptor
length");
    return;
  }

  /* Read 1 byte */
  i = stream_buffer-
>buffer[stream_buffer->descriptor_start
+ 2];

  /* mdv_valid_flag */
  fprintf(out,"| mdv_valid_flag
= %d\n",(i&128) >> 7);/*mask: 1000
0000*/

  /* multiplex_delay_variation, read 1
more byte */
  i &= 127;    /*mask: 0111 1111*/
  i =  i * 256 /* 2^8 */
     + stream_buffer-
>buffer[stream_buffer->descriptor_start
+ 3];
  fprintf(out,"|
multiplex_delay_variation = %d\n", i);

  /* Read 1 byte */
  i = stream_buffer-
>buffer[stream_buffer->descriptor_start
+ 4];

  /* multiplex_strategy */
  fprintf(out,"| multiplex_strategy
= %d\n",(i&224) >> 5);/*mask: 1110
0000*/

  if(detail < 1)
    fprintf(out,"| reserved = %d\n",
i&31);/*mask: 0001 1111*/
}


void
MAXIMUM_BITRATE_DESCRIPTOR(struct
buffer *stream_buffer, FILE *out)
/*************************************
 Reads the descriptor info contained in
the buffer (global struct).

*************************************/
{
  register int i;
  double rate;

  /* descriptor_length */
  i = stream_buffer-
>buffer[stream_buffer->descriptor_start
+ 1];
  fprintf(out,"| descriptor_length =
%d\n", i);
  if(i != 3)
  {
    Warn(out, "Invalid descriptor
length");
    return;
  }

  /* Read 1 byte */
  i = stream_buffer-
>buffer[stream_buffer->descriptor_start
+ 2];

  if(detail < 1)
    fprintf(out,"| reserved =
%d\n",(i&192) >> 6);/*mask: 1100 0000*/

  /* maximum_bitrate, read 2 more bytes
*/
  i &= 63; /*mask: 0011 1111*/
  rate = (double)i * 65536 /* 2^16 */
     + (double)stream_buffer-
>buffer[stream_buffer->descriptor_start
+ 3] *256 /* 2^8 */
     + (double)stream_buffer-
>buffer[stream_buffer->descriptor_start
+ 4];
  fprintf(out,"| maximum_bitrate = %g
(%g bytes/s)\n", rate, rate*50);
}

void
PES_PACKET (PMT_ptr PES, int
unit_start, int *bytes_read, FILE *in,
FILE *out)
{
  register int i,
               n_bytes = 0;/* bytes
counter */
```

```
      i = 188 - *bytes_read -n_bytes;
      fseek(in, i, SEEK_CUR); /*Ignores
rest of packet*/
     *bytes_read = 188;
}


void
Warn(FILE *out, char *str)
/*************************************
 Prints a warning in the parsing and in
a log file.

*************************************/
{
  /* boolean log;
       unsigned long n_TPs;
       char no_extension[80]; GLOBAL
VAR. !*/
  char errorstr[80], s[80];
  FILE *logfile;

  if( log )
  {
    sprintf(s,"%s.LOG", no_extension);

    if((logfile=fopen(s,"at")) == NULL)
    {
        sprintf(errorstr,"\nat - can't
open %s (log file)",s);
        perror(errorstr);
        return;
    }
    sprintf(s,"%ld\t! %s\n", n_TPs + 1,
str);
    fprintf(logfile, s);

    fclose(logfile);
  }

  fprintf(out,"! ");
  fprintf(out, str);
  fprintf(out,"\n");
}
```

Anexo B
Listagem do código implementado no Simulador

Neste apêndice apresenta-se a listagem do programa, realizado em C, que foi implementado e que testa o empacotamento em AAL5 das tramas MPEG-2 com a opção *PCR-aware* e *PCR-unaware*, com diferentes valores de N (números de pacotes MPEG-2 num PDU).

Listagem do código implementado:

```
/*************************************
*************************************
 MPEG2 system bitstreams parser:
ISO/IEC 13818-1
30/07/97

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

#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "ctype.h"
#include "math.h"
#define PES_BUFFER_SIZE 470 /*Maximum
buffer size (+1) for the PES_packets
                            header
(in PMT). Could include data_bytes. */

/* /// Compilation problem in UNIX
///// */

#ifndef SEEK_SET
  #define SEEK_SET 0
#endif
#ifndef SEEK_CUR
  #define SEEK_CUR 1
#endif

/* /// Type definitions
///////////////// */
typedef enum { NO, YES } boolean;
typedef struct PA_table *PAT_ptr;
typedef struct PM_table *PMT_ptr;
typedef struct CA_table *CAT_ptr;

/* /// Global variables
///////////////// */

boolean extract,
        logg, /* Creation of a log file
with errors found in the analysed
stream
                 Will also have some
statistics (e.g. overhead).
                 See FindSwitch(),
Main(), Warn(). */
        unconditional;/* Type of PES
extraction:
                                1 - all PIDs
                                0 - after
having the PSI tables.
                               See
TRANSPORT_STREAM(), FindSwitch().
                               In case 1 the
new table may require much memory and
                               make the
parsing to abort.*/

unsigned char detail;/* Detail level in
the parsing file.
                              Initialized in
FindSwitch. */

unsigned long n_TPs; /* TP counter.
Initialized in PROGRAM_STREAM() (due to
                             Warn()).
```

```
                        Initialized and
used in TRANSPORT_STREAM(). Used in
                        Warn(); */

char no_extension[80];

struct buffer
{
  unsigned char buffer[1024]; /*must be
unsigned !*/
  int p; /* buffer pointer */
  int descriptor_start; /* buffer
pointer; start of most recent
descriptor */
  int bytes_left; /* Remaining bytes to
read in order to complete the section*/
}
  PA_section, /*Used in
TS_PROGRAM_ASSOCIATION_SECTION() */
  CA_section, /*Used in TS_CA_SECTION()
*/
  PM_section; /*Used in
TS_PROGRAM_MAP_SECTION() */

struct
{
  char buffer[4096]; /* Maximum size of
a private section */
  int p; /* buffer pointer */
  int bytes_left; /* Remaining bytes to
complete the section */
}
  private_section; /*Used in
PRIVATE_SECTION() */
struct PA_table
{
  int program_number;
  int PID;
  PAT_ptr next;
};

struct PM_table
{
  int elementary_PID; /*Initialized in
TS_PROGRAM_MAP_SECTION()*/
  char continuity_counter;/*Used in
TS_TRANSPORT_PACKET(); Init.in
TS_PROGRAM_MA
P_SECTION()*/
  boolean old_stream_removed;
/*Activated in PES_PACKET(); Init.in
TS_PROGRAM_MA
P_SECTION()*/
  unsigned char
buffer[PES_BUFFER_SIZE];/*PES_packet
header buffer */
  int p, info_start, info_end; /*
buffer pointers */
  int field; /*Which part of PES_packet
is being read*//*Init. in
TS_PROGRAM_MAP
_SECTION()*/
  unsigned flag[7]; /*PES_packet flags
buffer (helps deciding field)*/
  int packet_length; /* If null then
length is unknown */
  int bytes_left; /* Remaining bytes to
complete PES_packet*/
  int header_data_length; /* To confirm
stuffing_bytes */
  PMT_ptr next;
};

struct CA_table
{
  int CA_PID;
  CAT_ptr next;
};
PAT_ptr PAT,
        new_PAT;
PMT_ptr PMT,
        new_PMT,
        ALL;/*Used in PES extraction
without PSI tables*/
CAT_ptr CAT,
        new_CAT;

int PAT_version; /* version_number of
new_PAT */
int PMT_version; /* version_number of
new_PMT */
int CAT_version; /* version_number of
new_CAT */

/* New variables*/
int
escolha,numspts,numcelulas,PAD,contpos,
contador,teste1,teste2,teste3;
long pos,
novapos,comp,aux1,numtramas,numtramas,n
umPCR,numCSPDU;
double tamanho,media;
FILE *trama;
char *sda;
boolean PATII,PCRpres;
int i,auxf;

/* /// Function prototypes
//////////// */

void    Program_Use
(void);
void    Expands_meta_ch
(int *, char ***);
char   *Obtains_path
(char *);
void    NameOut
(char *, char *, char *);
void    FindSwitch
(int *, char ***);
void    STARTCODE_prefix
(FILE *);
void    PACK
(FILE *, FILE *);
void    SYSTEM_HEADER
(FILE *, FILE *);
void    PACKET
(int, FILE *, FILE *);
/* Nova funcao */
void    trailertrama
(int);
void    PADtrama
(int);
void    copia
(FILE  *,int,int);
boolean TRANSPORT_STREAM
(FILE *, FILE *);
void    TS_TRANSPORT_PACKET
(FILE *, FILE *);
void    TS_ADAPTATION_FIELD
(int, int *, FILE *, FILE *);
void    Warn
(FILE *, char *);
```

```
/*
////////////////////////////////////
*/
void
Program_Use()
/************************************
 ISO 13818-1 streams parser.

************************************/
{
  fprintf(stderr,"----------------------
----------------------------------------
-\n");
  fprintf(stderr,"MPEG2 sistemas:
ISO/IEC 13818-1 \n");
  fprintf(stderr,"\nEntrada um ficheiro
SPTS\n");
  fprintf(stderr,"Saida um ficheiro CS-
PDU, aal5.spts\n");
  fprintf(stderr,"----------------------
----------------------------------------
-\n");
  exit(0);
}

void
NameOut (char *output, char *input,
char *executable_name)
/************************************
 Obtains the output filename by
adding/modifying an extension to the
input.
 The extension is made from the first 3
(or less) letters of the program name
 without the path.

************************************/
{
    register int i, j;
    char extension[80];
    strcpy(extension, executable_name);
    for(i = strlen(extension) - 1;
        i>=0 && extension[i]!='\\' &&
extension[i]!='/' && extension[i]!=':';
        i--) ;
    extension[4] = '\0';
    extension[0] = '.';
    for(j=1; j<4; j++)
    {
      extension[j] =
executable_name[i+j];
      if(extension[j]=='.')
      {
        extension[j]='\0';
        break;
      }
    }
    strcpy(output, input);
    for(i=strlen(output)-1; i>=0; i--)
    {
      if(output[i]=='.')
      {
        output[i]='\0';
        break;
      }
    }
    /*Global variable initialization.*/
    strcpy(no_extension, output);
    strcat(output, extension);
}
void
FindSwitch(int *argc, char **argv[])
/************************************
 Finds (and removes), in the command
line arguments, instructions to define
the
 program behaviour.

************************************/
{
  register int i,
               new_argc=0;
  char **argv_buffer = (char
**)malloc(*argc * sizeof(char *));
  /* Switches initialization (defaults)
*/
  extract = YES;
  unconditional = NO;
  logg = NO;
  detail = 1;
  /* Search */
  for(i=0; i < *argc; i++)
  {
    if( (*argv)[i][0] == '-') /* switch
indicator */
    {
        fprintf(stderr,"\nInvalid
switch !");
        Program_Use();
    }
    else
    {
      argv_buffer[new_argc] = (char
*)malloc(80 * sizeof(char));
      strcpy(argv_buffer[new_argc++],
(*argv)[i]);
    }
  }
  *argc = new_argc;
  *argv = argv_buffer;
}

main(int argc, char *argv[])
{
  FILE *in, *out ;
  char str[80], s[80];
  int file;
  FindSwitch(&argc, &argv); /*
Initializes global variables according
to the
                                command
line.*/
  if(argc < 2) Program_Use();
  for(file=1; file < argc; file++)
  {
    if((in=fopen(argv[file],"rb")) ==
NULL)
    {
      sprintf(str,"\nrb - can't open
%s", argv[file]);
      perror(str);
      continue;
    }
   NameOut(str, argv[file], argv[0]);
   sda="aal5.spts";
   if((out=fopen(str,"wt")) == NULL)
    {
      sprintf(s,"\nwt - can't open %s",
str);
      perror(s);
      fclose(trama);
      fclose(in);
      continue;
    }
    if (setvbuf(in, NULL, _IOFBF,
15360) != 0)
```

```
      fprintf(stderr,"\n(failed to set
up buffer for input file)");
    if (setvbuf(out, NULL, _IOFBF,
15360) != 0)
      fprintf(stderr,"\n(failed to set
up buffer for output file)");
    fprintf(stdout,"\n----------------
---------------------------------
\n");
    fprintf(stdout,"\n A processar a
informacao de %s para %s\n
   Ficheiro CS-PDU : aal5.spts",
argv[file], str);
    fprintf(stdout,"\n----------------
---------------------------------
\n");
    fprintf(out,"%s
parse:",argv[file]);
    if( logg ) /* file with errors and
stream statistics */
    {
      sprintf(s,"%s.LOG",
no_extension);
      if(remove(s))
      {
       sprintf(str,"(can't remove %s
(log file))", s);
       perror(str);
      }
    }


PCRpres=numtramas=numPCR=numCSPDU=0;
      media=0;
      contador=numspts;
      printf("\n Choose the type " );
      printf("(1-PCR aware  2-Non PCR
aware):");
      scanf("%d",&escolha);
      if (escolha!=2) escolha==1 ;
      if ( escolha==1) printf("\n  PCR
aware \n ");
      else printf( " \n Non PCR aware
\n");
     printf("\n Choose now the number
of SPTS(entre 1 e 348): ");
     scanf("%d",&numspts);
     printf("\nNumber of SPTS: %d\n
",numspts);
     printf("\nProcessing....\n");
       if( !TRANSPORT_STREAM(in, out) )
        fprintf(stderr,"\n%s isn't a
valid ISO13818-1 bitstream
!\n",argv[file]);
    printf("\nStreams read %d Streams
with %d PCR Written %d CS-
PDU",numtramas,numPCR,numCSPDU );
   if (escolha==2) printf("\n Mean
deviation of PCR  %f   in 188
bytes",media/numPCR);
   else printf("\n  PCR exception: %d
",teste1);
    printf("\n Process ended.\n");

    fclose(trama);
    fclose(in);
    fclose(out);
  }

  return 0;
}

void
copia(FILE *in,int pos,int comp)
{
  char str[80], s[80];
  long aux;

  aux=pos;
  fseek(in,aux,SEEK_SET);
  aux1=comp;
  while(aux1--)putc(getc(in),trama);
}

void
PADtrama(int numero)
{
 double x;
 tamanho=(double)numero*188+8;
 x=tamanho/48;
 numcelulas=ceil(x);
 PAD=numcelulas*48-tamanho;
// printf("\n num celulas%dPAD %d,\n
",numcelulas,PAD);
 if (PAD!=0) while(PAD--)
putc(0x0,trama);
}
void
trailertrama(int numero)
{
union
{
 char byte[2];
 int  comp;
} trail;
trail.comp=numero;
putc(0x0,trama);
putc(0x0,trama);
putc(trail.byte[1],trama);
putc(trail.byte[2],trama);
putc(0x0,trama);
putc(0x0,trama);
putc(0x0,trama);
putc(0x0,trama);
}

boolean
TRANSPORT_STREAM (FILE *in, FILE *out)
/*************************************
 In the Transport Packets (TPs)
identification I assume that there can
be trash
 between packets and without
synchronism byte emulation.
 Returns YES or NO depending on a valid
stream parsing.
 Counts the number of processed
packets.
 Initializes the PSI (Program Specific
Information) linked list pointers
 (global variables). An empty table is
a one element list.

*************************************/
{
  boolean aux; PAT_ptr pat_aux; PMT_ptr
pmt_aux; CAT_ptr cat_aux;/*Auxiliary*/
  while(getc(in) != 0x47)  if(feof(in))
return NO; /* Luck ? */
  if(
(new_PAT=(PAT_ptr)malloc(sizeof(struct
PA_table))) == NULL )
   {puts("TRANSPORT_STREAM(): malloc
error"); exit(1);}
  new_PAT->next=NULL;
```

```
  if(
(new_PMT=(PMT_ptr)malloc(sizeof(struct
PM_table))) == NULL )
   {puts("TRANSPORT_STREAM(): malloc
error"); exit(1);}
  new_PMT->next=NULL;
  if(
(new_CAT=(CAT_ptr)malloc(sizeof(struct
CA_table))) == NULL )
   {puts("TRANSPORT_STREAM(): malloc
error"); exit(1);}
  new_CAT->next=NULL;
  PAT = new_PAT; /* The current table
is new */
  PMT = new_PMT;
  CAT = new_CAT;
  PAT_version = 0;
  PMT_version = 0;
  CAT_version = 0;
  if( unconditional )
  {
    if(
(ALL=(PMT_ptr)malloc(sizeof(struct
PM_table))) == NULL )
    {puts("TRANSPORT_STREAM(): malloc
error"); exit(1);}
    ALL->next=NULL;
  }
  else ALL = NULL;
  TS_TRANSPORT_PACKET(in, out);
  numtramas=1;
  contador=numspts;
  trama=fopen(sda,"wb");
  n_TPs = 1;
  teste1=teste2=teste3;
  while( !feof(in) ) if(getc(in) ==
0x47)
                     {
                       if (
escolha==1) /* PCR aware */
                       {
                        pos=ftell(in);
                        novapos=pos-
188;

copia(in,novapos,188);

fseek(in,pos,SEEK_SET);
                        contador--;
                        if (PCRpres)
                             {

fprintf(out,"  %d \n",contador);

PCRpres=0;
                                 if
(contador!=0){

teste1++;

}

auxf=numspts-contador;

PADtrama(auxf);

trailertrama(auxf*188);

contador=numspts;

numCSPDU++;
                                    }
                                    /* igual ao Non
 PCR aware */

 if(contador==0){ PADtrama(numspts);

 trailertrama(numspts*188);

 numCSPDU++;

 contador=numspts;

 }
                                    }

                              if ( escolha==2) /*
Non PCR aware */
                                 {
                                  pos=ftell(in);
                                  novapos=pos-
188;

copia(in,novapos,188);

fseek(in,pos,SEEK_SET);
                                 contador--;
                                  if (PCRpres){

media=media+contador;

fprintf(out,"   %d \n",contador);

PCRpres=0;
                                                }

if(contador==0){ PADtrama(numspts);

trailertrama(numspts*188);

numCSPDU++;

contador=numspts;
                                                }
                                 }
                                numtramas++;

TS_TRANSPORT_PACKET(in, out);
                     }
 /* permite copiar a ultima trama */
                     pos=ftell(in);
                     novapos=pos-188;

copia(in,novapos,188);

fseek(in,pos,SEEK_SET);
                     contador--;

                     auxf=numspts-
contador;
                     PADtrama(auxf);

trailertrama(auxf*188);
                     numCSPDU++;
  printf("\a");
  /* Memory release */
  if(ALL != NULL)
  {
    while(ALL->next != NULL)
    {
      pmt_aux = ALL->next;
      free(ALL);
      ALL = pmt_aux;
    }
```

```
      free(ALL);
    }
    if(PAT == new_PAT) aux = NO;
    else aux = YES;
    if(PAT != NULL)
    {
      while(PAT->next != NULL)
      {
        pat_aux = PAT->next;
        free(PAT);
        PAT = pat_aux;
      }
      free(PAT);
    }
    if(aux && new_PAT != NULL)
    {
      while(new_PAT->next != NULL)
      {
        pat_aux = new_PAT->next;
        free(new_PAT);
        new_PAT = pat_aux;
      }
      free(new_PAT);
    }
    if(PMT == new_PMT) aux = NO;
    else aux = YES;
    if(PMT != NULL)
    {
      while(PMT->next != NULL)
      {
        pmt_aux = PMT->next;
        free(PMT);
        PMT = pmt_aux;
      }
      free(PMT);
    }
    if(aux && new_PMT != NULL)
    {
      while(new_PMT->next != NULL)
      {
        pmt_aux = new_PMT->next;
        free(new_PMT);
        new_PMT = pmt_aux;
      }
      free(new_PMT);
    }
    if(CAT == new_CAT) aux = NO;
    else aux = YES;
    if(CAT != NULL)
    {
      while(CAT->next != NULL)
      {
        cat_aux = CAT->next;
        free(CAT);
        CAT = cat_aux;
      }
      free(CAT);
    }
    if(aux && new_CAT != NULL)
    {
      while(new_CAT->next != NULL)
      {
        cat_aux = new_CAT->next;
        free(new_CAT);
        new_CAT = cat_aux;
      }
      free(new_CAT);
    }
  return YES;
}
void
TS_TRANSPORT_PACKET (FILE *in, FILE
*out)
```

```
/*************************************
 sync_byte = 1 byte (already read).

 transport_error_indicator = 1 bit
\
 payload_unit_start_indicator = 1 bit
\
 transport_priority = 1 bit
\
 PID = 13 bits
> 3 bytes.
 transport_scrambling_control = 2 bits
/
 adaptation_field_control = 2 bits
/
 continuity_counter = 4 bits
/

 adaptation_field and/or data_bytes =
until 184 bytes.


*************************************/
{
  struct header_bits
  {
    unsigned transport_error_indicator
:  1;
    unsigned
payload_unit_start_indicator :  1;
    unsigned transport_priority
:  1;
    unsigned PID_12_8
:  5;
    unsigned PID_7_0
:  8;
    unsigned
transport_scrambling_control :  2;
    unsigned adaptation_field_control
:  2;
    unsigned continuity_counter
:  4;
  };

  union
  {
    struct header_bits bits;
    char byte[3];
  } header;

  int PID;
  register int i;
  char str[80];
  int bytes_read;
  int PM_flag;
  /*Read 3 bytes of TP header.*/
  for(i=0; i<3; i++)
header.byte[i]=(char)getc(in);
  if( feof(in) )
  {
    Warn(out,"Premature EOF in TP
header");
    return;
  }
  bytes_read = 4;
  /* Packet Identifier */
  PID = header.bits.PID_12_8 * 256
      +header.bits.PID_7_0;
  if(PID == 0x1FFF)
  {
    i = 188 - bytes_read;
    fseek(in, i, SEEK_CUR); /*Ignores
rest of packet*/
```

```
    bytes_read = 188;
    return;
  }

  /* adaptation_field */
  i =
header.bits.adaptation_field_control;
  if( i == 2  ||  i == 3 )
TS_ADAPTATION_FIELD(i, &bytes_read, in,
out);
  /* payload */
 if(
header.bits.adaptation_field_control ==
1 ||

header.bits.adaptation_field_control ==
3    )
  {
    if(bytes_read >= 188) Warn(out,"No
space for payload");
    else
    {
              i = 188 - bytes_read;
              fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
              bytes_read = 188;
    }
  }
}

void
TS_ADAPTATION_FIELD (int control, int
*bytes_read, FILE *in, FILE *out)
/*************************************
 adaptation_field_length = 1 byte

 descontinuity_indicator = 1 bit
\
 random_access_indicator = 1 bit
\
 elementary_stream_priority_indicator =
1 bit   \
 PCR_flag = 1 bit
> 1 byte.
 OPCR_flag = 1 bit
/
 splicing_point_flag = 1 bit
/
 transport_private_data_flag = 1 bit
/
 adaptation_field_extension_flag = 1
bit     /

 program_clock_reference_base = 33 bits
\
 reserved = 6 bits
> 6 optional bytes.
 program_clock_reference_extension = 9
bits /

 original_program_clock_reference_base
= 33 bits     \
 reserved = 6 bits
> 6 optional bytes.

original_program_clock_reference_extens
ion = 9 bits /

 splice_countdown = 1 optional byte.

 transport_private_data_length = 1
optional byte.
 private_data_byte = n optional bytes.

 adaptation_field_extension_length = 8
bits \
 ltw_flag = 1 bit
\
 piecewise_rate_flag = 1 bit
> 2 optional bytes.
 seamless_splice_flag = 1 bit
/
 reserved = 5 bits
/

 stuffing_byte = n optional bytes.


*************************************/
{
  struct flag_bits
  {
    unsigned descontinuity_indicator
:  1;
    unsigned random_access_indicator
:  1;
    unsigned
elementary_stream_priority_indicator :
1;
    unsigned PCR_flag
:  1;
    unsigned OPCR_flag
:  1;
    unsigned splicing_point_flag
:  1;
    unsigned
transport_private_data_flag          :
1;
    unsigned
adaptation_field_extension_flag      :
1;
  };

  union
  {
    struct flag_bits bits;
    char byte;
  } flags;

  struct PCR_bits /* Used for PCR and
OPCR */
  {
    unsigned PCR_base_32_25     :  8;

    unsigned PCR_base_24_17     :  8;

    unsigned PCR_base_16_9      :  8;

    unsigned PCR_base_8_1       :  8;
    unsigned PCR_base_0         :  1;
    unsigned reserved           :  6;
    unsigned PCR_extension_8    :  1;
    unsigned PCR_extension_7_0  :  8;
  };

  union
  {
    struct PCR_bits bits;
    char byte[6];
  } PCR;

  struct extension_flag_bits
  {
    unsigned ltw_flag               :
1;
```

```
    unsigned piecewise_rate_flag    :
1;
    unsigned seamless_splice_flag   :
1;
    unsigned reserved               :
5;
  };

  union
  {
    struct extension_flag_bits bits;
    char byte;
  } extension;

  struct ltw_bits
  {
    unsigned ltw_valid_flag  :  1;
    unsigned ltw_offset_14_8 :  7;
    unsigned ltw_offset_7_0  :  8;
  };

  union
  {
    struct ltw_bits bits;
    char byte[2];
  } ltw;

  struct piecewise_rate_bits
  {
    unsigned reserved              :  2;
    unsigned piecewise_rate_21_16 :  6;
    unsigned piecewise_rate_15_8   :  8;
    unsigned piecewise_rate_7_0    :  8;
  };

  union
  {
    struct piecewise_rate_bits bits;
    char byte[2];
  } piecewise_rate;

  struct seamless_splice_bits
  {
    unsigned splice_type        :  4;
    unsigned DTS_next_au_32_30 :  3;
    unsigned marker_bit_1       :  1;
    unsigned DTS_next_au_29_22 :  8;
    unsigned DTS_next_au_21_15 :  7;
    unsigned marker_bit_2       :  1;
    unsigned DTS_next_au_14_8  :  8;
    unsigned DTS_next_au_7_0    :  7;
    unsigned marker_bit_3       :  1;
  };

  union
  {
    struct seamless_splice_bits bits;
    char byte[5];
  } splice;

  int    adaptation_field_length;
  double PCR_base, PCR_extension,
         OPCR_base, OPCR_extension;
  int    splice_countdown,
         transport_private_data_length,

adaptation_field_extension_length,
         stuffing_bytes;
  char str[80];
  register int i,
               n_bytes;/*counter of
bytes in the adaptation_field*/
  /*Read adaptation_field_length.*/
    adaptation_field_length = getc(in);
    if( feof(in) )
    {
      Warn(out,"Premature EOF in
adaptation_field_length");
      exit(1);
    }
    n_bytes = 1;
    if(control == 3 &&
(adaptation_field_length > 183 ||
adaptation_field_length < 0) ||
       control == 2 &&
adaptation_field_length != 183 )
    {
      i = 188 - *bytes_read -n_bytes;
      sprintf(str,"Invalid length: %d
bytes ignored",i);
      Warn(out, str);
      fseek(in, i, SEEK_CUR); /*Ignores
rest of packet*/
      *bytes_read = 188;
      return;
    }
    if(!adaptation_field_length) /*
single stuffing_byte*/
    {
      *bytes_read += n_bytes;
      return;
    }

    /*Read flags byte.*/
    flags.byte = (char)getc(in);
    if( feof(in) )
    {
      Warn(out,"Premature EOF in flags");
      exit(1);
    }
    n_bytes++;

    /* descontinuity_indicator */
    i =
flags.bits.descontinuity_indicator;
    /* PCR_flag */
    i = flags.bits.PCR_flag;
    /* OPCR_flag */
    i = flags.bits.OPCR_flag;
    /* Program Clock Reference */
    if( flags.bits.PCR_flag )
    {
   PCRpres=1;
   numPCR++;
   fprintf(out,"\n   %d",numPCR);
    /*Read 6 bytes of PCR.*/
      for(i=0; i<6; i++)
PCR.byte[i]=(char)getc(in);
      if( feof(in) )
      {
        Warn(out,"Premature EOF in PCR");
        exit(1);
      }
      n_bytes += 6;

      PCR_base =
(double)PCR.bits.PCR_base_32_25 *
33554432

+(double)PCR.bits.PCR_base_24_17 *
131072

+(double)PCR.bits.PCR_base_16_9  * 512

+(double)PCR.bits.PCR_base_8_1   * 2
```

```
+(double)PCR.bits.PCR_base_0;
    PCR_extension =
(double)PCR.bits.PCR_extension_8 * 256

+(double)PCR.bits.PCR_extension_7_0;

  }

  /* Original Program Clock Reference
*/
  if( flags.bits.OPCR_flag )
  {
    /*Read 6 bytes of OPCR.*/
    for(i=0; i<6; i++)
PCR.byte[i]=(char)getc(in);
    if( feof(in) )
    {
      Warn(out,"Premature EOF in
OPCR");
      exit(1);
    }
    n_bytes += 6;

    OPCR_base =
(double)PCR.bits.PCR_base_32_25 *
33554432

+(double)PCR.bits.PCR_base_24_17 *
131072

+(double)PCR.bits.PCR_base_16_9  * 512

+(double)PCR.bits.PCR_base_8_1   * 2

+(double)PCR.bits.PCR_base_0;
    OPCR_extension =
(double)PCR.bits.PCR_extension_8 * 256

+(double)PCR.bits.PCR_extension_7_0;
  }

  if( flags.bits.splicing_point_flag )
  {
    splice_countdown = getc(in);
    if( feof(in) )
    {
      Warn(out,"Premature EOF in
splice_countdown");
      exit(1);
    }
    n_bytes++;
  }
  /*Read private data.*/
  if(
flags.bits.transport_private_data_flag
)
  {
    /*Read length*/
    transport_private_data_length =
getc(in);
    if( feof(in) )
    {
      Warn(out,"Premature EOF in
transport_private_data_length");
      exit(1);
    }
    n_bytes++;
    if((transport_private_data_length +
n_bytes -1) > adaptation_field_length)
    {
      i = 188 - *bytes_read -n_bytes;
      sprintf(str,"Invalid length: %d
bytes ignored",i);
      Warn(out, str);
      fseek(in, i, SEEK_CUR); /*Ignores
rest of packet*/
      *bytes_read = 188;
      return;
    }

    /*Read data*/
    fseek(in,
transport_private_data_length,
SEEK_CUR);
    if( feof(in) )
    {
      Warn(out,"Premature EOF in
private data");
      exit(1);
    }
    n_bytes +=
transport_private_data_length;
  }

  /*Read adaptation_field extension.*/
  if(
flags.bits.adaptation_field_extension_f
lag )
  {
    /*Read length*/
    adaptation_field_extension_length =
getc(in);
    if( feof(in) )
    {
      Warn(out,"Premature EOF in
adaptation_field_extension_length");
      exit(1);
    }
    n_bytes++;

if((adaptation_field_extension_length +
n_bytes -1) > adaptation_field_length)
    {
      i = 188 - *bytes_read -n_bytes;
      sprintf(str,"Invalid length: %d
bytes ignored",i);
      Warn(out, str);
      fseek(in, i, SEEK_CUR); /*Ignores
rest of packet*/
      *bytes_read = 188;
      return;
    }


if(adaptation_field_extension_length >
0)
    {
      /*Read extension flags.*/
      extension.byte = getc(in);
      if( feof(in) )
      {
        Warn(out,"Premature EOF in
extension flags");
        exit(1);
      }
      n_bytes++;

adaptation_field_extension_length--;
      /* Legal Time Window */
      if(extension.bits.ltw_flag)
      {
        /*Read 2 bytes of ltw*/
        ltw.byte[0] = getc(in);
        ltw.byte[1] = getc(in);
```

```
        if( feof(in) )
        {
          Warn(out,"Premature EOF in
ltw");
          exit(1);
        }
        n_bytes += 2;
        adaptation_field_extension_lengt
h -= 2;

      }

      /* Piecewise Rate */

if(extension.bits.piecewise_rate_flag)
      {
        /*Read 3 bytes of
piecewise_rate*/
        for(i=0; i<3; i++)
piecewise_rate.byte[i]=(char)getc(in);
        if( feof(in) )
        {
          Warn(out,"Premature EOF in
piecewise_rate");
          exit(1);
        }
        n_bytes += 3;
        adaptation_field_extension_lengt
h -= 3;

      }

      /* Seamless Splice */

if(extension.bits.seamless_splice_flag)
      {
        /*Read 5 bytes.*/
        for(i=0; i<5; i++)
splice.byte[i]=(char)getc(in);
        if( feof(in) )
        {
          Warn(out,"Premature EOF
prematuro in Seamless Splice");
          exit(1);
        }
        n_bytes += 5;
        adaptation_field_extension_lengt
h -= 5;

        if( !splice.bits.marker_bit_1
||
            !splice.bits.marker_bit_2
||
            !splice.bits.marker_bit_3
)
        {
          i = 188 - *bytes_read -
n_bytes;
          sprintf(str,"Splice: invalid
marker(s) - %d bytes ignored",i);
          Warn(out, str);
          fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
          *bytes_read = 188;
          return;
        }
      }

if(adaptation_field_extension_length <
0)
      {
        i = 188 - *bytes_read -n_bytes;
        sprintf(str,"adaptation_field_ex
tension length doesn't fit: %d bytes
ignored",i);
        Warn(out, str);
        fseek(in, i, SEEK_CUR);
/*Ignores rest of packet*/
        *bytes_read = 188;
        return;
      }
      fseek(in,
adaptation_field_extension_length,
SEEK_CUR);
      if( feof(in) )
      {
        Warn(out,"Premature EOF in
reserved bytes");
        exit(1);
      }
      n_bytes +=
adaptation_field_extension_length;
    }
  }

  for(stuffing_bytes=0, i=0; i <
adaptation_field_length +1 - n_bytes;
i++)
    if(getc(in) == 0xFF)
stuffing_bytes++;

  *bytes_read +=
(adaptation_field_length +1);
}


void
Warn(FILE *out, char *str)
/*************************************
 Prints a warning in the parsing and in
a log file.

*************************************/
{
  /* boolean logg;
       unsigned long n_TPs;
       char no_extension[80]; GLOBAL
VAR. !*/
  char errorstr[80], s[80];
  FILE *logfile;

  if( logg )
  {
    sprintf(s,"%s.LOG", no_extension);

    if((logfile=fopen(s,"at")) == NULL)
    {
        sprintf(errorstr,"\nat - can't
open %s (log file)",s);
        perror(errorstr);
        return;
    }
    sprintf(s,"%ld\t! %s\n", n_TPs + 1,
str);
    fprintf(logfile, s);

    fclose(logfile);
  }

  fprintf(out,"! ");
  fprintf(out, str);
  fprintf(out,"\n");
}
```